



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 A ÚPRAVA 3D TERÉNU PODLE MAP
TERRAIN GENERATION AND ADJUSTEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ OPLATEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ STARKA,

BRNO 2022

Zadání bakalářské práce



20024

Student: **Oplatek Tomáš**

Program: Informační technologie

Název: **Generování a úprava 3D terénu podle map**
Terrain Generation and Adjustement

Kategorie: Počítačová grafika

Zadání:

1. Natudujte API poskytovatelů map a terénních dat.
2. Navrhněte algoritmus, který na základě metadat (např. z mapového podkladu) upraví a případně rozšíří 3D model terénu, opět získaný od třetí strany. Vstupní metadata by měla zahrnovat minimálně definici vodních ploch, cest a budov.
3. Vytvořte demonstrační aplikaci, která umožní parametrizovat vstupy a prohlížet výstup.
4. Vytvořte krátké video k práci.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Bod 1 a část bodu 2.

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 2021

Datum odevzdání: 11. května 2022

Datum schválení: 1. listopadu 2021

Abstrakt

Cílem této práce bylo vytvoření takové aplikace, která koncovému uživateli dovolí vygenerovat libovolnou oblast na zemi jako 3D mapu, kterou si bude uživatel moci prohlédnout a seznámit se s touto oblastí, než se tam případně vydá. Aplikace byla vyvíjena jako desktopová aplikace v jazyce C++ s využitím OpenGL na zobrazení výsledku a Qt frameworku na uživatelské rozhraní. Přínosem aplikace je, že dovoluje nastavit například hustotu vegetace, a uživateli se tak zobrazí i běžně skryté cesty, a i na méně známých místech poskytuje reálné cesty.

Abstract

The goal of this thesis was to create an application, which allows the end user to generate any area on earth as a 3D map, which they could look at and get to know, before they set out on a journey there. The application was developed as a desktop application in the C++ language with the use of OpenGL to render the result and Qt framework for user interface. Added value of this application is, that it allows to set for example the density of vegetation, so that small paths, that are usually hidden are visible and it provides real paths even in lesser known areas.

Klíčová slova

3D mapa, desktopová aplikace, procedurální generace, triangulace polygonu, post-process efekty, deferred shading, C++, OpenGL

Keywords

3D map, desktop application, procedural generation, polygon triangulation, post-process effects, deferred shading, C++, OpenGL

Citace

OPLATEK, Tomáš. *Generování a úprava 3D terénu podle map*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka,

Generování a úprava 3D terénu podle map

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.

.....
Tomáš Oplatek

30. července 2022

Poděkování

Chtěl bych poděkovat vedoucímu této práce panu Ing. Tomáši Starkovi za rady a konzultaci při tvoření této práce. Dále bych chtěl poděkovat všem, kteří mě při tvoření práce podpořovali a poskytli důležité rady. V poslední řadě bych chtěl poděkovat těm, kteří otestovali aplikaci a tím pomohli najít a vyřešit řadu nedostatků.

Obsah

1	Úvod	2
2	Analýza současného stavu a grafické technologie	3
2.1	Dvourozměrné mapy	3
2.2	Trojrozměrné mapy	4
2.3	Metody zvyšování kvality modelu	5
2.4	Metody Ambient Occlusion	8
2.5	Triangulace polygonu	10
2.6	Procedurální generování vegetace	13
2.7	Deferred shading	16
3	Výběr technologií a návrh funkcionality aplikace	18
3.1	Architektura aplikace	18
3.2	Výběr zvolených technologií	19
3.3	Návrh knihovny	24
3.4	Návrh uživatelské aplikace	25
4	Implementace aplikace	28
4.1	Generování mapy	28
4.2	Zobrazení výsledku	33
4.3	Uživatelské rozhraní	35
4.4	Externí dependence aplikace	37
5	Vyhodnocení a testování	39
5.1	Testování uživateli	39
5.2	Výkon aplikace	39
5.3	Nedostatky v aplikaci	41
6	Závěr	42
	Literatura	43
A	Obrázky s vygenerovanou mapou	45
B	Konfigurační soubor	47
C	Obsah paměťového média	48

Kapitola 1

Úvod

Digitální mapy dnes používá již téměř každý. Není divu, že popularita papírových map klesá. Papírové mapy obsahují jen určitou oblast, jsou rozměrné, neukazují polohu, a člověk je musí koupit a nesmí je zapomenout doma. Digitální mapy jsou naproti tomu snadno dostupné v mobilním telefonu, který má v dnešní době naprostá většina lidí. Jejich užívání je jednoduché a praktičtější, mapy jsou stále aktuální a poskytují uživateli i dodatečné informace o provozu. Populární je i využití map pro plánování cesty či tras, a stoupá i využití 3D map – ty jsou užívány zejména pro prozkoumání daného prostředí, často nějaké části města, což uživateli může pomoci se na daném místě lépe zorientovat. A proč tuto možnost, která je často dostupná ve městech a známých místech, nerozšířit i na odlehlejší místa, například turistické cesty? Uživatel by tak mohl lépe prozkoumat i místa, která jsou skryta v lesích či horách, a díky tomu by mohl například i lépe odhadnout, zda je zvolená trasa v jeho silách, nebo zda je vhodné zvolit jinou.

Motivací pro tuto bakalářskou práci je rozšířit existující 3D mapy o podrobnější a uživatelsky přívětivější zobrazení, které by vhodně doplňovalo již existující mapy.

Práce je členěna do čtyř kapitol. První z nich se zabývá analýzou současných řešení a popisem technologií používaných k zobrazování grafické aplikace. Druhá obsahuje primárně výběr použitých technologií a návrh různých implementačních částí. V pořadí třetí kapitola se věnuje implementaci aplikace, a poslední kapitola se věnuje vyhodnocení a testování aplikace.

Kapitola 2

Analýza současného stavu a grafické technologie

Tato kapitola má za účel přiblížit čtenáři základní princip zobrazení map, a to jak klasických 2D, tak i map s 3D zobrazením. Dále přiblížuje porovnání různých typu zobrazení dat, mapových podkladů, a různých technologií a metod, které se využívají na generování a zobrazení mapy. Nejedná se o podrobný popis uvedených problematik, ale jen o seznámení se základními fakty a postupy.

2.1 Dvourozměrné mapy

Dvourozměrnou (2D) mapou rozumíme klasickou mapu světa, která je buď digitální, nebo klasická papírová. Poskytovatelů online digitálních podkladů je dostupných celá řada. Většina z nich funguje tak, že když si uživatel otevře online mapu, většinou získá pouze několik obrázků, které jsou na sebe skládány a dohromady tvoří mapu. Každý z poskytovatelů nabízí jiné možnosti, jak data získat a jaký typ dat vrací. Nejznámější z nich jsou popsány v následujících sekcích.

Google Maps

Google Maps obsahují velmi přesné mapové podklady téměř celého světa – obzvláště pak měst a zabydlených oblastí. Chybí zde ale menší lesní stezky. Další velkou výhodou jsou funkce typu Street View, které uživateli dovolí se procházet ulicemi na místech, kde je tato funkce dostupná. Dostupnost není ale z daleka všude, a uživatel se může pohybovat pouze po předem daných místech.

Google mapy nabízí API, které obsahuje endpointy na získávání informací přímo z Google map. Příkladem se jedná o informace jako výšková data, která je ale možné získat pouze pro konkrétní body nebo jako trasu mezi dvěma různými body¹. Používání tohoto rozhraní je zdarma, pokud uživatel nepřekročí určitý počet požadavků².

OpenStreetMap

OpenStreetMap jsou otevřené, kolaborativní mapy, které jsou zcela zdarma. Dobrou vlastností je právě kolaborace, kdy data do map zadávají uživatelé samotní. Díky tomu jsou

¹<https://developers.google.com/maps/documentation/elevation/overview>

²<https://mapsplatform.google.com/pricing/>

mapy většinou přesnější a obsahují i ty nejmenší cesty. OpenStreetMap jsou také používány velkými společnostmi jako je Amazon, Facebook, nebo Tesla, které přispívají na vznik kvalitních mapových podkladů.

Pro práci s OpenStreetMap je dostupné oficiální API, které je zcela zdarma, a dovoluje uživatelům čtení a zápis přímo do a z map. Nevýhodou je, že je optimalizované pro editaci map. Pro případ, kdy uživatel chce z aplikace pouze získávat data a nedělat žádné úpravy, není rychlosť ideální. Jeho otevřenosť ale dovolila vznik dalších aplikací, které jsou postavené na OpenStreetMap API, a dovolují lépe optimalizovaný přístup pro čtení. Příkladem je Overpass API, které je optimalizované na rychlosť a na to, aby uživatel získal přesně taková data, která potřebuje, pomocí vlastního dotazovacího jazyku.

Data získána z API jsou standartně ve formátu XML³ a jsou složeny z částí node, way, relation. Node symbolizuje nějaký bod na mapě a jeho vlastnosti (např. roh budovy). Way je způsob, jakým jsou body pojmenované nodes propojeny – seznam několika bodů může tvořit třeba hranice budovy. Relation popisuje vztahy mezi jednotlivými cestami (například budova je součástí zahrady). Právě tyto vlastnosti nabízejí jednoduchou práci s těmito daty, a proto jsou využívány řadou aplikací a projektů. Informace o OpenStreetMap jsou získány z [8].

Bing Maps

Bing Maps jsou mapy od společnosti Microsoft. Poskytuje API obsahující podrobné výškové informace o libovolné oblasti v relativně vysokém rozlišení, zjištění adresy z bodu a bodu z adresy (geocoding), a spousty dalších funkcí. Je možné získat také informace o oblasti ve formátu obrázku. Mezi další důležité funkce patří například možnost jednoduše integrovat mapy do webových aplikací pomocí dostupného SDK, nebo do aplikací vyvíjených pro knihovny WPF a UWP, které jsou součástí frameworku .NET.

2.2 Trojrozměrné mapy

Digitální trojrozměrné (3D) mapy mají za účel zobrazit uživateli svět tak, jak skutečně vypadá. Většinou toho docílí skládáním informací z mnoha různých zdrojů jako je LIDAR, fotogrammetrické snímky pořízené ze se satelitů, letadel, nebo i klasických topografických 2D map. Existuje řada poskytovatelů digitálních 3D map, mezi které patří například Google Earth a Esri ArcGIS.

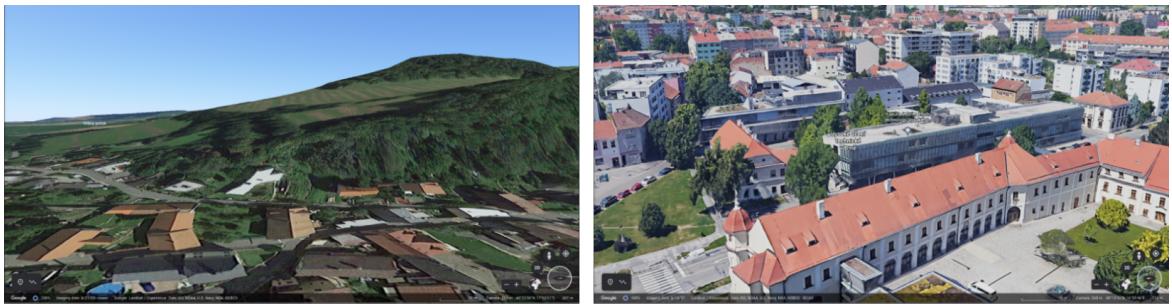
Google Earth

Google Earth je v dnešní době jednou z nejpopulárnějších služeb na zobrazení reálného světa jako 3D objektu. Je multiplatformní, základem je webová aplikace, ale má také spoustu nativních aplikací pro všechny hlavní, ať už mobilní, nebo desktopové platformy. Svá data sbírá pomocí satelitních a leteckých snímků, které jsou snímány po dobu několika měsíců pro každou oblast⁴. To dovoluje Googlu mít velmi kvalitní a detailní 3D mapu, která téměř perfektně replikuje reálný svět. V okolí velkých měst funguje většinou bez sebemenšího problému – budovy vypadají reálně, jsou vidět detaily jako jednotlivé stromy, nebo dokonce auta zaparkována na ulici. Ovšem u míst, která jsou od měst vzdálena, je kvalita map většinou podstatně horší. Například zdaleka ne všechny budovy jsou 3D, lesy bývají často

³Extensible Markup Language

⁴<https://support.google.com/earth/answer/6327779>

vykresleny špatně – obzvláště pak v případě, že je les na kopci. Mapy také nejsou vhodné pro člověka, který se chce například připravit na túru – malé cesty v lese či horách většinou nejsou viditelné. Porovnání mapy města a vzdálenější oblasti je ukázáno na obrázku 2.1.



Obrázek 2.1: Porovnání detailů vesnice (vlevo) a města (vpravo) na Google Earth.

Mimo jiné má Google Earth i svoje vlastní rozhraní, které je součástí Google Maps API. Nabízí oproti 2D mapám funkce pro 3D. Má tedy stejný ceník a podmínky používání.

Esri ArcGIS

ArcGIS je sbírka produktů nabízející 2D a 3D mapové podklady. Obsahuje metody, kterými si uživatel může sám vytvořit vlastní mapy. Má funkce na skládání dat z různých podkladů, vizualizaci vlastních mapových dat, nebo knihovnu pro jazyk Python, ve které uživatelé mohou zpracovávat data. Základní formou je desktopový klient fungující na všech hlavních platformách, ale existuje i online varianta.

2.3 Metody zvyšování kvality modelu

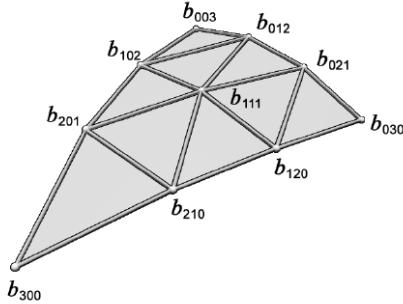
Pro realistické zobrazení scény je nutné, aby byly modely ve scéně v dostatečně vysokém rozlišení. Když je rozlišení moc nízké, začne docházet k tomu, že mezi jednotlivými polypyramidy jdou vidět tvrdé přechody. Tomu se dá předejít použitím detailnějšího modelu, ale když model s lepšími detaily není k dispozici (at řež z důvodu že se šetří paměť, nebo jednoduše prostě neexistuje), je nutné udělat approximaci, jak by model mohl vypadat ve vyšším rozlišení. Na to je možné použít několik různých metod, z nichž nejpoužívanější jsou Curved PN Triangles, Patching Catmull-Clark Meshes a ACC Patches.

Curved PN Triangles

Curved PN Triangles⁵ je metoda, která používá rozdelení trojúhelníků na několik menších, čímž prakticky vzniknou nové trojúhelníky, které se vykreslují. Tyto nové (menší) trojúhelníky jsou rozmištěny do prostoru podle původních bodů a normálových vektorů těchto bodů. Body jsou rozdeleny kubickým způsobem, takže jedna hrana původního trojúhelníku je rozdělena na tři menší, zatímco normálové vektory původních bodů jsou naopak rozděleny kvadratickým způsobem, takže jedna hrana je rozdělena na dvě. Tímto lze dosáhnout dostatečně kvalitní nové geometrie i lepších možností stínování, a to bez vytváření zbytečných dat. Obzvláště důležitý je prostřední bod, který je na obrázku 2.2 označen jako b₁₁₁,

⁵Curved Point Normal Triangles

který dovoluje vypouknutí středu nové geometrie. Informace o této metodě byly získány z práce Curved PN triangles [12].



Obrázek 2.2: Ukázka rozdělení trojúhelníků pomocí techniky Curved PN triangles.⁶

Pokud jsou známé rohové body trojúhelníku P_1, P_2, P_3 a jejich normály N_1, N_2, N_3 , tak lze vypočítat síť zvanou control net, jež souřadnice b_{ijk} $i, j, k \in \{0, 1, 2, 3\}$ (viz obrázek 2.2) se získávají následujícím způsobem:

$$\begin{aligned} b_{300} &= P_1 & b_{210} &= \frac{(2P_1 + P_2 - w_{12}N_1)}{3} & b_{120} &= \frac{(2P_2 + P_1 - w_{21}N_2)}{3} \\ b_{030} &= P_2 & b_{012} &= \frac{(2P_3 + P_2 - w_{32}N_3)}{3} & b_{021} &= \frac{(2P_2 + P_3 - w_{23}N_2)}{3} \\ b_{003} &= P_3 & b_{102} &= \frac{(2P_3 + P_1 - w_{31}N_3)}{3} & b_{201} &= \frac{(2P_1 + P_3 - w_{13}N_1)}{3} \\ && b_{111} &= \frac{E + (E - V)}{2} && \end{aligned}$$

$$w_{ij} = (P_j - P_i) \cdot N_i \text{ kde } \cdot \text{ značí skalární součin}$$

$$E = (b_{210} + b_{120} + b_{021} + b_{012} + b_{102} + b_{201})/6$$

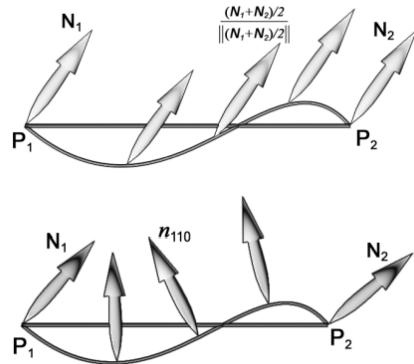
$$V = (P_1 + P_2 + P_3)/3$$

Následně jsou každému takovému nově vygenerovanému bodu v síti přiřazeny hodnoty u, v, w . Tyto hodnoty jsou v rozmezí od 0 do 1, a symbolizují hodnotu nového bodu podle barycentrických koordinátů na své Béziér ploše. Z těchto bodů je pak možné vypočítat body nových trojúhelníků tímto způsobem:

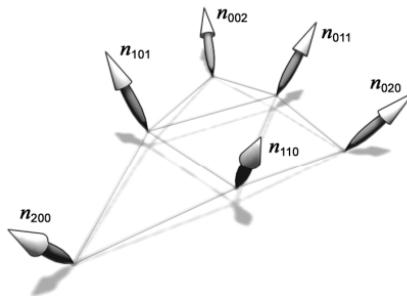
$$b(u, v) = \sum_{i+j+k=3} \frac{3!}{i!j!k!} u^i v^j w^k$$

Také je nutné vypočítat hodnoty normál pro takto vytvořené nové body, a to tak, že se vytvoří síť bodů, které jsou buď lineárně nebo kvadraticky interpolovány. Lineární interpolace narozdíl od kvadratické nebene v potaz nově vytvořené zakřivení mezi jednotlivými body, jak je ukázáno na obrázku 2.3. Jednodušší na implementaci je lineární interpolace, ale díky nižší přesnosti se většinou využívá kvadratické.

⁶Obrázek převzat z: [12]



Obrázek 2.3: Porovnání lineárního (nahore) a kvadratického (dole) výpočtu normál. Šipky značí směr normálního vektoru.⁸



Obrázek 2.4: Ukázka kvadratického rozdělení normál na celém trojúhelníku.⁹

Výpočet hodnot normál $n_{i,j,k}$, $i, j, k \in \{0, 1, 2\}$ pro jednotlivé body na obrázku 2.4 kvadratickou metodou je definován takto:

$$\begin{aligned}
 n_{200} &= n_1 & n_{110} &= \frac{h_{110}}{\|h_{110}\|} & h_{110} &= N_1 + N_2 + v_{12}(P_2 - P_1) \\
 n_{020} &= n_2 & n_{011} &= \frac{h_{011}}{\|h_{011}\|} & h_{011} &= N_2 + N_3 + v_{23}(P_3 - P_2) \\
 n_{002} &= n_3 & n_{101} &= \frac{h_{101}}{\|h_{101}\|} & h_{101} &= N_3 + N_1 + v_{31}(P_1 - P_3) \\
 && n_{ijl} &= 2 \frac{(P_j - P_i) \cdot (P_i + P_j)}{(P_j - P_i) \cdot (P_j - P_i)}
 \end{aligned}$$

Normály pro body nově vytvořeného trojúhelníku jsou definovány následovně:

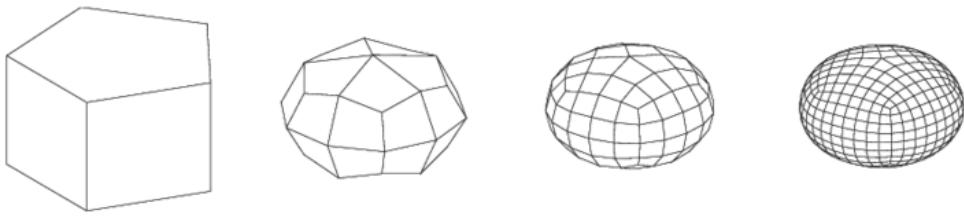
$$n(u, v) = \sum_{i+j+k=2} n_{ijk} u^i v^j w^k$$

⁸Obrázek převzat z: [12]

⁹Obrázek převzat z: [12]

Patching Catmull-Clark Meshes

Narozdíl od metody PN Triangles se tato metoda zabývá rozdělováním čtverců. Ty jsou rozděleny bikubickým způsobem, takže z jednoho čtverce vznikají 4 nové, a při vytváření se bere v potaz i okolí původních bodů. Při výpočtu nové geometrie tedy nejsou využívány normálové vektory, ale pouze pozice bodů. Metoda vytváří NURBS¹⁰ plochy. Jedná se o plochu tvořenou body, které mají nějakou pozici a váhu a podle toho je vypočteno zaoblení plochy. Touto metodou je možné získat plochy s hladkými přechody na všech částech dělené plochy, vyjma okrajů, na kterých je nedostatek informací (právě proto, že nemají okolní body). Zdrojem této metody je [9].



Obrázek 2.5: Ukázka rozdelení modelu na detailnější pomocí metody PCCM.¹¹

Metoda se dá na stejný objekt volat opakovaně a model je tak rozdělován a approximován stále silněji. Toto je možné vidět na obrázku 2.5.

ACC Patches

Metoda ACC Patches¹² je kombinací výše popsaných metod. Informace o ní jsou čerpány z [5]. Oproti metodě Patching Catmull-Clark Meshes je méně náročná na výpočty, protože ji pouze approximuje. Pro každý čtverec geometrie je vytvořena plocha s novou geometrií (geometry patch), a dvě plochy složeny z tečen (tangent patch), podle kterých jsou určeny normály. Jak už bylo zmíněno, původní metoda nevytváří hladké přechody u okrajů. ACC Patches toto řeší pomocí použití pole tečen, podle jejichž hodnoty se počítají normály nového povrchu. Metoda také vyžaduje méně větvění, takže je vhodnější pro výpočty a má lepší podporu pro SIMD.

2.4 Metody Ambient Occlusion

Ambient Occlusion (AO) je technika nasvětlení scény, která approximuje množství světla dosahující na určitý povrch podle přímo viditelných dalších povrchností v jeho okolí. Používá se k tomu, aby scéně přidala jemné stíny, které připomínají globální iluminaci v místech jako například rohy místnosti. Zde nemusí dopadat klasické stíny, ale je zde méně světla. Standardní AO je velmi pomalý proces. Pro každý bod se vyšle několik paprsků z polokoule (ta je sestrojena v okolí bodu), pomocí kterých se zjistí tvar okolní geometrie. Používá se tedy hlavně v aplikacích, kde není kladen důraz na vysoký výkon. V kontextu této práce

¹⁰non-uniform rational basis spline

¹¹Obrázek převzat z [9]

¹²Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches

je AO důležité pro přidání reálnějšího zobrazení ať už generované vegetace, budov, nebo povrchu samotného.

Screen Space Ambient Occlusion

SSAO používá k zjištění informací o okolí hloubkový buffer, do kterého jsou zapisovány informace o fragmentech geometrie při jejím renderování, a poté jsou z něj čteny vzorky s informacemi o okolí. Pro SSAO algoritmus jsou potřeba dva průchody. V prvním jsou vygenerovány textury, které jsou později použity pro výpočet AO. V druhém kroku je vyrenderována samotná scéna, na kterou se přidá SSAO. Vygenerované textury potřebné pro výpočet SSAO jsou následující:

- **SSAO Depth Texture** – textura, do které se zapisuje hloubkový buffer scény
- **SSAO Normal Texture** – textura, do které se zapisuje normálový buffer scény
- **SSAO Random Texture** – textura, do které se zapisuje náhodný šum
- **SSAO Texture** – textura, do které se zapisuje výsledek SSAO

Pro každý fragment se provede výpočet SSAO funkce. Pomocí vzorkování okolních fragmentů se zjistí tvar okolní geometrie z Depth Texture. Jaké fragmenty jsou vzorkovány je zjištěno z Random Texture. Následně se vypočítají rozdíly v hloubce aktuálního fragmentu, a všech vzorkovaných okolních fragmentů. Tyto hodnoty jsou normalizovány pomocí maximální hloubky okluze. Následně se vyvzorkované hodnoty zprůměrují, a výsledek je faktor okluze. Takto získaná hodnota generuje relativně vysoký šum jak je vidět na obrázku 2.6.



Obrázek 2.6: Ukázka SSAO na scéně ze hry Starcraft II (náhodný šum).¹³

Tento šum je možné odstranit pomocí funkce na rozmazávání obrazu, například upraveným filtrem **Gaussian Blur**. Tento filtr se musí upravit tak, aby se nezávislé objekty vzájemně po rozmazání neovlivnily. Na to se použijí Depth Texture a Normal Texture, které jsou vzorkovány v okolí rozmazávaného bodu. Pokud je okolní hloubka moc vysoká, nebo je skalární součin hodnoty normály bodu a hloubky moc malý, tak je výsledek nahrazen hodnotou 0. Šum je možné aplikovat několikrát, aby se scéna vyhladila co nejlépe.

¹³Obrázek převzat z: <https://developer.amd.com/wordpress/media/2013/01/Chapter05-Filion-StarCraftII.pdf>

Alternativním způsobem aplikací šumu je vybrat několik okolních pixelů a jednoduše je vyprůměrovat, ale to způsobí, že efekt vypadá mírně rozmazaně. Zdrojem informací o této metodě je [13].

Horizon Based Ambient Occlusion

Horizon Based Ambient Oclusion (HBAO) je metoda, která dosahuje přesných výsledků za cenu poměrně vyššího času zpracování. Je to fyzikálně založený algoritmus, jehož princip je takový, že approximuje integrál pomocí vzorkování hloubkového bufferu scény. Na určitém povrchu, jsou měřeny jisté úhly, pomocí kterých se zjistí, jak silná bude míra světelné okluze. V potaz se bere nejen vzdálenost (hloubka), ale i směr normálových vektorů povrchu oproti světu (horizontu). Počítá se tedy i se směrem povrchu, díky čemuž se dají získat přesnější výsledky. Časová náročnost algoritmu způsobí, že je efekt většinou renderován v nižším rozlišení, a kvůli tomu vznikají určité renderovací artefakty. Ty se dají řešit pomocí postupného filtrování dat a následného odstranění šumu pomocí rozmazaní obrazu podobně jako u SSAO, viz 2.4. Zdrojem těchto informací je [1].



Obrázek 2.7: Ukázka HBAO ze hry Battlefield 3. Na obrázku je zobrazen pouze HBAO průchod scénou.¹⁵

Na obrázku 2.7 je ukázka jak vypadá pouze tento efekt v praxi. Rozšířením této metody je metoda HBAO+, která je přesnější oproti HBAO a dokonce řeší problémy s výkonem.

2.5 Triangulace polygonu

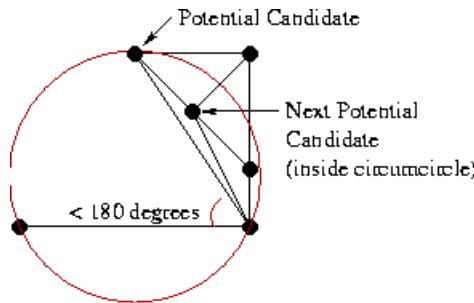
Pro vykreslování budov je nejdříve potřeba budovy převést na trojúhelníky tak, aby je bylo možné vykreslit na grafické kartě. To je nutné z toho důvodu, že budovy mají v reálném světě tvar polygonu o n hranách. Jedním z algoritmů pro triangulaci je Constrained Delaunay Triangulation (CDT). Je to rozšíření algoritmu Delaunay Trinagulation, který neumí pracovat s konkávními tvary. Tento algoritmus to umí, ale vstupem je kromě jednotlivých bodů také seznam hran (constrained edge – omezující hrana), popisující, jak má výsledný objekt vypadat.

¹⁵Obrázek převzat z: https://developer.nvidia.com/sites/default/files/akamai/gamedev/files/gdc12/GDC12_Bavoil_Stable_SSAO_In_BF3_With_STF.pdf

Delaunay Triangulation

Proces generování Deaunay Triangulation má několik možných algoritmů. Nejrychlejší metodou¹⁶ je *divide and conquer*, která triangulaci vypočítá v čase $O(n * \log(n))$. Ta funguje dle [10] následovně:

1. Všechny body jsou seřazeny podle jejich koordinátů x. V případě, že jsou dva koordináty x stejné, jsou seřazeny podle bodu y.
2. Seznam bodů je postupně rozdělován na poloviny, dokud nejsou v polovinách dva, nebo tři body. Ze dvou bodů se vytvoří segment, ze tří přímo trojúhelník.
3. Jednotlivé poloviny bodů jsou rekurzivně spojeny s druhou polovinou, a výsledkem je triangulace, která se skládá z několika typů hran: LL – bod je z levé triangulace a konec má také v levé triangulaci; LR – bod je z levé triangulace a konec má v pravé; RR – začátek i konec jsou v pravé triangulaci. Při spojování těchto polovin bodů jsou vytvářeny LR hrany. Spojení polovin se dělá následovně:
 - 3.1. Vloží se základní LR hrana, která spojuje nejnižší body triangulací tak, aby neprotínala žádnou LL ani RR hranu.
 - 3.2. Je třeba vytvořit další hranu nad touto nově vytvořenou hranou. Na to se musí zvolit další bod, buď z levé, nebo pravé triangulace, který bude spojen s jedním z původních bodů nové hrany. Potencionální body na spojení jsou vybrány v pořadí velikosti úhlu mezi hranou vytvořenou v bodě 3.1 a jednotlivými body v této polovině. Každý z těchto bodů je zkонтrolován dvěma pravidly. Prvním pravidlem je, že úhel mezi body musí být menší než 180° . Druhým pravidlem je, že opsaná kružnice okolo dvou bodů původní LR hrany a aktuálního potencionálního bodu nesmí obsahovat další potencionální bod. Výběr je zobrazen na obrázku 2.8.



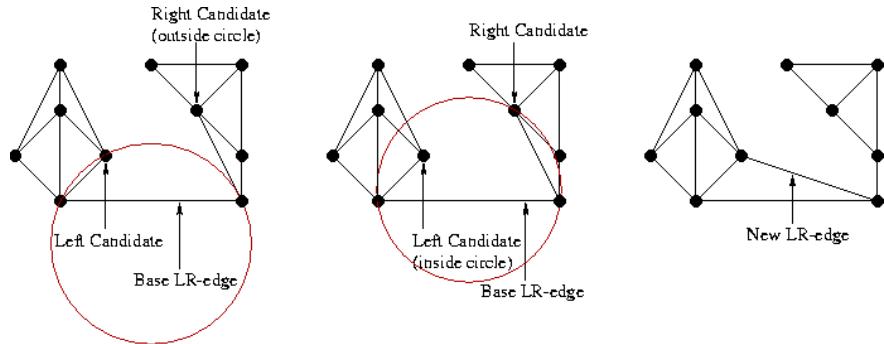
Obrázek 2.8: Výběr potencionálního kandidáta pro spojení dvou polovin.¹⁷

- 3.3. Pokud jsou obě podmínky splněny, je vybrán první kandidátní bod. Pokud není první podmínka splněna, musí být vybrán bod z druhé poloviny. Pokud není splněna druhá podmínka, je odebrána RR hrana mezi kandidátním bodem a původním pravým bodem. Tento proces je opakován se všemi body, dokud není vybrán kandidát, nebo se nepočítá s druhou stranou.
- 3.4. Pokud na pravé straně nebyl žádný kandidát vybrán, proběhne výběr pro levou stranu stejným způsobem.

¹⁶https://en.wikipedia.org/wiki/Delaunay_triangulation

¹⁷Obrázek byl převzat z: http://www.geom.uiuc.edu/~samuel/del_project.html

- 3.5. Když není z pravé ani levé poloviny získán žádný bod, spojení je kompletní a pokračuje se dalšími polovinami. V případě, že je získán jen jeden bod, znamená to, že je automaticky spojen, a přeskočí se na krok 3.7. V případě, že jsou získány body z obou stran, provede se test, podle kterého bude bod vybrán.
- 3.6. Tento test je následující: pokud pravý kandidátní bod není v kruhu okolo původní LR hrany a levého kandidátního bodu, je vybrán levý bod a naopak. Této podmínce bude vždy vyhovovat právě jeden z bodů. Pro představu je výběr zobrazen na obrázku 2.9.



Obrázek 2.9: Test jestli bude vybrán levý nebo pravý kandidátní bod. Na obrázku byl vybrán levý, protože vyhovuje podmínce.¹⁹

- 3.7. Jakmile jsou body propojeny a je vytvořena nová LR hrana, propojování pokračuje s dalšími hranami s tím, že za původní hranu se považuje nově vytvořená hrana.
4. Poté, co jsou spojeny i poslední dvě poloviny, je triangulace dokončena.

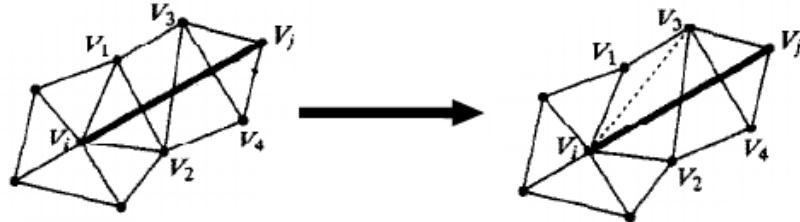
Constrained Delaunay Triangulation

Existuje řada algoritmů pro sestrojení trojúhelníku z polygonu pomocí metody CDT, pro ilustraci je uvedena jedna z jeho variant, pro kterou bylo čerpáno z [4] a [11]:

1. Proběhne iterace přes všechny omezující hrany, které jsou definovány body V_i a V_j . Pro každou takovou tu hranu budou provedeny kroky 2-4.
2. Pokud je omezující hrana $V_i - V_j$ již přítomná, pokračuje se další hranou. Jinak se ve vytvořené triangulaci vyhledají všechny hrany, které prochází touto hranou, a nad nimi se provádí krok 3.
3. Pro všechny hrany procházející hranou $V_i - V_j$ provádime následující kroky:
 - 3.1. Hrana je odebrána ze seznamu hran, které procházejí $V_i - V_j$. Tato hrana je definovaná body V_k a V_l .
 - 3.2. Tato hrana definuje ve vytvořené triangulaci dva trojúhelníky. Pokud tyto dva trojúhelníky netvoří čtyřúhelník, který je konvexní, je hrana umístěna zpět do

¹⁹Obrázek byl převzat z: http://www.geom.uiuc.edu/~samuel/del_project.html

seznamu hran, a algoritmus se vrací k bodu 3.1. Jinak proběhne prohození diagonály tohoto čtyřúhelníku. Prohození diagonály je znázorněno na obrázku 2.10. Tato nová diagonála je definována body V_m a V_n . Pokud $V_k - V_l$ stále protíná hranu $V_i - V_j$, pak je hrana umístěna do seznamu protínajících hran. Pokud hranu neprotíná, je vložena do seznamu vytvořených hran.



Obrázek 2.10: Otočení diagonální hrany $V_1 - V_2$ za $V_j - V_3$. Na obrázku body V_i a V_j značí omezující hranu.²¹

4. Dokud dochází k prohození hran v kroku 4.3, provádí se následující kroky:
 - 4.1. Probíhá iterace přes všechny hrany ze seznamu nově vytovřených hran.
 - 4.2. Nově vytovřená hrana je definována pomocí bodů $V_k - V_l$. Pokud se nová hrana rovná hraně $V_i - V_j$, vrátíme se ke kroku 4.1
 - 4.3. Pokud jsou dva trojúhelníky tvořené hranou $V_k - V_l$ umístěny tak, že jeden z trojúhelníků je uvnitř opsaného kruhu druhého trojúhelníku, potom tyto trojúhelníky tvoří čtyřúhelník s diagonálou, která stále protíná hranu $V_i - V_j$. V tom případě je hrana $V_k - V_l$ nahrazena druhou diagonálou tohoto čtyřúhelníku se jménem $V_m - V_n$. Tímto jsou nahrazeny tyto dva trojúhelníky za dva nové trojúhelníky, a hrana $V_k - V_l$ je nahrazena za $V_m - V_n$ v seznamu nových hran.
5. Odeberou se všechny trojúhelníky, které leží mimo supertriangle, což je takový trojúhelník, který opisuje všechny původní body před triangulací.

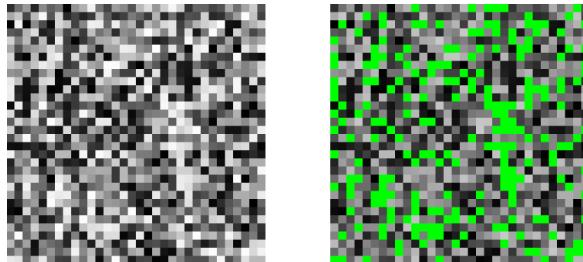
2.6 Procedurální generování vegetace

Aby bylo dosaženo více realistického zobrazení scény, je třeba vygenerovat vegetaci - stromy v lese, trávu na loukách a zahradách, obilí na polích. Pozice, kam může být vegetace rozšířena, se dá spočítat několika způsoby, z nichž je několik popsáno v následujících kapitolách. Poznatky použity v následujících sekci byly získány z [7] a z [2].

Náhodné rozložení

Při náhodném rozložení je vytvořena textura, do které se pomocí libovolné pseudonáhodné funkce od 0 do 1 zapíší hodnoty. Tato textura se poté celá projde, a na každé místo, kde je hodnota textury vyšší než libovolná hodnota x , je umístěn nějaký typ vegetace. Tato metoda je výpočetně velmi jednoduchá, ale je s ní spojena řada problémů. Vegetace nevypadá realisticky (v reálném světě není vegetace umístěna plně náhodně), a může se stát to, že se kusy vegetace budou protínat, protože jsou moc blízko u sebe.

²¹Obrázek byl převzat a upraven z: [11]

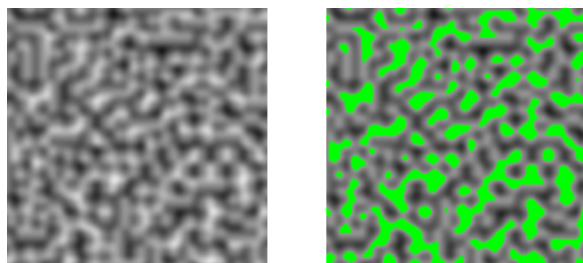


Obrázek 2.11: Ukázka rozdelení pomocí náhodného šumu. Vlevo je zobrazen šum samotný a vpravo pozice vegetace v šumu

Na obrázku 2.11 jde vidět jak vypadá takové rozdelení v nižším rozlišení 32x32. Světle zelená barva značí pozice stromů, kde hodnota x byla zvolena přibližně 0.78. Stromy jsou zde moc blízko u sebe a nevypadají přirozeně. To se dá řešit tak, že pokud by dva stromy byly moc blízko sebe, jeden se odstraní. To ale stále vede k nerealistickému zobrazení.

Rozložení pomocí Perlinova šumu

Perlinův šum je typ šumu, který se v 3D grafice běžně využívá na generování celé řady efektů jako plameny ohně, kouř, oblaka, nebo také realisticky vypadající procedurálně generované terény, či planety. Svou popularitu získal díky jednoduché a rychlé implementaci a vysoké realističnosti při procedurálném generování. Většinou je skládáno více vrstev šumu dohromady, aby se dosáhlo realističtějšího zobrazení. Pro generování vegetace je možné Perlinův šum využít podobně jako obyčejného šumu. Musí se zde také řešit překrývání vegetace. Generování probíhá takovým způsobem, že je vybrána hodnota z šumu, a jestli je vyšší než vybraná hodnota x , tak je zde umístěna vegetace. Použití Perlinova šumu vede k skupinám vegetace, které mají vysokou hustotu, a skupinám, které mají naopak hustotu nízkou. Tohoto je využíváno například ve hrách, kdy se tyto velké plochy stromů využijí pro oddělení herních oblastí, a hráč pak přes les neprojde.



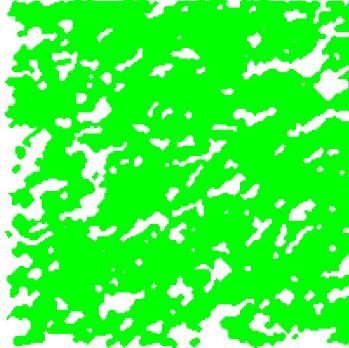
Obrázek 2.12: Ukázka rozdelení pomocí Perlinova šumu. Vlevo je zobrazen šum samotný a vpravo pozice vegetace v šumu

Na obrázku 2.12 je vidět, jak by se vygenerovala vegetace s použitím Perlinova šumu s hodnotou x nastavenou na 0.58.

Rozložení s užitím celulárních automatů

Další možností je využití celulárních automatů. Buňky automatu můžou mít stav 1, nebo 0, které označují, jestli na bodě je, nebo není vegetace. Musí být určen počáteční stav automatu, přičemž je možné jej nastavit náhodně. Automat běží několik iterací tak, aby

bylo docíleno co nejlepšího výsledku. Tímto způsobem vzniká vegetace, která má velmi realistické rozložení. Hlavní problém je, že vegetace je umístěna do mřížky podle toho, jak byl vytvořen automat. To se dá řešit tak, že se každý prvek vegetace posune, avšak výsledky pak vypadají méně opět méně realisticky, a stále není vyřešeno překrývání.



Obrázek 2.13: Ukázka rozdelení pomocí celulárních automatů.²²

Obrázek 2.13 je ukázkou jak může vypadat vegetace po rozdelení s užitím celulárních automatů.

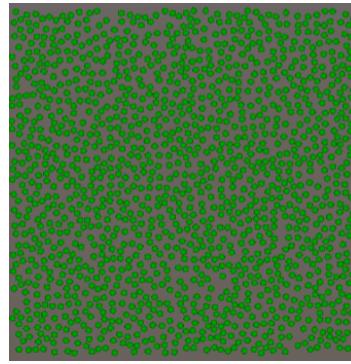
Poisson disk sampling

Další metodou je Poisson disk sampling. Ta povoluje generovat body v n dimenzionálním prostoru a zajíšťuje, že body budou vygenerovány s určitou vzdáleností od sebe, takže se výsledně generovaná vegetace nepřekrývá. Vstupem algoritmu je vzdálenost r mezi body, velikost generovaného prostoru a počet pokusů k . Hodnota k určuje maximální počet neúspěšných pokusů o vytvoření nového bodu. Jakmile algoritmus provede pro daný bod tento počet pokusů, přesune se na jinou pozici. Body jsou pak vygenenrovány pomocí následujícího algoritmu:

1. Vypočte se velikost buňky $c = \frac{r}{\sqrt{n}}$.
2. Vytvoří se n rozměrná mřížka s délkou hrany $\frac{max}{c}$, kde max značí maximální rozlišení mřížky s tím, že délka hrany je zaokrouhlená nahoru k nejbližšímu celému číslu.
3. Vytvoří se dva seznamy. Jeden na uložení vytvořených bodů a druhý na vytvoření aktivních bodů. Aktivní body jsou takové body, které mohou ještě mít další nové sousední body.
4. Vytvoří se první bod, který je umístěn do mřížky a do seznamu aktivních bodů.
5. Dokud není aktivní seznam prázdný, probíhá iterace přes body i následovně:
 - (a) Vytvoří se nový bod na pozici vypočítané pomocí náhodného směrového vektoru d , který lze získat z náhodného úhlu α . $\alpha = x * \pi * 2$, kde x značí náhodnou hodnotu. Pro dvouzměrnou mřížku je vektor d vypočítán tak, že jedna složka je rovna $\sin(\alpha)$ a druhá složka je rovna $\cos(\alpha)$. Nový bod má pak pozici definovanou jako $p = i + d * rand(r, r * 2)$, kde $rand(y, z)$ značí funkci generující rovnoměrné rozdelení v intervalu $\langle y; z \rangle$.

²²Obrázek převzat z: [7].

- (b) Proběhne kontrola, zda se body překrývají. Pokud ano, vrátí se algoritmus k předchozímu bodu, a je odebrán jeden pokus (z celkových k pokusů).
 - (c) Pokud byl bod vytvořen, je přidán do aktivního seznamu, a je generován další bod.
 - (d) Pokud po k pokusech nebyl žádný bod vytvořen, znamená to, že bod i nemůže mít další sousedy, a je tedy odebrán z aktivního seznamu.
6. Pokud již nejsou dostupné žádné nové body, generování je ukončeno.



Obrázek 2.14: Ukázka rozdělení pomocí metody Poisson disk sampling.²³

Na obrázku 2.14 je ukázáno, jak vypadá výsledné rozdělení vegetace na 2D ploše. Zelené body značí, kam tato metoda umístí vegetaci. Postup pro tuto metodu čerpá z [3]

2.7 Deferred shading

Klasickým způsobem renderování scény je forward shading. U této metody je pro každý objekt ve scéně vypočítáno osvětlení tak, že se iteruje přes každý zdroj světla, a objekt je postupně nasvětlen. To je výpočetně velmi náročné, protože komplexnější scény s více světly musí provést zbytečně vysoké množství iterací. Zároveň když se dva objekty vzájemně překrývají, může se stát, že je část výsledného obrazu několikrát přepracovávána znovu právě díky překrytí objektů. Toto se dá vyřešit pomocí techniky deferred shading. Tato technika je založena na tom, že se odloží zobrazení veškerého nasvícení scény na později. Deferred shading je rozdělen na dvě fáze.

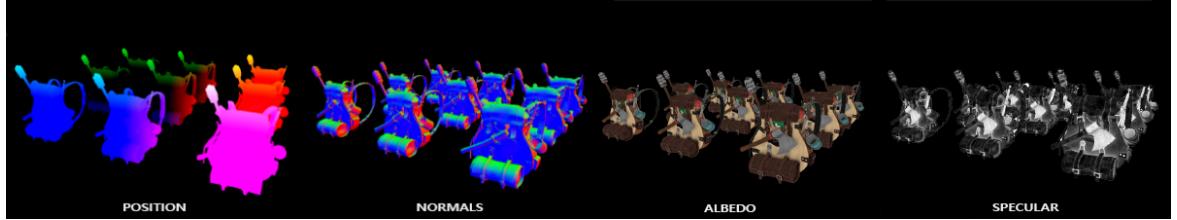
V první fázi se sesbírají různé informace o geometrii objektů ve scéně, a ty jsou uloženy do speciálního bufferu s názvem G-Buffer (geometry buffer). G-Buffer obsahuje několik textur se stejným rozlišením jako výsledný obraz. Do textur jsou uloženy informace o pozici, normálový vektor, barva²⁴ a lesklost²⁵ každého pixelu (fragmentu) scény. Lesklost se dá uložit do alpha složky barevné textury, a je tím možno ušetřit jednu texturu a snížit tím paměťovou náročnost metody, která je relativně vysoká. Pro přesnost uložení dat jako pozice a normálové vektory nestačí použít malé datové typy jako byte, ale je třeba větších, které zabírají více paměti. V případě rozlišení 4K (3840x2160) a využití všech textur se jedná o téměř 1.6 GB paměti, což je poměrně hodně i na moderní grafické karty. Když se využije

²³Obrázek převzat z: [7].

²⁴albedo

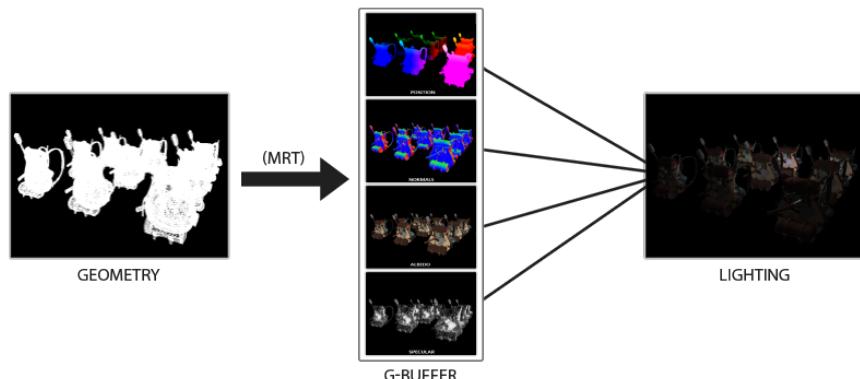
²⁵specular

deferred shading, je také velmi jednoduché implementovat celou řadu screen space post process efektů. Příkladem takového efektu je metoda SSAO, jejíchž dvě vstupní textury jsou přímým výstupem právě deferred shadingu viz 2.4. Jak takové textury mohou vypadat je ukázáno na obrázku 2.15.



Obrázek 2.15: Ukázka textur v G-Bufferu.²⁶

V druhé fázi deferred shadingu nastane výpočet nasvícení scény. Přes celou obrazovku se zobrazí pouze obdélník, na který se ve fragment shaderu vypočítá výsledná textura reprezentující finální scénu. Iteruje se přes každý pixel v texturách z G-bufferu, a je vypočítáno nasvícení světly ve scéně. Tím, že je nasvícení počítáno na texturách, proběhne výpočet pouze jednou pro každý pixel scény. V případě, že scéna obsahuje větší množství světel a překrývajících se objektů, je díky tomu ušetřeno relativně vysoké množství výkonu. Nasvícení na takovýchto texturách může být počítáno naprostě stejně jako u klasického forward shadingu, kde je počítáno pro každý objekt zvlášť. Informace v této sekci byly získány z [13].



Obrázek 2.16: Proces deferred shadingu.²⁷

Na obrázku 2.16 je vidět, jak probíhá deferred shading. Geometrie je rozdělena na jednotlivé textury, jsou provedeny výpočty, a výsledky jsou vloženy do výsledné scény včetně nasvícení.

²⁶Obrázek převzat z: <https://learnopengl.com/Advanced-Lighting/Deferred-Shading>

²⁷Obrázek převzat z: <https://learnopengl.com/Advanced-Lighting/Deferred-Shading>

Kapitola 3

Výběr technologií a návrh funkcionality aplikace

Tato kapitola popisuje návrh aplikace, který je vytvořen tak, aby aplikace byla co nejvíce přínosná koncovému uživateli. V případě této aplikace se jedná o obyčejného člověka, který se chystá na výlet do nějaké oblasti, a chce si tuto oblast vizualizovat v 3D prostoru. Aplikace by tedy měla být jednoduchá na použití i pro někoho, kdo není informatik.

3.1 Architektura aplikace

Jelikož má být aplikace využívána běžnými uživateli, tak je potřeba, aby byla multiplatformní, a byla přístupná jako desktopová aplikace. Vhodným jazykem pro tuto práci je jazyk C++, protože nabízí framework Qt pro uživatelské rozhraní. Hlavní logika aplikace by měla být samostatná knihovna, která má co nejméně dependencí, a ideálně je i přenosná mezi různými systémy. Samotná spustitelná aplikace je navržena jako pouze frontend volající funkce z knihovny, a zobrazení výstupu knihovny. Tímto je možné docílit toho, že aplikaci bude jednoduše možné v budoucnosti zkompilovat i do technologií jako je WebAssembly, a vytvořit z práce webovou aplikaci.

Vstupem této aplikace budou 2 koordináty na libovolné pozici na zemi ve formátu zeměpisné výšky a šířky. Dohromady tvoří rohy obdélníku okolo oblasti, která bude knihovnou vygenerována. Uživatel tyto koordináty zadá pomocí uživatelského rozhraní, na kterém se zobrazí klasická 2D mapa, a nějakou akcí (například potažením myši) vybere oblast. Poté se získají data pro tuto oblast a započne výpočet. Ten by měl běžet asynchroně a postupně vracet detailnější informace o oblasti, aby uživatel čekal co nejkratší dobu na minimální možné informace. Výstupem bude scéna obsahující jednotlivé modely, textury, shadery, další grafické objekty, a popisy, jak se má scéna zobrazit. Součástí výstupu, už ale nebudou postprocess efekty, jako je například SSAO, které nejsou potřebné pro funkčnost, a uživatel knihovny si je dokáže nadefinovat sám. Pro případ této práce bude na zobrazování dat využita knihovna GPUEngine¹, vyvíjena na VUT FIT, která staví nad grafickým rozhraním OpenGL.

¹<https://github.com/Rendering-FIT/GPUEngine>

3.2 Výběr zvolených technologií

Z výše zmíněných technologií v sekci 2.1 je třeba zvolit vhodnou technologii pro každou část implementace. Každá má své výhody a nevýhody, podle kterých je posouzena vhodnost použití v této práci.

API pro mapové podklady

Jako mapové podklady je možné použít API, které byly zmíněny v sekci 2.1. Porovnání jejich vlastností je uvedeno v tabulkách 3.2, 3.1 a 3.3.

- **OpenStreetMap** – Hlavní výhodou je, že data jsou zadávána uživateli. Z toho plyne, že data jsou velmi přesná (když uživatel vidí někde chybu, může ji sám opravit). Zároveň to ale znamená, že uživatel může zadat i data, které jsou nějakým způsobem nepřesná. Vlastnost, že je služba vedena uživateli, vede také k tomu, že s ní pracuje spousta lidí, a existuje tedy rozsáhlá dokumentace a řada uživatelských projektů, které na těchto mapách staví. Drobou nevýhodou je, že API vrací opravdu velmi detailní údaje. Pro přibližnou oblast města Brna API vrací přes 400MB dat, což na pomalejším internetovém připojení znamená, že uživatel na data musí dlouho čekat. Požadavky se dají zo optimalizovat tak, aby byly pro stejnou oblast přibližně 200MB dat, ale to je stále hodně. Pro představu na průměrném českém internetu v roce 2020, tj. na přibližně 74 Mbps², by se data stahovaly okolo 20 sekund. Za tu dobu uživatel pouze čeká na načtení aplikace.
- **Google Maps** – Google Maps jsou většinou mírně přesnější než zmíněné OpenStreetMap. Příkladem je například budova blízko místa mého bydliště, která ovšem reálně není budova, ale auto, které tam někdy parkovalo. Na Google mapách jsou data zadána správně. Nevýhodou je, že Google Maps API neobsahuje vhodný endpoint na získání metadat o oblasti. Jsou zde endpointy na získání statického obrázku, který obsahuje slepu mapu oblasti, a z té je možné (např. dle barvy) získat informace o tom, o jaký typ oblasti se jedná. Zde ale může docházet k nepřesnosti daným tím, že je aplikace limitována pouze na odhad. Google Maps API obsahuje i několik způsobů jak získat výškové data. Problém ale je, že to je buď získání informací o jednom bodu, nebo o čáře mezi dvěma body. V kontextu této práce to znamená, že by bylo nutné opravdu velmi vysoké množství požadavků aby byly data použitelná. Vhodným využitím API je zobrazování statických 2D map, nebo výškového profilu.
- **Bing Maps** – Bing Maps nejsou vhodné na získávání metadat o oblasti – nemají na to žádné endpointy. Na co jsou Bing Maps velmi dobré, je získávání výškových dat. Krom toho že data jsou velmi přesná, je i přístup k nim jednoduchý. Jedním požadavkem lze získat až 1024 výškových bodů v libovolné obdélníkové oblasti. Pro získání výškových dat z nějaké oblasti je tedy nutné zastlat pouze jeden požadavek obsahující několik ohraňujících koordinátů a API vrátí rovnoměrně rozložené výškové body pro tu oblast. V případě větší oblasti (několik km) ale není 1024 bodů zdaleka dostačující. Jedná se o rozlišení 32x32 bodů a to je velmi málo. Je možné zaslat více požadavků po sobě a následně data poskládat do jedné větší oblasti, ale to také není ideální. Hlavní problém s tímto přístupem je, že Bing Maps mají nastavený limit na požadavků v určitém čase. Když je tento limit překročen, je API nedostupné až přes 10 vteřin.

²<https://www.statista.com/statistics/1155589/internet-connection-speeds-czechia/>

Limit není v dokumentaci zmíněn, ale z testování vyplynulo, že je to přibližně 16 rychle po sobě poslaných požadavků, ale často i méně. Při 16 požadavcích se jedná o rozlišení 128x128. Bing Maps API je tedy vhodné limitovat na maximální rozlišení 96x96, které je vždy v pořádku. V případě využití tohoto API bude v práci nutné využít některou z metod popsaných v sekci 2.3.

Google Maps	
Výhody	Nevýhody
Přenost a kvalita dostupných dat	Cena za použití může být vysoká
Rychlosť	Dokumentace není ideální
	Nevhodné endpointy pro tuto práci

Tabulka 3.1: Porovnání výhod a nevýhod Google Maps

OpenStreetMap	
Výhody	Nevýhody
API je uživatelsky přívětivé	Data nejsou z daleka vždy perfektní
Data jsou zadávána uživateli	API vrací velmi vysoké množství dat
Spousta dostupné dokumentace	
Zcela zdarma	

Tabulka 3.2: Porovnání výhod a nevýhod OpenStreetMap

Bing Maps	
Výhody	Nevýhody
Kvalita výškových dat	Bez možnosti získat metadata
Výborná dokumentace	Velmi tvrdé limity na počet požadavků
Rychlosť	
Možnosti získávání výškových dat	

Tabulka 3.3: Porovnání výhod a nevýhod Bing Maps

Z výše popsaných výhod a nevýhod plyne, že použití Google Maps v práci nebude vhodné, a budou využity Bing Maps pro získání výškových dat a OpenStreetMap pro získání metadat. Obě dvě použité API vrací data ve formátu JSON, který bude do objektu serializován pomocí knihovny **JSON for Modern C++**³.

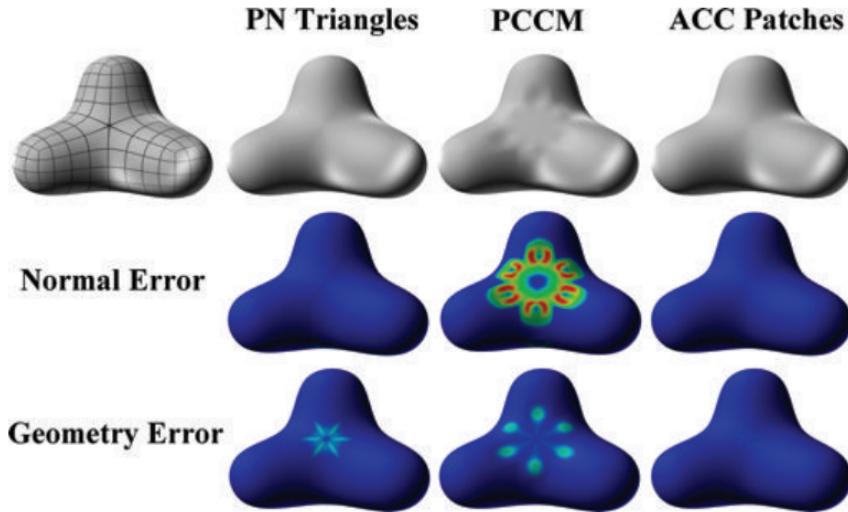
Zvýšení kvality modelů

Jelikož pro plochy, které mají v reálném světě několik kilometrů čtverečních, je využití výškových podkladů s rozlišením maximálně 96x96 bodů nevhodné, je třeba povrch zlepšit. Takto malé rozlišení může způsobovat nepřesnosti nebo dalších problémy, které mohou nastat například s nasvícením scény. To v práci bude implementováno pomocí jedné z metod popsaných v sekci 2.3. Metoda poběží na grafické kartě tak, aby ušetřila co nejvíce výpočetního výkonu na procesoru, kde by metoda zabrala podstatně více času. Porovnání metod pro zvýšení kvality modelů:

³<https://github.com/nlohmann/json>

- **Curved PN Triangles** – Výhoda této metody je, že je relativně jednoduchá na implementaci jako tessalační shader v OpenGL. Vytváří ovšem chybu (drobné artefakty) v geometrii a v normálách u některých tvarů objektů. Další výhodou je, že výpočty probíhají na úrovni trojúhelníků, což opět vyhovuje OpenGL, které pracuje při renderu také primárně s trojúhelníky.
- **Patching Catmull-Clark Meshes** – Hlavní nevýhodou tohoto algoritmu je, že pracuje se čtverci. To v kontextu práce není problém – povrch je možné generovat ať už jako čtverce nebo jako trojúhelníky. Problém to je ale při zobrazování povrchu na grafické kartě s využitím knihovny OpenGL, který pracuje primárně s trojúhelníky. Oproti metodě Curved PN Triangles je ale složitější na implementaci a v některých případech má vyšší chybovost. Algoritmus je možné implementovat na grafické kartě, ale není k tomu vhodný z důvodu nižší rychlosti.
- **ACC Patches** – Tato metoda má velmi vysokou přesnost – téměř nevytváří chyby jak v geometrii tak v normálách geometrie, ale oproti ostatním metodám má nejsložitější implementaci. Je velmi vhodná na implementaci na grafických kartách a dosahuje i velmi dobré rychlosti – obzvlášť oproti PCCM metodě, kterou tato metoda approximuje.

Na obrázku 3.1 je vidět porovnání chybovosti všech metod, oproti geometrii, která už byla ve vyšším rozlišení.



Obrázek 3.1: Porovnání metod zvyšování rozlišení geometrie. Každý z povrchníků byl rozdělen dvakrát, každý jinou metodou. U PCCM vzniká relativně vysoká chyba normál, narozdíl od ostatních metod.⁵

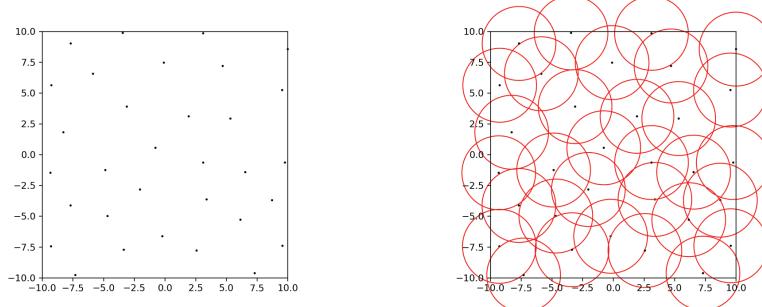
V práci bude použita metoda Curved PN Triangles, která dosahuje dostatečně dobrých výsledků a je jednochá na implementaci.

⁵Obrázek převzán z: [5]

Generování vegetace

Práce bude reprezentovat realistické zobrazení nějaké oblasti ve skutečném světě. Nereálně vypadající zobrazení náhodného rozdělení vegetace je tedy i přes to, že je velmi rychlé na implementaci, nevhodné použít. Rozložení pomocí perlinova šumu generuje reálněji vypadající vegetaci, ale na generování vegetace oproti jiným přístupům je méně realistický, a proto také nebude využit. Rozložení za pomocí celulárních automatů generuje relativně reálně vypadající vegetaci, ale jejich implementace je složitější a výsledky stále nejsou ideální. Zároveň se zde musí řešit řada problémů, jako například překrývání, a fakt, že vegetace je přesně umístěna do mřížky. Pro nejrealističtější zobrazení je proto vhodné použít algoritmus Poisson Disk Sampling 2.6, který i sám řeší překrývání vegetace a podobné problémy.

Ten bude implementován tak, aby příjemal několik různých typů vegetace (např. jehličnaté a listnaté stromy, tráva, obiloviny). Algoritmus bude moct míchat i více různých typů vegetace tak, aby mohly vzniknout i například smíšené lesy. O každém druhu vegetace bude předem známá informace o tom, jak daleko od sebe mají být jednotlivé kusy a jak jsou velké. Algoritmus pak zaplní všechny oblasti, kde se má vegetace objevit ve finální reprezentaci světa, a pokud budou nějaké vygenerovány mimo, budou odstraněny. Generování samotných vzorků bude zprostředkováno pomocí knihovny `poisson-disk-sampling`⁶, která je zabalená pouze do jednoho hlavičkového souboru, a bude s ní tedy jednoduché pracovat. Výsledek generování pomocí knihovny je na obrázku 3.2.



Obrázek 3.2: Ukázka jak knihovna zmíněna zde 3.2 generuje vzorky. Vlevo výsledné body, vpravo body včetně kruhů, zajišťující že se body nepřekrývají.⁸

Zobrazení vegetace bude probíhat pomocí instanced renderingu, který probíhá tak, že procesor dá instrukci grafické kartě na vykreslení pouze jednou (pro každý typ vegetace), ale předem bude vytvořen buffer, který obsahuje dodatečné informace, jako například offsety a rotace. Offsety říkají, o kolik se má vegetace posunout oproti základnímu modelu, rotace jsou určeny úhlu, který se má vegetace orotovat, aby bylo zobrazení realističtější a všechny kusy nevypadaly přesně stejně. Tímto se podstatně zvedne výkonnost renderování vegetace. Pro vysoké množství vegetace (přes 150 000 kusů) by bylo nemožné docílit rozumného výkonu klasickým způsobem, kde se každý kus vegetace vykresluje zvlášť, alespoň ne na aktuálním hardwaru.

⁶<https://github.com/thinks/poisson-disk-sampling>

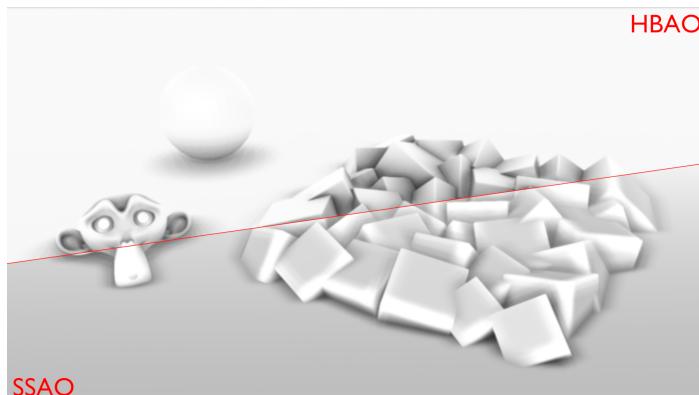
⁸Obrázek převzat z: <https://github.com/thinks/poisson-disk-sampling>

Generování budov

Pro generování budov bude v práci použit algoritmus CDT popsaný v sekci 2.5. V práci bude použita open-source knihovna CDT⁹. Výstup této knihovny bude využit na seskládání podlahy a jednotlivých střech budovy, přičemž budova může mít různý počet pater. Výška budovy bude určena podle počtu jejích pater. Počet pater se dá u některých budov získat z OpenStreetMap API. U budov, kde není zadán počet pater, bude počet pater nastavený na hodnotu 1. Výška patra bude nastavena na 4.2 metrů, což je přibližně průměrná výška patra, která činí 14 stop¹⁰. Stěny budovy budou sestrojeny tak, že se projdou původní body reprezentující podlahu a body s offsetem vysokým podle výšky budovy, a pro každou stěnu mezi těmito body se sestrojí dva trojúhelníky reprezentující tuto stěnu.

Výběr Ambient Occlusion metody

Jelikož Ambient Occlusion v reálném čase je na současném hardwaru nereálný, je na výběr buď metoda SSAO nebo HBAO. Jako metoda, která v aplikaci bude používána, byla zvolena metoda SSAO. Oproti HBAO nabízí méně reálně vypadající výsledky, ale je jednodušší na implementaci. Pro tuto práci budou i tak výsledky více než dostatečné. HBAO je také náročnější na výpočetní hardware – například ve hře Battlefield 3 se jedná o 28-35 snímků za sekundu, oproti 35-40, kterých dosahuje SSAO na stejném hardwaru¹¹. SSAO bude v práci implementováno jako volitelný efekt, který bude moct uživatel zapínat a vypínat. Obrázek 3.3 nabízí porovnání těchto dvou metod na stejné scéně.



Obrázek 3.3: Porovnání SSAO a HBAO. Metoda HBAO nabízí reálnější zobrazení více připomínající reálný svět. Obrázek zobrazuje pouze ambient occlusion.¹³

Jak bylo zmíněno v sekci 2.4, musí se výsledek SSAO funkce rozmažat, jinak je viditelný šum. To bude v práci vyřešeno zprůměrováním okolních hodnot. Tím se docílí sice mírně rozmazaného obrazu, a může se stát, že bude okluze v některých místech silnější nebo slabší než by měla být, ale efekt je to stále dostatečný.

⁹<https://github.com/artem-ogre/CDT>

¹⁰<https://theskydeck.com/how-tall-is-a-storey-in-feet/>

¹¹https://www.reddit.com/r/battlefield3/comments/kwnip/my_battlefield_3_screenshot_comparison_ssao_vs/

¹³Obrázek převzat z: <https://blenderartists.org/t/bge-hbao-ambient-occlusion-shader/690374>

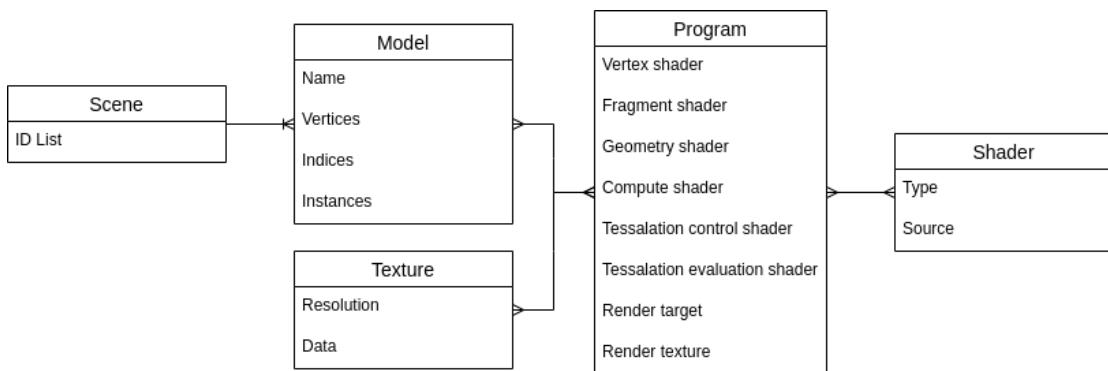
3.3 Návrh knihovny

Knihovna bude vytvořena tak, že jejím vstupem bude konfigurace a koordináty oblasti, která se bude generovat. Uživatelsky dostupné budou hlavně funkce na nastavení vlastní konfigurace, zapnutí generace (která poběží asynchroně) a zrušení generování, pokud ještě nedoběhlo. Knihovna bude zpočátku vracet vše v nízkém rozlišení, které bude s postupem času zvyšováno. To urychlí generování alespoň základního modelu. Zároveň bude s postupem času model stále více kvalitní a reprezentace světa bude přesnější. Generování vegetace proběhne právě jednou, a to až bude generovaná mapa o dostatečně vysokém rozlišení, aby bylo zobrazení relativně přesné.

Ke knihovně budou také dostupné soubory s texturami pro generovanou oblast a veškerými modely vegetace. Modely se kterými bude knihovna pracovat budou ve formátu OBJ¹⁴, který je jednoduše zpracovatelný a dostatečný pro tuto práci. Knihovna bude obsahovat logovací funkce, do které si uživatel knihovny bude moci vložit vlastní funkci jako výstup logu a bude moci takto postupně získávat informace o tom, co knihovna zrovna generuje. Po volání funkce na generování knihovna vrátí prázdný objekt reprezentující mapu, která bude postupně zaplněna daty podle toho co bude zrovna vygenerováno.

Výstup knihovny

Výstupem bude scéna obsahující veškeré modely, jejich textury, shadery, programy (sbírka zkompilovaných shaderů pro grafické karty) a další popis, jak scénu správně zobrazit (uniformní proměnné atd.). Každý model může využívat více programů, každý program musí obsahovat minimálně vertex a fragment shader, ale může nepovinně obsahovat i geometry tessellation control, a tessellation evaluation shader. Každý takovýto shader bude moci být využit u více různých programů. Dále bude u programu nastavitelné, jestli se má zobrazit do nějaké konkrétní textury, nebo do libovolného bufferu, který je zrovna aktivní. Krom toho budou na program vázány textury, které budou stejně jako shadery využitelné pro více programů zároveň. Na obrázku 3.4 je dostupný diagram této scény.



Obrázek 3.4: Diagram popisu jak vypadá scéna, která je výstupem knihovny. Zakončení čar symbolizuje vztah mezi prvky. Jednoduché značí 1:n, větvené m:n

Takto vytvořená scéna nemusí nutně obsahovat pouze mapu, ale teoreticky cokoliv jiného.

¹⁴[Wavefront OBJ \(Wikipedia\)](#)

Konfigurace knihovny

Knihovna by měla být co nejvíce konfigurovatelná tak, aby uživatel mohl nastavit co nejvíce parametrů generování. Bude se jednat o parametry:

- maximální a minimální rozlišení textury, určující typ oblasti
- jak velký je krok mezi jednotlivými generovanými rozlišeními
- rozlišení výškové mapy oblasti
- minimální rozlišení potřebné k vygenerování vegetace
- počáteční hodnota generátoru náhodných čísel použitého při generování vegetace

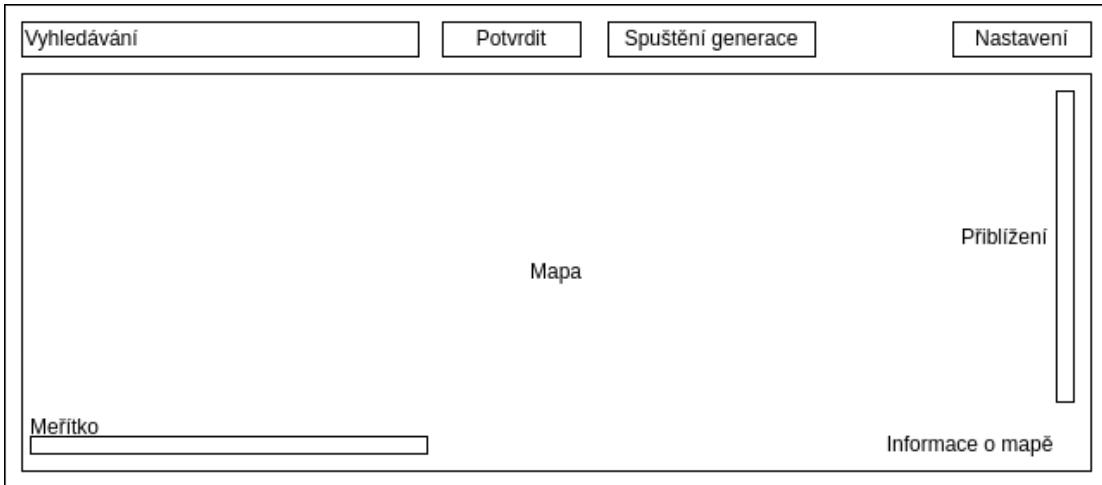
Krom toho bude muset uživatel dodat API klíč pro Bing Maps, které nedovolují přístup bez něj. Konfigurace bude předána jako JSON soubor, který se předá jako argument při spouštění aplikace, a bude obsahovat všechna tato nastavení.

3.4 Návrh uživatelské aplikace

Uživatelská aplikace bude rozdělena na dvě hlavní části. V první z nich bude na výběr klasická 2D mapa, na které si uživatel bude moct vybrat jakou oblast chce zobrazit, a další prvky pro ovládání a potvrzení výběru. Druhá část bude obsahovat prvky, které budou nastavovat chování zobrazovaného modelu, model samotný a tlačítko zpět, kterým se uživatel dostane zpět k části s výběrem oblasti.

Rozhraní mapy

Mapa bude zabírat většinu prostoru okna. Kromě mapy zde bude dostupná lišta pro uživatelský vstup, do které uživatel zadá vyhledané místo, a tlačítko na potvrzení vyhledávaného dotazu. Dále bude k dispozici potvrzení vybrané oblasti a zahájení generování oblasti. Pro zobrazení 2D mapy bude využito knihovny **Marble**, která je součástí aplikace pod stejným názvem. **Marble** je open-source přenositelná aplikace, přestože je součástí KDE, což je desktopové prostředí pro operační systém Linux. Výběr oblasti proběhne pomocí podržení pravého tlačítka myši a potáhnutí po části mapy. Když uživatel tlačítko pustí, zobrazí se, která oblast byla vybrána. Dále na mapě budou dostupné ovládací a informační prvky jako měřítko a přiblžení či oddálení mapy. Návrh tohoto rozhraní je dostupný na obrázku 3.5.

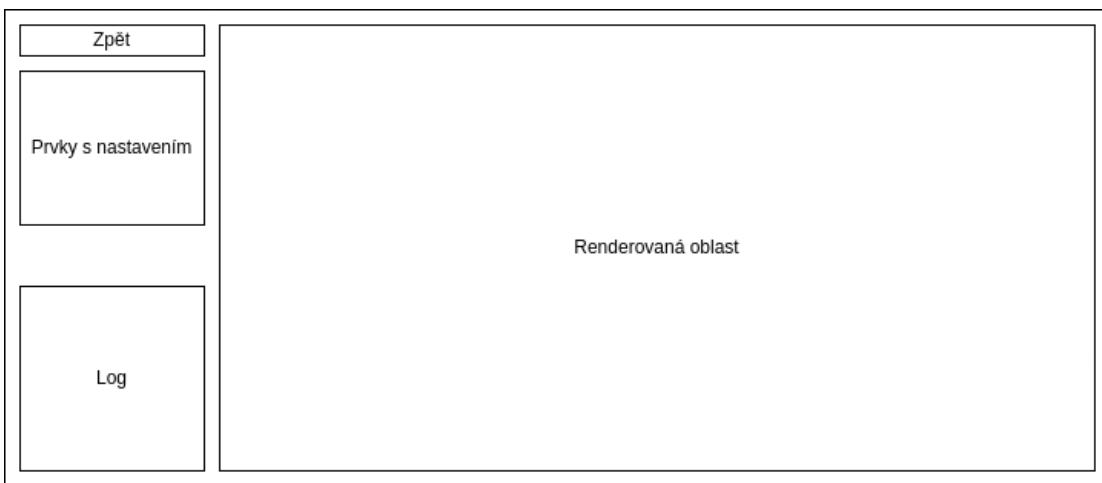


Obrázek 3.5: Návrh uživatelského rozhraní pro vybírání oblasti ke generaci

Většina ovládacích prvků mapy bude možné schovat a zobrazit po kliknutí pravým tlačítkem myši bez potáhnutí tak, aby nedošlo k výběru oblasti.

Rozhraní pro generovanou oblast

Generována oblast bude mít své vlastní rozhraní, obsahující výřez s OpenGL kontextem (zvaná renderer) a ovládací prvky pro tento kontext. Bude zde možnost zapnout a vypnout prvky, o které se stará renderer, a ne knihovna samotná. Součástí bude i tlačítko zpět, aby se uživatel mohl vrátit do výběru oblasti a nebyl nucen aplikaci restartovat pokaždé když chce vybrat jinou oblast. Zároveň zde bude dostupná plocha pro výstup z logovací funkce knihovny, protože zde může uživatel čekat delší dobu, než se například stáhnou metadata pro oblast.



Obrázek 3.6: Návrh uživatelského rozhraní pro zobrazení generované oblasti

Na obrázku 3.6 je návrh toho, jak tato oblast bude vypadat.

Zobrazení generované oblasti

V této oblasti zvané renderer se bude zobrazovat výstup z knihovny. Vzhledem k tomu, že výstupem jsou pouze objekty, a ne další post-process efekty jako AO, je nutné tyto efekty přidat později. Renderování výstupu bude provedeno metodou deferred shading, která dovoluje jednoduché přidávání post process efektů. Zároveň se bude starat o další funkcionality typu odebírání počtu vegetace, což je vhodné pro zvýšení výkonosti aplikace (větší oblast obsahující vysoké množství vegetace může mít problémy), nebo také pro zlepšení viditelnosti například lesních cest, které mohou být jinak hůře viditelné přes vegetaci. Mimo jiné bude možné nastavit sílu tohoto odebírání tak, aby uživatel dosáhl požadovaného výsledku. Renderer bude schopný zobrazit libovolnou scénu, která obsahuje libovolný počet modelů, shaderů a dalších grafických objektů. Renderované scéně bude poskytovat informace jako uplynulý čas, časový rozdíl mezi jednotlivými snímky, pozici kamery, a další. Obnovovací frekvence bude limitována podle obrazovky na uživatelském zařízení. To zde bude z důvodu, aby aplikace zbytečně nezatěžovala výpočetní zdroje, které nejsou nutně potřebné. Zobrazení oblasti bude nastavené tak, aby směr, kterým je sever, byl po vytvoření aplikace vždy směrem vpřed, východ vpravo atd. Toto zobrazení bude využívat metodu deferred shading pro jednodušší implementaci post-process efektů.

Kapitola 4

Implementace aplikace

V této kapitole je popsána praktická část této práce. Práce je rozdělená na 3 hlavní části – generování mapy, zobrazení výsledku a uživatelské rozhraní. Generování mapy je umístěno do svého vlastního projektu, který nemá žádnou závislost na zbytku.

4.1 Generování mapy

Generování mapy je rozděleno do několika fází. V první z nich jsou získány informace o oblasti, a pak je postupně generováno detailnější zobrazení těchto dat. Generování samotné je rozděleno tak, že generování jednotlivých typů objektů (povrchy, budovy, vegetace atd.) běží postupně každé zvlášť.

Generování povrchu

V první fázi jsou stažena data z Bing Maps API. Aplikace pošle 1 – 9 požadavků na API dle nastavení, a poté je poskládá do mřížky, ze které jsou vypočteny trojúhelníky a jejich normálové vektory. Takto poskládaný povrch je umístěn mezi koordináty 0 – 1 jak pro x, tak pro z. Výšková data jsou rozděleny tak, aby prostřední bod výšky byl umístěn na pozici 0.5. Jelikož data získána z API jsou v metrech nad mořem, musí se provést normalizace tak, aby data odpovídala rozměrům modelu, a dala se zobrazit. To je provedeno tak, že k normalizaci, která by standartně vygenerovala nejvyšší bod na pozici 1 a nejnižší na pozici 0, jsou před počítáním minimální a maximální výšky přidána ještě data o délce a šířce oblasti samotné. Jelikož mají známý rozdíl od 0 do 1, odpovídají takto normalizované hodnoty realitě.

$$realHeight = \frac{currentHeight - minHeight}{maxHeight - minHeight}$$

Jelikož data jsou v této fázi v relativně nízkém rozlišení, maximálně 96x96 výškových bodů, je při jejich zobrazení provedena tessalace¹. Implementace tessalace je v práci provedena jako metoda PCCM, která běží jako tessalační shader na grafické kartě. Implementace byla inspirována článkem z [6].

Protože je výšková mapa interpolována až při renderování, může se stát, že změna tvaru je dostatečná na to, že některé prvky vykreslované později mohou být vytvořeny

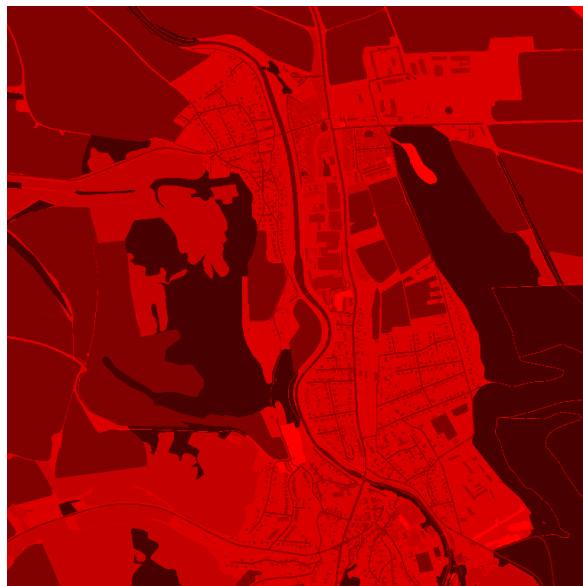
¹Proces rozdělení jednoho polygonu na více menších.

mimo povrch. To je řešeno tak, že veškerý povrch je vyrenderován dvakrát. První render proběhne z pohledu tak, aby uživatel viděl mapu. Druhý probíhá ze shora nad povrchem, a zobrazí do textury pouze výškovou složku povrchu. Výsledná textura z tohoto renderu je následně předána dalším objektům, které jsou renderovány na povrch, a jejich pozice je založena právě na hodnotách získaných z této textury.

Získávání a generování metadat

Poté, co je vygenerován povrch, jsou stažena metadata z OpenStreetMap. Jakožto OpenStreetMap samotné nejsou vhodné na čisté získávání informací o oblasti, je v projektu využito Overpass API², které je optimalizované na čtení dat. Pro libovolnou oblast se pošle požadavek, který získá veškerou geometrii pro každý objekt v oblasti. Geometrie je definována jako seznam bodů, které jsou definovány svou zeměpisnou výškou a šířkou. Dále jsou získány informace o každém takovém objektu. Mezi tyto informace patří hlavně typ objektu a další metadata (u lesu například typ, u budovy počet pater apod.). API tyto informace vrací ve formátu JSON, a pro konverzi do C++ objektu je využito knihovny JSON for Modern C++³.

Tyto informace jsou následně zpracovány do textury, která dle barvy každého pixelu určuje typ oblasti, která je zde obsažena. Textura obsahuje 4 hodnoty typu float, a je tedy teoreticky omezena na 2^{128} oblastí, což je podstatně více než API poskytuje. Jednotlivých typů oblastí, které aplikace rozpoznává, je 42, a jsou navíc rozděleny do 12 kategorií. Jedná se o lesy, louky, zahrady, vodní plochy, farmy, cesty (asfaltované, chodníky, lesní cesty), písčité oblasti a okraje měst. Lesy jsou dále rozděleny za listnaté, jehličnaté a smíšené. Podle toho se v aplikaci chová generování vegetace.



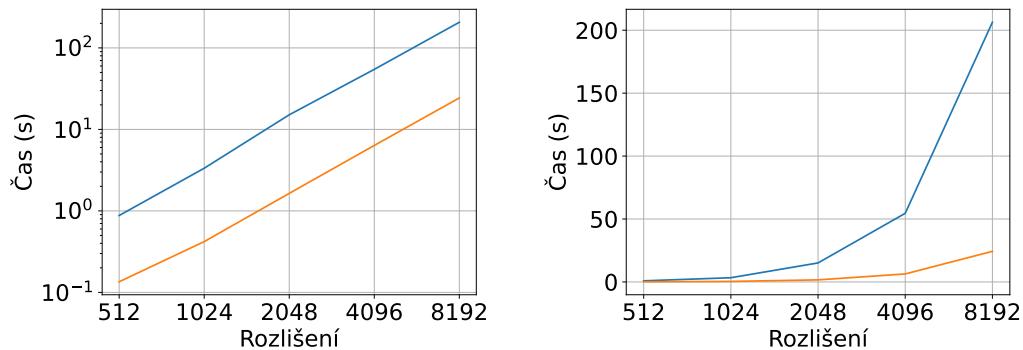
Obrázek 4.1: Ukázka vygenerovaných metadat o typu oblasti. Obrázek je zesvětlen pro lepší viditelnost.

²https://wiki.openstreetmap.org/wiki/Overpass_API

³<https://github.com/nlohmann/json>

Na obrázku 4.1 je vidět ukázka metadat o takto vygenerované oblasti. Všechny informace jsou zde uloženy pouze v červeném kanále, který je dostatečný na uložení všech typů oblastí, které práce rozpoznává.

Textura je vytvořena tak, že pro každý pixel je získána oblast, a její ID je uloženo do textury. Každý pixel této textury může obsahovat více oblastí, a je proto nutné zvolit správnou oblast. To probíhá primárně podle priority (např. cesty mají vyšší prioritu než vodní plochy, aby se nestalo, že řeka je nad mostem), a pokud má více oblastí stejnou prioritu, vybere se ta, která zabírá nejmenší plochu. Takto je vybrán nejvhodnější typ oblasti pro každý pixel. Na mapě obsahující velké množství oblastí (např. vysoký počet domů) tento přístup může trvat dlouhou dobu, a v aplikaci je proto výpočet rozdělen na počet dostupných procesorových jader kromě dvou, které zůstávají volné na další výpočty.



Obrázek 4.2: Ukázka zrychlení na 16 vláknovém procesoru AMD Ryzen 7 2700. Modré části grafů znázorňují jedno vlákno, oranžové všechny. Vlevo je graf s logaritmickým měřítkem, vpravo s lineárním.

Na paralelizaci výpočtu byla použita knihovna OpenMP⁴. Díky ní bylo dosáhnuto téměř lineárního škálování, které je zobrazené na obrázku 4.2.

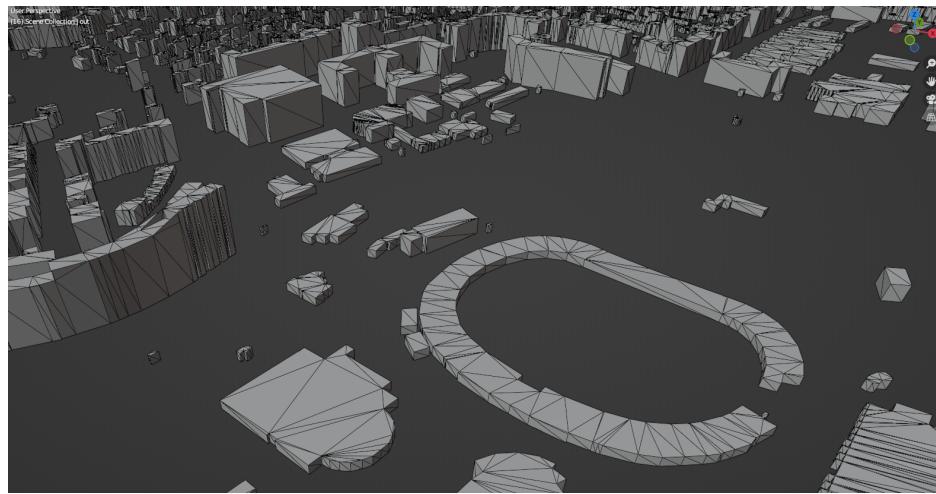
Před generováním mapy je třeba data získaná z API připravit na výpočet. Příprava probíhá tak, že každé oblasti je přiřazeno její ID. Oblasti, které aplikace neumí rozpoznat, jsou zahozeny, a oblasti rozdělené na několik polygonů jsou pospojovány do jednoho velkého, což zjednoduší výpočet. Takto vytvořená textura je pak přiřazena vygenerované ploše, která při následném vykreslování přiřazuje textury a barvy podle ID oblasti v textuře. Tyto textury byly získány z <https://www.textures.com/> a <https://polyhaven.com/>. Pokud existuje takový pixel, který nemá známou oblast, je mu přiřazena šedá barva, která působí neutrálně.

Generování budov

Generování budov probíhá tak, že se ze stávajících metadat upravených ve fázi 4.1 vyfiltrují pouze budovy, a na nich se provádí výpočet. Pro každou budovu se nejdříve zkontroluje, jestli nějaké její rohy nejsou mimo generovanou oblast. V případě, že by byly, dojde k tomu, že textura s informací o výšce popsaná v sekci 4.1 nebude obsahovat tyto rohy, a budova nebude mít informaci o tom, jak má být vysoko. Rohy budovy jsou posunuty tak, aby se blížily k okraji generované plochy. Aby byl zachován co nejpřesnější tvar budovy, je zde

⁴<https://www.openmp.org/>

vypočítána rovnice přímky mezi posledním bodem, který je stále uvnitř, a prvním bodem mimo. Nový bod je pak umístěn na tuto přímku co nejblíže okraji plochy. Pokud je mimo generovanou oblast více bodů, jsou odebrány všechny, a místo nich jsou vytvořeny pouze dva nové body na hranici plochy. Po této úpravě tvaru budovy je kombinace původních a nově vytvořených bodů zaslána jako vstup knihovně CDT⁵, která provede triangulaci, a výsledné trojúhelníky jsou použity na sestrojení podlahy a stropu budovy. Stěny jsou vytvořeny postupným propojením bodů střechy a stropu. Všechny budovy jsou postupně vkládány do jednoho modelu. I velmi vysoký počet budov na mapě má nízký vliv na výkon, a není potřeba je tedy odebírat, aby se snížil potřebný výkon. Celé město Brno obsahuje 4,5 milionu trojúhelníků v budovách, což moderní grafické karty zvládají bez problémů. Protože budovy není potřeba odebírat, jsou všechny poskládány do jednoho modelu pro jednoduchost zobrazování, nastavování textur a podobných vlastností.



Obrázek 4.3: Ukázka vygenerovaných budov po exportu z aplikace a importu do aplikace Blender. Na obrázku je možné vidět wireframe budov.

Generování budov proběhne hned po vygenerování základní mapy s nejnižším rozlišením. Triangulace i pro velké oblasti jako je město Brno proběhne pro všechny budovy pod sekundu a není tedy nutné ji nijak zvlášť paralelizovat. Na obrázku 4.3 je ukázka generovaných budov.

Generování vegetace

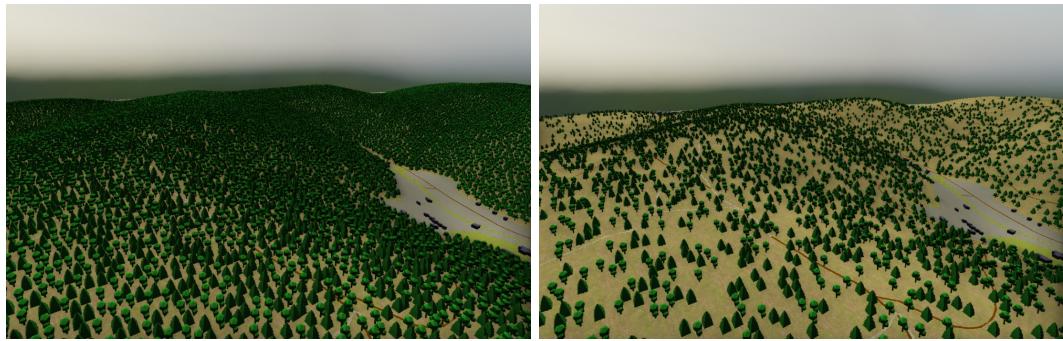
Generování vegetace proběhne až poté, co je rozlišení textury určující typ oblasti v dostatečně vysokém rozlišení. Výchozím nastavením je, že generování proběhne, až když je rozlišení textury alespoň 1024 pixelů. To je v aplikaci z toho důvodu, aby generování vegetace bylo co nejpřesnější, protože na rozdíl mezi oblastmi využívá právě tuto texturu.

Generování samotné pak probíhá tak, že prvně jsou při vytvoření třídy načteny všechny modely, které budou při generování použity. Na jednotlivý typ vegetace je možné použít několik různých modelů uložených do `.obj`⁶ souborů. Ty stačí vložit do správné podsložky ve složce `lib/assets/models` a program je automaticky načte a náhodně promícha s ostatními stejněho typu. Po vytvoření generátoru je možné začít generovat vegetaci. Na každý

⁵<https://github.com/artem-ogre/CDT>

⁶https://en.wikipedia.org/wiki/Wavefront_.obj_file

typ vegetace je nutné zavolat funkci, která generuje vegetaci s jiným nastavením, a to proto, že například stromy mají jinou hustotu umístění a výšku oproti například obilí. Podporované typy vegetace jsou jehličnaté, listnaté a smíšené lesy a pole. Generování probíhá tak, že je vypočtena hustota a výška vegetace, a poté jsou zavolány funkce z knihovny *poisson-disk-sampling*⁷. Z této knihovny je možné získat seznam bodů, které značí umístění jednotlivých vygenerovaných pozic. Pro každý takový bod je zkontovalo, jestli je umístěn uvnitř nebo mimo oblast, pro kterou se generuje vegetace. Tato kontrola se provádí oproti textuře s metadaty. Body, které jsou mimo, jsou zahrozeny. Poté je pro každý bod určený náhodný druh vegetace – například v případě smíšeného lesa je určeno, zda se jedná o listnatý nebo jehličnatý strom, a k modelu stromu je přidána informace o pozici bodu. Tato pozice je generována od 0 do 1 a s její hodnotou se počítá jako s offsetem vegetace oproti její výchozí pozici.



Obrázek 4.4: Ukázka vygenerovaného smíšeného lesa. Vlevo jsou vidět všechny stromy, vpravo je les po odebrání vegetace, nastaveném na maximální hodnotu.

Modely jsou vytvořeny tak, aby obsahovaly co nejméně trojúhelníků (celý jehličnatý strom má pouze 90), ale to samotné není dostatečné na to, aby byl výkon přijatelný. V případě větších oblastí může vygenerovaná vegetace zabírat velké množství výkonu, a z toho důvodu je při zobrazování odebírána vzdálená vegetace. Pro celé město Brno je generováno téměř půl milionu jednotlivých kusů vegetace. Po přepočtu na polygony je to přibližně 1,5 miliardy trojúhelníků. Ani moderní grafický hardware na toto nemá dostatečný výkon – okolo 10 snímků za sekundu na grafické kartě *Nvidia GTX 1080*. Odebereme-li přebytečnou vegetaci, počet snímků za sekundu se zvýší na přibližně 50 pro stejnou scénu.

Další výhodou toho, že jsou stromy odebírány, je to, že po odebrání je možné vidět nové lesní cesty, které jinak nemusí být viditelné. Ukázka vygenerované vegetace je na obrázku 4.4.

Skládání výsledků

Každý z vygenerovaných objektů je vrácen do třídy, která se stará o skládání jednotlivých objektů. Ta objekty vygeneruje a vloží do scény (mapy). Tato třída se zároveň stará o veškeré načítání všech složení programů, nastavení, které textury patří ke kterým modelům, shaderům (které jsou při komplikaci zabaleny do binárního souboru samotného), atd. Zároveň se stará o správné předávání dat mezi jednotlivými generátory, a výslednou scénu aktualizuje pokaždé, když je to potřeba.

⁷<https://github.com/thinks/poisson-disk-sampling>

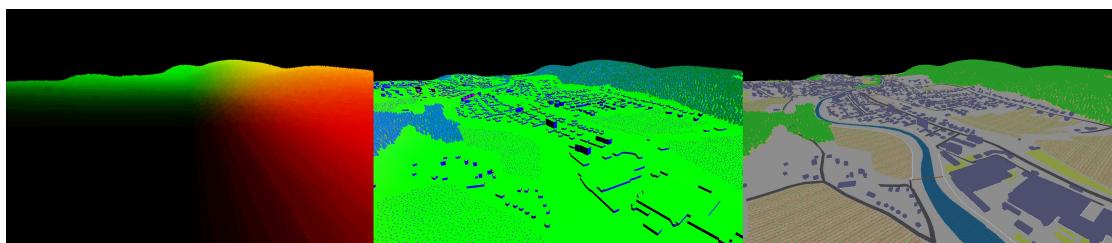
4.2 Zobrazení výsledku

O zobrazování výsledků se starají dvě hlavní, a několik pomocných tříd. Hlavní třídy jsou **Renderer** a **Scene3D**. Mezi pomocné třídy patří implementace SSAO, skyboxu a kamery, které jsou umístěny do tříd se stejným názvem jako je jejich funkčnost.

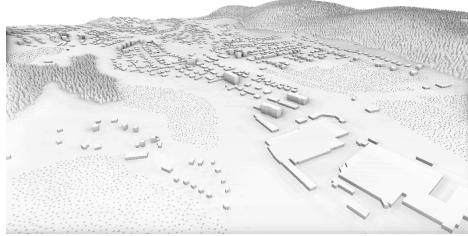
Renderer

Třída **renderer** se stará o zobrazování scény a ovládání pomocných tříd jako SSAO, které přidávají do scény další efekty. Dědí ze třídy **QOpenGLWidget**, která je součástí knihovny Qt a obsahuje většinu věcí potřebných k práci s OpenGL. Obsahuje zejména funkce pro implementaci virtuálních metod, a funkci, která se stará o inicializaci OpenGL kontextu, kamery, post-process efektů a veškerých součásti G-bufferu, který je potřeba pro správné zobrazení scény. Když je vše správně inicializováno, je možné aby obsah mohl být zobrazován. Funkce na zobrazení scény se volá periodicky pokaždé, když je třeba nového zobrazení snímku. V každém takovém volání proběhne nastavení správných bufferů (Qt si vytváří vlastní), vyčištění obrazovky, a následně vykreslení obrazu. Vykreslení probíhá v několika fázích:

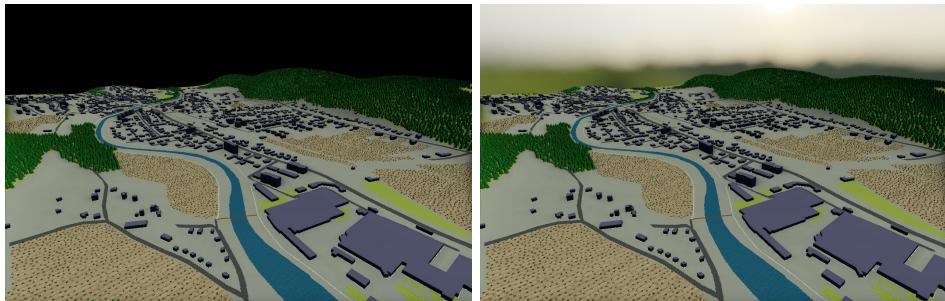
1. Geometry pass – vykreslí veškerou geometrii, její normály, složku s lesklostí a barvu do textur obsažených v G-bufferu. Výsledek průchodu je zobrazen na obrázku 4.5.
2. SSAO pass – vykreslí podle geometrie vrstvu ambient occlusion do vlastní textury. Renderuje se zde pouze jeden obdélník přes celou obrazovku.
3. Blur pass – rozostří původní výsledek SSAO tak, aby nebyl vidět šum. Výsledek tohoto a předchozího průchodu je na obrázku 4.6.
4. Lightning pass – vypočte se veškeré nasvícení scény z informací získaných v předešlých průchodech, a jako výstup použije výsledný buffer.
5. Skybox pass – na pixely, na kterých nebylo nic umístěno v předešlých průchodech, je umístěna obloha. Výsledná scéna po a před tímto průchodem je na obrázku 4.7



Obrázek 4.5: Výstup průchodu, který vypočte geometrii. Vlevo je pozice jednotlivých pixelů, ale zesvětlena pro lepší viditelnost. Uprostřed jsou normálové vektory, vpravo je barevný výstup. Alpha složka barevného výstupu obsahuje informaci o lesklosti.



Obrázek 4.6: Výstup SSAO průchodu scénou. Zde je zobrazena verze bez rozmazání.



Obrázek 4.7: Spočítané nasvícení scény. Vlevo je scéna po zkombinování textur z obrázků 4.6 a 4.5. Vpravo byla na chybějící místa do scény přidána obloha.

Také je zde funkce, která se stará o vytvoření nových textur, do kterých se zobrazují výsledky pokaždé, když dojde ke změně velikosti okna. To je důležité z toho důvodu, že některé z fází zobrazování ukládají svůj výsledek do textur, jejichž rozlišení je závislé na velikosti okna. Je tedy nutné, aby měly správnou velikost.

Dále tato třída dává vstup kamere. Vstupem pro kameru je pouze informace, že je myš na určité pozici, nebo zda je zmáčknuté tlačítko na klávesnici a na myši. Kamera pak 60x za sekundu vyhodnotí tyto vlastnosti, a podle nich vypočte novou pozici.

Scene

O zobrazení veškeré geometrie na správné pozice se stará třída `Scene3D`. Jejím vstupem je přímý výstup mapy z knihovny. Jelikož propojení jednotlivých objektů ve scéně je vyřešeno pomocí toho, že každému z nich je přiřazeno nějaké unikátní ID, je jednoduše určeno, jestli se scéna od posledního průchodu geometrie nezměnila. Pokud je propojení jednotlivých ID na sebe jiné (např. model 8 již není propojen na texturu 6, ale 7), scény vytvoří nové objekty tak, aby bylo zobrazení vždy správné. `Scene3D` se stará zároveň o zobrazení do různých cílů (např. přímo do textury), nebo o odebírání vzdálených kusů vegetace. Odebírání vzdálených objektů není aplikováno jen na vegetaci, ale na cokoliv, co se vykresluje pomocí metody `instanced rendering`. Odebírání funguje tak, že každému objektu je přiřazena náhodná hodnota mezi 0 a 1, a následně je tato hodnota porovnána oproti vzdálenosti kamery. Pokud je vzdálenost nad toto náhodně vygenerované číslo, objekt je odebrán. Tím je docíleno postupného odebírání vegetace, které je úměrné vzdálenosti od kamery. K náhodné hodnotě je buď přičtena nebo odečtena hodnota, kterou je určena síla odebírání. Nevýhoda tohoto přístupu je ta, že se s každým dalším zobrazeným snímkem musí přepočítat vzdálenost

kamery od všech jednotlivých instancí. To v praxi trvalo až 6 milisekund, což významně snížilo počet snímků za sekundu, které bylo možné zobrazit. Pro docílení lepšího výkonu je tato činnost rozdělena mezi všechna jádra procesoru, čímž je potřebný čas snížen pod 1 ms na moderním hardwaru⁸.

Implementace SSAO

Efekt SSAO je v práci implementován do vlastní třídy `SSAO`. Ta inicializuje všechny potřebné textury a buffery pro generování tohoto efektu. Jako šum je použita náhodná textura s rozlišením 4x4 pixely s tím, že každý z nich obsahuje náhodnou barvu. Jako vstup s informacemi o scéně jsou použity textury, které jsou výstupem metody `deferred shading`. Třída na práci s efektem si pak sama zpracuje všechny buffery tak, aby nedocházelo ke konfliktům při zobrazení dalších efektů. Jako rozmaření výsledného obrazu, který obsahuje vysoké množství šumu, je použita funkce, která zprůměruje okolní pixely ve scéně. Tím je efekt sice mírně rozmařaný, ale stále dostatečný. Výsledek efektu je zobrazen na obrázku 4.6. Implementace byla vytvořena dle [13] a [6]. Pro použití výsledné textury efektu třída obsahuje také metodu, která přiřadí výsledek na uživateli zadanou texturovací jednotku.

Implementace skyboxu

Implementace skyboxu je vytvořena jako poslední fáze zobrazení scény a je ve třídě `Skybox`. Ta je podobně jako SSAO nezávislá na okolí, a všechny své textury a buffery nastavuje sama. Funkčnost je taková, že se kolem kamery vytvoří dostatečně velká a vzdálená kostka s tím, že kamera se udržuje uprostřed tak, aby uživatel nikdy nemohl vyletět mimo tuto kostku. Dostatečná velikost kostky je důležitá proto, aby se nemohlo stát, že uživatel vidí pouze část zobrazené mapy, protože zbytek je již mimo kostku. Pro sestavení výsledné skybox textury bylo využito 6 textur, které jsou poskládány na sebe, a každá z nich je využita na jednu stěnu kostky. v OpenGL tvoří pouze jednu texturu, jejíž typ je nastaven jako `GL_TEXTURE_CUBE_MAP`. OpenGL samotné se stará o mapování jednotlivých stěn kostky na jednotlivé textury. Jak vypadá scéna pouze se spočítaným nasvícením a po přidání skyboxu je vidět na obrázku 4.7.

4.3 Uživatelské rozhraní

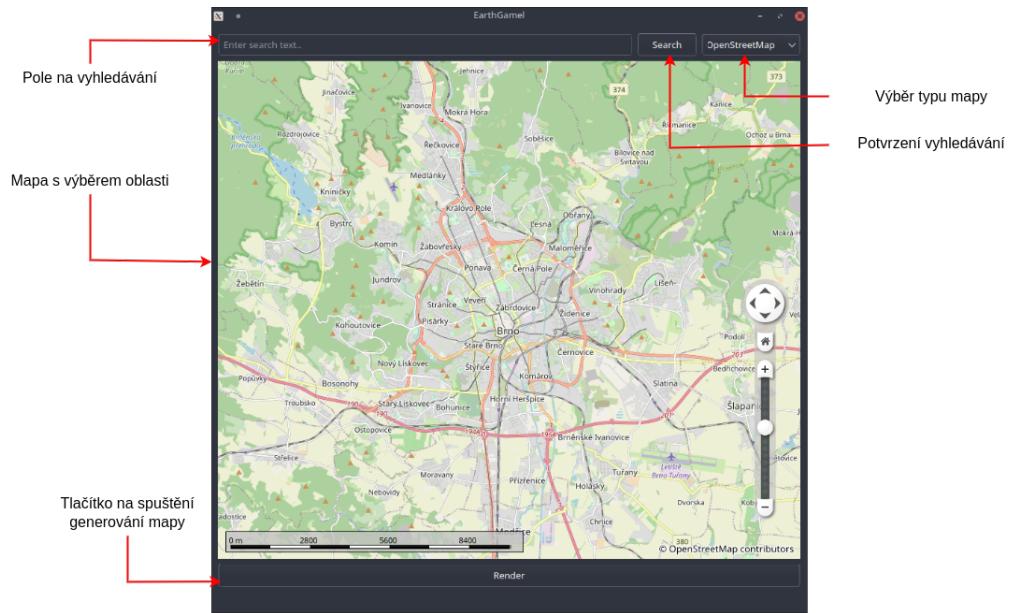
Uživatelské rozhraní je napsáno s využitím knihovny `Qt5`. Nepoužívá jazyk QML, ale `Qt Widgets`. Celé UI je rozděleno na 3 hlavní části. První částí je hlavní okno, které pouze obsahuje další objekty, a nemá v sobě žádnou významnou logiku (kromě nastavení, které další prvky se mají zobrazovat). Dále jsou v aplikaci prvky pro zobrazení 2D mapy a generované 3D mapy.

Rozhraní na zobrazení 2D mapy

Rozhraní funguje tak, aby bylo co nejvíce uživatelsky přívětivé. Stisknutím pravého tlačítka myši a jejím potažením si uživatel může vybrat libovolnou oblast na mapě k zobrazení. Pokud oblast přesahuje velikost 100km^2 , je uživateli zobrazeno varování, že generování a stahování oblasti může trvat dlouhou dobu. Dále je zde dostupná lišta na zadávaní vyhledá-

⁸AMD Ryzen 4500U, 15W

vané oblasti. Vyhledávaný dotaz se dá potvrdit buď pomocí zmáčknutí tlačítka enter, nebo potvrzením pomocí tlačítka v grafickém rozhraní.

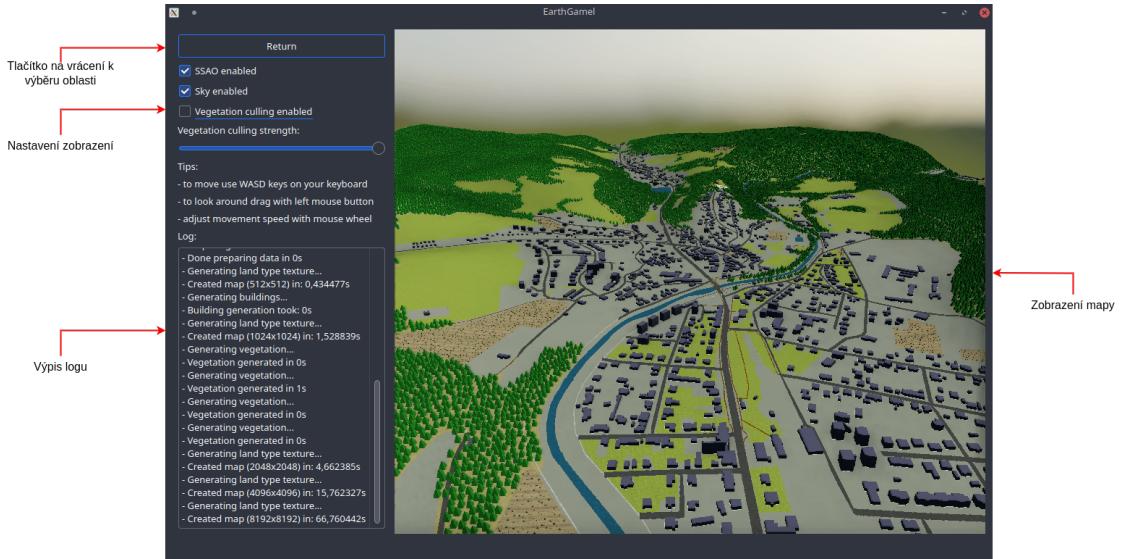


Obrázek 4.8: Uživatelské rozhraní pro výběr oblasti na mapě.

Dále jsou na návrhu, který je zobrazený na obrázku 4.8 prvky na výběr typu mapy (podporované jsou OpenStreetMap, satelitní se snímky od společnosti NASA a topografická mapa), a obsahuje i mapu samotnou. Na mapě jsou ovládací a informační prvky jako například měřítko. Ty jsou součásti knihovny **Marble**, pomocí které se mapa zobrazuje.

Rozhraní na zobrazení výsledku

Výsledek aplikace se zobrazuje do svého vlastního Qt Widgetu. Ten se stará o svůj OpenGL kontext a o veškeré ovládání a posílání událostí mezi prvky na zobrazování a Qt framework. Jsou v něm obsaženy třídy popsané v sekci 4.2. Mezi ovládací prvky patří ovládání zobrazení (zapnutí a vypnutí jednotlivých efektů), log s výpisem toho, co zrovna knihovna dělá, a tlačítko zpět. Ovládání je zde vytvořeno tak, že po tisknutí po generovaném okně se posune pohled směrem, kterým uživatel potáhl. Po mapě se dá pohybovat stisknutím tlačítka W, A, S, D na klávesnici. Alternativně se lze pohybovat i pomocí šipek. Stisknutím tlačítka Shift a Ctrl se nastavuje rychlosť pohybu. To je vhodné použít v případě, že je vygenerována velká oblast, ale uživatel se chce podívat pouze na malou část oblasti. Pro pohyb nahoru a dolů (přiblížení) je možné použít kolečko myši jako kdyby uživatel používal 2D mapu.



Obrázek 4.9: Uživatelské rozhraní pro zobrazení generovaného výsledku.

Popis jednotlivých prvků grafického rozhraní této části je dostupný na obrázku 4.9.

Krom toho, že výstup části logu je vypisován do okna, je vypsán automaticky také do konzole, pokud uživatel spustil program z příkazové řádky.

4.4 Externí dependence aplikace

Jak knihovna, tak uživatelské rozhraní aplikace, má několik externích dependencí. Většina z nich je automaticky stažena a zkompilována pomocí projektu CPM⁹. I přes to, že většina dependencí je stažena, je potřeba mít několik dependencí instalovaných přímo v systému.

Dependence knihovny

Knihovna má pouze dvě povinné a jednu nepovinnou dependenci, které musí být nainstalovány v systému. Mezi povinné patří knihovna `libpng` psaná v jazyce C a je využita na čtení png obrázků textur. Je dostupná ze stránky <http://www.libpng.org/pub/png/libpng.html>. Další povinnou knihovnou je `assimp`, která dostupná ze stránky <https://github.com/assimp/assimp>, a je využita na načítání `.obj` souborů. Nepovinnou systémovou dependencí je knihovna `OpenMP`, bez které lze projekt zkompilovat, ale knihovna poté nebude své výpočty rozdělovat mezi všechna jádra systému. Zbytek dependencí je automaticky stažen z internetu a zkompilován zároveň s knihovnou. Celý seznam těchto dependencí je zde:

- `nlohmann_json` – parsování json textu; dostupná z <https://github.com/nlohmann/json>
- `cpr` – odesílání požadavků na API; dostupná z <https://github.com/libcpr/cpr>
- `fmt` – formátování textu; dostupná z <https://github.com/fmtlib/fmt>
- `cdt` – triangulace polygonů; dostupná z <https://github.com/artem-ogre/CDT>
- `boolinq` – nahrazení STL práce s vektory za podobný jazyk LINQ; dostupná pouze jako jeden header soubor z <https://github.com/k06a/boolinq>

⁹<https://github.com/cpm-cmake/CPM.cmake>

- `poisson-disk-sampling` – implementace algoritmu poisson disk sampling; dostupná z <https://github.com/thinks/poisson-disk-sampling>

Dependence uživatelského rozhraní

Uživatelské rozhraní má celkově 6 dependencí. Z toho všechny až na jednu musí být instalované v systému. Jedná se o knihovnu KDE `Marble` (<https://marble.kde.org/>) využitou na zobrazení 2D map. Dále je využit framework `Qt5`, knihovna `libpng` a nepovinně `OpenMP`. Automaticky se stáhne dependence na `GPUEngine`, který je využit na jednodušší práci s `OpenGL`. Ten je stažen ze stránky <https://github.com/Rendering-FIT/GPUEngine>. Poslední dependencí je knihovna, která je součástí této práce.

Kapitola 5

Vyhodnocení a testování

V této kapitole je vyhodnoceno, jak je aplikace přívětivá pro koncové uživatele, jaký je výsledný výkon aplikace, a její současné nedostatky a chyby.

5.1 Testování uživateli

Aplikace byla předvedena dvěma skupinám uživatelů. První z nich jsou uživatelé, kteří studují informatiku, a teoreticky tak dokáží s aplikací lépe pracovat. Druhá z nich jsou běžní uživatelé, kteří jsou v tomto ohledu méně zdatní. Výsledky testování jsou, že první skupina neměla téměř žádné problémy s ovládáním aplikace. Hlavní připomínkou bylo, že se nezobrazuje výběr oblasti do momentu, kdy uživatel pustí tlačítko myši. To bylo opraveno, a oblast se zobrazuje už při výběru.

Druhá skupina uživatelů neměla žádný problém s výběrem oblasti, ale vznikl problém s pohybem v prostoru okolo mapy. Hlavní problémem bylo stávající nastavení ovládání. Uživatelům nevyhovalo nastavení, kde se nahoru a dolů na mapě pohybovalo pomocí tlačítka `Shift` a `Ctrl`, a kolečkem myši se nastavovala rychlosť pohybu. Uživatelé se snažili pohybovat nahoru a dolů právě pomocí kolečka myši, jako kdyby to byla 2D mapa. Podle toho bylo nastavení v aplikaci upraveno tak, že se nahoru a dolů pohybuje pomocí tlačítka myši, a rychlosť pohybu je řešena na klávesnici. To lépe vyhovuje uživatelům, pro které je tento přístup více intuitivní.

U obou skupin uživatelů vznikl problém, že si nevšímaly tipů na ovládání, které jsou vedle mapy nad logem. Ty byly proto přesunuty na jinou pozici a zvýrazněny tak, aby byly lépe viditelné.

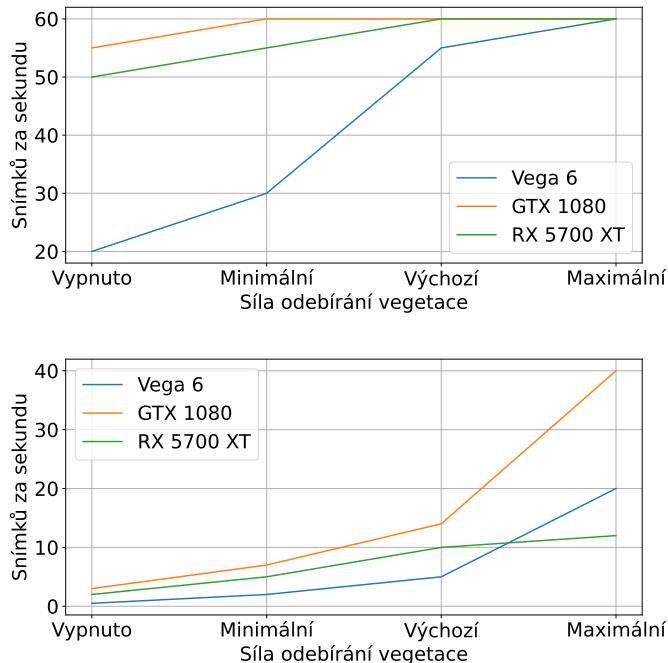
Testování dalšími uživateli ukázalo několik chyb. Nejvýznamnější z nich byla ta, že pokud při generování mapy uživatel klikl na tlačítko zpět před tím, než byly staženy metadata o oblasti, aplikace spadla, protože data se již neměla kam uložit. Tato chyba je opravena, a pád v této situaci již nenastává. Další chybou odhalenou testováním těmito uživateli bylo, že na grafických kartách společnosti AMD jsou odlesky zobrazeny špatně.

5.2 Výkon aplikace

Výkon aplikace je přímo závislý na velikosti generované oblasti a na uživatelském hardwaru. Testování bylo provedeno na třech zařízeních. Prvním z nich je stolní počítač vybaven procesorem **AMD Ryzen 7 2700** a grafickou kartou **Nvidia GTX 1080**. Dalším je stolní počítač vybaven stejným procesorem, ale grafickou kartou **AMD Radeon RX 5700XT**. Tímto

je funkčnost a výkon testován na obou hlavních výrobcích moderního grafického hardwaru. Pro testování na zařízeních s nižším výkonem bylo využito notebooku s procesorem AMD Ryzen 4500U a integrovanou grafickou kartou Vega 6.

Testování výkonu bylo provedeno na dvou přesně stejných oblastech. První z nich je menší, a je v oblasti definované koordináty 49,89242; 17,86897; 49,87692; 17,88640. Tato oblast má přibližně $2,2\text{km}^2$. Druhá je podstatně větší – konkrétně 222.34km^2 . Jedná se o město Brno a okolí definované koordináty 49,25440; 16,48189; 49,14353; 16,72984. Výkon v těchto oblastech se podstatně liší.



Obrázek 5.1: Porovnávání výkonu pro malou (nahoře) a velkou (dole) oblast. Počet snímků za sekundu je limitován na 60.

Dle grafů na obrázku 5.1 jde vidět, že velké oblasti jsou na notebooku i na grafické kartě RX 5700XT špatně použitelné, a použitelné jsou pouze v případě, že si uživatel nastaví maximální úroveň odebírání vegetace. Zároveň lze vidět chybu u škálování s RX 5700XT. I přes vysoké odebírání vegetace u velké oblasti dosahuje pouze 12 snímků za sekundu. Grafická karta GTX 1080 zde dosahuje 40 snímků za sekundu, přestože v moderních hrách má karta RX 5700XT mírně vyšší výkon. Hlavní problém s výkonem zde vzniká tím, že oblast obsahuje velmi vysoké množství vegetace, kterou některé grafické karty nestíhají zobrazovat. Na testovaném notebooku jde zároveň poznat rozdíl v počtu snímků za sekundu i u malých oblastí, ale zde je výkon přijatelný, pokud není odebírání vegetace úplně vypnuté. V případě menší oblasti je hlavním zpomalením procesorový výkon, který je ve fázi generování mapy plně využit. Snímkový rozdíl u notebooku při generování oblasti, oproti chvílím kdy je oblast již vygenerovaná, je téměř dvojnásobný. Při generování malé oblasti notebook dosahuje pouze 35 snímků za sekundu. Když je oblast již vygenerována dosahuje 60.

5.3 Nedostatky v aplikaci

Mezi nedostatky patří několik chyb. Odrazy vodních ploch se nechovají správně. Výkon aplikace, obzvlášt v případech, kde generování dat může trvat velmi dlouhou dobu, také není ideální. To lze zlepšit jiným přístupem generování oblasti – místo určování správné oblasti pro každý pixel textury je možné postupně zaplňovat oblast rasterizací jednotlivých oblastí na mapě. Jako další nedostatek je uživatelské rozhraní pro vyhledávání na mapě, které nenabízí seznam výsledků, ale pouze vyhledá první možný. Dále velmi zřídka dochází k pádům aplikace v případě, že je generování oblasti zrušeno dřív, než je oblast dogenerována do konce. Dalším nedostatkem je generování tunelů, u kterých se občas stane, že přestože je část cesty myšlena jako tunel, je zobrazena na povrchu místo pod.

Kapitola 6

Závěr

Cílem práce bylo vytvořit takovou aplikaci, která pomůže koncovému uživateli zobrazit, a tím si i lépe přestavit, jak vypadá libovolná oblast na mapě v reálném světě. Obzvlášt důležitou částí byly různé lesní cesty, které bývají na konkurenčních aplikacích, jako je Google Earth, skryty pod vrstvou stromů. Tento cíl byl splněn.

Aplikace generuje povrch, na povrch vygeneruje několik různých textur a materiálů, dále vygeneruje několik typů vegetace, a budovy. Zobrazení je obohaceno o několik efektů, které dělají zobrazení realističtější a hezčí. Bylo vytvořeno video, které je ukázkou toho jak vypadají výsledky práce. To je přidáno na paměťové médium.

Při vytváření práce jsem se naučil práci s knihovnou `OpenGL` a s knihovnou `GPUEngine`. Knihovnu `OpenGL` jsem dotedl využil jen několikrát na velmi jednoduché projekty, a v práci jsem se naučil pokročilejší zobrazovací techniky a metody. Dále jsem si v práci lépe vyzkoušel asynchronní programování, obzvlášt pak s využitím knihovny `OpenMP`.

Aplikace byla původně navržena tak, aby měla co nejméně dependencí, ale při implementaci se ukázalo že nejlepším způsobem bylo využít na některé věci knihovny s ohledem na čas a náročnost projektu.

Aplikace byla několikrát upravena podle uživatelského testování. Hlavní změny přišly v uživatelském rozhraní a v ovládání aplikace, které uživatelům nevyhovovavly. Většina uživatelů ale instinktivně zjistila, jak se v generovaném modelu pohybuje s kamerou, a jak se vybírá na mapě oblast, která se bude generovat.

Jako další rozšíření práce je kromě vyřešení aktuálních nedostatků přidání další funkcionality. Další funkcionalitu může být například možnost uložit si generovanou oblast, uložit si pouze metadata k libovolné velké oblasti, ze které by aplikace mohla získávat data i bez přístupu k internetu. Dále bych chtěl přidat detekci více různých oblastí (hlavně ledovce a sníh) a přidání různých typů oblohy podle toho, kde je oblast generována. Dalším možným rozšířením je přepsání aplikace tak, aby byla zkompilovatelná do technologoií jako je `Wasm` a zobrazování do `WebGL`, tak aby mohla aplikace být použita například na webových stránkách.

Literatura

- [1] ARVI VR. *Ambient occlusion: An extensive guide on its algorithms and use in VR*. Listopad 2018 [cit. 2022-05-06]. Dostupné z: <https://vr.arvilab.com/blog/ambient-occlusion>.
- [2] BIAGIOLI, A. *Understanding Perlin Noise*. Srpen 2014 [cit. 2022-03-09]. Dostupné z: <https://adrianb.io/2014/08/09/perlinnoise.html>.
- [3] BRIDSON, R. Fast Poisson Disk Sampling in Arbitrary Dimensions. In: *ACM SIGGRAPH 2007 Sketches*. New York, NY, USA: Association for Computing Machinery, 2007. SIGGRAPH '07. DOI: 10.1145/1278780.1278807. ISBN 9781450347266. Dostupné z: <https://doi.org/10.1145/1278780.1278807>.
- [4] BRUNNETT, G., HAMANN, B., MÜLLER, H. a LINSEN, L. *Geometric Modeling for Scientific Visualization*. Leden 2004. 241-257 s. ISBN 978-3-642-07263-5.
- [5] LOOP, C. a SCHAEFER, S. Approximating Catmull-Clark Subdivision Surfaces with Bicubic Patches. *ACM Trans. Graph.* New York, NY, USA: Association for Computing Machinery. mar 2008, sv. 27, č. 1. DOI: 10.1145/1330511.1330519. ISSN 0730-0301. Dostupné z: <https://doi.org/10.1145/1330511.1330519>.
- [6] MEIRI, E. *OpenGL Step by Step*. [cit. 2022-01-07]. Dostupné z: <https://ogldev.org/>.
- [7] MELNYCHUK, V. *Landscape generation using procedural generation techniques*. 2020. Bakalářská práce. Ukrainian Catholic University. Dostupné z: https://s3.eu-central-1.amazonaws.com/ucu.edu.ua/wp-content/uploads/sites/8/2021/07/Melnychuk-Vladyslav_188572_assignsubmission_file_VladyslavMelnychuk.pdf.
- [8] OPENSTREETMAP. *OpenStreetMap Wiki*. [cit. 2021-12-28]. Dostupné z: https://wiki.openstreetmap.org/wiki/Main_Page.
- [9] PETERS, J. Patching Catmull-Clark Meshes. *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. Duben 2002. Dostupné z: <https://www.cise.ufl.edu/research/SurfLab/papers/00pccm.pdf>.
- [10] PETERSON, S. *Computing constrained Delaunay triangulations*. [cit. 2022-05-08]. Dostupné z: http://www.geom.uiuc.edu/~samuelp/del_project.html.
- [11] VIGO, M. An Improved Incremental algorithm for constructing restricted Delaunay triangulations. Březen 1997, sv. 21. DOI: 10.1016/S0097-8493(96)00085-4.
- [12] VLACHOS, A., PETERS, J., BOYD, C. a MITCHELL, J. Curved PN triangles. *Bi-Annual Conference Series*. Únor 1970, sv. 2001. DOI: 10.1145/364338.364387.

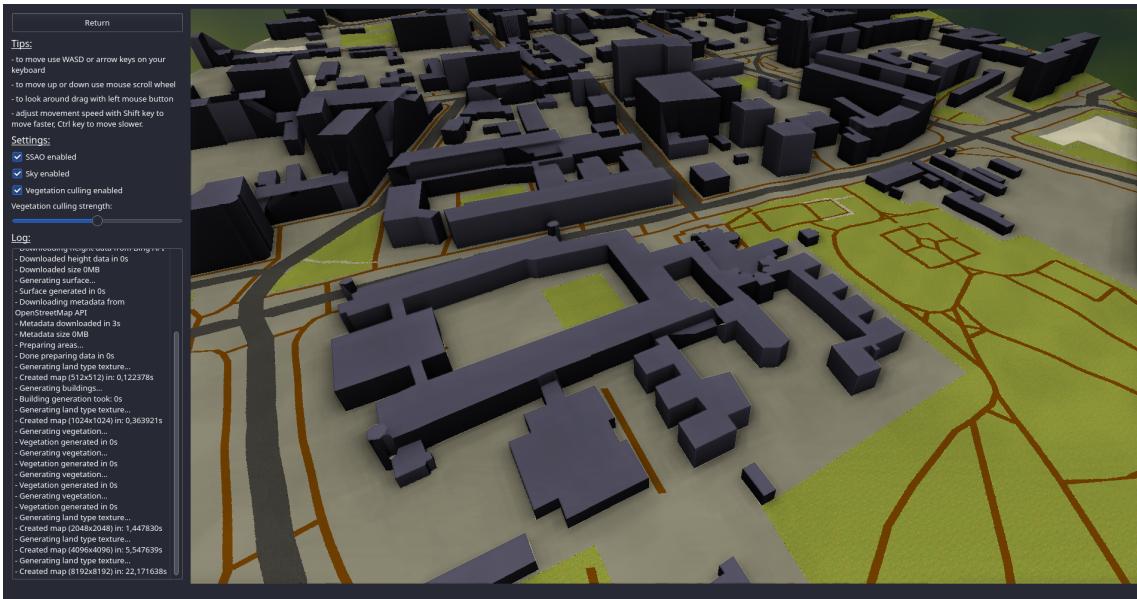
- [13] VRIES, J. de. *LearnOpenGL*. [cit. 2022-03-07]. Dostupné z: <https://learnopengl.com/>.

Příloha A

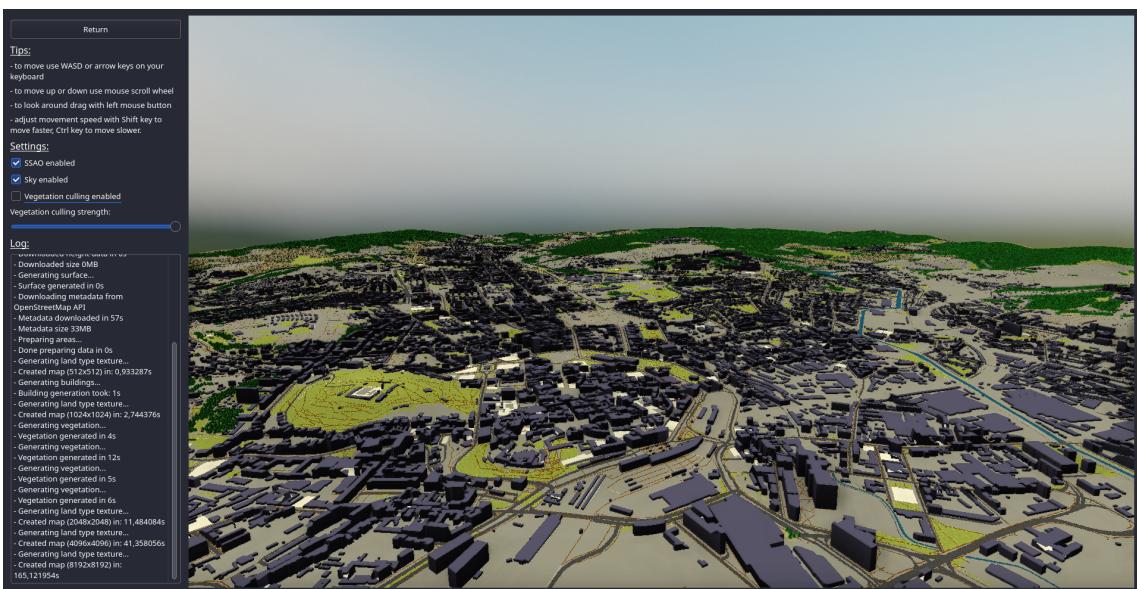
Obrázky s vygenerovanou mapou



Obrázek A.1: Hornatá oblast v Rakousku okolo města Schladming.



Obrázek A.2: Přiblžený pohled na VUT FIT ze západu.



Obrázek A.3: Pohled na centrum města Brna a dále na sever.

Příloha B

Konfigurační soubor

Konfigurační soubor musí být aplikaci při spuštění přiřazen přes argument v příkazové řádce. Jeho formát je následující:

```
{  
  "keys": [  
    {  
      "service": "<name>",  
      "key": "<value>"  
    }  
  ], "options": {  
    "terrainResolution": <number>,  
    "minTextureResolution": <number>,  
    "maxTextureResolution": <number>,  
    "textureResolutionStep": <number>,  
    "generateVegetationAt": <number>,  
    "randomSeed": <number>  
  }  
}
```

- **keys** – seznamů klíčů k autorizace na různé API
 - **service** – název služby, ke které patří klíč
 - **key** – hodnota klíče
- **options** – nastavení generátoru mapy
 - **terrainResolution** – rozlišení generovaného povrchu
 - **minTextureResolution** – minimální rozlišení textury s metadaty pro oblast
 - **maxTextureResolution** – maximální rozlišení textury s metadaty pro oblast
 - **textureResolutionStep** – násobič o kolik se změní každé další generování textury
 - **generateVegetationAt** – hodnota rozlišení textury s metadaty, při které bude vygenerována vegetace
 - **randomSeed** – vstup pro náhodné generátory čísel

Příloha C

Obsah paměťového média

Paměťové médium ve formě CD má následující strukturu:

- **src/** – složka, obsahující veškeré zdrojové soubory práce
- **bin/** – sestavené binární soubory pro systém Manjaro Linux
- **text/** – zdrojové soubory k textové části práce
- **CMakeLists.txt** – soubor, sloužící k překladu projektu pomocí programu cmake
- **Makefile** – soubor, který využívá program make k tomu, aby vytvořil všechny potřebné složky a pomocí volání cmake sestavil celou aplikaci
- **config.json** – ukázkový konfigurační souboru aplikace.
- **video.mp4** – výsledek práce v podobě videa
- **thesis.pdf** – tento text práce v zkompilovaném digitálním formátu
- **doc.pdf** – návod ke komplikaci a použití projektu