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

GENEROVÁNÍ PROCEDURÁLNÍCH PLANET

PROCEDURAL GENERATION OF PLANETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR FUSEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ STARKA,**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student:	**Fusek Petr**
Program:	Informační technologie
Název:	**Proceduralní generování planet** **Generating Procedural Planets**
Kategorie:	Počítačová grafika

Zadání:

1. Nastudujte problematiku generování planet a knihovny potřebné k práci.
2. Navrhněte knihovnu pro proceduralní generování planet.
3. Implementujte knihovnu.
4. Implementujte demonstrační aplikaci.
5. Vytvořte plakát nebo krátké video demonstrující práci.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a část bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Starka Tomáš, Ing.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2019
Datum odevzdání:	28. května 2020
Datum schválení:	17. března 2020

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací jednoduše rozšiřitelné knihovny pro vytváření generátorů procedurálních povrchů planet. Cílem knihovny je umožnit použití původně dvourozměrných metod procedurálního generování map pro generování kontextuálně bohatého povrchu planet. Knihovna klade důraz na rozšiřitelnost a jednoduchost práce s generovaným povrchem. Umožňuje uživateli si vytvářet a parametrizovat vlastní generátory a využívat v nich vlastní algoritmy. Obsahuje také implementovaný generátor planetární výškové mapy, který využívá zjednodušeného modelu simulace kolizí tektonických plátů a tím generuje povrch s topologií obsahující pohoří, zálivy a souostroví. Takovýto povrch by měl vykazovat vizuální výsledky bližší realitě, nežli umožňuje klasický přístup s užitím procedurálních šumů. Knihovna je implementována společně s vizualizační aplikací prezentující vygenerované povrchy a umožňuje nastavit všechny možné vstupy generátoru pomocí GUI.

Abstract

This bachelor's thesis deals with the design and implementation of an easily extensible library for creating generators of procedural planet surfaces. The aim of the library is to enable the use of originally two-dimensional methods of procedural map generation to generate a contextually rich planet surface. The library emphasizes the extensibility and simplicity of working with the generated surface. It allows the user to create and parameterize their own generators and use their own algorithms in them. It also includes an implemented planetary elevation map generator that uses a simplified model of simulating tectonic plate collisions to generate a surface with a topology containing mountains, bays and archipelagos. Such a surface should show visual results closer to reality than the classical approach using procedural noises allows. The library is implemented together with a visualization application presenting the generated surfaces and allowing to set all possible inputs of the generator using the GUI.

Klíčová slova

cubemapy, procedurální mapy, procedurální generování, procedurální textury, planety, planetární povrchy, planetární terén, tektonika, Voroného diagramy, C++, OpenGL

Keywords

cubemaps, procedural maps, procedural generation, procedural textures, planets, planetary surfaces, planetary terrain, tectonics, Voronoi diagrams, C++, OpenGL

Citace

FUSEK, Petr. *Generování procedurálních planet.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka,

Generování procedurálních planet

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Starky. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Fusek
28. května 2020

Poděkování

Velice rád bych poděkoval Ing. Tomáši Starkovi za odbornou pomoc a jeho vedení při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Vizualizace vesmírných těles je součástí počítačové grafiky prakticky od jejích počátků, kdy Jim Blinn na přelomu sedmdesátých a osmdesátých let vytvořil slavné video průletu sondy Voyager 1 kolem Jupiteru a Saturnu[1]. Od té doby se toto téma v grafice neustále rozšiřuje a vyvíjí. Dnes je populárnější než kdy dřív hlavně díky rostoucímu počtu počítačových her s vesmírnou tématikou.

Technologie se v profesionální sféře zaměřuje na vizualizaci a zpracování reálných dat především v astronomii. Pro samotnou vizualizaci existují technologie v podobě virtuálních glóbů, které vykreslují velké množství geografických map na virtuální interaktivní planetě. Nejznámější je aplikace *Google Earth*, která prezentuje planetu zemi, lze si ale zobrazit i povrchy Měsíce či Marsu.

Velká herní scéna podporuje vývoj technologií pro vizualizaci planet, jejich generování a interakci hráčů s nimi. Koncept otevřených her, kde hráč prozkoumává nekonečný procedurálně generovaný svět, je v celku běžný a nejvíce jej zpopularizovala hra *Minecraft*. Procedurální generování planet povyšuje myšlenku prozkoumávání jednoho nekonečného světa na prozkoumávání nekonečného množství konečných světů a přidává tak nový rozměr, což je ve své podstatě bližší realitě našeho světa. Herní vývojáři proto vyvíjí a vylepšují metody, jak procedurálně vygenerovat a vizualizovat interaktivní planetu, a téma se stává velice širokým.

Planety mohou být celé světy, na které je možno přistát a prozkoumávat (např. *No Man's Sky*). Mohou představovat herní mechaniku, kde planeta není ve skutečném měřítku, neobsahuje úroveň detailů a hra se odehrává na povrchu koule, kde můžeme využít její topologie (např. *Planetary Annihilation*). Planety mohou být využity jako celistvé herní prvky (uzly), které slouží k interakci s hráčem, třeba v různých *real-time* strategiích (např. *Stellaris*). Nebo kombinace zmíněných (např. *Spore*). Ve všech případech je ale nutné nějakým způsobem planety vykreslovat a ve většině případů i náhodně generovat.

Způsoby procedurální vygenerování planet se odvíjí od požadavků, které jsou na výsledek kladeny. Pokud je potřeba planeta o (částečně) realistické velikosti, na kterou je možno sestoupit a pohybovat se po jejím povrchu, řešení bude spíše používat techniky pro generování terénů a krajin přizpůsobené na povrch koule. Nezbytností bude také implementace nějakého *level of detail* (*LOD*) algoritmu. Proto většina řešení pro generování terénu používá nějaké náhodné šumové funkce, které se dobře škálují napříč měřítky. Celkový zjev tělesa z dálky většinou reflektuje skutečnou podobu jeho povrchu a výsledný dojem je tedy

[1] Více na Blinnových osobních stránkách: *Jim Blinn's Web Corner*, Url: http://www.jimblinn.com/ [Navštíveno 19. 5. 2020]

závislý na metodě generování terénu. Při použití šumových funkcí dostaneme většinou generický náhodně vypadající kopcovitý povrch s nejvyšším bodem vždy uprostřed ostrova či kontinentu. Užívají se ovšem i složitější postupy a jejich kombinace pro vytvoření co nejkvalitnější procedurálně generované planety. Většina výzkumu a prací je zaměřena právě na tuto oblast, jelikož takovéto modely planet jsou velice rozšířené a populární, lépe zachycují realitu, jsou do velké míry interaktivní a často působí až „ohromujícím“ dojmem.

Pokud planety budou viděny pouze z vesmíru jako celek, či jen jejich obrázky, většina technických problémů okolo geometrie odpadá a prioritu získá celkový dojem. Generování planet zde znamená vytvoření hezké textury či vizualizace koule tak, aby při nasvícení hvězdou připomínala skutečnou planetu. Tak jako ve výše zmíněném průletu Voyageru sluneční soustavou. Tato oblast se většinou zaměřuje na realistickou topografii kontinentů, pohoří a oceánů, atmosférické jevy a vliv klimatu na povrchu v různých zeměpisných šířkách (lesy, pouště, ledovce a pod.). Zde se mohou aplikovat metody pro generování herních map. Mapa může být vygenerována na povrchu koule, a tím vytvořit koherentní herní svět s biomy dávajícími smysl v jeho celém kontextu, i když nemusí bý stěžejní součástí herní mechaniky (např. hra *RimWorld*).

Cílem této práce je navrhnout knihovnu, která umožní jednoduše implementaci dvourozměrných diskrétních technik procedurálního generování map na povrchu koule. Její součástí bude datová struktura popisující diskrétní povrch na kouli včetně souřadného systému řešícího kulatou topologii. Dále bude obsahovat možnost vytvářet modulární a rozšiřitelné generátory procedurálních povrchů nad touto strukturou. Bude zahrnovat i implementaci generátoru výškové mapy pomocí simulace kolizí tektonických plátů. Výšková mapa by měla mít řadu vizuálních výhod oproti klasickému přístupu za použití šumových funkcí. Současně implementovaná demonstrační aplikace bude sloužit pro vizualizaci povrchu produkovaného knihovnou a umožní nastavení všech jejích možných vstupů a konfigurací.

Struktura práce je následující: Druhá kapitola je věnována obecnému přehledu planet v počítačové grafice. Stručně je zde rozebráno, co všechno tato problematika většinou obnáší. Kapitola třetí rozebírá problematiku generování map na planetách a techniky, které jsou v tomto směru používané. Popsány jsou metody, které bude možno využít v implementaci knihovny a aplikace. Čtvrtá kapitola popisuje návrh knihovny a její teoretický princip. Je zde popsána datová struktura pro práci s povrchem a s tím související souřadný systém. Dále je navržen koncept modulárního generátoru procedurálních povrchů a princip generátoru planetárních výškových map, který je součástí knihovny. Pátá kapitola podrobně rozebírá implementaci knihovny a aplikace. Popisuje objektový návrh knihovny jakožto aplikačního rozhraní, se kterým uživatel bude pracovat. Dále jsou popsány zvolené technologie a principy vizualizace povrchu v demonstrační aplikaci. Poslední část se věnuje konkrétní implementaci procedurálního generátoru výškové mapy a demonstruje dosažené výsledky. Na závěr jsou výsledky práce zhodnoceny a jsou navrhnuta témata k jejímu pokračování. Zmíněna jsou i možná praktická užití knihovny a metod zde navržených.

Kapitola 2

Planety v počítačové grafice

Tato kapitola se zaměřuje na problematiku okolo planet v grafice obecně. Výzkum na toto téma je obsáhlý a zasahuje do mnoha jiných odvětví počítačové grafiky. Cílem kapitoly je probrat existující metody, které by mohly být relevantní v kontextu vizualizace planety a procedurálního generování jejího povrchu jako celku. Primárním zdrojem pro tuto tématiku byla kniha *3D Engine Design for Virtual Globes* [6]. Je zde věnována pozornost reprezentaci planety jako koule a jejího vykreslování a jsou popsány metody užívané při procedurálním generování planet.

2.1 Reprezentace planety v počítači

Planety v realitě nejsou dokonalé koule, ale vlivem nerovnoměrných hustot jejich objemu a rotaci vzniká nepravidelné těleso nazvané geoid vytvořené nerovnoměrným gravitačním působením [26]. Ve většině grafických aplikací mimo vědecký výzkum tento fakt není příliš významný a běžně se aproximuje na pravidelnou kouli, se kterou je práce daleko jednodušší a výpočetně efektivnější. Výsledky jsou téměř vždy dostatečné. Výjimkou mohou být virtuální glóby a navigační systémy zobrazující skutečná geografická data, jejichž reprezentace planety musí odpovídat realitě, v takovém případě je nutné správě implementovat její skutečnou geometrii [6].

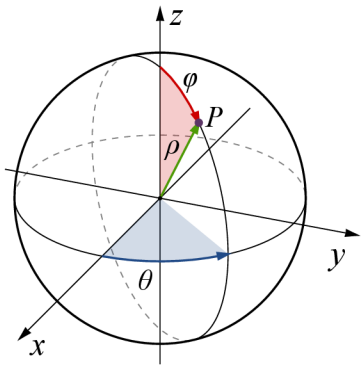
Soustava souřadnic

Pro geograficky přesný popis Země je používaným standardem *World Geodetic System 1984* (*WGS84*), který je definován třemi osami se středem v zemském těžišti [16]. Planeta země je zde reprezentována pomocí elipsoidu. V této práci bude ale dostačovat reprezentace v podobě koule, jelikož jejím cílem není zobrazovat žádná reálná data.

Relevantní budou dvě soustavy trojrozměrných souřadnic, kartézská a sférická. Kartézská soustava souřadnic se klasicky skládá ze tří vzájemně kolmých os x, y a z se stejným měřítkem a lineární stupnicí. Sférická, nebo také kulová, soustava souřadnic slouží pro určení bodu v trojrozměrném prostoru pomocí vzdálenosti ρ od středu a dvou úhlů θ a ϕ. Hodí se především pro popis pohybu na povrchu koule, kde při známém průměru lze jakoukoliv pozici na povrchu zapsat jako dvourozměrný vektor (θ, ϕ). V podstatě jde o trojrozměrné doplnění polární soustavy souřadnic (o úhel ϕ).

Bod v prostoru P je definován jako vektor:

$$P = (\rho, \theta, \phi)$$



Obrázek 2.1: Bod ve sférické soustavě souřadnic [27]

.

, kde ρ je vzdálenost bodu od počátku, θ je úhel mezi osou x a projekcí P na rovinu xy a ϕ je úhel mezi osou z a úsečkou z počátku do bodu P [17]. Obrázek 2.1 znázorňuje význam jednotlivých souřadnic. Převod ze sférické soustavy do kartézské soustavy lze vyjádřit takto:

$$\begin{aligned} x &= \rho\, sin(\phi)\, cos(\theta) \\ y &= \rho\, sin(\phi)\, sin(\theta) \\ z &= \rho\, cos(\phi) \end{aligned} \tag{2.1}$$

a z kartézské do sférické takto:

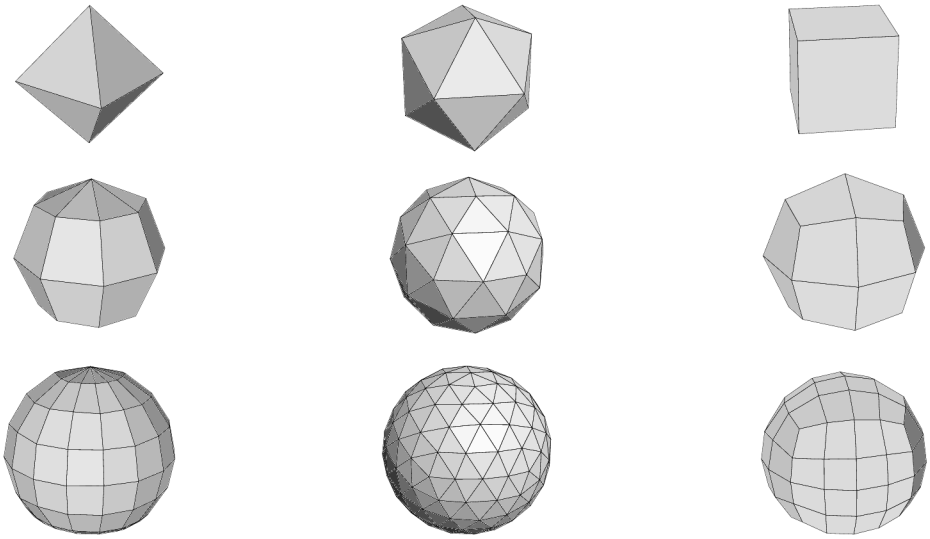
$$\begin{aligned} \rho &= \sqrt{x^2 + y^2 + z^2} \\ \theta &= arctan\left(\frac{y}{x}\right) \\ \phi &= arccos\left(\frac{z}{\rho}\right) \end{aligned} \tag{2.2}$$

2.2 Generování kulovité geometrie

Jak bylo zmíněno, planety budou reprezentovány pomocí koule. Práce s planetou se tím tedy značně zjednoduší a ušetří se výkon, na rozdíl od snahy adaptovat komplexní standard jako *WGS84*. Co zbývá vyřešit je tedy, jakým způsobem reprezentovat kouli v počítači.

Existuje mnoho různých způsobů, jak vytvořit kulovitou geometrii pomocí vrcholů a hran. Hlavní rozdíl mezi jednotlivými reprezentacemi je hlavně v rozložení a tvaru trojúhelníků. Problematiku dobře shrnuje Paul Bourke ve svém přehledu „Circles and spheres“ [2]. Jednotlivé metody lze také hodnotit z pohledu uniformity trojúhelníků, čili jak moc rovnoměrně jsou jednotlivé vrcholy rozmístěny po povrchu. Různé geometrie jsou vhodné pro různé metody *LOD* či další algoritmy, pracující s povrchem. V této sekci jsou stručně představeny pouze základní a často užívané typy kulatých geometrií a stručně popsány jejich výhody a nevýhody.

Koule generovaná pomocí sférických souřadnic je jednoduchá a přímočará metoda, jak rychle vygenerovat kouli, kterou zobrazuje na obrázek 2.2a. Vrcholy jsou generovány pomocí lineárního iterování přes úhly θ a ϕ. Změnou délky kroku získáme kontrolu nad počtem vrcholů tělesa. Hlavním rysem této metody jsou nepravidelné trojúhelníky hlavně na pólech, kde vzniká „vějíř trojůhelníků“. Metoda je vhodná pro rychlé vytvoření koule a její otexturovaní cylindrickou texturou, jelikož souřadnice (θ, ϕ) lze přímo mapovat na texturovací



(a) Koule generovaná pomocí θ a ϕ (b) Koule generovaná pomocí tesselace dvacetistěnu (c) Koule generovaná dělením krychle

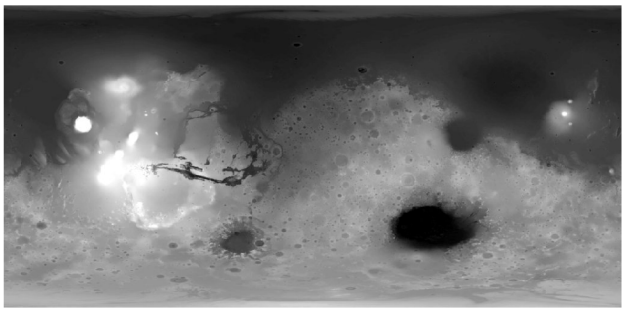
Obrázek 2.2: Různé způsoby generování koule

souřadnice (u, v), a proto se této reprezentaci obecně říká „*uv-sphere*“. Nehodí se však pro složitější práci s jejím povrchem kvůli nepravidelné geometrii.

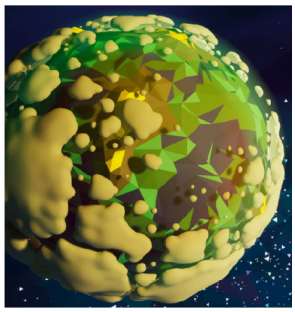
Další metodou je dělení povrchu nějakého pravidelného mnohostěnu a následná normalizace jeho vrcholů. Často užívaný bývá dvacetistěn (icosaherdon), kde jeho dělením vznikne těleso obecně známé jako „*ico-koule*“ (*icosphere*), jejíž vytvoření ukazuje obrázek 2.2b. Lze ale dělit i jiná tělesa[1]. Tato metoda má výhodu, že produkuje relativně rovnoměrné trojúhelníky po celém povrchu a implicitně vytváří hierarchickou strukturu pro každý trojúhelník tělesa během dělení. Tato struktura je vhodná pro *ROAM* (*Real-time Optimally Adapting Meshes*) algoritmus kontinuální úrovně detailů, který zjemňuje geometrii rekurzivním dělením jednotlivých trojúhelníků [9]. Nevýhodou ovšem může být exponenciální růst počtu trojúhelníků při zvyšování úrovně rozdělení.

Třetí metodou je dělení stěn krychle na čtverce (quads) a normalizace jejich vrcholů. Tím vznikne takzvaná *QuadSphere* či *QuadCube*, zachycena na obrázku 2.2c. Tento způsob je přímočarý a vyžaduje pouze jednoduchou geometrii pro implementaci. Topologie krychle je také značně intuitivní na rozdíl od předešlé metody. Podstatnou výhodou je, že každá stěna krychle se dá chápat jako dvourozměrná čtvercová mřížka pouze jinak orientovaná v prostoru a umožňuje tak relativně snadnou adaptaci různých algoritmů vyvinutých pro 2D povrchy. Hlavní pozornost při adaptaci 2D algoritmu vyžaduje hlavně řešení návazností mezi stranami krychle a kompenzování zkreslené geometrie v blízkosti jejích vrcholů. *QuadSphere* také umožňuje jednoduchou implementaci pomocí hierarchických *LOD* algoritmů používající *quadtree* datové struktury, jako je například *Chuncked LOD* [25].

[1] Více o problematice se lze dočíst zde: https://en.wikipedia.org/wiki/Spherical_polyhedron [Navštíveno 16.04.2020]



(a) Výšková mapa Marsu, získaná ze stránek *NASA*



(b) Planeta s volumetrickým terénem ve hře *Astroneer*.

Obrázek 2.3: Různé způsoby reprezentace geometrie povrchu planety

2.3 Reprezentace terénu planety

Výškové mapy

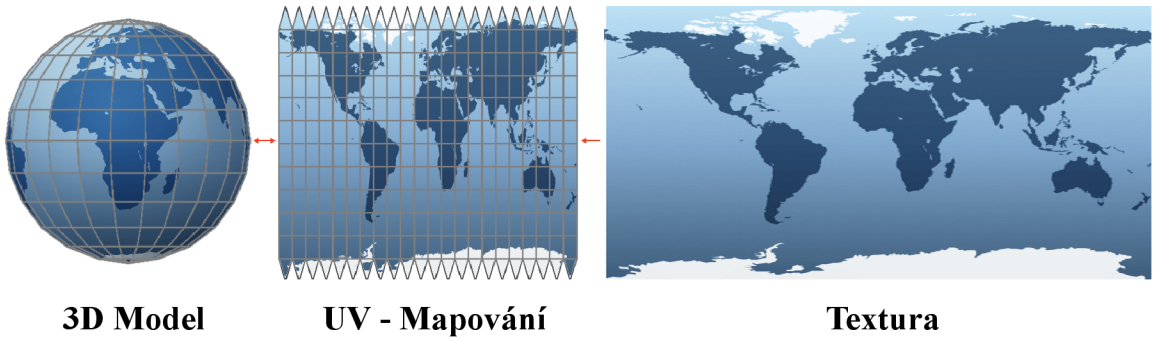
Velmi užívanou technikou pro reprezentaci terénů jsou výškové mapy (*heightmaps*). Tato rozšířená metoda ve světě 2D terénů se přirozeně adaptovala i na terény kulaté a pořád se užívá jako efektivní způsob ukládání a reprezentace povrchových dat jak v profesionální sféře, tak ve hrách. Výšková mapa je textura, která místo barev uchovává výšky jednotlivých bodů v rovině, kde hodnota jednoho pixelu určuje výšku jednoho bodu terénu. Problém mapování hodnot pixelů na výšky jednotlivých vrcholů je prakticky stejný jako problém klasického texturování. V případě dvourozměrného povrchu lze výškovou mapu namapovat přímo projekcí prostorových souřadnic (x, y) na souřadnice texturovací (u, v) s určitým měřítkem a získáním tak výšky, tedy souřadnice z. Namapování výškové mapy na planetu je složitější. Znamená to promítnou pozici na povrchu koule na texturovací souřadnice (u, v) a získat vzdálenost od středu. K tomu lze využít metody texturování koule, kterých existuje celá řada a těmi významnějšími se zabývá sekce 2.4.

Planetární výškové mapy se využívají hlavně pro ukládání reálných dat například v astronomii k mapování povrchů planet. Tyto data jsou většinou velice objemná. Například *NASA* nabízí ke stažení výškové mapy pro různé planety sluneční soustavy v různých rozlišeních[2].Ukázka cylindrické textury zobrazující reálná data je na obrázku 2.3a. Mnohdy ale žádná data ani neukládáme a k získání výškové mapy využijeme procedurální generování. To se využívá především v počítačových hrách pro účely získání nekonečného světa či neomezeného množství planet a také při počítání velice jemných úrovní detailů. Více o procedurálním generování planet v sekci 2.5.

Volumetrické planety

V určitých situacích jsou výškové mapy nevhodné, hlavně kvůli nedostatku komplexity a nemožnosti existence dutin, či převisů. Terén v takovém případě lze reprezentovat volumetricky, tedy pomocí objemových dat. Volumetrický terén je výpočetně náročnější jak z pohledu generování, tak ukládání a vyžaduje i složitější *LOD* algoritmus. Užívá se především ve hrách, kde planeta bývá hlavním tématem, pro umožnění hráči prozkoumávat různé jeskyní komplexy, či rovnou měnit její tvar. Například hra *Astroneer* využívá procedurálně

[2] *Annex Products Search, NASA* [navštíveno 17. 12. 2019] Dostupné z: https://astrogeology.usgs.gov/search?pmi-target=mercury&pmi-scope=PDS



Obrázek 2.4: Mapování cylindrické textury na planetu[4].

generované volumetrické planety s *LOD*, se kterými hráč může interagovat a s možností upravovat její terén[3]. Obrázek 2.3b ukazuje planetu z této hry.

2.4 Texturování planet

V případě, že není potřeba skutečná geometrie terénu a planety jsou zobrazovány z dálky tak, že nerovnosti terénu nejsou patrné, lze planetu vykreslit s pomocí textury promítnuté na hladký povrch koule. Textury mohou obsahovat různá data (Výškové mapy jsou textury nesoucí informaci o výšce.), kromě barvy fragmentu můžeme definovat i textury popisující například oblačnost nad planetou nebo odrazivost povrchu. I takovéto textury mohou být produktem sběru reálných dat či být procedurálně generované. V počítačové grafice se užívají dva hlavní způsoby jak namapovat texturu na povrch koule. A to buď užitím klasické cylindrické textury, anebo pomocí *cube-mappingu*.

Cylindrická textura je dvourozměrnou obdélníkovou projekcí povrchu s poměrem stran $2:1$, jejíž souřadnice (u, v) přímo odpovídají (θ, ϕ) [6]. Souřadnice v kartézském prostoru na povrchu koule, dané jednotkovým vektorem (x, y, z), lze namapovat na texturovací souřadnice (u, v) takto:

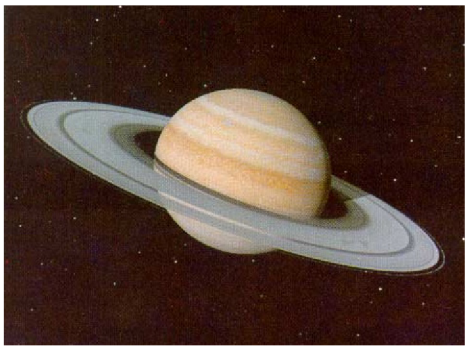
$$\begin{aligned} u &= \frac{atan2(y, x)}{2\pi} + 0.5 \\ v &= \frac{arcsin(z)}{\pi} + 0.5 \end{aligned} \tag{2.3}$$

Obrázek 2.4 ukazuje mapování cylindrické textury na planetu.

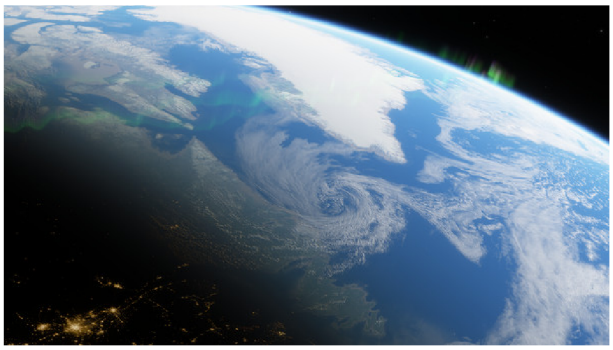
Další metodou je *cube-mapping* navržený N. Greenem jako metoda pro *enviroment mapping* díky své výpočetní efektivitě [12]. Nepromítá se zde jedna textura na celý povrch koule, ale celkem šest čtvercových textur na strany jednotlivé krychle. Z krychle lze vytvořit *Quad-Sphere* a získat tak plně otexturovanou planetu. Ačkoliv se nejedná o standardní přístup jak otexturovat kouli, v případě planet je tento způsob docela praktický, jelikož lze jednoduše na čtvercové strany aplikovat hierarchickou úroveň detailů. Tuto techniku využili například ve hře *Spore* pro procedurální generování planet v galaxii [4] nebo také simulátor vesmíru *SpaceEngine*, vyvinutý Vladimírem Romanyukem [24]. V rámci *SpaceEngine* byl

[3] „Astroneer is a space-themed exploration and survival game developed by System Era.“ – Astroneer wiki. Více na: https://astroneer.space

[4] Originální obrázek: *Obrázek 8.* z článku:*How does 3D scanning technology work?* [Artec3D]. Dostupný z https://www.artec3d.com/learning-center/3d-scanning-technology [navštíveno 17.04.2020].



Obrázek 2.5: Planeta Saturn vykreslená při simulaci průletu sondy Voyager sluneční soustavou.



Obrázek 2.6: Atmosférické efekty v programu *SpaceEngine*[5].

vyvinut program, který dokáže zpracovat cylindrické textury planet na *cubemapy* s různými volitelnými úrovněmi detailů [23].

Atmosféra a post-processing

Při aplikacích zobrazující planety se většinou dbá na věrnost zobrazení tak, aby pozorovatel měl pocit, že se skutečně dívá na planetu z vesmíru. K tomuto účelu bývá většinou planeta nasvícena světlem z hvězdy, okolo které obíhá, a jsou simulovány efekty, jak světlo proniká a je rozptylováno atmosférou. Simulace působení světla hvězd na planety je řešená od samého počátku vizualizace planet Jimem Blinnem [1]. Jeho výsledky jsou vidět na obrázku 2.5. Problematice rozptylování světla v atmosféře je věnováno mnoho dalšího výzkumu. Nad povrchem planety mohou být rovněž vykreslovány atmosférické jevy jako oblačnost, bouře apod. Obrázek 2.6 ukazuje vykreslování atmosférických jevů nad virtuální planetou ve *SpaceEngine*.

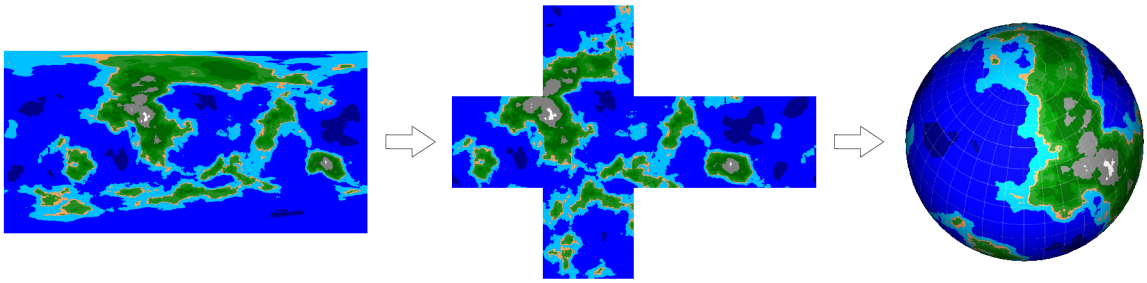
2.5 Procedurální generování planet

Rozsáhlou oblastí, která je dnes velice populární, je možnost zobrazit uživateli zdánlivě nekonečnou řadu různých planet v rámci vesmíru, který může prozkoumávat. *SpaceEngine* například umožňuje prozkoumávat celý vesmír v reálné velikosti a navštěvovat planety, hvězdy a další vesmírné objekty [24]. Jde o komplexní 3D engine planetária, který modeluje a procedurálně generuje vesmír se všemi jeho objekty na základě skutečných vědeckých dat. Tato sekce stručně popíše metody jaké se užívají k procedurálnímu generování vesmírných objektů.

Užití šumových funkcí

Velice užívanou metodou v procedurálním generování obecně je užití šumových funkcí. Jedná se o funkce, které jsou hojně využívané v počítačové grafice pokud je potřeba vytvořit dojem náhodnosti. Jsou často užívány v procedurálním generování textur, pro zanášení nedokonalostí na povrchy, při procedurálním generování terénů či vytváření animací. Šumovou funkci jako první definoval K. Perlin jako *Perlinův šum*. Od té doby bylo vyvinuto

[5] Obrázek z produktové stránky *SpaceEngine* ve službě *Steam*.



Obrázek 2.7: Planeta s procedurálně vygenerovaným povrchem s užitím šumové funkce od P. Diraca [8].

mnoho jeho variací a různých dalších implementací a jejich využití se najde téměř v každé grafické aplikaci [13].

Procedurální generování planet se od výše zmíněných aplikací v podstatě nijak neliší. Většinou jde o vygenerování výškové mapy (viz sekce 2.3) právě pomocí šumu a aplikování na kulatý terén. Tato metoda je často užívaná jako první volba, když je cílem vytvořit procedurálně generované světy. Díky dobré škálovatelnosti z nich lze jednoduše získat data pro různé úrovně detailů. Například Sean O'Neil vytváří procedurální planetu právě tímto způsobem [19]. Na obrázku 2.7 je ukázáno, jak lze vytvořit procedurální planetární texturu právě za pomocí šumu. Jak je vidět, byla vygenerována cylindrická mapa, která byla následně převedena na *cubemapu* a namapována na planetu. Nevýhodou této metody je vizuální charakteristika šumu, byť bylo použito více oktáv. Vždy lze čekat, že nejvyšší bod bude uprostřed pevninského masivu a od něj se terén bude snižovat ve všech směrech až k pobřeží. Povrch postrádá prvky, které vznikají přirozeným pohybem zemské kůry, jako jsou pohoří či souostroví.

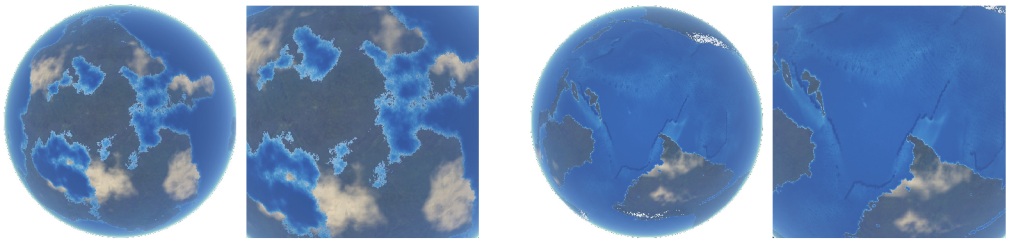
Fault lines

Zajímavou alternativní technikou je užití *fault* algoritmu[6]. Jedná se o iterační metodu jak procedurálně vytvořit výškovou mapu. Její 3D varianta byla užita na kouli nejdříve H. Eliasem [10] a poté P. Bourkem [3]. Pro vytvoření planety touto metodou se koule postupně rozděluje náhodnými rovinami a jedné ze dvou vzniklých částí se poloměr zvýší a druhé sníží. Po dostatečném počtu iterací dostaneme reliéf planety, který velice připomíná náhodný šum s vysokým počtem oktáv. Proces tvoření takovéto planety ukazuje obrázek 2.8.

(a) $i = 1$ (b) $i = 10$ (c) $i = 100$ (d) $i = 1000$ (e) Finální výsledek.

Obrázek 2.8: Planeta, jejíž reliéf byl generován pomocí *fault lines* algoritmu z práce P. Bourkera. S vizualizovaným procesem postupného zvyšování počtu iterací i.

[6] Více na: http://www.lighthouse3d.com/opengl/terrain/index.php?fault



(a) Planeta generovaná pomocí skládaných šumových funkcí.

(b) Planeta generovaná s užitím procedurálních tektonických plátů.

Obrázek 2.9: Obrázek srovnávající procedurálně vygenerovanou planetu pomocí simulace tektoniky oproti užití šumů, převzatý z [5].

Simulace fyzikálních procesů

Další metodou generování planet je použití fyzikální simulace geologických procesů na jejím povrchu. Můžeme simulovat tektonickou, vulkanickou činnost a koroze na povrchu v časovém měřítku v řádu desítek až stovek miliónů let. Simulace může mít různé úrovně abstrakce, tedy jak přesně následuje tyto procesy. Přirozeně platí, že čím vědecky přesnější, tím je výpočet náročnější. Při snaze vytvořit realisticky vypadající procedurální planetu není nutné přesné simulace a zaměřujeme se spíše na efektivitu výpočtů, které produkují uvěřitelně vypadající terén.

Nedávná práce s název *Procedural Tectonic Planets* [5] představuje model simulace pohybů tektonických plátů vhodný pro generování procedurálních planet. Práce je zaměřena na vizuálně realistické výsledky s efektivním výpočtem a možností uživatelského vstupu do procesu generování pro nastavení požadovaného vzhledu. Obrázek 2.9 ukazuje srovnání této metody oproti generování za použití šumu. Je zde vidět, jak šumové planety působí umělým dojmem oproti kontinentům a zlomům při simulaci tektoniky. Další zajímavou prací je model tektoniky planety *Tectonics.js* od Carla Davidsona, který nabízí online interaktivní aplikaci simulující pohyby tektonických plátů v čase [7].

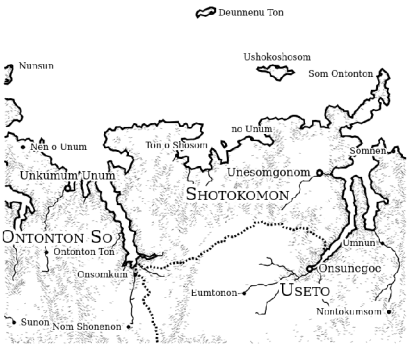
Kapitola 3

Generování map na povrchu planety

Tato kapitola se zabývá adaptací metod procedurálního generování map na povrch koule jakožto alternativu k více klasičtějším metodám procedurálního generování planet.

3.1 Procedurální generování map

Vytváření map je disciplína, která se zabývá návrhem mapy fiktivního světa, do kterého autor zasazuje své příběhy. Procedurální generování nám umožňuje automatizovat tento proces a ze vstupních parametrů si nechat vygenerovat už hotovou mapu. Generování map je široké a populární téma v kruzích nezávislých vývojářů, kteří tyto metody vyvíjí a zdokonalují. Mapy jsou ovšem vytvářené tak, jak nad nimi většina lidí uvažuje: Svět zakreslený ve 2D s různými prvky, jako jsou hory, pohoří, řeky, města, moře, cesty apod. Například O'Leary vytvořil twitter robota, který každou hodinu publikuje unikátní mapu fantasy světa[1] (příklad na obrázku 3.1a). Proces, jak toho docílil, popsal ve svém článku [18]. Existují online aplikace, kde si kdokoliv může podle svých potřeb vygenerovat mapu pro svůj fantasy svět, například *Azgaar*[2], jehož výsledky jsou vidět na obrázku 3.1b.



(a) Mapa generovaná podle O'Learyho.

(b) Politická mapa ostrova vygenerovaná v aplikaci *Azgaar*.

Obrázek 3.1: Příklady procedurálně generovaných map

[1] https://twitter.com/unchartedatlas

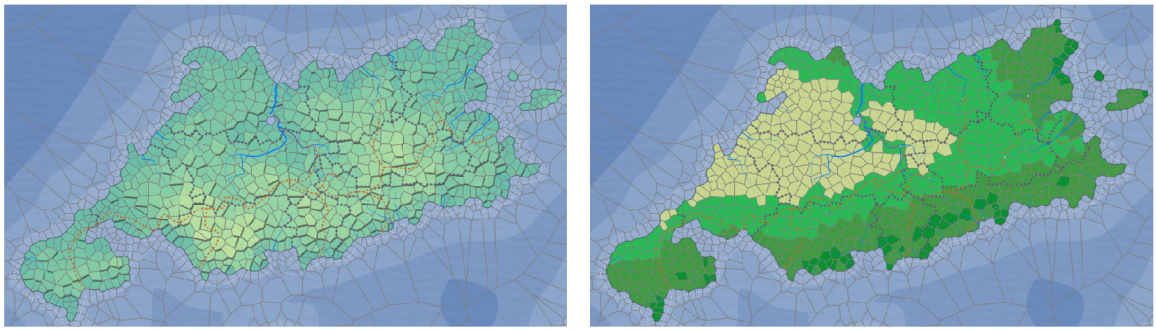
[2] *Fantasy Maps for fun and glory*: https://azgaar.wordpress.com/ [navštíveno 18. 12. 2019]

Jak je vidět, procedurálně vygenerované světy jsou daleko bohatší co se týče kontextu, vytváří dojem postupného vzniku s historickým vývojem oproti generickým světům z konvenčních metod procedurálního generování (viz sekce 2.5).

3.2 Diskretizace povrchu koule

Základem generátorů map je rozdělit plochu, na které generátory pracují, do diskrétních regionů pravidelného či častěji nepravidelného tvaru a iterativně nad nimi provádět různé transformace a celulární algoritmy. Nejčastěji se k rozdělení roviny užívají Voroného diagramy, které vytvoří regiony nepravidelného tvaru, lze ale mít mapy vygenerované na pravidelných regionech jako jsou čtvercové nebo šestiúhelníkové mřížky.

Klíčové je získat prohledávatelný prostor či graf s možností rychlé indexace jednotlivých regionů/uzlů a určení jejich sousedství a okolí. Při práci s regiony na dvourozměrném povrchu je navigace přímočará a efektivní. Regiony lze ukládat do dvourozměrného pole a indexovat pomocí (x, y) souřadnic. Toto řešení je dostatečné pro čtvercové, šestiúhelníkové i Voroného regiony. Přístup do sousedních buněk a procházení okolí se děje v konstantním čase bez ohledu na velikost pole. Takovéto prostory při procházení mohou být v obou směrech nekonečné, nebo mít jasně definovanou hranici. Obrázek 3.2 ukazuje jak byla vygenerována mapa na obrázku 3.1b pomocí Voroného regionů.



(a) Výšky regionů (b) Biomy regionů

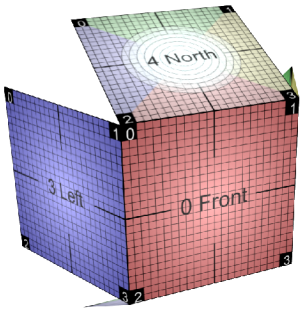
Obrázek 3.2: Voroného regiony a struktura povrchu pomocí kterých byla vygenerována politická mapa z obrázku 3.1b.

Vytvořením takovéhoto prohledávatelného prostoru na povrchu koule není úplně triviální záležitostí. Je třeba dodržet topologii koule, zároveň určitou pravidelnost a rovnoměrnost regionů na jejím povrchu a vyvarovat se všem zkreslením, které mohou vyvstat z použité geometrie. Tato sekce popisuje techniky, které lze právě pro tyto účely použít.

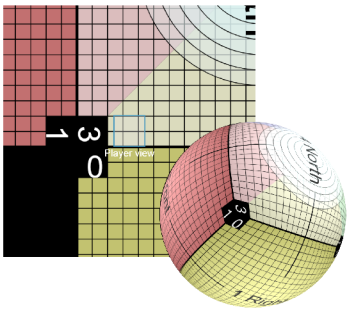
Mapování pravidelné čtvercové sítě

Jednou z populárních metod je vytváření mapy na čtvercové síti, kde všechny regiony jsou čtverce a mají vždy 8 sousedů. Pro namapování takového světa na kouli musíme docílit správné cykličnosti při držení směru pohybu. Vytvořením cyklického pole z původní matice ovšem vzniká torus nikoli koule. Je ale možné namapovat čtvercovou síť na jednotlivé strany kulaté krychle s pomocí znalostí z *cubemappingu* a správně ji propojit ve hranách.

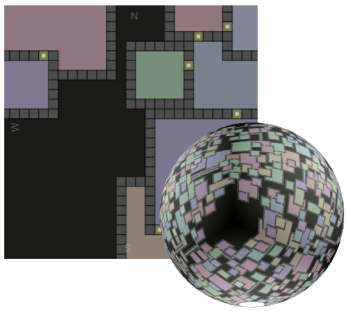
Amit Patel na svém blogu takto vytvořil herní generátor *„planetárních dungeonů“*, čili procedurální mapu sklepení tvořenou na čtvercové síti, která se při procházení chová



(a) Namapování čtvercové mřížky na strany krychle s použitím *cubemappingu*



(b) Propojení stran na povrchu koule a vykreslení aktuální pozice na rovinu.



(c) „Pohled hráče“ - Demonstrace, jak se generátor mapy vyhne rohům kulaté krychle.

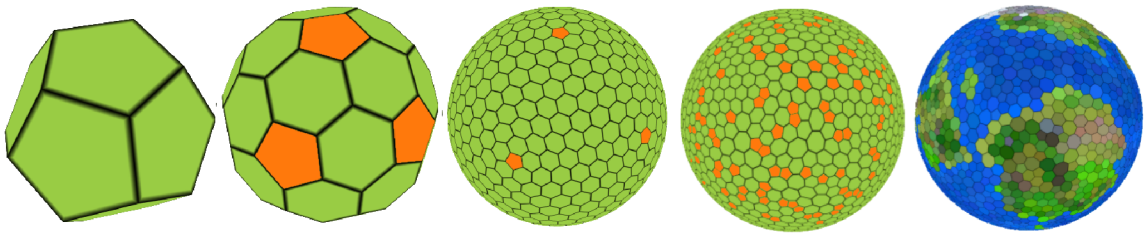
Obrázek 3.3: Řešení A. Patela namapování čtvercové herní mapy na povrch koule.

jako by byla na povrchu koule [21] (viz obrázek 3.3c). Hráč přechází mezi procedurálně generovanými místnostmi a při udržení směru se nakonec dostane tam odkud začal. Mapa se vykresluje do roviny rozložením pláště krychle a všechny buňky jsou tedy dokonale čtvercové. Problém s nekonzistentním počtem sousedů v rozích krychle vyřešil tím, že se generátor rohům jednoduše vyhnul a hráč se k nim nemůže přiblížit. Jeho postup, jak vytvořit čtvercovou mapu tak, aby se chovala jako na povrchu koule, popisuje ve své práci *Wraparound square tile maps on a sphere* [22] a je demonstrován na obrázcích 3.3a a 3.3b.

Diskretizace dělením dvanáctistěnu

Zajímavým alternativním přístupem, jak dělit kouli na buňky, je použít pravidelný dvanáctistěn složený z pětiúhelníků. Při jeho dělení získáme aproximaci koule složenou z pravidelných šestiúhelníků, kde vždy na dvanácti původních vrcholech bude pětiúhelník, a znovu získáme nepravidelné okolí v určitých místech koule, ale prakticky bez žádného zkreslení oproti normalizované krychli.

Andy Gainey použil tento způsob a do výsledného 3D modelu zavedl umělé nepravidelnosti, získal tak planetu složenou z nepravidelných pěti- a šestiúhelníků [11]. Na obrázku 3.4 je vidět postupná transformace z dvanáctistěnu na výslednou planetu. Jeho práce představuje alternativu k Voroného diagramům na povrchu koule a rovněž využívá metody procedurálního generování mapy na planetě.

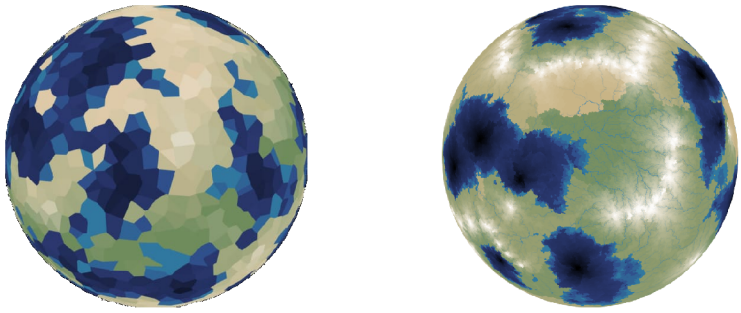


Obrázek 3.4: Způsob rozdělení koule do buněk a procedurální generování povrchu planety podle A. Gaineyho. (Barevně jsou zvýrazněny pětiúhelníky.)

Voroného diagramy a implicitní hranice regionů

Pravidelné regiony při vytváření mapy zpravidla nedokáží zajistit přirozený vzhled. K docílení realistického efektu je třeba, aby regiony byly dostatečně malé, náhodného tvaru a rovnoměrně velké po celém povrchu. Oko pak nebude hledat pravidelnosti a výsledný efekt bude na první pohled z dálky více připomínat přirozeně vzniklý terén. Možností, jak toho docílit, je rozmístit po povrchu koule body namísto celých regionů. Ty pak ponesou informace o povrchu v jejich okolí, vypočteném při vizualizaci. Velikost a tvar regionů tedy bude určen rozmístěním těchto bodů a jejich koncentrací. Hranice mezi nimi mohou být například definovány právě pomocí Voroného diagramu.

Problematika tohoto přístupu spočívá v metodě rovnoměrného rozmístění bodů (viz sekce 3.3) a ve výpočtu hranic regionů. Amit Patel, ve své práci o procedurálně generované planetě [20], řeší rozmístění bodů pomocí Fibonacciho spirály[3] a hranice regionů vykresluje ve formě Voroného diagramů nebo konstrukcí polygonů pomocí Delaunayho triangulace[4]. Výsledek je na obrázku 3.5.



Obrázek 3.5: Planeta vygenerovaná za pomoci Voroného diagramu (vlevo) a Delaunayho triangulace (vpravo) od A. Patela

3.3 Procedurální texturování s použitím kulaté krychle

Relativně nedávno (2018) vyšel článek s názvem „*Cube-to-sphere Projections for Procedural Texturing and Beyond*“ [28], autory jsou Matt Zucker a Yosuke Higashi. Studie se zabývá problematikou mapování dvourozměrných algoritmů pro procedurální generovaní textur na kulatý povrch a řeší také problémy jako je rovnoměrné rozmístění bodů či konstrukce Voroného diagramů na kouli. Tato sekce se zabývá rozborem studie a jejími metodami, které jsou přínosné pro téma práce. Poskytuje totiž robustní matematický aparát jak efektivně přetvořit většinu dvourozměrných algoritmů generování map založených na regionech a jednoduše jej promítnout na povrch koule. Využívá k tomu znalosti z oblasti „*cubemappingu*“ a řeší problémy procházení regionů („*nearest-neighbor*“).

Projekce krychle na kouli

Podstatná část práce je věnována srovnání a výběrem vhodné projekce bodů na povrchu krychle na kouli, aby zkreslení při normalizaci bylo co nejméně zřetelné. Cílem je namapovat každý bod $\overrightarrow{p}$ na povrchu krychle s vrcholy od -1 do 1 na povrch jednotkové koule pomocí

[3] Fibonacciho spirální disk je způsob jakým krokovat okolo polárních souřadnic a vytvořit tak množinu vesměs ekvidistantních bodů.

[4] Delaunayho triangulace je metoda jak spojit množinu vrcholů do co nejpravidelnější sítě trojúhelníků.

mapovací metody, která dostatečně dobře kompenzuje zkreslení obsahu stran, vzniklé při normalizaci krychle („*area-preserving* mapping“).

Nejdříve je nutné parametrizovat povrch krychle. Každá strana je dvourozměrná plocha s vlastním lokálním 2D souřadným systém s počátkem v levém horním rohu a koncem v protějším. Každá strana je jinak orientovaná v prostoru a tuto orientaci lze vyjádřit 3×3 maticí P s prvky -1, 0 nebo 1, která transformuje lokální souřadnice ve straně do globálního 3D prostoru a zpět. Mějme lokální souřadnice strany určené vektorem $(a, b) \in \langle -1, 1\rangle \times \langle -1, 1\rangle$, potom:

$$\overrightarrow{p} = P \cdot \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \tag{3.1}$$

Například bod ve 3D na povrchu krychle $\overrightarrow{p} = (-0.3, -1, 0.2)$ může být reprezentován takto:

$$P = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{kde:} \quad \begin{matrix} a = 0.2 \\ b = 0.3 \end{matrix}$$

Vektor $(0.2, 0.3)$ představuje texturovací souřadnice pro danou stranu bez jakékoliv informace o orientaci v prostoru. Matice P je sestavena pro každou stranu a slouží pro jednoduchou reverzibilní transformaci z 2D lokálních souřadnic strany do 3D globálního prostoru.

Dále je zavedena „*warp*“ funkce, která zajistí, že po následné normalizaci bodu $\overrightarrow{p}$ bude zachována plocha a zkreslení bude co nejmenší. Z lineárních souřadnic (a, b) na straně krychle jsou vypočteny souřadnice (u, v) jako $u = f_u(a, b)$ a $v = f_v(a, b)$, kde $f_u, f_v : \mathbb{R}^2 \to \mathbb{R}$ jsou *warp* funkce. Potom vektor $\overrightarrow{q}$ představuje bod na povrchu krychle se započteným kreslením:

$$\overrightarrow{q} = P \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \cdot \begin{bmatrix} f_u(a, b) \\ f_v(a, b) \\ 1 \end{bmatrix} \tag{3.2}$$

Nakonec jednotkový vektor $\overrightarrow{\omega}$ představuje správně promítnutý bod na kouli:

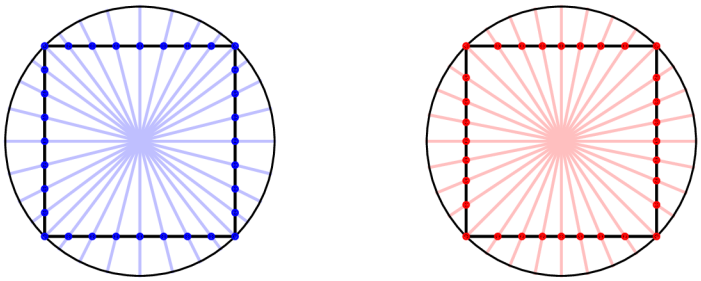
$$\overrightarrow{\omega} = \frac{\overrightarrow{q}}{\| \overrightarrow{q} \|} \tag{3.3}$$

Volba vhodné „warp“ funkce

Zucker a Higasi zavádí způsob jak měřit míru jejich zkreslení pomocí „*Root Mean Square Error*“ (*RMSE*) a takto porovnávají různé možnosti jejich implementací a zavádí rovněž jednu novou vlastní [28]. Provádí srovnání také z pohledu jejich rychlosti a složitosti.

Funkce musí být symetrické, zachovávat polohu krajních bodů ve hranách a polohu středu, tedy $f_u(0, b) = f_v(a, 0) = 0$ a $f_u(1, b) = f_v(a, 1) = 1$. Pro některé implementace obecných *warp* funkcí dvou proměnných lze nalézt jednoduší funkci jedné proměnné $f : \mathbb{R} \to \mathbb{R}$, pro kterou platí: $f_u(a, b) = f(a)$ a $f_v(a, b) = f(b)$. Problém se tedy zjednoduší na hledání jedné funkce f.

Pokud není vyžadována naprostá přesnost (nulové zkreslení $RMSE = 0$), lze použít metodu *tangent-warp*, která je relativně jednoduchá, a tedy rychlá, s dostatečnou přesností, aby lidský pozorovatel nezaznamenal zkreslení. Funkce byla použita v řadě aplikací, například v práci Leboura o *real-time* renderování planet [15]. Podrobná srovnání *warp* funkcí



Obrázek 3.6: Promítnutí bodů čtverce na kružnici. Vlevo jsou rovnoměrné body na čtverci, zkreslené projekcí na kružnici. Vpravo jsou body na čtverci transformované pomocí *tangent-warp* a rovnoměrně promítnuty na kružnici. Obrázek převzat z [28].

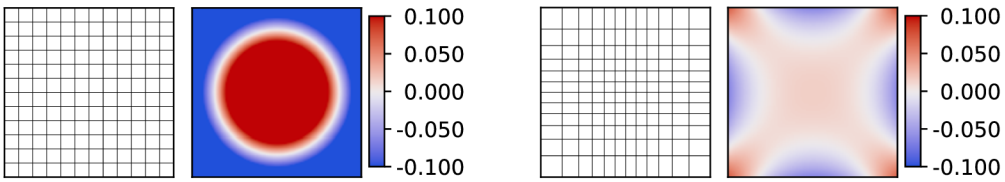
s využitím *RMSE* nabízí Lambers [14]. Její základní podobou je:

$$\begin{aligned} u = f(a) &= tan\left(\frac{\pi}{4}a\right) \\ a = f^{-1}(u) &= \frac{\pi}{4}tan^{-1}(u) \end{aligned} \tag{3.4}$$

Metoda vychází z 2D analogie, že pokud chceme rovnoměrně rozmístit $2n$ bodů na stranu jednotkového čtverce, která leží na přímce $x = 1$, tak, aby všechny svíraly mezi sebou stejný úhel a po promítnutí na opsanou kružnici byly rozmístěny rovnoměrně, rozmístíme body $(1, y)$, kde $y = f(\frac{i}{n}) = tan\left(\frac{\pi}{4}\frac{i}{n}\right)$ pro každé $i \in [-n, n]$. Princip ilustruje obrázek 3.6. Konstanta $\frac{\pi}{4}$ v tomto případě zajistí, že promítnuté body budou skutečně všechny rovnoměrné, tedy s nulovou chybou. Ovšem při aplikaci ve 3D na strany krychle a projekce na kouli, se začíná objevovat zkreslení. *RMSE tangent-warp* pro krychli v porovnání oproti klasické identitě ($u = f(a) = a$) je znázorněno na obrázku 3.7. Zucker a Higasi proto zavedli volný parametr θ, který byl v předchozím případě právě $\frac{\pi}{4}$, a vyjádřili funkci takto:

$$\begin{aligned} u = f(a) &= \frac{tan(a\theta)}{tan(\theta)} \\ a = f^{-1}(u) &= \frac{1}{\theta}tan^{-1}(u\ tan(\theta)) \end{aligned} \tag{3.5}$$

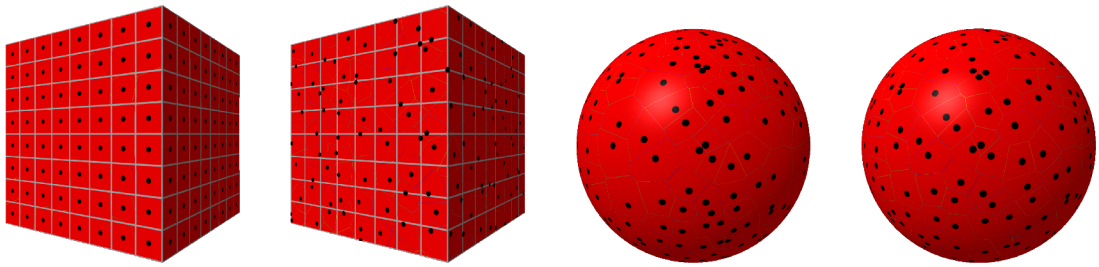
Experimentálně poté optimalizovali θ na hodnotu $\theta \approx 0.86873$, která vykazovala nejnižší *RMSE*.



(a) *RMSE* bez *warp* funkce ($maximum = 0,476$*)

(b) *RMSE* pro *tangent-warp* ($maximum = 0,0738$)

Obrázek 3.7: Porovnání relativních ploch chyb při použití *tangent-warp* a bez něj. Sytost odstínu ukazuje míru zkreslení na povrchu strany krychle. *Maximální hodnota chyby překročila limit barevného značení. Převzato z [28]. (Původní obrázek obsahuje srovnání vícero možných *warp* funkcí.)



(a) Rovnoměrně rozložené body na krychli.

(b) Rozechvění bodů je rovnoměrné, protože žádný bod neopustí svůj čtverec.

(c) Projekce na kouli bez *tangent-warp*, v rozích jsou body více nahuštěny.

(d) Projekce z použitím *tangent-warp*, body jsou náhodně rozložené s rovnoměrnou hustotou.

Obrázek 3.8: Proces rozmístění náhodných bodů pomocí projekce krychle na kouli a *tangent-warp*. Obrázky byly vygenerovány pomocí příkladu v aplikaci *Shadertoy* implementované Zuckerem a Higashim [28].

Rozmístění náhodných bodů na kouli

Nyní je ustanovená spolehlivá reverzibilní metoda projekce povrchu krychle na kouli. Lze tak místo práce se zakřiveným povrchem pracovat na rovných stranách krychle a tím zjednodušit algoritmy a využít metod určených pro 2D rovinu.

Jak bylo zmíněno v sekci 3.2, pro rozdělení povrchu koule na regiony s implicitní hranicí, je třeba rovnoměrně rozmístit po celém povrchu body tak aby vzniklé regiony byly zhruba stejně velké a nedalo se tak vizuálně odlišit různé části planety od jiných. Rovnoměrné rozmístění bodů na povrch koule není triviální záležitost a existuje mnoho aproximačních metod a algoritmů[5]. Projekce krychle na kouli nabízí relativě efektivní aproximaci řešení tohoto problému. Na každé straně lze vygenerovat rovnoměrnou mřížku bodů, jak je vidět na obrázku 3.8a, a pomocí *warp* funkce je promítnout na kouli. Body budou na jejím povrchu rozmístěny dostatečně rovnoměrně s maximální odchylkou charakteristickou pro *warp* funkci.

Pro náhodné rozmístění bodů s neměnnou hustotou, lze strany rozdělit do mřížky čtvercových buněk a do každé vložit právě jeden bod na náhodné místo (obrázek 3.8b). Jemnost mřížky potom určí celkový počet rozmístěných bodů. Obrázek 3.8c ukazuje, jak se náhodné body na krychli promítnou na kouli, není-li použita *warp* funkce, a na obrázku 3.8d je finální výsledek.

Vygenerování Voroného diagramu a „wrap“ funkce

Diskretizace povrchu koule se dokončí zkonstruováním Voroného diagramu. Zucker a Higashi nabízí způsob jak toto efektivně provést [28]. Středy jednotlivých Voroného regionů rozmístíme na kouli podle předchozí metody a nazvěme je *těžiště*. Každý bod na kouli lze vyjádřit jako jednotkový vektor $\overrightarrow{\omega}$ a je třeba jej zařadit do příslušného regionu. Musíme tedy zjistit, které těžiště $\overrightarrow{t}$ je bodu $\overrightarrow{\omega}$ nejblíže. Srovnávat všechny body se všemi těžišti by bylo při rostoucím rozlišení výkonově neúnosné. Využijeme tedy krychli a způsobu jakým jsou těžiště rozmístěny. Každá čtvercová buňka obsahuje právě jedno těžiště, stačí tedy zjistit do které buňky se $\overrightarrow{\omega}$ promítne a porovnat vzdálenosti s těžišti v okolních buň-

[5] Obecně se jedná o tzv. Thomson problem

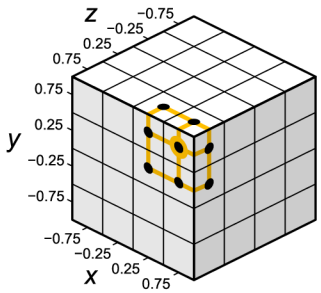
kách. Tím docílíme konstantního času přiřazení $\overrightarrow{w}$ k regionu při libovolně velkém rozlišení. Algoritmus 1 popisuje tento proces vyhledání nejbližšího těžiště.

Algoritmus 1: Vyhledání nejbližšího těžiště pro libovolný bod na kouli

Vstup: $\overrightarrow{w} = (w_x, w_y, w_z)$ – bod na jednotkové kouli
Výstup: $\overrightarrow{t} = (t_x, t_y, t_z)$ – nejbližší těžiště

```
1:  P = getFaceP(w)                    /* Nalezení permutační matice strany */
2:  w = w · P                              /* Orientace do prostoru strany */
3:  uv = w.xy / w.z                      /* Lokální 2D souřadnice ve straně */
4:  uv = unwarp(uv)
5:  cell = uvToCell(uv)    /* cell ∈ ℕ × ℕ - diskrétní koordináty buňky. */
6:  t = massCenterOfCell(P, cell)
7:  d = distance(w, t)
    /* Prohledání okolí 2 × 2                                            */
8:  for s_x = cell.x − 2 to cell.x + 2 do
9:      for s_y = cell.y − 2 to cell.y + 2 do
10:         c_uv = cellToUV((s_x, s_Y))
            /* Výpočet nového c_uv ležící v sousední straně s permutační
               maticí P_new, pokud byla překročena hrana.                */
11:         if wrap(P, c_uv, P_new) then        /* False při neplatném kroku */
12:             cell_new = uvToCell(c_uv)
13:             t_n = massCenterOfCell(P_new, cell_new)
14:             d_n = distance(w, t_n)
15:             if d_n < d then
16:                 d = d_n
17:                 t = t_n
18:             end if
19:         end if
20:     end for
21: end for
```

Funkce s názvem *wrap* (řádek 11. algoritmu 1) řeší problém hledání okolí k okrajovým buňkám, které z části leží na sousedních stranách. Každá strana má jinak orientované lokální souřadnice a funkce zajistí změnu orientace při překročení hrany. Rohové buňky mají okolí dokonce menší než ostatní, jak ukazuje obrázek 3.9, a tedy diagonální krok přes roh je neplatný a funkce vrací false. Touto funkcí se více zabývá sekce 4.4 při návrhu knihovny.



Obrázek 3.9: Menší okolí v rohových buňkách

Kapitola 4

Návrh knihovny pro procedurální generování planet

Cílem této kapitoly je popsat návrh dobře konfigurovatelné a rozšiřitelné knihovny, která bude umožňovat jednoduchou implementaci generátorů procedurálních planetárních povrchů ve formě modifikovatelných vektorových dat. V této kapitole se využívají poznatky popsané v kapitole 3 o generování procedurálních map na planetách.

4.1 Cíl knihovny

Generovaná povrchová data by měla popisovat, v kterých místech má povrch jaké vlastnosti. Samotná vizualizace těchto dat je však ponechána čistě na uživateli knihovny. Povrch by měl být snadno modifikovatelný a jednoduchý na zpracování. Samotý proces generování by měl být modifikovatelný uživatelem tak, aby ho v případě potřeby mohl doplnit o vlastní algoritmy a hodnoty. Měla by nabízet rovněž vysokou abstrakci nad nastavením výchozího generátoru pro uživatele, kteří se nechtějí zabývat algoritmickými podrobnostmi, jako je třeba „hornatost“, „průměrná teplota“, „členitost povrchu“ apod. Zároveň by ale každý algoritmus měl být otevřen jemné konfiguraci svých parametrů, pro případné doladění podle představ uživatele.

4.2 Základní koncepce

Knihovnu lze koncipovat do dvou hlavních částí - datovou (reprezentace povrchu) a algoritmickou (generování povrchu). Datová část bude sloužit pro uložení informací o povrchu, rozmístění regionů, jejich atributy, rozdělení do plátů atd. Část algoritmická neboli generátor potom bude provádět operace nad takto definovanou datovou strukturou a bude jí plnit daty popisujícími generovaný povrch. Existuje zde i mezistupeň „souřadný systém“, který provádí transformace informací uložené v datech povrchu do hodnot, se kterými pracuje generátor. Následující sekce popisují návrh jednotlivých částí knihovny.

4.3 Datová část a reprezentace povrchu

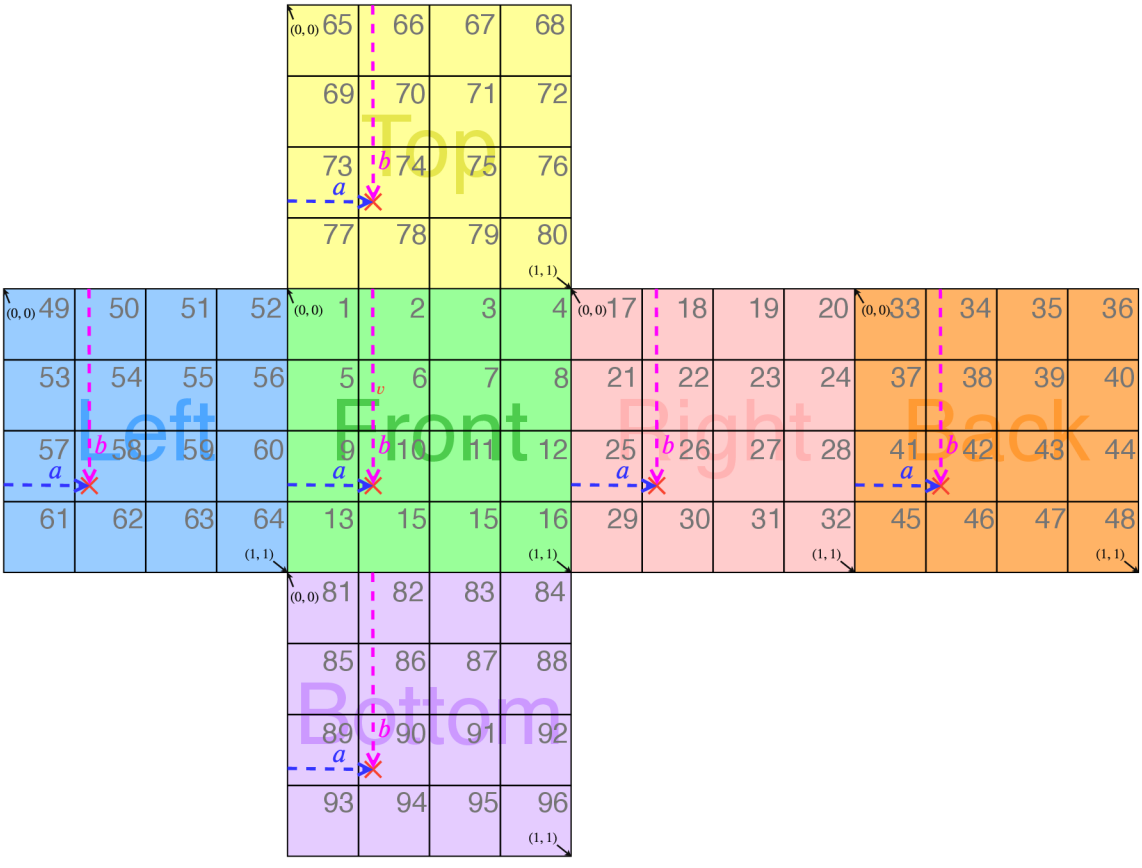
Tato část knihovny slouží pro uchování dat o kontextu povrchu do struktury vhodné pro aplikování algoritmů procedurálního generování a předáním takto strukturovaných dat uživateli. V kapitole 3 bylo popsáno, jak namapovat krychli na kouli a zpět. Bylo zmíněno,

že pracovat s povrchem krychle je jednodušší a rychlejší a intuitivnější než na zakřiveném povrchu. Toho můžeme při návrhu datového modelu povrchu využít.

Povrch a indexace regionů

Povrch bude tvořen regiony s implicitní hranicí (viz 3.2) a budou tedy definovány jako body na povrchu koule. Implementace implicitní hranice je už na uživateli, ten bude muset vhodně data zpracovat (Řeší demonstrační aplikace viz 5.3.). K regionům je třeba vyhledat jejich okolí a musí být jasná navigace. Měli by být taky rovnoměrné po celém povrchu.

Pro tyto účely jsou využity znalosti ze sekce 3.3 a regiony tedy budou rozmístěny na povrch dělené krychle. Míru rozdělení krychle nazvěmě rozlišení a značme R. Pokud povrch má rozlišení R, regiony budou rozmístěny v $R \times R$ mřížce buněk pro každou stranu. Každá buňka bude obsahovat právě jeden náhodně umístěný region. Lze vytvořit pravidlo, jak unikátně indexovat buňky s regiony na krychli pro libovolné R: Při rozložení povrchu krychle na rovinu se postupně indexují buňky počínaje v levém horním rohu přední strany až do poslední buňky v pravém spodním rohu spodní strany. Obrázek 4.1 demonstruje indexaci a fixní označení stran pro $R = 4$. Celý povrch bude tedy množina regionů $S = \{r_1, r_2, r_3, \ldots, r_n\}$, kde $n = 6R^2$. Index (pozice) regionu pak plně definuje buňku a stranu krychle, ve které se nachází.



Obrázek 4.1: Indexování regionů na dělené krychli pro rozlišení $R = 4$ v pořadí stran: „přední“ - „pravá“ - „zadní“ - „levá“ - „horní“ - „spodní“.

Atributy regionů

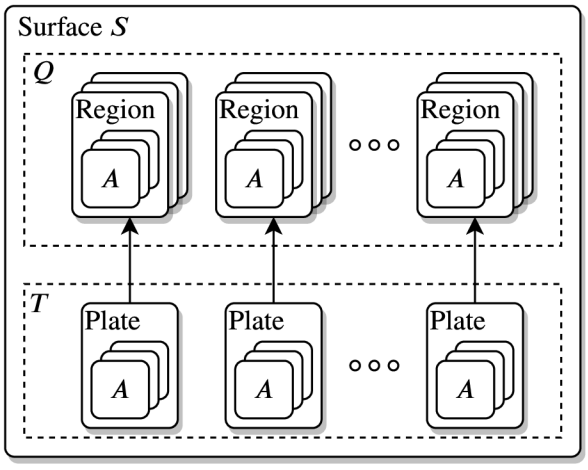
Regiony na povrchu musí uchovávat data o vlastnostech té části povrchu, kterou vymezují. Jedná se jednoduše o kolekci atributů, kde každý nese hodnotu nějaké specifické vlastnosti povrchu. Region proto koncipujeme jako asociativní pole, mapující název/typ na hodnoty různých datových typů. Tento pár *typ* $\rightarrow$ *hodnota* tvoří právě jeden atribut. V regionu lze atributy dynamicky měnit, přidávat a odebírat pomocí operací „*nastav/odeber atribut*“. Uživatel tak může terén upravovat a zanášet do něj své vlastní informace.

Jedním z klíčových atributů je relativní poloha samotného těžiště regionu v rámci buňky „*offset*“ $\rightarrow (p_a, p_b)$, kde $p_a, p_b \in \langle 0, 1 \rangle$. Hodnota $(0, 0)$ při pohledu na obrázek 4.1 značí levý horní bod buňky a $(1, 1)$ pravý spodní. Pokud region tento atribut nebude obsahovat, implikujeme polohu těžiště na střed: „*offset*“ $\rightarrow (0.5, 0.5)$.

Tektonické pláty

Jelikož procedurální generování planety bude v našem případě vyžadovat seskupování regionů do tektonických plátů, definujeme na povrchu novou kolekci pro shlukování regionů do disjunktních oblastí. Tektonický plát bude podobně jako povrch S kolekcí regionů jemu náležících. Samotnému plátu lze přidávat také vlastní atributy. Zároveň si musí pamatovat, které regiony jsou na jeho okraji, a ty pak budou sloužit jako rozhraní plátu při interakci s okolím. Tektonický plát T definujeme tedy takto: $T = (Q_t, Q_e, A_t)$, kde Q_t je množina regionů patřící plátu, $Q_e \subseteq Q_t$ je množina hraničních regionů plátu a A_t je množina atributů plátu se stejnými vlastnostmi jako atributy regionů.

Díky možnosti přiřazovat atributy plátům a regionům se povrch stává vysoce dynamický a lze tak nad mím implementovat množství algoritmů, které mohou používat i vlastní definované atributy. Jako příklad atributu plátu může být jeho střed pro určení jeho polohy v rámci planety, nebo jeho celková elevace pro výpočet výšky jednotlivých vnitřních regionů. Definovaná datová struktura popisující povrch je znázorněna na obrázku 4.2.



Obrázek 4.2: Schéma datové struktury modelu povrchu S. Množina T představuje pláty povrchu, které odkazují na disjunktní shluky regionů v množině Q. Celková velikost Q je tedy: $|Q| = \sum_{j=1}^{|T|} |Q_{t_j}| = 6R^2$

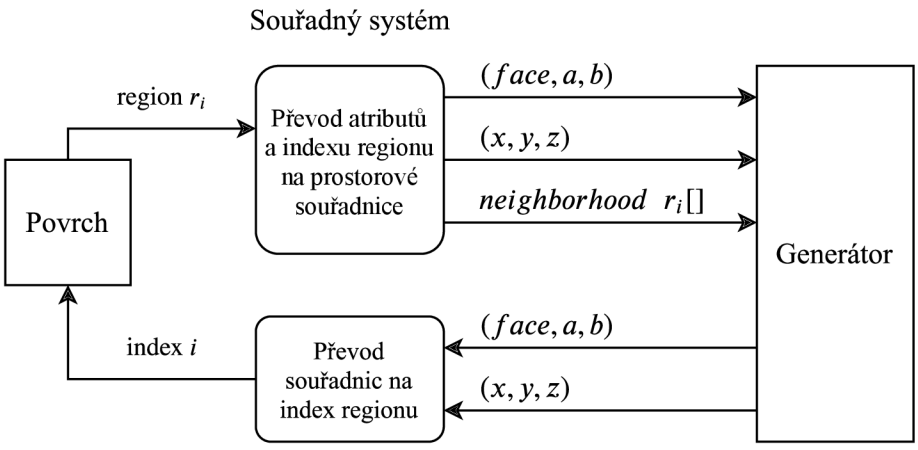
4.4 Souřadný systém a mapování regionů na povrch planety

Tato část knihovny implementuje samotnou navigaci po povrchu pro vyhledání regionů a jejich sousedů a zajišťuje transparentnost přechodů přes jednotlivé hrany krychle. Cílem je odstínit a zapouzdřit výpočty související s kulatou geometrií. Tato sekce popisuje navržený souřadný systém vycházející ze znalostí ze sekce 3.3.

Pro každý region je klíčové, aby generátor mohl určit jeho přesnou polohu nejen na krychli, ale i jeho finální polohu na kouli. Každý region by měl o sobě poskytovat následující informace:

- Index buňky i, ve které se nachází.
- Relativní pozici těžiště v rámci buňky (p_a, p_b).
- Lineární souřadnice $(face, a, b)$, kde (a, b) jsou souřadnice v rovině strany $face$.
- Globální 3D poloha na planetě (x, y, z) v kartézských souřadnicích.
- Nejbližší okolí „*neigborhood*“, tj. všechny okolní regiony. (Okolí může mít různou velikost v závislosti na umístění buňky).

Pro region r_i byl zmíněn povinný atribut „*offset*“, který určuje relativní polohu těžiště v rámci buňky, dáné indexem i. Více informací v regionu k pozici by bylo redundantní. Všechny výše jmenované reprezentace polohy lze dopočítat. Pro generátor by související výpočty měly být transparentní a nejlépe jako součást rozhraní regionu. Definujeme tedy modul „Souřadný systém“, který vložíme na rozhraní mezi povrch a generátor. Modul zapouzdří všechny operace transformací indexu a *offsetu* regionu na potřebné tvary zápisů polohy. Dojde tak k oddělení dat, výpočtů souvisejících s mapováním regionů do prostoru a samotných metod generování. Modul bude rovněž převádět požadavky generátoru pro vyhledání regionu na konkrétním místě na odpovídající indexy pomocí inverzních operací. Princip je znázorněn na obrázku 4.3.



Obrázek 4.3: Schéma transformace dat mezi generátorem a povrchem pomocí souřadného systému.

Transformace v souřadném systému

Zde jsou uvedeny operace, které souřadný systém provádí pro získání informací o poloze regionu včetně jejich inverzí. Příklady počítají s rozlišením R a regionem:

$$r_i = \{„offset“ \to (p_a, p_b)\}$$

Lokální souřadnice na krychli

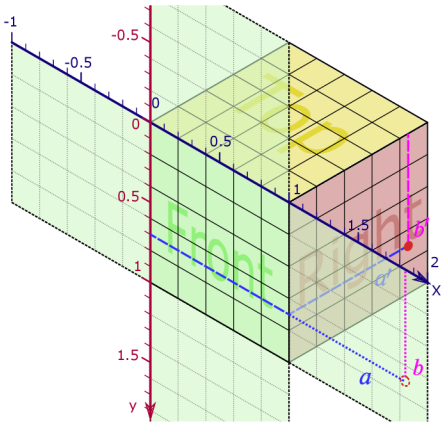
Lokální souřadnice na povrchu krychle nabízí možnost, jak pracovat v rámci jedné strany jako s dvourozměrnou rovinnou. Libovolný bod (x, y, z) ležící na straně krychle lze vyjádřit v lokálních souřadnicích oné strany jako bod l:

$$l = (face, a, b)$$

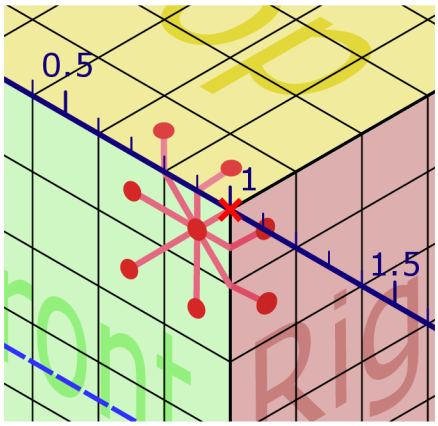
kde:

- $face$ je index strany krychle na které bod leží. Indexy jsou pevně dané v pořadí podle indexačního systému: 0: „přední“, 1: „pravá“, 3: „zadní“, 4: „levá“, 5: „horní“, 6: „spodní“)
- $a, b \in \mathbb{R}$ jsou lineární souřadnice na rovině v rámci strany $face$. Rovina má počátek $(0, 0)$ v levém horním rohu strany a bod $(1, 1)$ leží v pravém spodním.

Dokud a i b leží v intervalu $\langle 0, 1\rangle$, bod leží na straně $face$. Jakmile jedna ze souřadnic z intervalu vystoupí, bod bude ležet za hranou a tedy na jedné ze sousedních stran. V tom případě je třeba souřadnice transformovat na odpovídající lokální souřadnice příslušné sousední strany. Konkrétní příklad překročení hrany (v nejjednodušším případě) demonstruje obrázek 4.4a. Pohyb po rovině strany je omezený počtem přechodů přes hranu na maximálně jeden. Pohybovat se tak dá pouze na ploše, kterou lze přímo promítnout na stávající, či sousední stranu pomocí maximálně jednoho ohybu o 90°. V praxi omezení znamená, že a i b nesmí vystoupit z intervalu $\langle -1, 2\rangle$ a zároveň vždy jen jedna ze souřadnic má



(a) Lokalizace bodu na pravé straně krychle pomocí lokálních souřadnic dvou sousedních stran. $(0, a, b) \approx (1, a', b')$, kde $a > 1$ a $b' = b$ a $a' = 1 - a$.



(b) Vyhledání nejbližšího okolí rohové buňky. Krok přes 2 hrany (diagonálně vpravo nahoru) je zakázaný.

Obrázek 4.4: Lokální prostor v rámci přední strany a jeho chování při překročení hrany.

dovoleno překročit interval $\langle 0, 1\rangle$. Na obrázku 4.4a je plocha, po které je dovoleno se pohybovat, vyznačena zeleně. Takto definovaný systém nám umožní pracovat v rámci roviny (x, y) jedné strany, a neřešit její orientaci v prostoru a vůči sousedním stranám. Při hledání nejbližších sousedů lze jednoduše inkrementovat (a, b) souřadnice a zahazovat kroky, které padnou mimo validní prostor. Toto demonstruje obrázek 4.4b.

Pokud máme region r_i, dají se vypočítat jeho souřadnice následovně:

$$\begin{aligned} face &= \left\lfloor \frac{i}{R^2} \right\rfloor \\ a &= p_a + \frac{1}{R}\left(i \mod R\right) \\ b &= p_b + \frac{1}{R}\left(\left\lfloor \frac{i}{R} \right\rfloor \mod R\right) \end{aligned} \tag{4.1}$$

Zpětně lze z l dopočítat index a *offset* regionu takto:

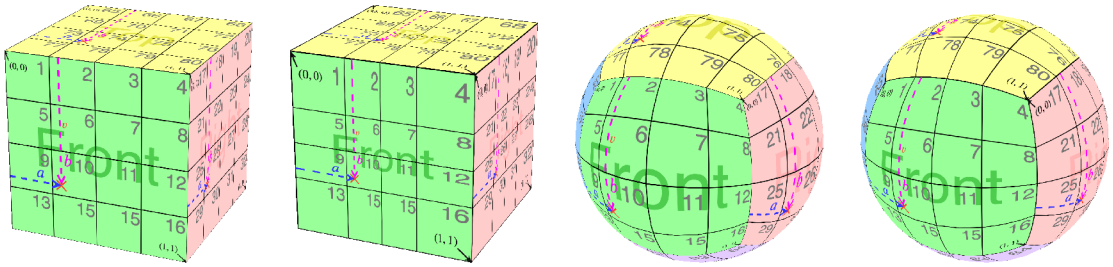
$$\begin{aligned} i &= face * R^2 + R(\lfloor bR \rfloor) + \lfloor aR \rfloor \\ p_a &= a - \frac{1}{R}\lfloor aR \rfloor \\ p_b &= b - \frac{1}{R}\lfloor bR \rfloor \end{aligned} \tag{4.2}$$

Zde musí platit, že $(a, b) \in \langle 0, 1\rangle$. To zajistí „*wrap*“ funkce zmiňovaná v sekci 3.3. Pokud lokální souřadnice l překročí hranu, funkce je transformuje na odpovídající souřadnice příslušné sousední strany podle algoritmu 2. Ten se od původního *wrap* algoritmu z [28] liší tím, že se zde užívají výše definované lokální souřadnice.

Algoritmus 2: *Wrap* funkce

Vstup: $l = (face, a, b)$, kde $a, b \in \mathbb{R}$
Výstup: $l' = (face', a', b')$, kde $a', b' \in \langle 0, 1\rangle$ nebo *error* pokud a, b nejsou validní

`/* pracujeme na přední straně dvojkové krychle */`
$\overrightarrow{p} = (2a - 1,\ 2b - 1)$ $\overrightarrow{p_c} = clamp(\overrightarrow{p},\ -1,\ 1)$
$\overrightarrow{p_e} = \|\overrightarrow{p_c} - \overrightarrow{p}.xy\|$
$d_e = \overrightarrow{p_e}.x + \overrightarrow{p_e}.y$
if $d_e == 0$ **then**
  return l
else if $min(\overrightarrow{p_e}.x,\ \overrightarrow{p_e}.y) > 0$ **then**
  return *error*
else
  `/* Vypočteme pozici na sousední straně ve 3D */`
  $\overrightarrow{p_g} = getFaceP(face) \cdot (\overrightarrow{p_c}.x,\ \overrightarrow{p_c}.y,\ 1 - d_e)$
  `/* Získáme id strany, kterou protíná směr vektoru` $\overrightarrow{p_g}$ `*/`
  $face' = getFaceId(p_g)$
  $\overrightarrow{p_g} = p_g \cdot getFaceP(face')$
  `/*` $\overrightarrow{p_g} \to l'$ `zpět na 2D souřadnice` $a', b' \in \langle 0, 1\rangle$ `*/`
  return $l' = \left(face', \frac{\overrightarrow{p_g}.x}{2} + 1, \frac{\overrightarrow{p_g}.y}{2} + 1\right)$
end if



(a) Globální pozice bodu na krychli před *warp*

(b) Globální pozice bodu na krychli po aplikování *warp*

(c) Globální (normalizovaná) pozice na kouli bez použití *warp*

(d) Globální (normalizovaná) pozice na kouli s použitím *warp*

Obrázek 4.5: Význam globální pozice na krychli a kouli s použitím *warp* funkce.

Globální pozice na kouli

Pro každý region lze nyní spočítat jeho lokální pozici l. Ta v sobě nese informace o poloze na krychli. Tuto polohu tedy promítneme na kouli pomocí projekcí ze sekce 3.3 a získáme tak jeho globální pozici. Globální souřadnice na povrchu koule (planety) jsou dány trojrozměrným normalizovaným vektorem:

$$\overrightarrow{p_g} = (x, y, z), \text{ kde } \|\overrightarrow{p_g}\| = 1$$

Lokální souřadnice pracují s jednotkovou krychlí s počátkem v levém horním rohu přední strany. Krychli je tedy potřeba přesunout na střed a normalizovat za použití „$tangent - warp$“. Nejdříve se určí globální poloha $\overrightarrow{p_k}$ na dvojkové krychli pro region r_i s lokální pozicí $l = (face, a, b)$:

$$\overrightarrow{p_k} = P_{face} \cdot \begin{bmatrix} f(2a-1) \\ f(2b-1) \\ 1 \end{bmatrix} \tag{4.3}$$

P_{face} je permutační matice strany $face$ a f je „$tangent - warp$“ funkce. Povrch krychle je tedy připravený k normalizaci:

$$\overrightarrow{p_g} = \frac{\overrightarrow{p_k}}{\|\overrightarrow{p_k}\|} \tag{4.4}$$

Obrázek 4.5 demonstruje význam globální pozice $\overrightarrow{p_k}$ na krychli a $\overrightarrow{p_g}$ na kouli bez a s *warp* funkcí.

Zpětně lze určit lokální souřadnice $l = (face, a, b)$ z jakéhokoliv bodu v (x, y, z) prostoru tak, že se vyhledá $\overrightarrow{p_k}$, který protíná polopřímka ve směru vektoru $\overrightarrow{p_g}$:

$$\overrightarrow{p_k} = \frac{\overrightarrow{p_g}}{max(|x|, |y|, |z|)} \tag{4.5}$$

Z tohoto bodu určíme index strany $face$, na které leží (která souřadnice je 1 nebo -1), a pro $face$ dohledáme permutační matici P_{face}. Pomocí matice transformujeme $\overrightarrow{p_k}$ a získáme $\overrightarrow{p_l} = (u, v, 1)$, kde $u, v \in \langle -1, 1 \rangle$ a nakonec vypočteme l:

$$\begin{aligned} \overrightarrow{p_l} &= (u, v, 1) = \overrightarrow{p_k} \cdot P_{face} \\ l &= \left(face, \frac{u}{2} + 1, \frac{v}{2} + 1\right) \end{aligned} \tag{4.6}$$

4.5 Generátor povrchových dat

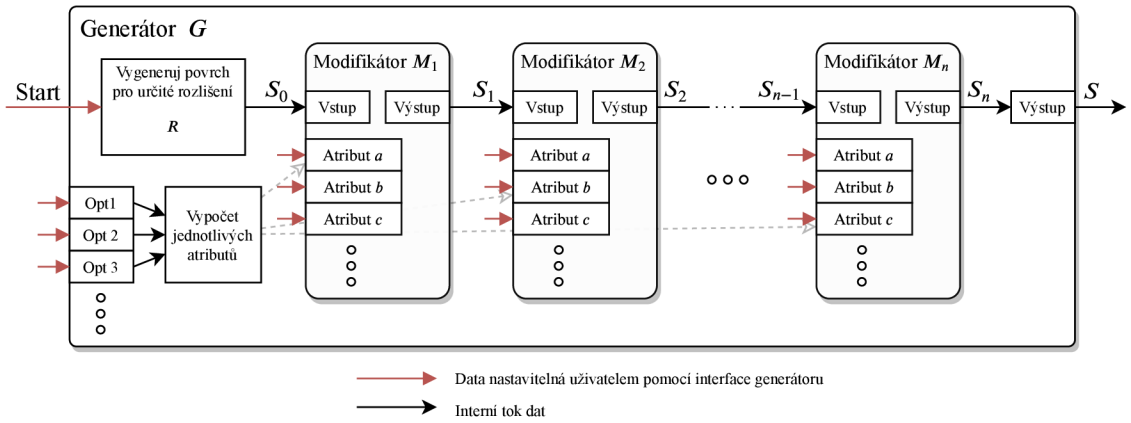
Samotným procedurálním generováním planet se zabývá „algoritmická“ část knihovny. Ta bude primárně poskytovat „generátory“, které budou schopny vytvářet libovolné množství procedurálních povrchů. Generátor je chápán jako objekt, který se jednou vytvoří a poté jej bude možné opakovaně spouštět. Při každém spuštění získáme na výstupu kompletní povrch S popisující planetu (viz. 4.3) a všechny takto vygenerované planety budou sdílet vlastnosti nakonfigurované uživatelem. Každý region tak po vygenerování bude mít například přiřazenou elevaci, teplotu, vlhkost, biom, směr proudění větru a podobně. Konfigurace by měla být co nejvíce otevřená a uživatel knihovny by měl mít možnost sám rozhodovat, jak proces generování bude probíhat, s jakými vstupními parametry a co bude na jeho výstupu. Měla by také existovat možnost, jak do procesu zanášet vlastní algoritmy a na povrch zapisovat vlastní data. Na druhou stranu, pro uživatele, kteří chtějí mít planetu připravenou bez většího úsilí a nechtějí se zabývat vnitřní koncepcí knihovny, by měl být připraven generátor předem nakonfigurovaný, který funguje již po prvním spuštění. Měl by nabízet jednoduché abstraktní rozhraní pro parametrizaci výstupu, například jako „hornatost“, „průměrná teplota“ či „míra vlhkosti“.

Struktura generátoru

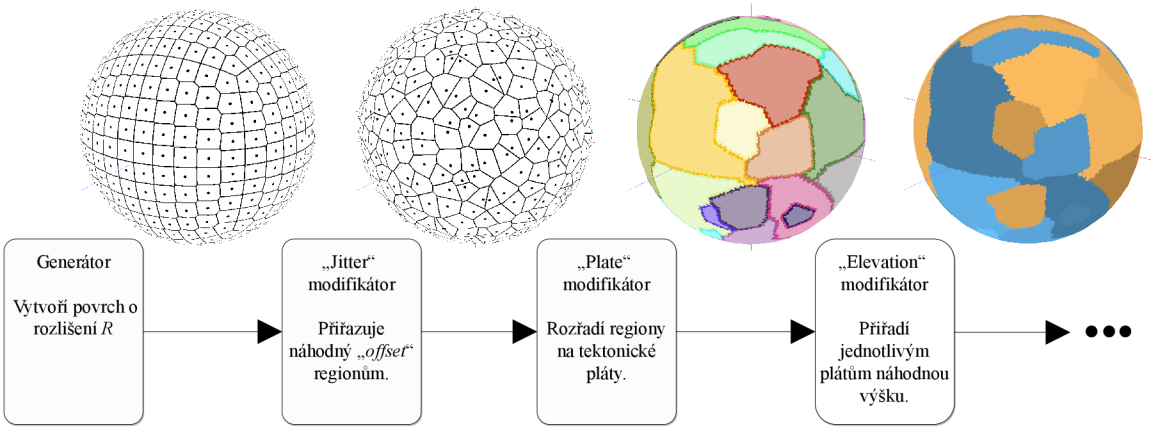
Generátor pracuje s datovou částí knihovny a o výpočty správných dat se stará *souřadný systém* (viz sekce 4.4). Na vstupu nám stačí určit rozlišení R, které určí fixní počet regionů. Generátor dále nebude měnit jejich počet ani uspořádání. Bude pracovat pouze v rámci jejich atributů.

Logická vnitřní struktura je popsaná na obrázku 4.6. Jedná se především o sérii (*pipeline*) modifikátorů, které si postupně předávají povrch S jako mezivýsledek výpočtu. Každý modifikátor může pro svůj algoritmus využít atributy přidané na region v modifikátorech předchozích. Generátor tuto *pipeline* udržuje, umožňuje modifikátory přidávat, zakazovat a odebírat.

Samotná logika procedurálního generování je čistě záležitostí modifikátorů, generátor je pouze spouští v definovaném pořadí, případně nabízí rozhraní jejich přidávání a odebírání. Je tedy nutné udržovat modifikátory logicky seřazené. Dále může poskytovat vlastní vstupní



Obrázek 4.6: Logická struktura generátoru povrchů. Dělení do modifikátorů a možnost uživatelské konfigurace.



Obrázek 4.7: Základní struktura minimálního příkladu funkčního generátoru obsahující 3 modifikátory a ukazující jejich výstupy. Obrázky jsou generovány pomocí implementované demonstrační aplikace.

parametry (na obrázku 4.6 označeno jako „Opt n“), které slouží jako abstrakce nad atributy v celé *pipeline* (například „hornatost“ může být kombinací atributů „frekvence“, „počet plátů“ a „síla pohybu plátů“).

Modifikátory povrchu

Modifikátor reprezentuje implementaci jednoho konkrétního algoritmu procedurálního generování. Jedná se například o výpočet elevace regionů. Tento cíl je možný implementovat mnoha způsoby, například pomocí šumů nebo tektonických plátů (viz. sekce 2.5). Na vstupu je povrch ve stavu definovaným předchozím modifikátorem ve frontě. Pomocí svého algoritmu potom může libovolně měnit atributy regionů a povrchu. Takto změněný povrch poté předá na výstup. Zároveň nabízí rozhraní, pomocí kterého lze jeho algoritmus konfigurovat, například nastavení konstant pro interní výpočty (frekvence šumu, počet tektonických plátů apod., na obrázku 4.6 znázorněno jako atributy a, b, c, ...). Modifikátor by měl tedy definovat fixní rozhraní atributů, které generátor, či přímo uživatel může nastavovat.

Základní struktura generátoru a minimální příklad

Na obrázku 4.7 je konceptuální příklad, jak by mohl takový generátor vypadat, případně jak by uživatel měl postupovat, když by si chtěl od základu definovat generátor svůj vlastní. Obrázek naznačuje generátor pro planety „zemského“ typu s použitím tektoniky, který postupně aplikuje modifikátory a tím tvoří povrch. Konkrétní návrh předdefinovaného generátoru a jeho modifikátorů je součástí implementace v sekci 5.4.

Kapitola 5

Implementace

Jak je patrné z návrhu, projekt je dělen do dvou částí: Knihovna pro generování procedurálních planet a demonstrační aplikace. Implementace těchto dvou částí je řešena odděleně jako dva samostatné projekty `GeoPlanetLib`[1] a `GeoPlanetDemo`[2].

Demonstrační aplikace `GeoPlanetDemo` je grafická okenní aplikace určená pro vykreslování planet vygenerovaných na základě vstupů od uživatele. K tomuto účelu využívá knihovnu `GeoPlanetLib`, která slouží pro procedurální generování planetárních povrchů.

Tato kapitola popíše jak tyto dva projekty byly implementovány s důrazem na samotnou knihovnu, jelikož právě knihovna řeší procedurální generování planet a produkuje vektorový popis povrchu. Na každém uživateli knihovny už je ponecháno, jak bude tento popis vizualizovat, tudíž implementace demonstrační aplikace je pouze jeden ze způsobu zpracovaní výstupu knihovny, kterým se uživatel může inspirovat.

5.1 Zvolené technologie

Oba projekty jsou implementovány v programovacím jazyce `C++` s použitím sestavovacího systému `CMake`. Repozitáře jsou spravovány systémem `Git` a volně dostupné na portálu GitHub.

`GeoPlanetLib` je sestavována jako dynamicky linkovaná knihovna (`dll`) za pomocí systému `CMake`. Jako externí závislost využívá grafickou matematickou knihovnu `glm`.

Aplikace používá více knihoven třetích stran pro svůj chod. Planety jsou vykreslovány za použití `OpenGL` s nadstavbou `GPUEngine`[3]. Pro práci s okny a vstupem od uživatele používá knihovnu `SDL` s nadstavbou `SDL2CPP`[4]. Grafické uživatelské rozhraní je vykreslováno za použití `Dear ImGui`.

5.2 Objektový návrh a implementace knihovny

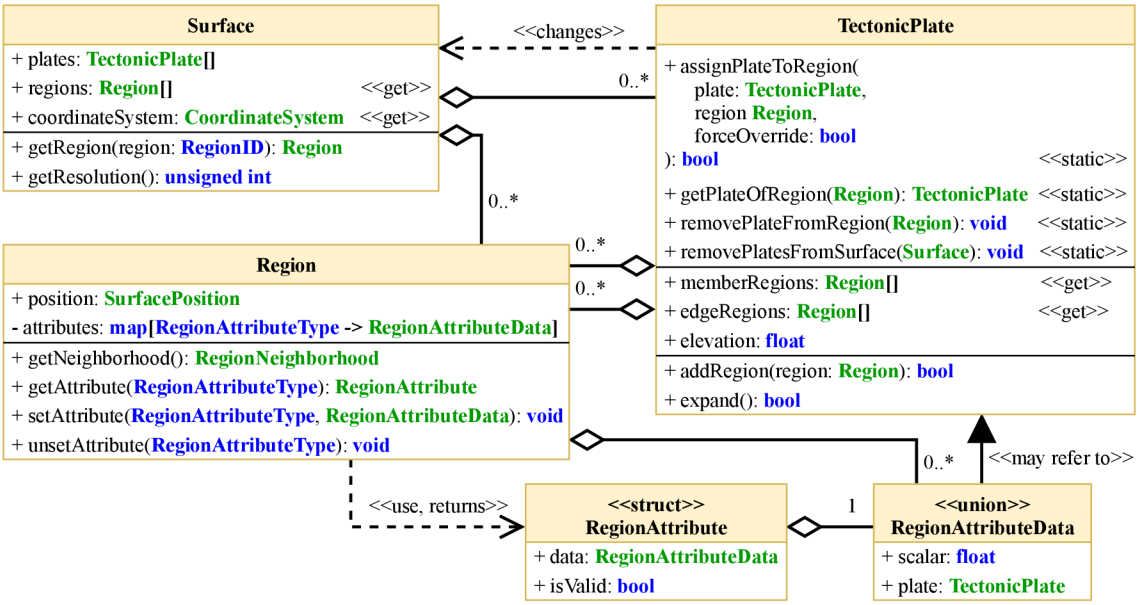
Rozhraní knihovny tvoří datové objektové struktury vyplývající z návrhu v kapitole 4. Jelikož uživatel s těmito objekty (třídami) bude muset pracovat, měl by mít jasnou představu o jejich struktuře a významu, aby je mohl správně využívat ve svém projektu. Tato sekce

[1] Repozitář na GitHub: https://github.com/xfusek08/GeoPlanetLib

[2] Repozitář na GitHub: https://github.com/xfusek08/GeoPlanetDemo

[3] Knihovna pro objektovou práci s `OpenGL`, vyvíjena na Fakultě informačních technologií na VUT v Brně, dostupné na URL: https://github.com/Rendering-FIT/GPUEngine

[4] Objektová nadstavba v `C++` nad knihovnou `SDL`, dostupné na URL: https://github.com/dormon/SDL2CPP



Obrázek 5.1: Třídní diagram popisující objektovou strukturu reprezentace povrchu planety za pomocí regionů a tektonických plátů.

popíše základní objektovou strukturu pomocí třídního diagramu zvlášť pro tři základní logické celky (reprezentace povrchu, souřadný systém a generátor). Prezentované diagramy jsou podčástmi celkového třídního diagramu knihovny `GeoPlanetLib`, který je dostupný v příloze B a na přiloženém datovém médiu ve formátu `svg`.

Reprezentace povrchu

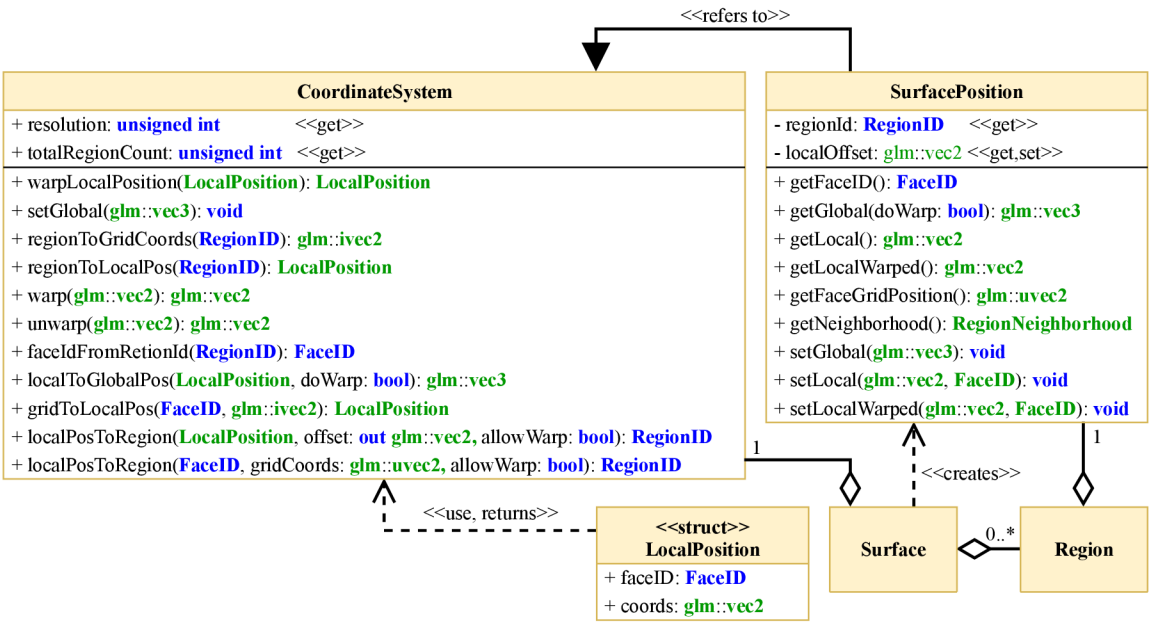
Na obrázku 5.1 je ukázána struktura tříd, která reprezentuje povrch. Je zde vidět, jak vychází z navržené struktury, která se skládá z regionů, tektonických plátů a jejich atributů.

Povrch reprezentovaný třídou `Surface` drží jednorozměrná pole regionů a plátů. Třída `Region` se chová jako mapa klíčů typu `RegionAttributeType` na `RegionAttributeData`, což je hodnota atributu implementovaná pomocí `union`, jehož složky odpovídají výčtu `RegionAttributeType`. Jak je vidět na diagramu, do atributu regionu lze definovat skalární hodnotu typu `float` a nebo ukazatel na tektonický plát.

`TectonicPlate` obsahuje statické metody, které dovolují manipulovat s atributem regionu typu „tektonický plát“. Tento atribut existuje z praktických důvodů jako způsob, jak okamžitě zjistit plát, do kterého jakýkoliv region patří, bez nutnosti prohledávání.

Tektonické pláty drží rovněž dvě pole, ve kterých si pamatují všechny příslušné a okrajové regiony. Tektonické pláty oproti návrhu neobsahují atributy jako regiony z důvodu zjednodušení implementace. Jejich funkci zastává zatím jediná vlastnost – `elevation`.

Regiony existují v paměti jenom jednou a všechny pole pracují s ukazateli na ně. `Surface::regions` je brán jako kanonický zdroj ukazatelů na všechny regiony na povrchu s neměnnou velikostí, seřazený podle jejich indexů (`RegionID`). Pláty pak obsahují podmnožiny těchto ukazatelů. Je důležité, aby regiony nebyli nikde duplikované a neměnil se jejich počet, jelikož jich může být velké množství při vyšších rozlišeních. Tím zabráníme zbytečným přesunům velkých kusů paměti.



Obrázek 5.2: Třídní diagram popisující implementaci transformací souřadného systému jako rozhraní regionu pomocí třídy `position` a `CoordinateSystem`.

Souřadný systém

Jak bylo popsáno v sekci 4.4, samotný souřadný systém by měl existovat na rozhraní datového modelu a být co nejvíce transparentní pro samotný generátor. Celá struktura implementace souřadného systému ve vztahu k povrchu a regionu je znázorněná na obrázku 5.2.

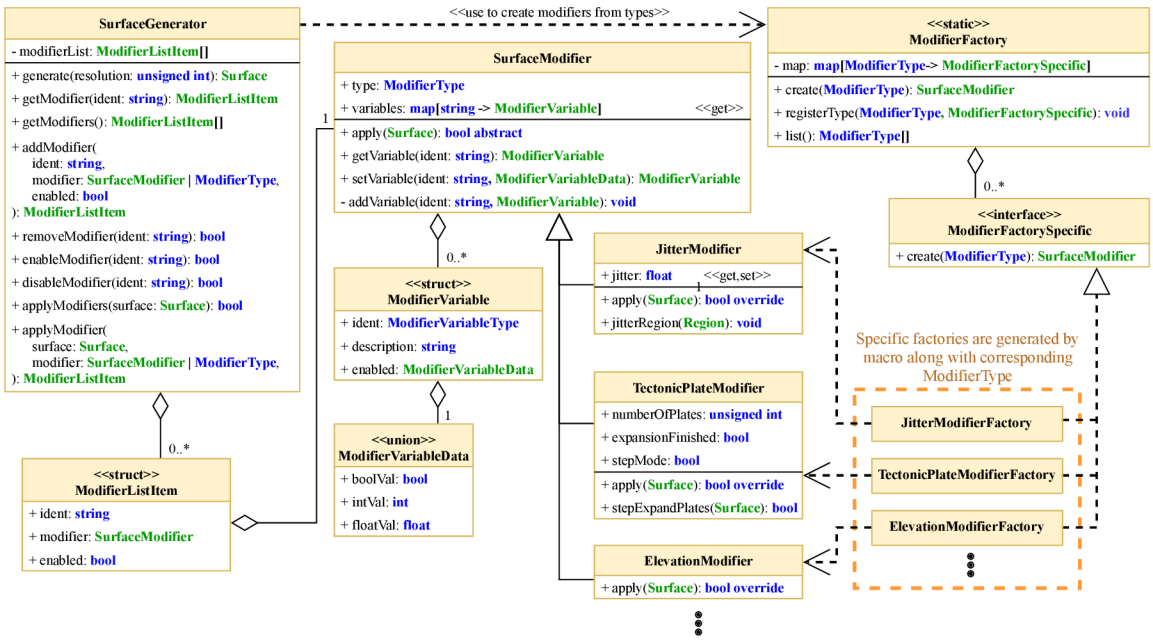
Každý povrch (`Surface`) obsahuje vlastnost `coordinateSystem`. Jedná se o instanci třídy `CoordinateSystem`, která implementuje a obaluje všechny matematické operace definované v návrhu. Tyto operace jsou charakteristické pro konkrétní povrch, jelikož jsou závislé na jeho rozlišení.

Každý region si drží ve vlastnosti `position` instanci třídy `SurfacePosition`. Tato instance je mu přidělena povrchem při vytvoření a jejím úkolem je udržovat a poskytovat jeho prostorové souřadnice. `SurfacePosition` k tomuto účelu využívá právě operace v souřadném systému povrchu a transformuje index a *offset* regionu na koordináty v požadovaném tvaru. Slouží tak jako rozhraní pro získávání prostorového umístění regionu a jeho nejbližšího okolí. Region jako samotný tak nemá žádnou informaci, kde a v jaké souřadném systému se nachází. Je tedy znovupoužitelný a otevřený k užití v jiných implementacích.

Generátor

Generátor, podle návrhu ze sekce 4.5, obsahuje modifikátory, které jednotlivě implementují algoritmy procedurálního generování. Diagram na obrázku 5.3 ukazuje jeho objektovou implementaci v rámci knihovny.

`SurfaceModifier` představuje společné rozhraní pro všechny specifické modifikátory (tři definované v diagramu). Generátor postupně používá jeho virtuální metodu `apply`, která dostane instanci celého povrchu a nějak ho změní. K parametrizaci vnitřního procesu modifikátoru slouží mapa `variables` indexovaná identifikátory, která je naplněna v imple-



Obrázek 5.3: Třídní diagram popisující implementaci generátoru užívající modifikátory, které jsou vytvářeny pomocí abstraktní továrny.

mentaci potomka. Uživatel může nastavovat pouze hodnoty jednotlivých proměnných podle identifikátoru, nikoliv však měnit jejich strukturu. Proměnné lze následně načíst a použít v implementaci uvnitř modifikátoru.

Generátor obsahuje pole struktur `ModifierListItem`, které mimo reference na samotný `SurfaceModifier` obsahují identifikátor a příznak, zda je tento modifikátor povolen. Identifikátor slouží pro vyhledání konkrétní instance modifikátoru a je vždy unikátní. Registrování pod různými identifikátory umožňuje existenci instancí modifikátorů stejného typu na různých místech v *pipeline* a jejich jednoznačné odlišení.

Za zmínku stojí použitý návrhový vzor *abstraktní továrny*[5], díky které je generátor schopen vytvářet instance modifikátorů pouze na základě typu `ModifierType`.

5.3 Implementace demonstrační aplikace

Aplikace `GeoPlanetDemo` je okenní aplikace s grafickým uživatelským rozhraním. Jejím hlavním cílem je vizualizovat data generovaná knihovnou `GeoPlanetLib` v podobě planety. Jedná se o aplikaci, která umožní uživateli vyzkoušet možnosti knihovny. Obsahuje grafické uživatelské rozhraní, pomocí kterého uživatel může nastavovat různé vstupní parametry pro knihovnu a získat tak představu, jak vypadají planetární povrchy generované knihovnou.

Kompletní objektové schéma demonstrační aplikace je zakresleno na třídním diagramu v příloze C. Struktura aplikace není detailně popsána, protože každý uživatel integruje knihovnu do vlastního procesu vykreslování scény. Aplikace by měla sloužit jako příklad pro nakládání s daty získanými z knihovny a jak správně nastavit její vstupy. Následující popis se bude převážně zabývat právě touto problematikou.

[5] Návrhový vzor *abstract factory* je způsob, jak vytvářet instance tříd, které nemají předem známou definici a používají pouze známé rozhraní.

Vizualizace povrchu planety

Hlavní náplní aplikace je vykreslit povrch generovaný knihovnou v podobě planety. Aplikace tak demonstruje způsob, jak vektorová data povrchu správně prezentovat uživateli. V návrhu knihovny (viz kapitola 4) bylo zmíněno, že povrch je generován s myšlenkou regionů s implicitní hranicí (viz sekce 3.2). Regiony jsou tedy pouhé body ve správně seřazeném poli podle indexačního systému. Tato sekce ze zabývá tím, jak takovéto pole vykreslit na povrchu koule a zobrazit tak celou planetu.

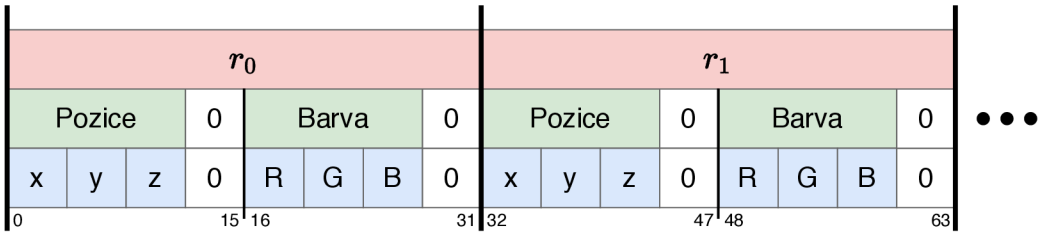
Aplikace k tomuto cíli využívá grafickou *pipeline*. Konkrétně vertex shader, který zajistí samotnou geometrii koule a fragment shader, který každý fragment přiřadí ke správnému regionu a získá tak jeho barvu. Grafická *pipeline* pracuje s polem regionů jako s uniform bufferem.

Příprava dat a nahrání na grafickou kartu

Aplikace obaluje obsluhu grafického API pomocí *vizualizačních technik*. Jedná se o objekt, který zaručí nastavení grafiky (shader programy, uniforms, VAO) a provede *drawcall* specifický pro konkrétní *entitu* scény (3D model, skybox, osy).

Vizualizační technika reprezentovaná třídou `PlanetVT` má za úkol zpracovat a připravit k vykreslení entitu planety `PlanetEntity`. Ta obsahuje právě vygenerovaný povrch (instance `Surface`). `PlanetVT` použije odpovídající vertex a fragment shader a nahraje povrchová data pomocí uniform bufferu do grafické paměti. Data budou strukturována jako jednorozměrné pole struktur, nesoucích globální pozici regionu a jeho barvu pro vykreslení, indexované podle indexačního systému z návrhu v sekci 4.3. Jedná se o minimální množství informací potřebné ke konstrukci Voroného diagramu na kouli. Obrázek 5.4 ukazuje rozložení dat regionů v uniform bufferu.

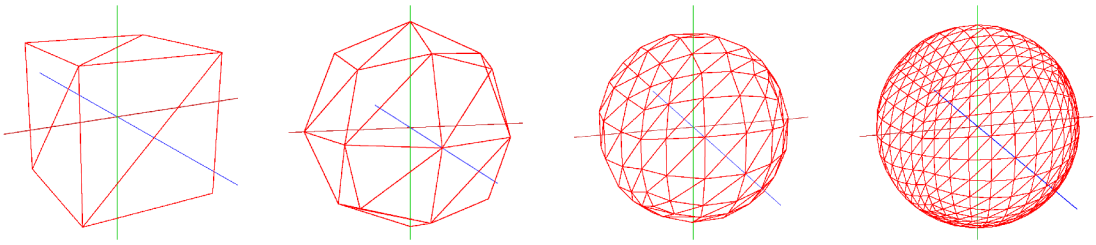
Pozici regionu poskytuje jeho rozhraní (viz sekce 5.2) a vizualizační technika zpracuje atributy v konkrétním regionu jako výslednou barvu. Lze tak definovat množství potomků `PlanetVT`, které budou implementovat pouze převod regionu na barvu a tak vizualizovat různý typ dat na povrchu (výšková mapa, teplota, vlhkost, biom, ...). Shader program následně zajistí vykreslení těchto zpracovaných dat jako Voroného diagram na kouli. Využívá přitom metody definované v návrhu.



Obrázek 5.4: Rozložení statických povrchových dat jako pole regionů v uniform bufferu na grafické kartě.

Generování kulaté geometrie ve vertex shaderu

Indexační systém i metody mapování povrchu koule vycházejí z normalizované dělené krychle a *cubemappingu*. Prakticky se jedná o texturování koule, nezáleží tedy příliš na samotné její geometrii. Proto je možné kulatou geometrii generovat stále stejnou, čistě na



(a) $R = 1$ (b) $R = 2$ (c) $R = 5$ (d) $R = 10$

Obrázek 5.5: Mesh koule generovaný ve vertex shaderu pro různá rozlišení R.

grafické kartě ve vertex shaderu, který proto nebere na vstup žádná data a jednoduše vypočítá polohy vrcholů na základě jeho indexu `gl_VertexID`. Tento shader používá vizualizační technika `PlanetVT`. Výstupem bude normalizovaná dělená krychle o určitém rozlišení, kde indexy jsou seřazeny podle stejných pravidel jako indexy regionů. Počet vrcholů, které vertex shader musí vygenerovat, je tedy roven $6 * 6 * R^2$ (každý čtverec ve straně tvoří dva trojúhelníky), přičemž R nemusí odpovídat R povrchu knihovny[6] a udává pouze počet stěn tělesa vzniklého po normalizaci (Aplikace používá $R = 10$.). Na obrázku 5.5 je vidět výstup vertex shaderu pro různá R.

Voroného diagram ve fragment shaderu

Samotné vykreslování povrchu planety v podobě různě barevných regionů je řešeno ve fragment shaderu. Pozice fragmentu je normalizovaná a každému se na povrchu koule přiřadí barva regionu, do kterého spadá (ke kterému těžišti je nejblíže). K tomu je využito indexačního systému a souvisejících transformací. Lze tedy v konstantním čase dopočítat index buňky, ve které fragment leží, a z jejího okolí vybrat nejbližší region porovnáním vzdáleností mezi těžišti ve 3D. K tomu byl v shaderu implementován souřadný systém (sekce 4.4) včetně *wrap* funkce podle algoritmu 2[7].

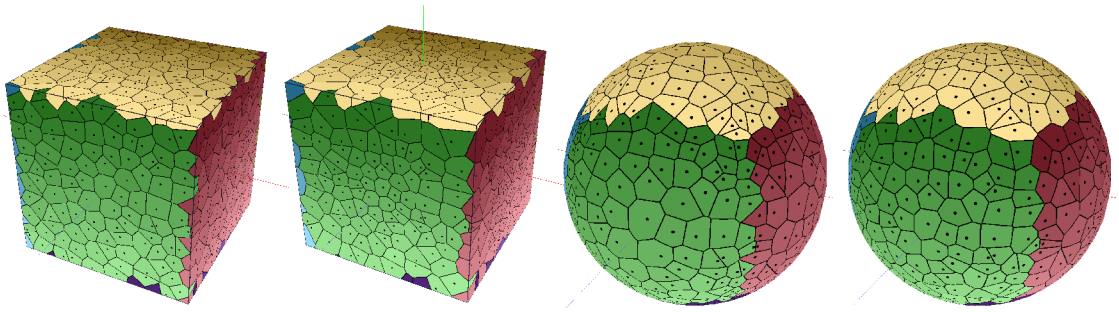
Shader a vizualizační techniky umožňují parametrizovat způsob vykreslení především pro demonstrační účely. Tyto parametry umožňují:

- Zapnout/vypnout vykreslování hranic a těžišť regionů.
- Zapnout/vypnout normalizaci přepínání mezi krychlí a koulí.
- Zapnout/vypnout použití *wrap* funkce.

Na obrázku 5.6 je demonstrována parametrizace shaderu za pomocí vizualizační techniky, která obarvuje regiony podle strany, na kterou spadají. Je zde dobře vidět, jak *wrap* algoritmus správně zaručí překročení strany a skutečný vliv *warp* funkce na tvar Voroného regionů. Obrázek 5.6d ukazuje výsledek používaný k finálnímu vykreslování planet.

[6] Implementace je řešená jako funkce v samostatném zdrojovém souboru:
`GeoPlanetDemo/resources/shaders/planet/calculateSphereVertex.glsh`

[7] Zdrojový kód fragment shaderu lze nalézt v souboru:
`GeoPlanetDemo/resources/shaders/planet/fragment.glsh`



(a) Diagram na krychli bez *warp* funkce.

(b) Diagram na krychli s *warp* funkcí.

(c) Diagram na normalizované krychli bez *warp* funkce.

(d) Diagram na normalizované krychli s *warp* funkcí.

Obrázek 5.6: Voroného diagram na povrchu geometrie planety generovaný ve fragment shaderu demonstrační aplikace za použití různé parametrizace vykreslování. Regiony jsou obarveny vizualizační technikou podle strany, na kterou spadají.

5.4 Generátor planet zemského typu

Součástí knihovny jsou implementované modifikátory, které slouží jako základní minimum k vytvoření povrchu planety. Tyto modifikátory jsou součástí generátoru planet zemského typu, který je také poskytován knihovnou. Tento generátor je prezentován v demonstrační aplikaci a jeho parametry (proměnné jeho modifikátorů) jsou nastavitelné uživatelem pomocí GUI.

Skládá se ze tří modifikátorů: „*Jitter*“, „*Tectonic plate*“ a „*Elevation*“. Každý modifikátor využívá data, která na povrch vygeneroval ten předchozí. Tato sekce popíše činnost algoritmů těchto klíčových modifikátorů.

Jitter

Modifikátor, který je aplikován jako první, je `JitterModifier`. Jeho úkolem je nastavit náhodný *offset* těžiště regionu v rámci jeho buňky (viz sekce 3.3). *Offset* se nastaví přímo do vlastnosti `localOffset` v objektu pozice regionu (`SurfacePosition`). Tuto operaci modifikátor provede pro každý region na povrchu individuálně. Je důležité, aby byl v *pipeline* jako první, protože je možné, že následující modifikátory budou pracovat s přesnou pozicí regionů. Poskytuje jednu nastavitelnou proměnnou `jitter`, která určuje *míru rozechvění regionů*.

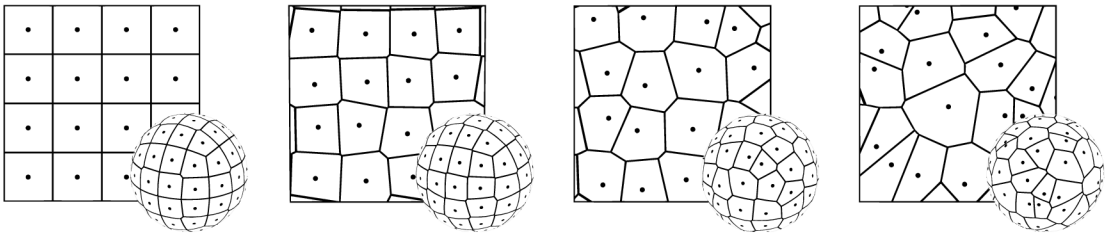
Jitter je reálné číslo v rozsahu $\langle 0, 1\rangle$, které nastaví odchylku souřadnic x a y *offsetu* těžiště od středu buňky $(0.5, 0.5)$. Pro *jitter* $j \in \langle 0, 1\rangle$ se nastaví $offset = (x, y)$ takto:

$$\begin{aligned} x &= 0.5 + random(0, j) - \frac{j}{2} \\ y &= 0.5 + random(0, j) - \frac{j}{2} \end{aligned} \tag{5.1}$$

Obrázek 5.7 ukazuje vliv proměnné *jitter* na *offset* regionu.

Tektonické pláty

V sekci 4.3 byl do návrhu datové reprezentace povrchu integrován koncept tektonických plátů pro možnost využít je během procedurálního generování. Z tohoto hlediska nám pláty



(a) *jitter* = 0 (b) *jitter* = 0.2 (c) *jitter* = 0.5 (d) *jitter* = 1

Obrázek 5.7: Efekt proměnné *jitter* modifikátoru `JitterModifier` na *offset* regionů.

poskytují užitečnou datovou strukturu umožňující vytvářet zajímavě vypadající procedurální povrchy planet. Uživatel však modifikátor nemusí použít, nepotřebuje-li v jeho procesu generování tektonické pláty.

Modifikátor `TectonicPlateModifier`, je stejně jako předchozí `JitterModifier`, základ, který nastavuje datovou strukturu povrchu do uceleného stavu. Po jeho aplikování bude mít povrch naplněné pole `plates` a jeho regiony budou obsahovat atribut odkazující na plát, ve kterém leží. Tato sekce popisuje jakým způsobem modifikátor pracuje s povrchem při vytváření plátů a ukazuje jeho možnosti parametrizace.

Algoritmus rozděluje regiony na povrchu do vzájemně disjunktních shluků, které jsou reprezentovány instancí třídy `TectonicPlate`. Musí platit, že každý region paří právě jednomu plátu. Za tímto účelem byl použit *flood fill* algoritmus[8]. Tento přístup rovněž použily ostatní práce využívající tektonické pláty na planetě [20, 11]. Zde je upravený, aby pracoval s datovou strukturou povrchu `Surface`, a je rozšířen o vstupní parametry měnící jeho vlastnosti. Těmito parametry jsou proměnné modifikátoru:

- `int plateNumber` – Udává celkový počet vytvořených plátů
- `float expansionRateRange` – Nastaví rozsah rychlostí expanzí plátům
- `bool randomDriven` – Příznak povolující náhodně řízenou expanzi
- `bool stepMode` – Příznak pro povolení krokovacího módu

Každá instance plátu `TectonicPlate` je schopna provést svoji vlastní expanzi voláním metody `expand`. Expanzí je myšleno, že si plát přivlastní okolní regiony (sousedy krajových regionů), které neleží na žádném plátu. Pokud při provádění expanze není přidán žádný nový region, plát se označí jako „dokončený“. Řízení expanzí zajišťuje modifikátor a končí tehdy, když jsou dokončeny všechny pláty.

Řídící proces modifikátoru v první fázi inicializace vytvoří `plateNumber` instancí plátů a každému určí náhodný počáteční region a „*rychlost expanze*“ (*rate*). Poté spustí cyklus, který provede *flood fill* opakovaným voláním metody `expand` jednotlivých plátů. Způsob provádění expanzí plátů je ovlivňován proměnnými `expansionRateRange` a `randomDriven`.

„Rychlost expanze“, nastavená `expansionRateRange`, je reálná hodnota přidělovaná každému plátu, která udává pravděpodobnost jeho expandování v jednom cyklu. Modifikátor si tyto rychlosti přiřazuje interně (mimo instanci plátu) jako náhodné reálné číslo z intervalu:

$$rate \in \begin{cases} \langle 1-e, 1+e\rangle, & \text{pro } e < 1 \\ \langle 0, e\rangle, & \text{pro } e > 1 \end{cases}, \text{ kde } e = \texttt{expansionRateRange}$$

[8] *Flood fill* algoritmus je hojně využívaný v grafických programech pro obarvení pixelů v určité souvislé oblasti.

Například při $rate = 1.5$ bude plát v jednom kroku expandován vždy aspoň jednou s 50% šancí na dodatečnou druhou expanzi. Jeho růst je tedy o polovinu rychlejší. Při $rate = 0.5$, bude mít plát 50% šanci, že nebude expandován vůbec, a jeho rychlost růstu je tedy poloviční. Algoritmus *flood fill* byl takto rozšířen, aby se jednotlivé pláty mohly lišit ve velikostech a měly více různorodé tvary, čímž byla narušena pravidelnost pro více přirozený vzhled. Řízení expanzí s použitím rychlostí *rate* popisuje algoritmus 3.

Algoritmus 3: *Flood fill* algoritmus plátů s použitím rychlosti expanzí

Vstup: $\mathcal{P} = \{p_0, p_1, p_2, \dots\}$ – množina tektonických plátů na povrchu, kde každý p_n obsahuje pouze jeden výchozí náhodný region na povrchu.

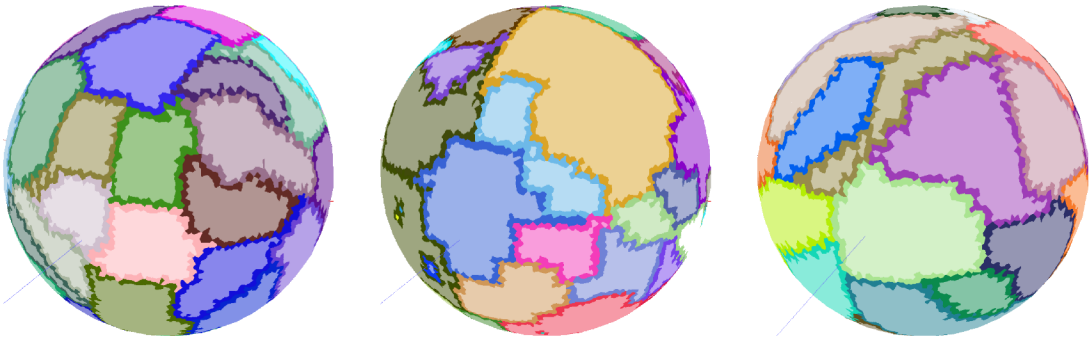
```
1:  while not allPlatesFinished(P) do
2:      foreach p in P do
3:          rate = getPlateExpansionRate(p)
            /* Expanze se provede tolikrát, kolik je celých jednotek v
               rychlosti tohoto plátu                                    */
4:          while rate > 0 do
5:              p.expand()
6:              rate = rate − 1
7:          end while
            /* Zbylé desetiny určí pravděpodobnost dodatečné expanze    */
8:          if random(0, 1) > rate then
9:              p.expand()
10:         end if
11:     end foreach
12: end while
```

Proměnná `randomDriven` určí, jestli pláty budou expandovány rovnoměrně, nebo budou řízeny náhodou. Rovnoměrná expanze znamená, že v jednom kroku cyklu je funkce `expandPlate` volaná jednou pro všechny pláty na povrchu. To zaručí, že jejich výsledný tvar je dán pouze jednotlivými náhodnými rychlostmi a počátečním rozložením (Rychlejší pláty mohou obklopit a zamknout pomalejší sousedy.). Náhodně řízená expanze (`randomDriven = true`) se snaží o více organicky tvarované pláty pomocí náhodného výběru jednoho kandidáta na expanzi v jednom kroku cyklu. Jedná se spíše o experimentální funkci, myšlenou jako alternativu k náhodným rychlostem expanzí. Ukázalo se ale, že ve srovnání s ní nepřináší znatelně lepší výsledky, jak je vidět na obrázku 5.8. Její neefektivita spočívá v tom, že může dojít k mnoha iteracím, během níž se algoritmus nebude moci trefit do ještě nedokončených plátu. Optimalizace by byla možná vyřazováním dokončených plátů, ta zatím však není implementována.

Proměnná `stepMode` slouží ke krokování expanze pro debugovací a demonstrační účely. Instance modifikátoru si v tomto případě zapamatuje stav a při volání metody `step` provede právě jeden krok algoritmu.

Výpočet elevace

Klíčovou vlastností každého terénu je jeho výšková mapa. V tomto případě výšková mapa znamená atribut „výška“ přidělený každému regionu. Jak bylo popisováno v sekci 2.5, výškové mapy se dají generovat mnoha způsoby. Významnou roli při vytváření reliéfu skuteč-



(a) Rovnoměrná expanze s $rate = 1$ pro všechny pláty.

(b) Náhodně řízená expanze generuje velmi nerovnoměrné tvary.

(c) Expanze s náhodnou rychlostí $rate \in \langle 0, 2 \rangle$ produkuje nerovnoměrné, ale souvislé pláty

Obrázek 5.8: Různé výstupy modifikátoru `TectonicPlateModifier` pro různé vstupní parametry. Vizualizace používá rozlišení povrchu $R = 50$ a `plateNumber` $= 50$. Výstupy jsou obarveny vizualizační technikou `PlanetPlatesVT`, která každému plátu přidělí barvu a zvýrazní jeho okrajové regiony.

ných pevných planet hrají tektonické pláty. Při pohledu z vesmíru díky nim lze pozorovat pohoří a souostroví táhnoucí se podél zlomů, kde se dva pláty střetávají, a příkopy a zálivy na místech, kde se oddalují.

Cílem modifikátoru `ElevationModifier` je využít definovanou datovou strukturu plátů a simulovat tyto geologické procesy při přidělování výšek jednotlivým regionům. Na výstupu potom v ideálním případě získáme právě zmiňovaná pohoří, souostroví a zálivy. Skutečná geologická simulace je ovšem výpočetně náročná a pro definovaný datový model postačí matematická abstrakce těchto procesů. Použité metody vychází z myšlenky, kterou použili Patel [20] i Gainey [11]. Spočívá v určení směru pohybu jednotlivým plátům a výpočtem jejich interakce v místech zlomů. Tam, kde se budou pohybovat směrem k sobě, bude výška regionů růst a naopak při pohybu od sebe bude klesat.

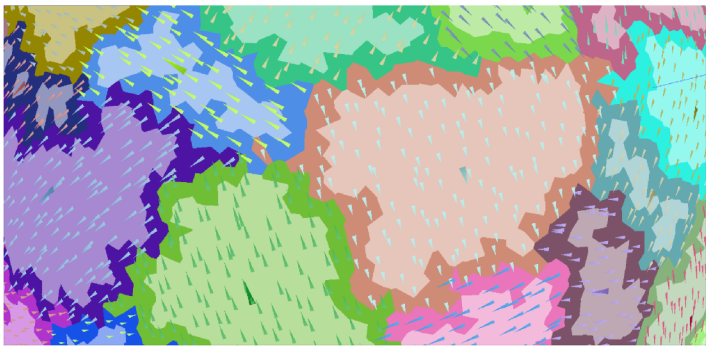
Pro vizualizaci výškové mapy je využita vizualizační technika `PlanetElevationVT`, která obarvuje regiony na základě atributu „elevation“. Jedná se o reálné číslo, které je chápáno jako nadmořská výška. Vizualizační technika potom bere hodnotu 0 jako hladinu oceánu, regiony s nižší výškou obarvuje modrou barvou a ty nad hladinou zase barvou hnědou. Odstín postupně tmavne s rostoucí vzdáleností od hladiny. Vytváří tak povědomou reliéfovou mapu pro intuitivnější vnímání terénu.

`ElevationModifier` vypočítává výšky regionů ve fázích:

1. Nastavení průměrné výšky plátů.
2. Provedení kolizí plátů.
3. Přičtení dodatečného šumu na výšku všech regionů.
4. Vyhlazení povrchu.

Stejně jako ostatní modifikátory i tento nabízí různé proměnné pro parametrizaci, které budou popsány při popisech jednotlivých fází.

Obrázek 5.9: Náhodně rozdělené výšky jednotlivých plátů při $R = 30$ a `plateNumber` $= 50$.



Obrázek 5.10: Vektory pohybů tektonických plátů a jejich regionů.

Nastavení průměrných výšek plátů

První fází je přiřazení průměrné výšky plátům. Tato výška bude počáteční pro všechny regiony v plátu a dostaneme tak představu kolik procent povrchu bude pod hladinou oceánu. Pro demonstrační účely modifikátor přiděluje výšku náhodně rovnoměrně z intervalu okolo 0. Tento interval je nastavitelný proměnnou modifikátoru `elevationRandomRange`. Výška plátu *elevation* bude náhodné reálné číslo z intervalu:

$$elevation \in \langle -e_r, e_r \rangle, \text{ kde } e_r = \texttt{elevationRandomRange}$$

Při této implementaci budou pláty z 50% pevninské a z 50% podmořské. Obrázek 5.9 ukazuje povrch po této první fázi.

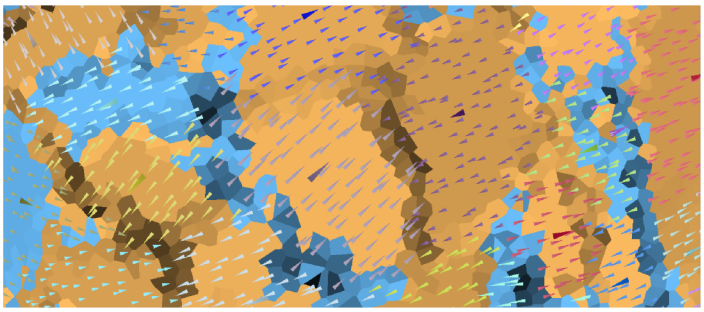
Simulace kolizí plátů

V této stěžejní části procesu generování dochází k výpočtu kolizí mezi jednotlivými tektonickými pláty. Jak bylo naznačeno, jedná se o algoritmus, který přidělí plátům pohybový vektor a v místech zlomů upraví výšku regionů podle směru jejich pohybu. K tomuto účelu je nutné definovat střed plátu p_c a vektor jeho pohybu p_s, který bude ležet na rovině tečné k jeho středu. Každý region náležící plátu potom zdědí tento vektor. Na obrázku 5.10 jsou tyto vektory vizualizovány. Pohybové vektory mohou být různě velké a tím značit „sílu“ s jakou tlačí na svého souseda, či se od něj vzdalují.

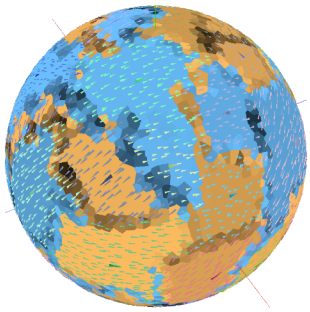
Výpočet středu plátu s_p je prostým průměrem všech těžišť jeho regionů. Tečný vektor pohybu v_p je vypočten jako náhodný vektor ležící v rovině, ke které je středový vektor plátu normálovým:

$$\begin{aligned} t_x &= p_c \times (1,0,0) \\ t_y &= p_c \times (0,1,0) \\ t_r &= random(-1,1) \cdot t_x + random(-1,1) \cdot t_y \\ v_p &= random(0.5,1) \cdot \frac{t_r}{\|t_r\|} \end{aligned} \tag{5.2}$$

Výpočet kolizí je proveden samostatně pro každý okrajový region na povrchu. Korekce výšky krajového regionu je provedena tak, že se určí celkový „tlak“ na něj působící. Tlak



(a) Detail kolize mezi pláty. Směřující od sebe vytváří rokli a proti sobě pohoří.



(b) Posuvy plátů v kontextu celé planety simulují reliéfy kontinentů.

Obrázek 5.11: Vizualizace kolizí plátů a jejich efekt na výšku regionů podél zlomů. Čím tmavší barva tmavší, tím je výška vzdálenější od hladiny.

působící na okrajový region r označíme jako $\rho_r \in \mathbb{R}$. Při $\rho_r < 0$ region klesá a při $\rho_r > 0$ stoupá a vypočítá se jako součet všech tlaků, působící na r z okolních regionů:

$$\rho_r = \sum_{i=0}^{|\mathcal{N}|} \left[C \cdot \left[\|v\| \cdot \big(v \cdot (p_i - p)\big) + \|v_i\| \cdot \big(v_i \cdot (p - p_i)\big) \right] \right] \tag{5.3}$$

, kde $\mathcal{N} = \{r_0, r_1, r_2, \dots\}$ je okolí regionu r, v a v_i jsou vektory pohybu regionů r a r_i, které jsou tečné k jejich těžištím se směrem a magnitudou zděděnou od jejich plátů a p a p_i jsou pozice těžišť regionů r a r_i. Platí, že tlak mezi dvěma regiony ležícími na stejném plátu bude 0. Konstanta $C \in \mathbb{R}$ nastavuje, jak moc silný efekt na výšku regionů bude kolize plátů mít. Lze ji nastavit proměnnou modifikátoru `collisionStrength`. Na obrázku 5.11 je ukázán výsledek tohoto procesu. Je zde vidět, jak pláty pohybující se proti sobě způsobují vyzvednutí okrajových regionů. Proměnná modifikátoru `usePlateCollisions` je příznak, který povolí výpočet kolizí plátů při generování povrchu.

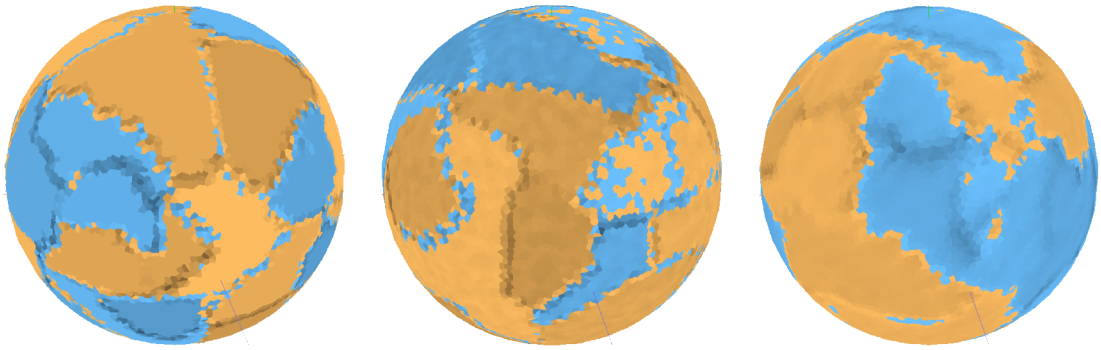
Zanesení šumu na výšky regionů

Jak je vidět při výpočtech kolizí, upravujeme výšku pouze okrajových regionů, vnitřek plátů je ovšem stále plochý. Modifikátor byl rozšířen o jednoduchý generátor trojrozměrného Perlinova šumu za použití hlavičkové knihovny `PerlinNoise.hpp`[9]. V demonstrační aplikaci je možné jeho frekvenci nastavit pomocí následujících proměnných:

- `usePerlin` – Příznak povolující aplikování šumu na povrch, povoleno ve výchozím stavu.
- `perlinFrequency` – Reálné číslo nastavující frekvenci šumu, 15.0 ve výchozím stavu.
- `perlinStrength` – Reálné číslo nastavující amplitudu šumu, 2.0 ve výchozím stavu.

Výchozí hodnoty jsou nastaveny tak, aby podávaly dobře vypadající výsledky při výchozích nastavení ostatních modifikátorů. Ve výsledku budou na povrchu drobné nerovnosti a pohoří a příkopy nebudou působit tak rušivým dojmem na jinak rovném povrchu. Obrázek 5.12b ukazuje efekt aplikování šumu na terén po provedení kolizí.

[9] Hlavičková open source knihovna od Ryo Suzuki dostupná z https://github.com/Reputeless/PerlinNoise



(a) Povrch po provedení kolizí (b) Povrch s aplikovaným šumem (c) Vyhlazený finální povrch

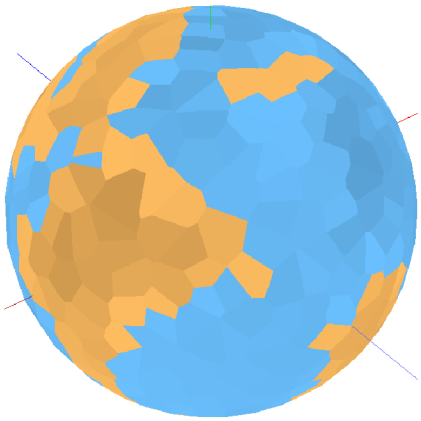
Obrázek 5.12: Efekt aplikování dodatečného šumu a vyhlazení na povrch s vypočtenými kolizemi. Vizualizace používá výchozí hodnoty generátoru.

Vyhlazení povrchu

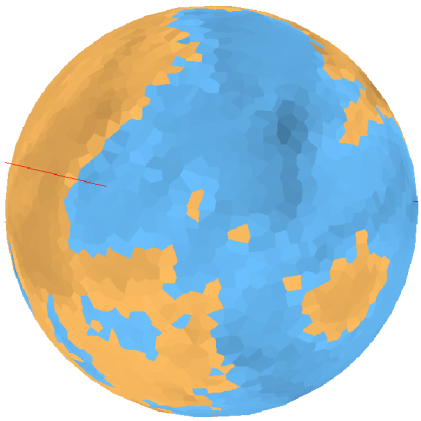
Jak je vidět na obrázcích 5.12a a 5.12b, změny výšek ve zlomech jsou poměrně ostré a příliš lokalizované. Šumy zase způsobují artefakty, kdy pláty s výškou okolo 0 oscilují nad a pod mořskou hladinou. Rovněž mezi různě vysokými pláty je změna výšky skoková. Proto jako finální úpravu provedeme vyhlazení aplikováním nízkofrekvenčního filtru, který upraví výšku regionu na průměr výšek jeho okolí. Po aplikování na všechny regiony vypadá povrch více plynule a realisticky. Toto je poslední fáze modifikátoru `Elevation modifier` a výsledná výšková mapa je vizualizovaná na obrázku 5.12c. Výsledek obsahuje všechny slíbené prvky jako jsou kontinenty, ostrovy, zálivy a podmořské příkopy.

Omezení současného řešení

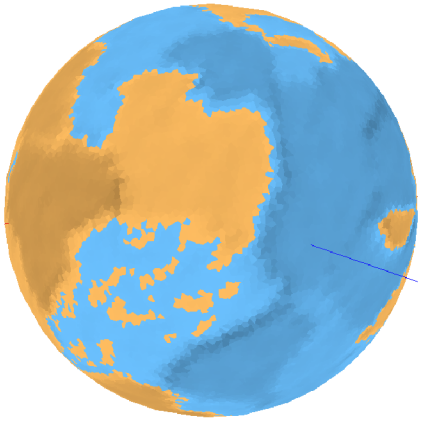
Generátor má v současném stavu jistá omezení, která za určitých podmínek způsobují úpadek kvality výsledného povrchu. Asi největším nedostatkem je špatná škálovatelnost. Pro vizuálně zajímavý terén je třeba udržet určitou velikost regionů (ve výchozím stavu $R = 50$) tak, aby jednotlivé tvary měly vliv na členitost okrajů plátů. Už při $R > 70$ jsou jednotlivé zlomy příliš úzké a vnitřní oblasti plátů se stávají monotónní. Rovněž se začínají projevovat artefakty vzniklé zanesením šumu, a plocha relativně nízko položeného plátu osciluje mezi zápornou a kladnou hodnotou (obrázek 5.13e). Proto je třeba udržovat R v přiměřeném rozsahu. Důležitý je poměr velikostí plátů a regionů, tedy pro vyšší rozlišení je třeba více plátů, tím ale vznikne členitější povrch (obrázek 5.13f). Tyto problémy by bylo teoreticky možné vyřešit zvětšením prohledávaného okolí, síly kolizí a amplitudy šumu při rostoucím R. Obrázek 5.13 ukazuje povrchy při různých rozlišeních.



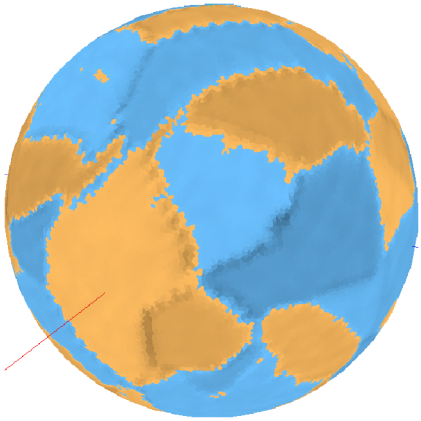
(a) $R = 10$ – Velmi nízká rozlišení vytváří dlaždicový povrch.



(b) $R = 25$ – Při nízkých rozlišeních kolize způsobují výškové gradienty.

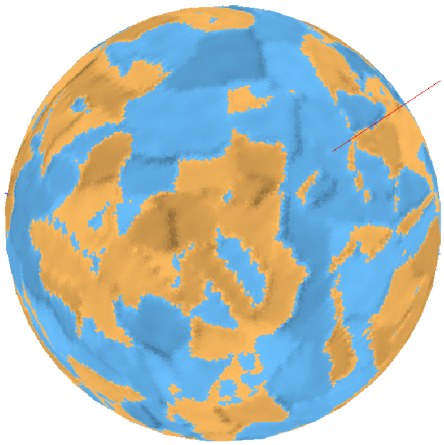


(c) $R = 60$ – Hrany kolizí jsou ostřejší. Lze si všimnout artefaktů šumové funkce.



(d) $R = 80$ – Začíná se vizuálně projevovat monotónnost velkých plátů.

(e) $R = 120$ – Pruhy změněné výšky ve zlomech. Projevují se artefakty šumové funkce.



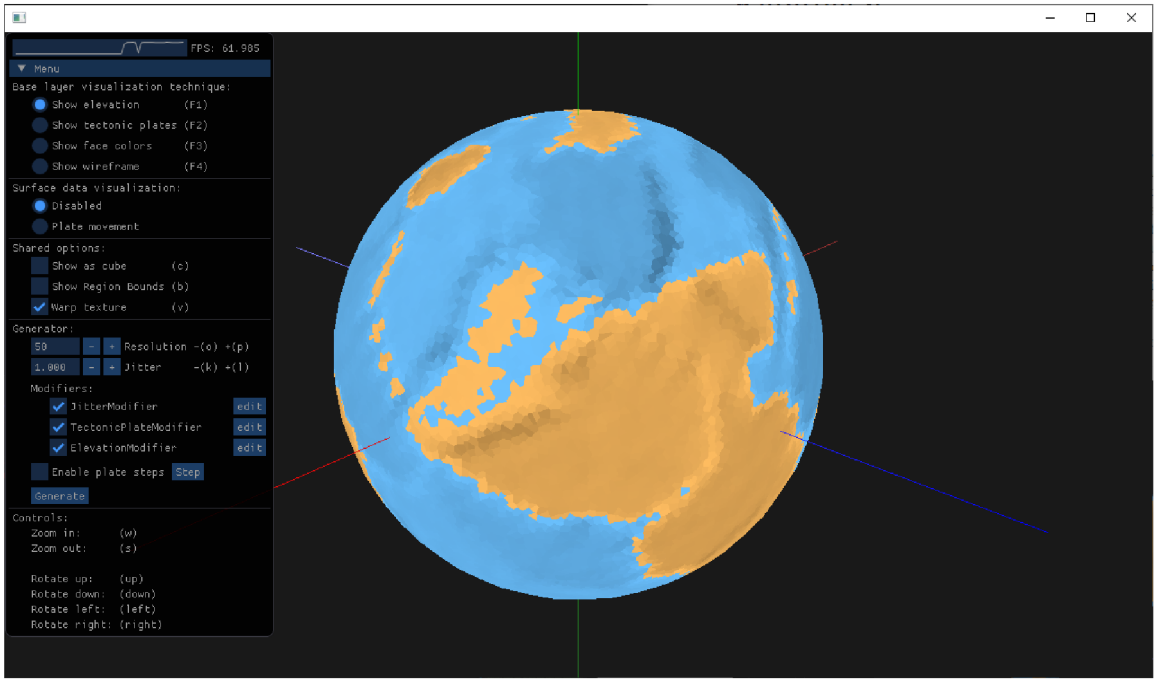
(f) $R = 120$ – Monotónnost je kompenzována vyšším množstvím plátů $P = 250$

Obrázek 5.13: Porovnání vzhledu povrchů při použití různých rozlišení a souvisejících nedokonalostí.

Kapitola 6

Závěr

V této práci byla prozkoumána metoda, jak adaptovat dvourozměrné generátory map na povrch koule a vytvořit tak knihovny pro procedurální generování planetárních povrchů. K tomu bylo využito poznatků z procedurálního texturování koulí za použití „kulaté krychle“ v kombinaci s metodami generování planetárních výškových map s aplikací kolizí tektonických plátů. Byla implementována knihovna `GeoPlanetLib`, poskytující abstrakci pro jednoduchou práci s diskreditovaným povrchem koule a zapouzdřující potřebné operace do unifikovaného souřadného systému. Funkci knihovny demonstruje aplikace `GeoPlanetDemo`, která umožňuje nastavení všech proměnných poskytovaných předpřipravenými modifikátory a zkoumání jejich efektu na generovaný povrch planety. Její uživatelské rozhraní s vizualizovanou planetou ukazuje obrázek 6.1. Planetární povrch je vykreslen jako textura ve fragment shaderu, který využívá operace souřadného systému knihovny. Aplikace nabízí uživateli různé vizualizační techniky pro zobrazení datových vrstev regionů, jako jsou výšky, tektonické pláty a vektory jejich pohybů. Tyto vrstvy byly



Obrázek 6.1: Snímek obrazovky s GUI aplikace `GeoPlanetDemo`

použity pro vygenerování obrázků v této práci, sloužících pro popis principu generování. V poslední řadě demonstrační aplikace nabízí krokovací režim, který krok za krokem demonstruje proces generování planety.

V rámci práce bylo vytvořeno krátké video, ukazující jak z krychle vytvářet procedurální planety. Video je obsaženo na přiloženém datovém nosiči. Obsah tohoto nosiče je popsán v příloze A.

Samotná knihovna ustanovuje základ pro vytváření komplexních generátorů povrchových dat o kontextu planety. Jejím cílem není nahradit stávající metody vizualizace ani generování planetární geometrie. Poskytuje ale způsob jak dát této geometrii určitý kontext a strukturu tak, aby procedurální planeta působila jako svět a nejen geometrická reprezentace různých matematických operací. Knihovna umožňuje využít procedurálních generátorů map, které jsou dnes ve 2D schopny vygenerovat svět do velkých detailů. Rovněž lze na navrženém povrchu provádět i různé další celulární algoritmy.

Knihovna momentálně postrádá některé klíčové funkcionality, které je třeba adresovat v nejbližší navazující práci. Kromě komplexnějšího generování je třeba implementovat export a import vytvořených povrchů a instancí generátorů. Knihovna by také měla nabízet možnost vytvořit z aktuálního povrchu obyčejnou cylindrickou texturu či cubemapu.

V budoucím vývoji knihovny by bylo možné nad existující výškovou mapou simulovat proudění vzduchu, které by mezi regiony distribuovalo vlastnosti jako je vlhkost a teplota a z těchto dat by se vypočítával typ biomu. Takto nyní pracuje aplikace *Worldbuilder*[1], která je svou podstatou velice podobná `GeoPlanetDemo`. Tu v roce 2015 publikoval A. Gainey a vychází z jeho řešení procedurální planety [11], které je jednou z inspirací této práce. Lze přidávat i další procesy generování map, například korozivní algoritmy jsou schopné vytvořit zálivy a fjordy. Pro procedurální mapy se rovněž užívají generátory procedurálních řek, měst, cest, států, národů apod.

Možná využití knihovny by mohla být následující: Exportování do textur by se dalo, při správné stylistické vizualizaci, využít pro „generátor procedurálních zeměpisných glóbů“. Aplikace využívající knihovnu by mohly produkovaná povrchová data použít jako kontext k procedurálně generovaným planetám s plnou geometrií, vizualizací a *LOD*, kde v rámci každého regionu bude použit jiný generátor terénu podle příslušného biomu. Například region obsahující město, spustí na daném místě procedurální generátor měst.

Co se týče stávající demonstrační aplikace, ta by, po implementaci exportu, mohla sloužit jako konfigurátor generátorů s živým náhledem. Pro aplikace nepracující s knihovnou by mohla být užita pro generování planetárních textur. Uživatel knihovnu používající by ji mohl využít pro nakonfigurování a exportování generátoru, který mu pak v jeho aplikaci bude produkovat náhodné planety v jednom konkrétním stylu. Mohl by také exportovat pouze jeden konkrétní povrch a vektorová data použít jako základ svého herního světa.

[1] Gainey A. *Worldbuilder* [Online, navštíveno 16.5.2020] Dostupné z https://experilous.com/1/product/worldbuilder-demo

Literatura

[1] Blinn, J. F. Light Reflection Functions for Simulation of Clouds and Dusty Surfaces. In: *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1982, s. 21–29. SIGGRAPH 82. DOI: 10.1145/800064.801255. ISBN 0897910761. Dostupné z: https://doi.org/10.1145/800064.801255.

[2] Bourke, P. *Circles and spheres: Sphere Generation*. 1992. [Online; navštíveno 16. 4. 2020]. Dostupné z: http://paulbourke.net/geometry/circlesphere/.

[3] Bourke, P. *Modelling fake planets*. 2000. [Online; navštíveno 19. 4. 2020]. Dostupné z: http://paulbourke.net/fractals/noise/.

[4] Compton, K., Grieve, J., Goldman, E., Quigley, O., Stratton, C. et al. Creating Spherical Worlds. In: *ACM SIGGRAPH 2007 Sketches*. New York, NY, USA: Association for Computing Machinery, 2007, s. 82–es. SIGGRAPH 07. DOI: 10.1145/1278780.1278879. ISBN 9781450347266. [Online; navštíveno 18. 4. 2020]. Dostupné z: https://www.cs.cmu.edu/~ajw/s2007/0251-SphericalWorlds.pdf.

[5] Cortial, Y., Peytavie, A., Galin, E. a Guérin, E. Procedural Tectonic Planets. *Computer Graphics Forum*. Květen 2019, sv. 38, s. 1–11. DOI: 10.1111/cgf.13614. [Online; navštíveno 21. 4. 2020]. Dostupné z: https://perso.liris.cnrs.fr/eric.galin/Articles/2019-planets.pdf.

[6] Cozzi, P. a Ring, K. *3D Engine Design for Virtual Globes*. 1st. CRC Press, June 2011. ISBN 978-1568817118. Dostupné z: http://www.virtualglobebook.com.

[7] Davidson, C. *Tectonics.js*. [Online; navštíveno 21. 4. 2020]. Dostupné z: http://davidson16807.github.io/tectonics.js/.

[8] Dirac, P. *Generating the planet*. 2019. [Online; navštíveno 18. 4. 2020]. Dostupné z: https://metriximor.wordpress.com/2019/01/19/mapping-planets/.

[9] Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C. et al. ROAMing terrain: Real-time Optimally Adapting Meshes. 1997, s. 81–88. [Online; navštíveno 23. 4. 2020]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.1811&rep=rep1&type=pdf.

[10] Elias, H. *Spherical landscapes*. [Online; navštíveno 19. 4. 2020]. Dostupné z: http://web.archive.org/web/19991102231524/http://freespace.virgin.net/hugo.elias/models/m_landsp.htm.

[11] Gainey, A. *Procedural Planet Generation*. 2014. [Online; navštíveno 18. 12. 2019]. Dostupné z: http://experilous.com/1/blog/post/procedural-planet-generation.

[12] Greene, N. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications*. 1986, sv. 6, č. 11, s. 21–29. [Online; navštíveno 17. 4. 2020]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4056759/authors#authors.

[13] Lagae, A., Lefebvre, S., Cook, R., Derose, T., Drettakis, G. et al. A Survey of Procedural Noise Functions. *Computer Graphics Forum*. 2010, sv. 29. DOI: 10.1111/j.1467-8659.2010.01827.x.

[14] Lambers, M. Survey of cube mapping methods in interactive computer graphics. *The Visual Computer*. Červen 2019. DOI: 10.1007/s00371-019-01708-4. [Online; navštíveno 19. 3. 2020]. Dostupné z: https://marlam.de/publications/cubemaps/lambers2019cubemaps.pdf.

[15] Lerbour, R., Marvie, J.-E. a Gautron, P. Adaptive Real-Time Rendering of Planetary Terrains. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*. Leden 2010, sv. 18, s. 89–96. [Online; navštíveno 19. 3. 2020]. Dostupné z: http://wscg.zcu.cz/WSCG2010/Papers_2010/!_2010_FULL-proceedings.pdf.

[16] National Geospatial-Intelligence agency (NGA). *World Geodetic System 1984 (WGS 84)*. 2014. [Online; navštíveno 16.11.2019]. Dostupné z: https://earth-info.nga.mil/GandG/update/index.php?dir=wgs84&action=wgs84.

[17] Nykamp, D. Q. *"Spherical coordinates." From Math Insight*. 2018. [Online; navštíveno 15. 12. 2019]. Dostupné z: https://mathinsight.org/spherical_coordinates.

[18] O'Leary, M. *Generating fantasy maps*. [Online; navštíveno 29. 4. 2020]. Dostupné z: http://mewo2.com/notes/terrain/.

[19] O'Neil, S. *A Real-Time Procedural Universe: Part One: Generating Planetary Bodies*. 2001. [Online; navštíveno 19. 4. 2020]. Dostupné z: https://www.gamasutra.com/view/feature/131507/a_realtime_procedural_universe_.php.

[20] Patel, A. *Procedural map generation on a sphere*. 2018. [Online; navštíveno 11. 1. 2020]. Dostupné z: https://www.redblobgames.com/x/1843-planet-generation/#rendering.

[21] Patel, A. *Planetary Dungeon*. 2019. [Online; navštíveno 05. 1. 2020]. Dostupné z: https://www.redblobgames.com/x/1939-planetary-dungeon/.

[22] Patel, A. *Wraparound square tile maps on a sphere*. 2019. [Online; navštíveno 10. 1. 2020]. Dostupné z: https://www.redblobgames.com/x/1938-square-tiling-of-sphere/.

[23] qotop. *Making addons: PLANET TEXTURES*. [Online; navštíveno 18. 4. 2020]. Dostupné z: http://spaceengine.org/manual/making-addons/planet-textures/.

[24] Romanyuk, V. *Space Engine*. [Online; navštíveno 17. 12. 2019]. Dostupné z: http://spaceengine.org/.

[25] Ulrich, T. Rendering Massive Terrains Using Chunked Level of Detail Control. 2000. [Online; navštíveno 23. 4. 2020]. Dostupné z: http://tulrich.com/geekstuff/sig-notes.pdf.

[26] United States. Defense Mapping Agency. Geodesy for the layman. In:. DMA technical report, 80-003. [Washington, D.C.]: Defense Mapping Agency, 1983, kap. 5 - PHYSICAL GEODESY. [Online; navštíveno 07. 12. 2019]. Dostupné z: https://www.ngs.noaa.gov/PUBS_LIB/Geodesy4Layman/TR80003C.HTM.

[27] Wikibooks. *Spherical Coordinates (Colatitude, Longitude).svg*. 2008. [Online; navštíveno 15. 12. 2019]. Dostupné z: https://commons.wikimedia.org/wiki/File:Spherical_Coordinates_(Colatitude,_Longitude).svg.

[28] Zucker, M. a Higashi, Y. Cube-to-sphere projections for procedural texturing and beyond. *Journal of Computer Graphics Techniques (JCGT)*. 2018, sv. 7, s. 11–22. [Online; navštíveno 05. 1. 2020]. Dostupné z: http://jcgt.org/published/0007/02/01/.

Příloha A

Obsah přiloženého paměťového média

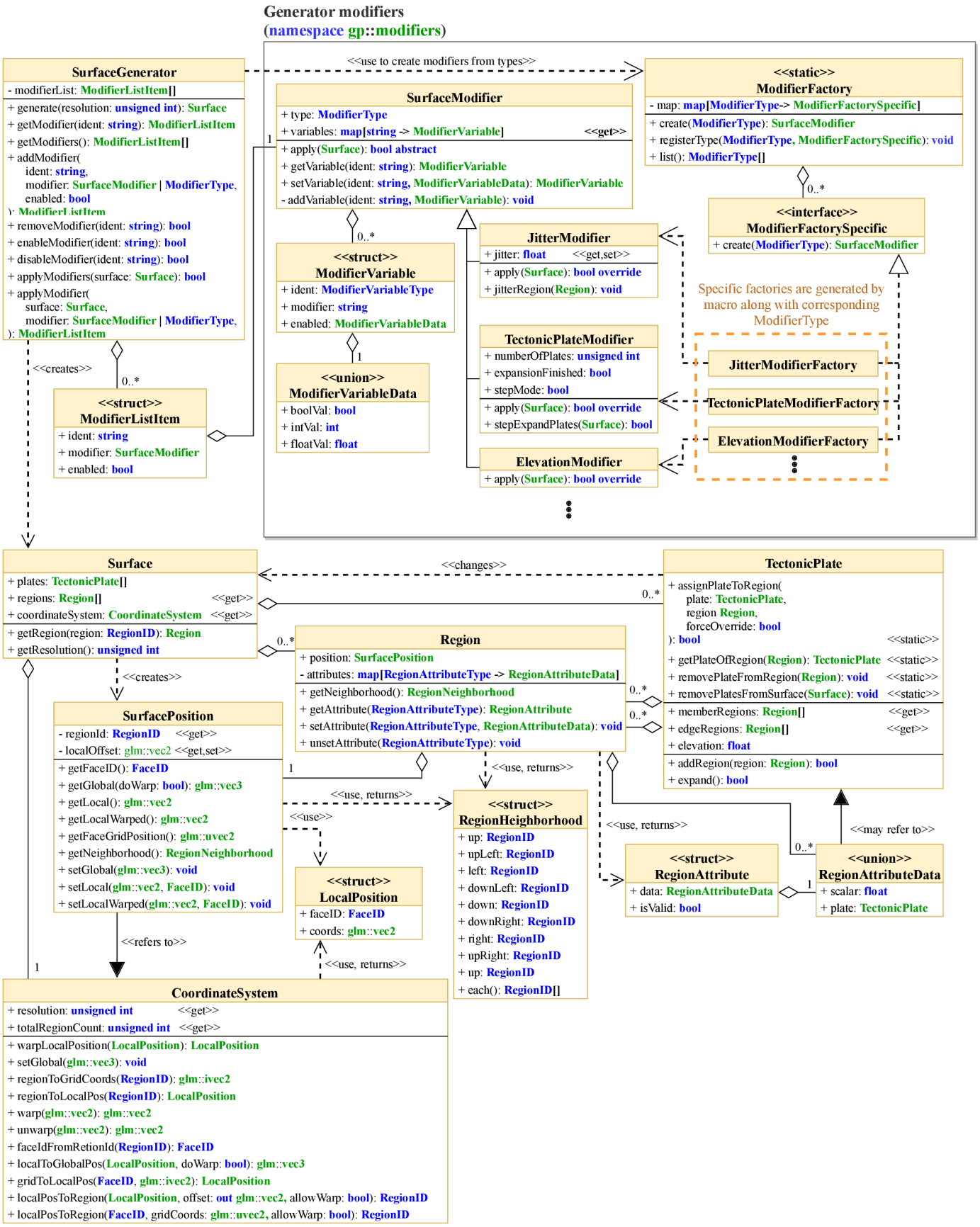
Na přiloženém paměťovém médiu naleznete následující:

- `/`
  - `Dokumnetace/`
    - `latex/` ..... složka se zdrojovými LaTeX soubory pro vygenerování této zprávy
    - `Demo_class_diagram.svg` ..... diagram struktury aplikace ve formátu `svg`
    - `Lib_class_diagram.svg` ..... diagram struktury knihovny ve formátu `svg`
    - `Projekt.pdf` ..... kopie této technické zprávy v `pdf`
  - `Projekt/`
    - `bin/` ..... složka obsahující spustitelný program pro platformu Win_x64
    - `source/` ..... složka obsahující veškeré zdrojové soubory projektu
      - `3rdPartyLib/` ..... přeložené knihovny třetích stran
      - `GeoPlanetDemo/` ..... kopie repozitáře projektu demonstrační aplikace
      - `GeoPlanetLib/` ..... kopie reposztáře projektu knihovny
      - `CMakeLists.txt` ..... konfigurace kompletního projektu pomocí CMake
      - `Readme.md`
  - `Video/`
    - `A_Procedural_Planet_From_Cube.mp4` ..... video ukazující výsledky práce
  - `Readme.md`

Přiložené zdrojové soubory ve složce `Projekt/source/` jsou konfigurovány pomocí *CMake* pro vygenerování projektu v *Microsoft Visual Studio 19*. Na této platformě byly přeloženy i poskytnuté knihovny třetích stran ve složce `3rdPartyLib`. `GeoPlanetLib` a `GeoPlanetDemo` jsou projekty a lze je přeložit samostatně s manuální instalací požadovaných knihoven i pro jiné platformy (netestováno).

Příloha B

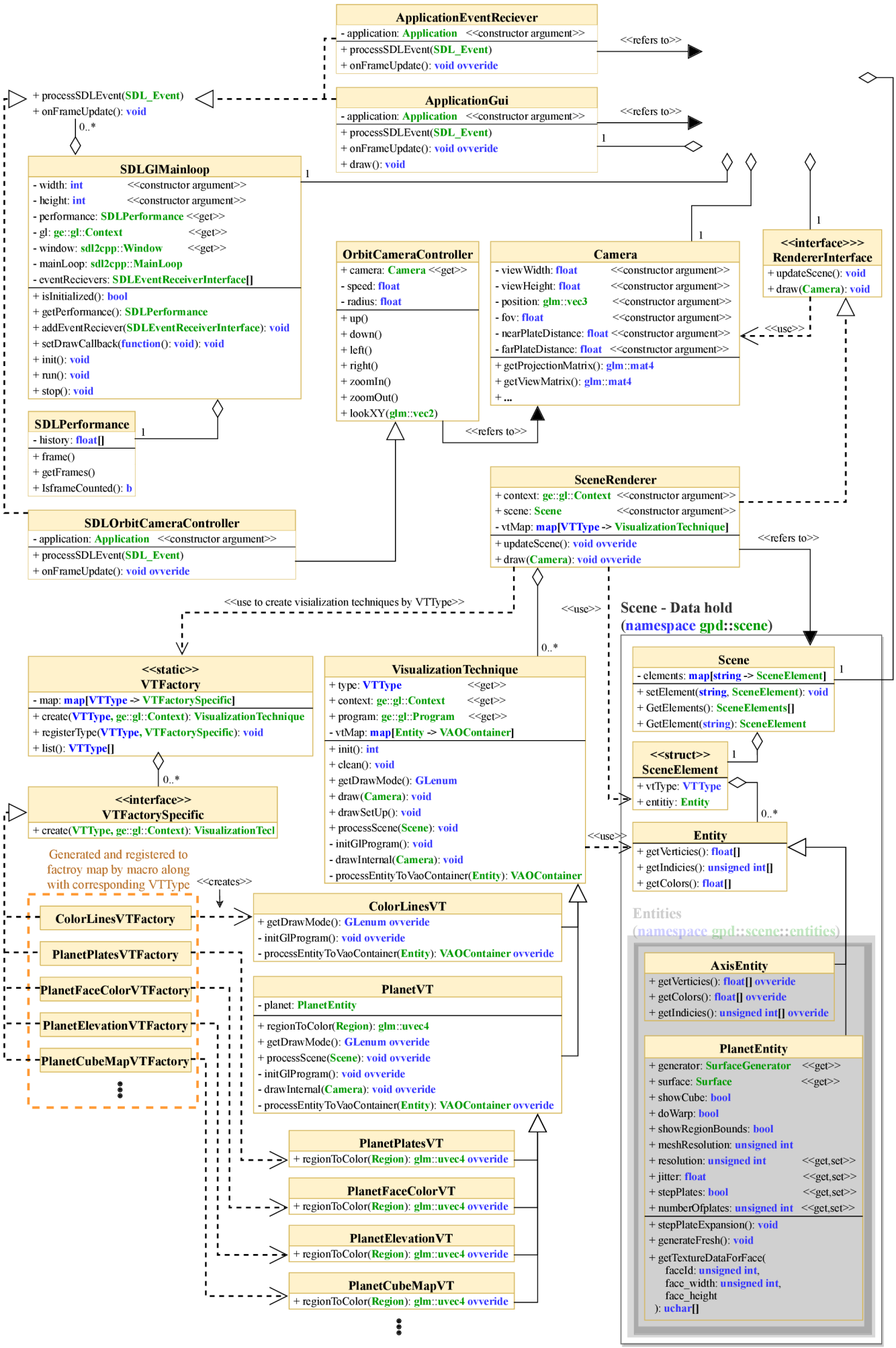
Diagram tříd `GeoPlanetLib`



Obrázek B.1: Třídí diagram struktury knihovny `GeoPlanetLib`.

Příloha C

Diagram tříd `GeoPlanetDemo`



Obrázek C.1: Třídí diagram struktury aplikace `GeoPlanetDemo`.