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

PROCEDURÁLNÍ MĚSTO

PROCEDURAL CITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **DANIEL DOLEJŠKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ MILET**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání bakalářské práce

Student: **Dolejška Daniel**
Program: Informační technologie
Název: **Procedurální město**
Procedural City
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte OpenGL, grafické efekty a metody procedurálního generování.
2. Navrhněte aplikaci umožňující generování a zobrazování měst.
3. Implementujte navrženou aplikaci.
4. Zhodnoťte, proměřte a vytvořte demonstrační video.

Literatura:

- Dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2 + kostra aplikace

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

Vedoucí práce: **Milet Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cílem této práce je implementovat konfigurovatelný nástroj (program), který bude schopen sestavit model města. Program používá knihovnu OpenGL pro vizualizaci a principů procedurálního generování k vytváření jednotlivých modelů (terén, budovy, silnice, ...). Proces generování má mnoho nastavitelných proměnných, které mají přímý dopad na vzhled, velikost i složitost jednotlivých modelů. Vytvořené modely je možné exportovat do souboru a dále upravovat v programech pro práci s 3D grafikou.

Abstract

The goal of this thesis is to implement configurable tool (program), which will be able to build a model of a city. The program uses OpenGL library and principles of procedural generation for model creation. The generation process itself has a number of customizable variables, which have direct impact on the visuals of the models, their complexity and size. Created models can be exported and further modified in almost any 3D graphics software.

Klíčová slova

C++, Windows, Linux, OpenGL, Perlinův šum, oktávy, quadtree, procedurální generování, město, budovy, terén, skybox, cubemap, textury

Keywords

C++, Windows, Linux, OpenGL, Perlin noise, octaves, quadtree, procedural generation, city, buildings, terrain, skybox, cubemap, textures

Citace

DOLEJŠKA, Daniel. *Procedurální město*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Milet

Procedurální město

Prohlášení

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

.....
Daniel Dolejška
16. května 2019

Poděkování

Rád bych poděkoval svému vedoucímu, Ing. Tomáši Miletovi, nejen za odborné vedení práce, nápady a inspiraci, ale také za jeho vždy pozitivní přístup a nekonečné nadšení pro toto odvětví počítačové grafiky.

Obsah

Seznam obrázků

Kapitola 1

Úvod

S využitím procedurálního generování v různých oblastech počítačové grafiky se můžeme setkat čím dál tím častěji. Cílem tohoto přístupu je zprostředkovat relativně snadné vytvoření velkého množství obsahu s různými předem stanovenými vlastnostmi. Je jej využíváno především v herním a filmovém průmyslu.

Ve filmovém průmyslu může být těchto principů využito například pro vytvoření komplexní a rozsáhlé scény. Typickým příkladem tak může být scéna na pozadí — ať už se jedná o městskou čtvrť, deštný prales či dechberoucí planetární scenérie ve vesmíru. V herním průmyslu může být procedurálního generování využito k zajištění vysoké různorodosti prostředí a to i za použití ručně vytvořených modelů. V neposlední řadě může sloužit k vytváření celých herních světů, kde i ty nejmenší detaily mohou být snadno nastavitelné.

Cílem této práce je navrhnout a implementovat nástroj, pomocí kterého bude možné sestavit a exportovat model města. Jeho hlavním rysem by měla být jeho konfigurovatelnost, tedy možnost upravit parametry generování (jako například výška budov, hustota zastavění nebo různé vlastnosti terénu) a tím dosáhnout požadovaných výsledků.

Správným použitím procedurálního generování obsahu můžeme potenciálně ušetřit velké množství času, který bychom jinak museli investovat do jeho ručního vytváření. V dnešní době existují komerční i open source programy, které poskytují různé nástroje pro procedurální generování terénu i jiných modelů. Tento nástroj kombinuje hned několik konkrétních úkolů a metod bude proto jeho použití vhodné pouze v konkrétních případech.

Samotné procedurální generování (myšleno vytváření zobrazitelných objektů) je zde možné rozdělit na několik částí — generování terénu, silnic a budov. K vytvoření tohoto nástroje ovšem nebude využito pouze principů procedurálního generování — ale také různých metod a postupů z počítačové grafiky.

1.1 Obsah kapitol

Seznámení se s principy, které jsou zapotřebí k vytvoření výše popisovaného nástroje, proběhne v kapitole 2 na straně 5. Jedná se například o představení osvětlovacích modelů (strana 5), používané principy procedurálního generování (strana 6), principy generátorů náhodných čísel (strana 7), používané optimalizační techniky (strana 9) nebo principy práce s texturami (strana 11).

V kapitole 3 na straně 16 jsou představena možná řešení problémů v rámci implementace tohoto nástroje. První podkapitola předkládá návrhy pro generování terénu (strana 16). Po ní následují kapitoly zabývající se generováním základu města — rozložením ulic (strana 20) a dále pak vytvářením parcel na základě vzniklého rozložení (strana 16). Kapitola návrhu bude uzavřena nápady na generování modelů budov a silnic, to na straně 16, a kapitolou o možném použití textur ve výsledné implementaci, strana 26.

Kapitola 4 na straně 29 bude popisovat konkrétní implementaci všech výše zmíněných modulů programu. Základní vykreslování objektů v knihovně OpenGL bude pokryto v kapitole 4.2 na straně 30. Samotné generování bude popsáno v několika oddělených kapitolách zabývajících se zvláště terénem (strana 31), silnicemi (strana 33), parcelami (strana 36) a budovami (strana 38). Posledním tématem, které bude v této kapitole zmíněno je generování a použití textur a to na straně 40.

V kapitole 5 na straně 43 bude krátce zhodnocena vzniklá implementace nástroje a to, co s tímto nástrojem lze a co nelze. Dále budou popsány i možnosti rozšíření jednotlivých částí programu.

Kapitola 2

Teorie

Tato kapitola slouží k seznámení se se základy všech různých odvětví, jejichž znalost je k implementaci tohoto nástroje nezbytná.

2.1 Osvětlovací modely

Osvětlovací model popisuje chování světla na povrchu modelu [11]. Díky těmto algoritmům je možné vykreslovat jednotlivé objekty scény v počítačové grafice.

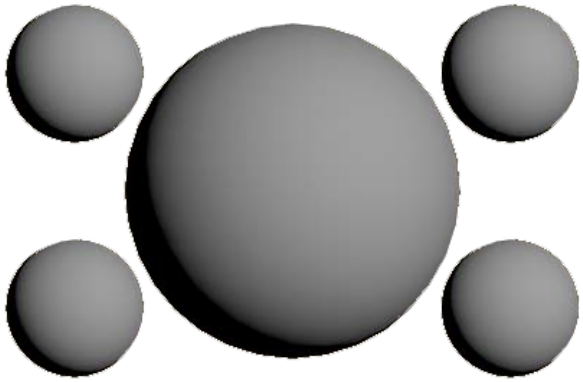
Mezi nejvýznamnější zástupce patří například Phongův a Lambertův osvětlovací model [13]. Phongův model je vhodný pro lesklé plochy, zatímco Lambertův model je použitelný především pro matné plochy.

Dále existují i odlišné modely pro vykreslování, např. PBR — physically based rendering (vykreslování založené na fyzice), tento přístup se zabývá co nejpřesnějším zachycením chování světla ve scéně.

2.1.1 Lambertův osvětlovací model

Tento osvětlovací model je jedním z nejpoužívanějších a nejednodušších. K výpočtu nevyžaduje pozici kamery, pouze světla (narozdíl např. od Phongova modelu). [13]

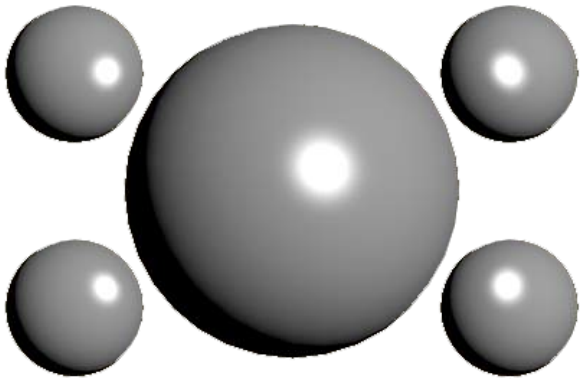
Tento model slouží jako základ pro výpočet difúzní barvy povrchu, nedochází tedy k výpočtům na základě odrazivosti a na ploše modelu proto nevznikají odlesky. Modely vykreslené pomocí tohoto modelu je možné vidět na obrázku 2.1.



Obrázek 2.1: Vykreslení Lambertovým osvětlovacím modelem, převzato a upraveno[1]

2.1.2 Phongův osvětlovací model

Tento osvětlovací model je vhodný především pro lesklé povrchy. Jeho výpočet se skládá z části difúzní barvy povrchu a z části spekulární barvy — odlesk. Výpočet odlesku je závislý na pozici kamery.



Obrázek 2.2: Zobrazení Phongovým osvětlovacím modelem, převzato a upraveno[1]

2.2 Principy procedurálního generování

Cílem procedurálního generování je algoritmické vytvoření velkého množství obsahu, popřípadě vytvoření obsahu, jehož manuální vytváření by vyžadovalo velké úsilí, mnoho, oboje nebo by to nebylo téměř možné. Tato kapitola dále čerpá především z [3, 12, 5].

Velkou část algoritmického generování tvoří určitá náhodnost. Tato náhodnost pomáhá zachování různorodosti a nemonotónnosti generovaného obsahu. Mezi nejdůležitější principy patří například šumy. Použité technologie a postupy se ovšem liší na volbě způsobu implementace.

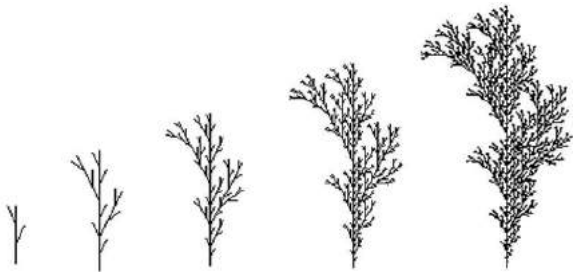
2.2.1 L-systémy

L-systémy jsou určitým typem formálních gramatik. Daná gramatika je specifikována tzv. přepisovacími pravidly, abecedou a axiomem — vstupním, počátečním, řetězcem. Při samotném generování je také určen počet iterací přepisu. [12]

K interpretaci vygenerovaného výsledku může být použita např. tzv. želví grafika. Ta bude používat výsledné axiomy jako instrukce pohybu v prostoru. Při svém pohybu za sebou bude zanechávat stopu, která je výslednou vizuální interpretací výsledku generování.

L-systémy se používají k procedurálnímu generování rostlin, keříků a různých dalších objektů především z rostlinné říše, viz obrázek 2.3.

[1] https://www.jordanstevenstechart.com/lighting-models



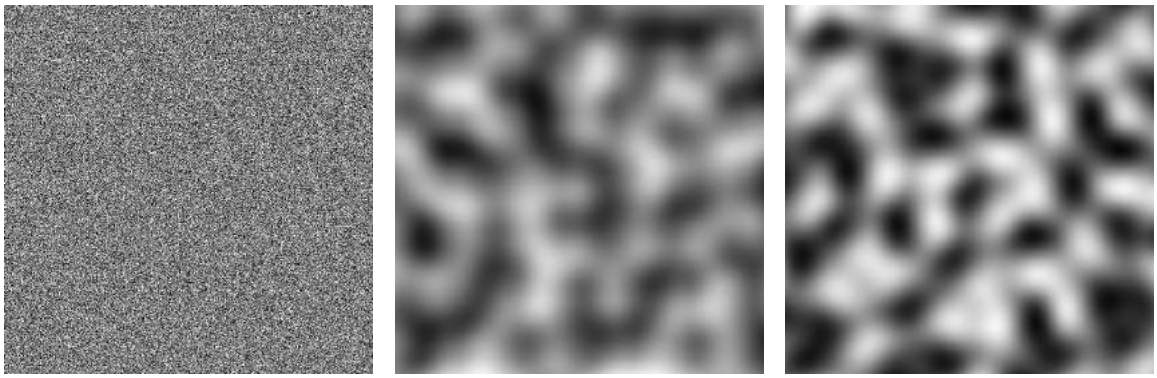
Obrázek 2.3: Několik iterací L-systému pro vykreslení rostliny, převzato[2]

2.2.2 Šumy

Šumy mohou být chápány jako výsledky (pseudo)náhodných generátorů čísel — vizualizují rozložení výsledných hodnot pro dané funkce.

Na obrázcích 2.4 je možné vidět porovnání různých typů šumů — šum náhodný (vytvořen čistě generátorem pseudonáhodných čísel), Perlinův šum a simplexní šum. Na první pohled je viditelný jasný rozdíl, minimálně mezi šumem náhodným 2.4(a) a dvěma ostatními, proto se jejich využití v procedurálním generování liší — není možné je využít ke stejným účelům.

Více o Perlinově šumu v kapitole 2.3.2.



(a) Náhodný šum (b) Perlinův šum (c) Simplexní šum

Obrázek 2.4: Náhodný, Perlinův a simplexní šum

Vizualizace několika typů šumů ve dvou rozměrech. Generované hodnoty jsou namapovány na čísla z intervalu $[0; 255]$ a použity jako barva jednotlivých pixelů.

2.3 Generátory náhodných čísel

Procedurální generování spoléhá na určitou míru nahodilosti, proto jsou generátory náhodných, respektive pseudonáhodných čísel, tak důležité.

[2] https://www.researchgate.net/publication/235816530

2.3.1 Pseudonáhodné generátory

Generování čísel pseudonáhodnými generátory není zcela náhodné, jak už název typu generátoru prozradil. To by se mohlo na první pohled zdát jako něco, co by mohl být potenciální problém, ovšem ve skutečnosti je to jednou z klíčových technik procedurálního generování.

Perioda generátoru a možnost nastavení jeho počátečního stavu zajistí generování vždy stejné posloupnosti čísel — toho může být využito jako prostředku k získávání stejných výsledků pro stejný vstupní *seed* (hodnota, podle které bude nastaven počáteční stav generátoru čísel).

2.3.2 Perlinův šum

Perlinův šum je speciální pseudonáhodný generátor čísel, jehož výsledky jsou určitým způsobem závislé na okolních hodnotách funkce Perlinova šumu — to znamená, že změny mezi hodnotami nejsou skokové, nýbrž plynulé (při vhodně zvoleném měřítku). Porovnání různých šumových funkcí v kapitole 2.2.2 na obrázku 2.4. Tato sekce čerpá z [1, 6].

Vylepšením Perlinova šumu jsou tzv. oktávy. Jejich cílem je do plynule měnících se výsledných hodnot šumu přidat další faktory náhodnosti a tím tak dosáhnout vyšší realističnosti. Porovnání výsledných hodnot ve dvou rozměrech je zobrazeno na obrázcích 2.5.

Výhodou těchto tzv. oktáv je relativně vysoká míra nastavitelnosti jejich chování. Generátor je řízen čtyřmi proměnnými `frequency`, `amplitude`, `persistence` a `lacunarity`. Ve výsledku se nejedná o nic jiného, než volání stejné funkce Perlinova šumu s mírně odlišnými parametry, vizte pseudokód 1.

Function `PerlinOctaves`$(x,\ y,\ i,\ f,\ a,\ p,\ l)$**:**
- $s \leftarrow 0$;
- **for** $y \leftarrow 1$ **to** i **do**
  - $s \leftarrow s + \texttt{Perlin}(x \cdot f, y \cdot f) \cdot a$;
  - $a \leftarrow a \cdot p$;
  - $f \leftarrow f \cdot l$;
- **end for**
- **return** s

Tato funkce vypočítá hodnotu Perlinova šumu s rozšířením oktáv v bodě (x, y). Funkce dále přijímá parametry: počet oktáv i, `frequency` f, `amplitude` a, `persistence` p a `lacunarity` l.

Algoritmus 1: Funkce pro výpočet hodnoty Perlinova šumu s oktávami

(a) 1 oktáva (b) 2 oktávy (c) 6 oktáv

Obrázek 2.5: Perlinův šum s oktávami

Obrázky zobrazují výsledné hodnoty Perlinova šumu za použití rozšíření oktávami. Výsledek s jednou oktávou, obrázek (a), je čistý Perlinův šum. Další obrázky ukazují změnu výsledku při použití dvou, obrázek (b), a šesti oktáv, obrázek (c).
Jako ostatní parametry generování byly použity: `frequency = 2`, `amplitude = 128`, `persistence = .5`, `lacunarity = 2`.

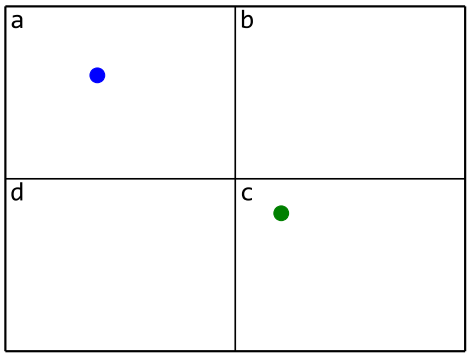
2.4 Optimalizační struktury

2.4.1 Quadtree

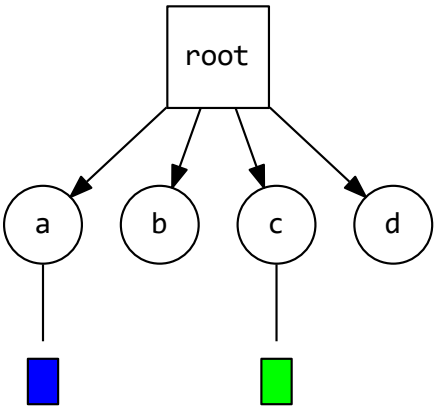
Quadtree je hierarchická stromová struktura pro dělení prostoru. Může být použita k optimalizaci algoritmů detekce kolizí ve dvourozměrném prostoru. Tato sekce čerpá především z [10, 4, 7].

Princip spočívá v rozdělení prostoru na menší segmenty — to dovolí určitý způsob vyhledávání objektů ve vzniklé struktuře. Tato vlastnost bývá principiálně použita jako filtr „nepotřebných“ objektů. Možnost získat výběr pouze relevantních objektů může při výpočetně náročných operacích extrémně urychlit jejich zpracování.

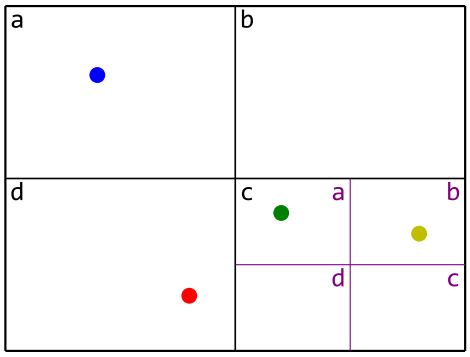
Ukázka dělení prostoru se dvěma body je ilustrována na obrázku 2.6(a), datová struktura, která odpovídá tomuto dělení je pak vidět na obrázku 2.6(b). Následně pak chování struktury po přidání dalších dvou bodů je ilustrováno obrázky 2.6(c) a 2.6(d).



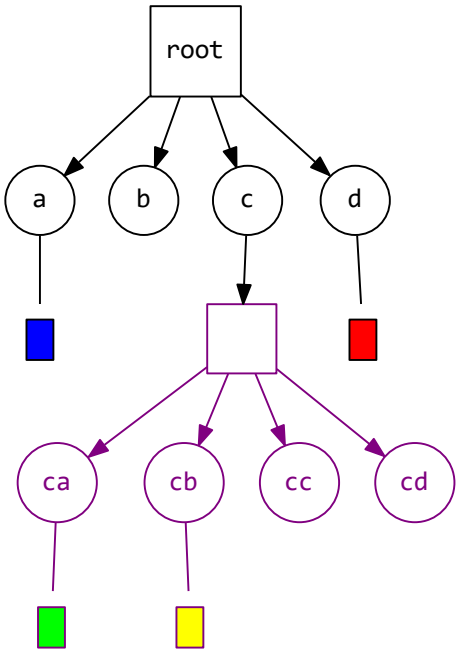
(a) Znázorněné dělení prostoru



(b) Struktura quadtree



(c) Znázorněné dělení prostoru



(d) Struktura quadtree

Obrázek 2.6: Segmentace prostoru se dvěma objekty pomocí quadtree

Na obrázku (a) je zobrazen prostor se dvěma objekty, který je pomocí quadtree rozdělen do čtyřech pomyslných segmentů. Výsledná datová struktura je zobrazena na obrázku (b). Obdobně pak případ čtyřech objektů v prostoru popisují obrázky (c) a (d).

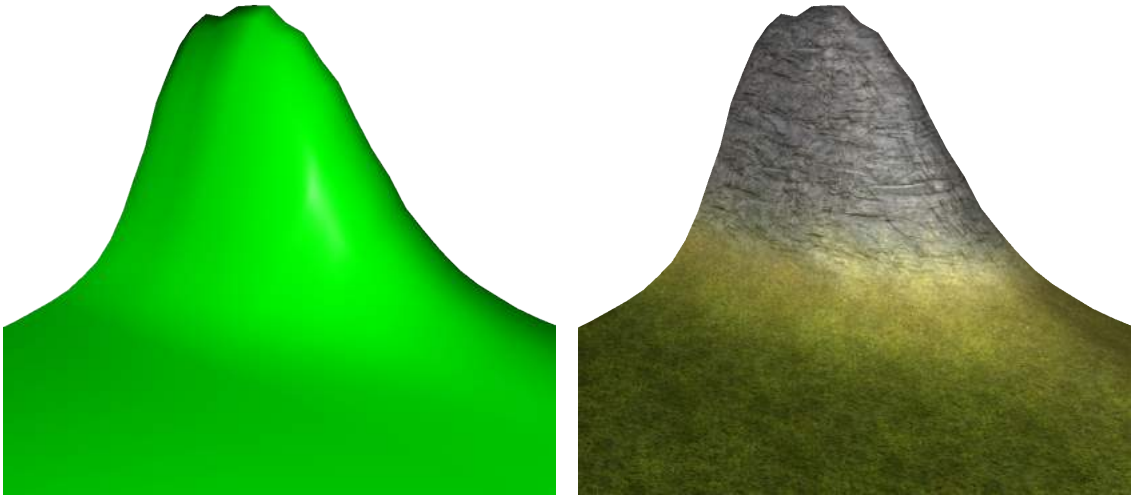
2.4.2 Octree

Octree je verze quadtree pro použití v třírozměrném prostoru. Dělení prostoru a vytváření podřízených uzlů stromu probíhá vždy na osm částí. Velice často se tato hierarchická struktura využívá k optimalizaci detekce kolizí v třírozměrném prostoru. Další použití mohou být např. pro ray-tracing, view frustum culling aj. [7]

2.5 Renderovací techniky a textury

Tato kapitola provádí bližší seznámení s různými typy a použitím textur. Dále prezentuje užitečné vykreslovací techniky a postupy.

Textury modelů jsou klíčovým prezentačním prvkem — rozdíl mezi modelem bez použití textur a modelem s texturami je markantní, viz Obrázek 2.7.



(a) Model bez textury (b) Model s texturou

Obrázek 2.7: Ukázka modelu před a po nanesení textury

Model na obrázku (a) je zobrazen pouze za pomoci Phongova osvětlovacího modelu (2.1.2). Na model na obrázku (b) byly naneseny tři textury na základě výšky jednotlivých vrcholů povrchu.

2.5.1 Typy textur

V počítačové grafice se využívá celé řady typů textur. Textury mohou ve výsledku obsahovat jakékoliv informace, pro které je obrázek ideálním úložným médiem.

Mezi často používané textury patří například:

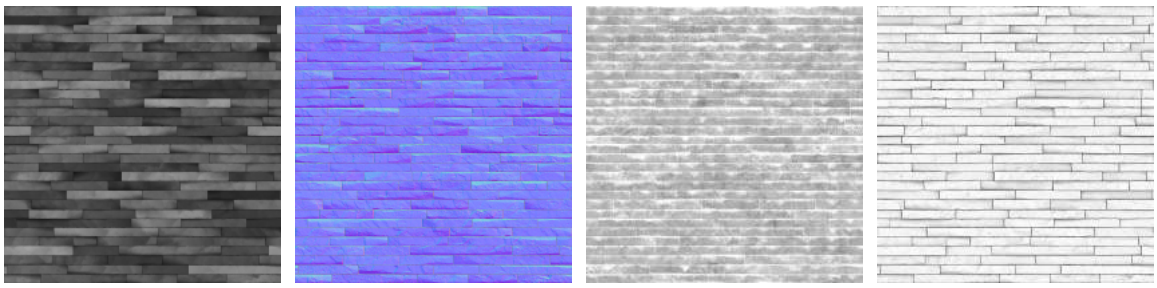
diffuse, albedo Textury tohoto typu nesou informaci o skutečné barevnosti povrchu. Hlavním rozdílem pak jsou odstraněné stíny v albedo texturách.

displacement Hodnoty této textury reprezentují skutečnou změnu ve výšce na povrchu modelu kam je textura aplikována. Tento proces se nazývá *teselace*. Obrázek 2.8(a).

normal Tato textura plně využívá všech třech barevných kanálů k reprezentaci normálových vektorů v daných pozicích na textuře. Použití této textury umožňuje aplikaci vykreslovacích technik jako je například *bump mapping*. Obrázek 2.8(b).

roughness Hodnoty uložené v této textuře reprezentují odrazivost/hrubost textury. Obrázek 2.8(c).

ambient occlusion Tato textura obsahuje stíny, které je možné aplikovat na difúzní nebo albedo texturu pro získání hlubších stínů. Obrázek 2.8(d).



(a) Displacement (b) Normal (c) Roughness (d) Ambient occlusion

Obrázek 2.8: Různé typy textur, převzato[3]

Kombinací nejen výše uvedených textur je možné dosáhnout velice zajímavých výsledků, jak je možné vidět na obrázku 2.9, a to vše pouze za použití těchto textur a specializovaného shaderu.

Obrázek 2.9: Výsledná kombinace textur, převzato a upraveno[3]

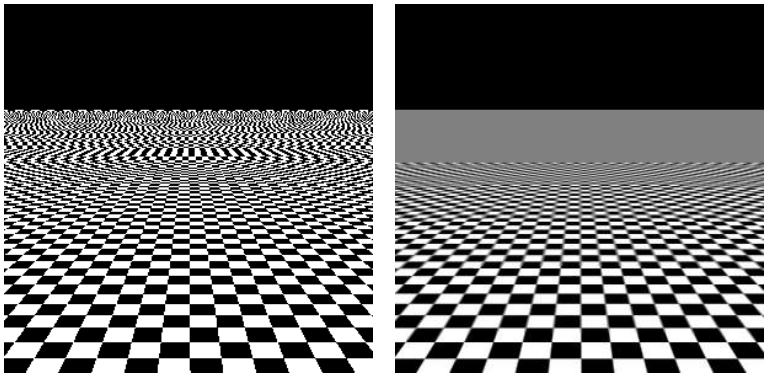
Výsledný materiál po nanesení na objekt. Tento materiál je sestaven z několika typů textur a dovoluje tak velice realistické zobrazení.

2.5.2 Mipmap

MIP textura, resp. mipmapa, je sada textur s postupně snižujícím se rozlišením. Všechny se ovšem snaží zachovat nejlepší možnou reprezentaci originální textury. Každá úroveň mipmapy má poloviční rozlišení než předchozí až do rozměru 1×1 pixel.

Této textury je využito ve chvíli, kdy je původní použitá textura v příliš velké vzdálenosti od kamery. Ve vysoké vzdálenosti dochází k problémům při mapování textur s vysokým rozlišení na příliš malé povrchy, tuto situaci lze vidět na obrázku 2.10(a). Tento problém může být vyřešen výběrem stejné textury s nižším rozlišením namísto té původní. Výsledné zobrazení s MIP texturami je možné vidět na obrázku 2.10(b).

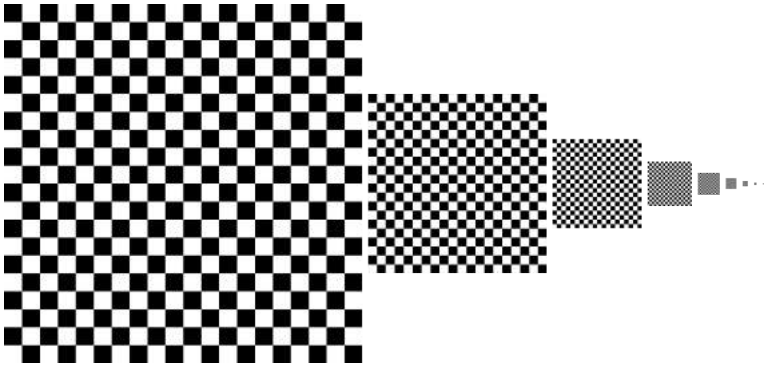
[3] https://www.textures.com/download/pbr0377/137154



(a) Textura bez mipmapy (b) Textura s mipmapou

Obrázek 2.10: Porovnání zobrazení textury s a bez mipmapy, převzato a upraveno[4]

Na obrázku (a) je zobrazení plochy s texturou bez mipmapy, je možné si všimnout „zrnění“ a deformací zvolené textury. Oproti tomu na obrázku (b) je možné vidět použití stejné textury s mipmapou — ve vyšší vzdálenosti je zvolena textura s nižším rozlišením.



Obrázek 2.11: Několik vygenerovaných úrovní mipmapy

Na obrázku je zobrazena původní textura a několik úrovní její mipmapy.

Mipmapy je pro jednotlivé textury možné předem vygenerovat ručně a následně je pouze používat. Moderní grafické karty však umí mipmapy textur vytvářet po jejich nahrání samy.

2.5.3 Cubemap, skybox

Tento typ textury je reprezentován šesti jednotlivými texturami, které lze namapovat na stěny krychle (výslednou texturu je možné vidět na obrázku 4.14). Sekce bude dále pojednávat o specifickém použití cubemap textury, kterou je skybox.

Principem skyboxu je vytvoření statického pozadí scény a tím tak dojmu, že zobrazená scéna je mnohem větší, než skutečně je. Textura tedy musí být bezpodmínečně vždy zobrazena *za* všemi vykreslovanými objekty. Dále se musí být možné v ní „rozhlížet“ — tedy musí podporovat rotační transformace. Její model se však sám o sobě při pohybu kamery hýbat nesmí — transformace posunutí nesmí probíhat.

Transformační matice pro posun v třírozměrném prostoru je uvedena ve vztahu 2.1. Čtvrtý řádek této matice reprezentuje vlastní posun modelů.

[4] http://tower22.blogspot.com/2011/10/compressor-part-22.html

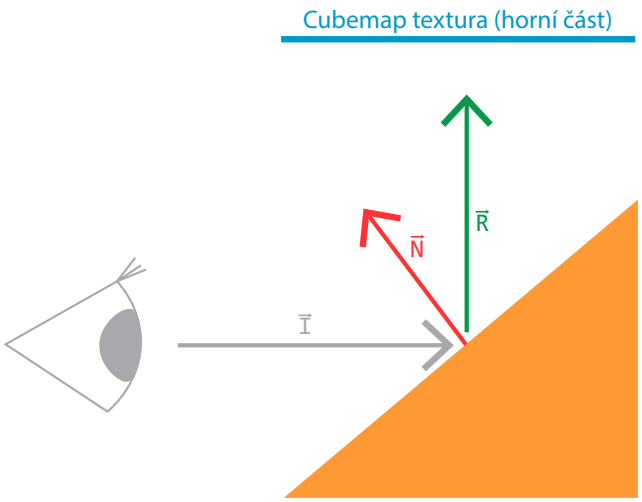
$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix} \tag{2.1}$$

Cílem je tedy v této transformační matici zajistit, že $d_x = d_y = d_z = 0$. Zbytek matice musí zůstat stejný aby bylo zajištěno funkčnosti všech dalších transformací.

Obrázek 2.12: Výsledná cubemap textura

Odraz

Specifikem cubemap textury je možnost adresace pozice v textuře pouze na základě směrového vektoru. Tato situace dovoluje snadné využití této textury k nenáročnému výpočtu statických odrazů na odrazivých plochách.



Obrázek 2.13: Odražený paprsek, převzato a upraveno[5]

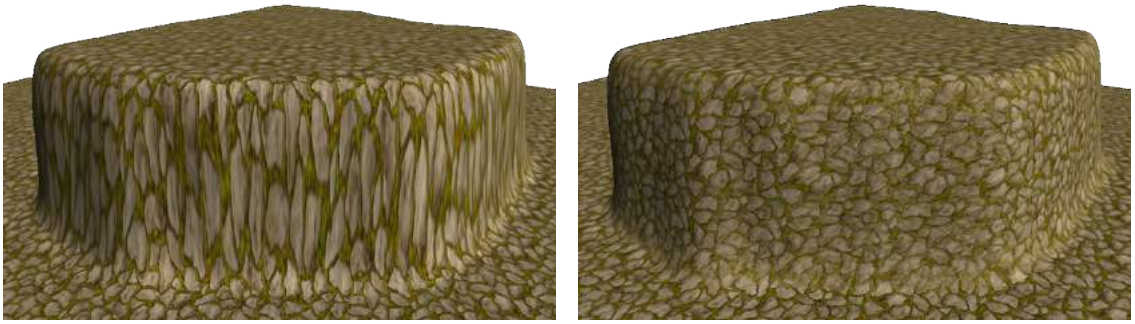
[5] https://learnopengl.com/Advanced-OpenGL/Cubemaps

2.5.4 Triplanární mapování

Tato podkapitola čerpá především z článku [8]. Princip triplanárního mapování textur řeší problémy vzniklé nanášením textur na terén pouze v jedné rovině. Tato technika využívá, tří rovin k mapování textur na povrch modelů.

Například případ modelu terénu — běžné mapování je prováděno v rovině `XY`. V této rovině ovšem bývají vrcholy modelu rovnoměrně rozloženy. Problém nastává především při prudkých změnách ve výškách, tedy při změnách hodnot na ose `Z`. Při těchto změnách dochází k deformaci modelu, všechny primitivy již nejsou stejně velké i přes to, že vzdálenosti mezi jednotlivými vrcholy na osách `X` a `Y` jsou stále stejné. V těchto místech dojde k typickému roztažení mapované textury, ilustrace problému na obrázku 2.14(a).

Tato technika provede vykreslení textury na povrch objektu třikrát, jednou pro každou ze tří zvolených os — `X`, `Y` a `Z`. Všechny tyto dodatečné operace a výpočty jsou implementovány v shaderech, to vše samozřejmě za cenu vyšší výpočetní náročnosti každého snímku. Výsledek vykreslení je možné vidět na obrázku 2.14(b).



(a) Triplanární mapování nepoužito

(b) Triplanární mapování použito

Obrázek 2.14: Ukázka efektu triplanárního mapování, převzato a upraveno[6]

Na obrázku (a) je zobrazen model terénu, na kterém je aplikována textura bez použití triplanárního mapování. Zde při nanášení textury na zkosený povrch dochází k viditelné chybě. K chybě nedochází při použití triplanárního mapování (b).

[6] https://gamedevelopment.tutsplus.com/articles/gamedev-13821

Kapitola 3

Návrh řešení

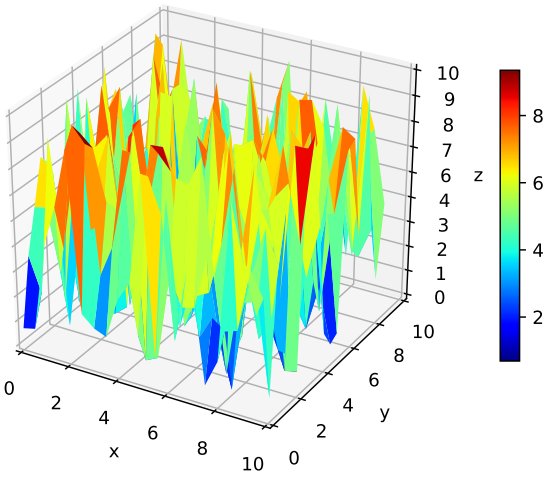
Tato kapitola prezentuje možnosti řešení — implementace — některých problémů spojených s procedurálním generováním města. Uvádí do kontextu také další techniky a možnosti spojené s navrhovanými řešeními.

3.1 Generování terénu

Základ prostoru, ve kterém bude generován model města, bude tvořit terén. Jeho klíčovými vlastnostmi by měly být změny výškách — terén by se měl stát „přírodní“ překážkou, se kterou se generátor bude muset umět vyrovnat. Pro základní demonstrační potřeby není realističnost terénu podstatná, značně ovšem může vylepšit vizuální kvality konečného výsledku. Zvýšení realističnosti by mohlo být dosaženo například implementací generování vodních ploch, které by mohly být využity u prohlubní v terénu, ty však nejsou v rámci tohoto nástroje řešeny. Tato kapitola dále čerpá především z [10, 2, 6, 3, 9].

K vytvoření modelu terénu by měl být použit (pseudo)náhodný model, pro který je možné specifikovat inicializační stav a následně vždy očekávat stejnou množinu výsledků. Pro tento účel jsou více než vhodné pseudonáhodné generátory (2.3.1), u kterých je možné nastavit tzv. *seed* (počáteční stav generátoru, dále pouze seed). Ten zajistí generování vždy stejné „náhodné“ posloupnosti čísel v rámci periody daného generátoru.

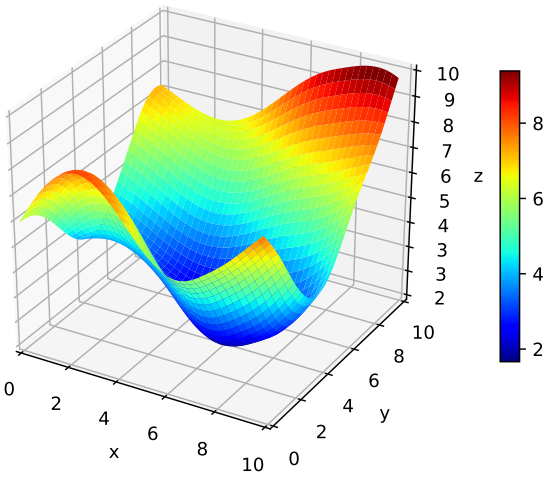
Použití pouze generátorů pseudonáhodných čísel k vytvoření celého modelu terénu ovšem není možné. Získaná čísla jsou sice při stejném počátečním stavu vždy identická, mají ovšem navzájem pramálo společného — což bychom samozřejmě od „náhodných“ čísel očekávali, ne ovšem od jednotlivých vrcholů terénu. Obrázek 3.1 demonstruje užití pseudonáhodného generátoru pro generování výšky vrcholů terénu.



Obrázek 3.1: Použití pseudonáhodného generátoru ke generování terénu

Vrcholy jsou generovány v rovnoměrné mřížce na osách `x` a `y`, jejich výška (hodnota na ose `z`) je určena náhodně.

Mnohem vhodnějším zdrojem čísel pro výpočet výšky terénu je například Perlinův (2.3.2) či simplexní šum, protože jejich výsledné hodnoty určitým způsobem souvisí s hodnotami z jejich okolí. Jsou tedy užitečnější pro generování modelu, jehož vrcholy mají stejné charakteristiky.



Obrázek 3.2: Použití Perlinova šumu ke generování terénu

Vrcholy jsou generovány v rovnoměrné mřížce na osách `x` a `y`, jejich výška (hodnota na ose `z`) je určena funkcí Perlinova šumu.

Při návrhu generátoru je nutné brát v potaz to, že získání výšky terénu v určitém bodě v prostoru bude častým úkonem, jelikož s touto informací budou konstantně pracovat téměř všechny části programu. Ideálním řešením je tedy vytvoření modulu pro výpočet výšky libovolného bodu, o tomto návrhu více v sekci 3.1.1. Samotné generování modelu pak může být rozděleno na tři další části — generování vrcholů (sekce 3.1.2), vytváření primitiv (sekce 3.1.3) a výpočet normál (sekce 3.1.4).

Jednou z dalších velice užitečných vlastností, které terén, resp. jeho implementace, může mít je možnost jeho segmentace na části. Tato možnost je velmi užitečná především v případech, kdy např. z různých implementačních důvodů není možné uložit a reprezentovat rozsáhlejší model jediným objektem.

Rozdělení terénu na menší části také zjednodušuje implementaci různých optimalizačních metod, například LOD (anglicky Level Of Detail). Tato metoda funguje na principu snižování detailu modelu dle vzdálenosti od kamery — tedy v případě, kdy je část modelu v dostatečné vzdálenosti, je detailnost celé této části snížena. Případně je také možné celé části modelu vůbec nezobrazovat (nepodstupovat je grafické kartě k vykreslení) na základě různých podmínek (vzdálenost, zakrytí překážkou, tzv. frustum culling aj.).

3.1.1 Výšková mapa

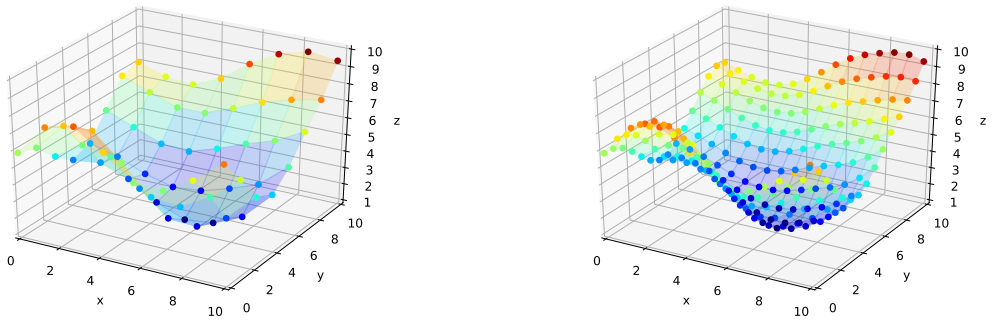
Generátor výšky terénu by měl využívat Perlinova šumu spolu s oktávami (2.3.2). Další velmi důležitou vlastností je zajištění náhodnosti výsledků na bázi seedu — výšková mapa bude pro stejný seed a vstupní hodnoty (koordináty) vždy generovat stejnou výšku.

Kvůli použití Perlinova šumu s oktávami může docházet k tomu, že vygenerované hodnoty již nebudou ležet v intervalu $[-1, 1]$. Ve funkcích proto bude nutné provádět vhodnou normalizaci vygenerovaných hodnot například zpět na interval $[-1, 1]$, popřípadě $[0, 1]$.

Výsledné hodnoty by také bylo možné mapovat na speciálně definovanou funkci definující výšku například právě pro hodnoty z intervalu $[-1, 1]$.

3.1.2 Generování vrcholů

Jednotlivé vrcholy terénu by měly být rovnoměrně rozmístěny nejlépe na čtvercové mřížce o předem stanovené velikosti. Vzdálenost mezi jednotlivými body může být určena na základě jejich počtu, například vztahem $distance = size/count$. Proměnná *count* se v této chvíli stává mírou detailnosti terénu — čím vyšší bude počet vrcholů, tím přesněji bude výsledný model kopírovat výškovou mapu, čím nižší, tím méně detailů bude možné v modelu zachytit.



(a) Nižší počet vrcholů

(b) Vyšší počet vrcholů

Obrázek 3.3: Porovnání detailnosti modelů

Znázornění detailnosti výsledného modelu v závislosti na zvoleném počtu vytvořených vrcholů.

Generování vrcholů ovšem nemusí zahrnovat pouze výpočet jejich souřadnic. S ohledem na to, že na model bude později mapováno několik textur, může být nutné pro jednotlivé vrcholy provést výpočet vzájemné intenzity daných textur. Tento proces je podrobněji navržen a popsán v sekci 3.5.1.

3.1.3 Vytváření primitiv

Primitivy modelu je možné vykreslit dvěma základními způsoby — pouze prostřednictvím vertexů, nebo za pomoci vertexů a indexů. V případě vykreslování pouze za použití vertexů bude nutné většinu z nich několikrát duplikovat a následně v konkrétním pořadí odeslat k vykreslení. Jednodušším způsobem řešení tohoto problému je použití indexování vertexů.

Indexování zamezí duplikacím vertexů za cenu vytvoření číselného seznamu (jednorozměrné pole čísel), ve kterém budou vertexy vybrány a použity. Duplicity se v poli s indexy samozřejmě vyskytují také, jde ovšem pouze o duplicity celočíselných hodnot.

3.1.4 Výpočet normál

Normálové vektory modelu jsou klíčové především ke správným výpočtům osvětlovacích modelů v shaderech. Pro každý vrchol je vypočtena hodnota normálového vektoru. Její výpočet je založen na pozici daného vrcholu a na pozicích vrcholů v blízkém okolí.

K výpočtu normálového vektoru je nutné vybrat vrcholy ve všech směrech a to nejlépe symetricky. V případě, kdy jsou vrcholy z okolí zvoleny chybně, mohou některé změny ve výškách způsobit chybný výpočet normály.

Pro tento výpočet byly zvoleny čtyři okolní vrcholy (tzv. čtyř-okolí), je možné také zvolit vrcholů osm (tzv. osmi-okolí) a tím tak výpočet zpřesnit, ale pro potřeby tohoto modelu dostačuje přesnost čtyř-okolí. Navržený postup výpočtu je dále popsán v Algoritmu 2.

Function `CalculateNormal`(x, y)**:**

`// get position vectors`

$\vec{u} \leftarrow$ `VertexPosition`(x, y); `/* current vertex position */`

$\vec{u}_1 \leftarrow$ `VertexPosition`$(x+1, y)$;

$\vec{u}_2 \leftarrow$ `VertexPosition`$(x-1, y)$;

$\vec{u}_3 \leftarrow$ `VertexPosition`$(x, y+1)$;

$\vec{u}_4 \leftarrow$ `VertexPosition`$(x, y-1)$;

`// calculate difference vectors`

$\vec{v}_1 \leftarrow \vec{u}_1 - \vec{u}$;

$\vec{v}_2 \leftarrow \vec{u}_2 - \vec{u}$;

$\vec{v}_3 \leftarrow \vec{u}_3 - \vec{u}$;

$\vec{v}_4 \leftarrow \vec{u}_4 - \vec{u}$;

`// calculate normal vectors with cross product`

$\vec{n}_1 \leftarrow \vec{v}_1 \times \vec{v}_2$;

$\vec{n}_2 \leftarrow \vec{v}_2 \times \vec{v}_3$;

$\vec{n}_3 \leftarrow \vec{v}_3 \times \vec{v}_4$;

$\vec{n}_4 \leftarrow \vec{v}_4 \times \vec{v}_1$;

`// sum and normalize`

$\vec{n} \leftarrow \vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4$;

$\hat{n} \leftarrow \dfrac{\vec{n}}{\|\vec{n}\|}$;

return $\hat{n}$

Tato funkce provádí výpočet normály v konkrétním vrcholu specifikovaného jeho souřadnicemi x a y. Funkce dále využívá funkce p, ta provádí výpočet pozičního vektoru pro vrchol v daných souřadnicích x a y.

Algoritmus 2: Funkce pro výpočet normálového vektoru pro specifický vrchol

3.2 Generování rozložení silnic

Rozložení silnic tvoří základní stavební kámen města. Cílem je vytvořit realistickou síť silnic, která umožní vhodné umístění budov a případně i dalších objektů. Je nutné, aby silnice vždy respektovaly terén, ve kterém se nacházejí a nepokračovaly do výšin strmých skal či nížin prohlubní. Silnice v kontextu této kapitoly by neměly být zaměňovány za modely silnic — generování rozložení a modelů jsou rozdílné věci (návrh pro generování modelů silnic je v sekci 3.4.2). Samotné rozložení definuje velikosti a tvary parcel, pozice modelů pozemních komunikací i jiných objektů ve městě.

Za vhodný vstupní bod generování se dá označit určitý počáteční stav silnic — silnice jsou generovány z předem definovaného, výchozího, stavu. Cesty postupují po krocích ve stanoveném směru a kontrolují svou pozici výpočtem kolizí. Vzhledem k časové náročnosti výpočtů kolizí pro všechny silnice se všemi ostatními, je velmi doporučena implementace některé optimalizační struktury pro prostorové vyhledávání, např. quadtree (sekce 2.4.1). Díky těmto strukturám je možné provádět hledání kolizí pouze pro relevantní podmnožinu silnic a výpočty tak velmi znatelně zrychlit.

Ve chvíli, kdy jsou kolize nalezeny, jsou kolidující silnice „opraveny“ — jedna ze silnic je ukončena a její další rozšiřování neprobíhá. Po určité vzdálenosti vždy dochází k vytváření dalších podřízených silnic s náhodně či deterministicky upravenými vlastnostmi (směr, rychlost, . . .).

Generování silnic může probíhat dokud existují silnice, jejichž generování nebylo ukončeno „přírodní překážkou“ (terénem), kolizí s jinou silnicí či postupem mimo mapu.

Výsledkem generování rozložení by měla být určitá datová struktura, přes kterou je možné v různých, nejlépe ve všech, směrech projít — tedy dostat se z jedné silnice „přes křižovatku“ na další a odtamtud moci pokračovat dále. Klíčovými částmi by v této struktuře měly být právě křižovatky — uzly spojující dva či více segmentů rozložení.

3.2.1 Výchozí stav

Při spuštění jsou do mapy vloženy počáteční silnice, podle kterých bude dále probíhat generování. Jejich vlastnosti (počáteční délka, pozice, směr, počet aj.) mohou být předem stanovené, či náhodně určené.

Pevně daný počáteční stav může zamezit správnému fungování generování na velkém množství terénů, které byly vytvořeny nestandardním nastavením. V případě náhodného nastavování těchto vlastností je důležité respektovat terén! Chybně nastavené výchozí silnice mohou kompletně znemožnit generování.

3.2.2 Postup silnic, generování

Jednotlivé silnice se mohou skládat z mnoha segmentů — nejsou tedy v průběhu generování vázány na jediný fixní směr, mohou proto směr průběžně měnit a stále reprezentovat jedinou ulici. Jeden krok generování je tvořen třemi dalšími kroky:

1. „**Protažení**“ neukončených silnic v aktuálním směru.

   V aktuálním směru jsou vytvořeny nové nebo rozšířeny existující segmenty všech silnic, které prozatím nebyly ukončeny (nedošlo ke kolizi s jinou silnicí, terénem atd.). Délka kroku závisí na nastavené rychlosti generování silnice. Provedení tohoto kroku je možné vidět na obrázku 3.4(a).

   Pokud by tento krok měl překonat např. přednastavené limity převýšení, nebo délky nebude proveden a silnice bude ukončena.

2. **Kontrola** všech nových či „protažených“ segmentů, výpočet kolizí.

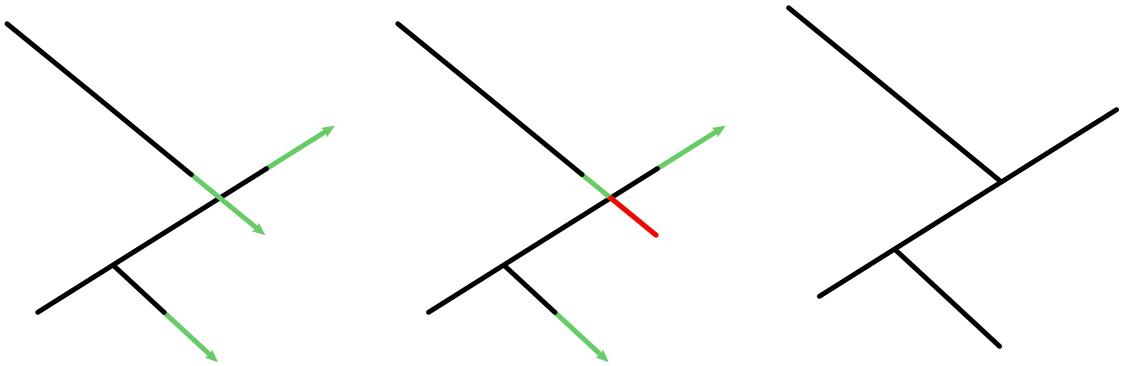
   Pro všechny změněné segmenty je proveden výběr blízkých segmentů silnic (např. pomocí quadtree, popřípadě je kontrola uskutečněna se všemi existujícími). Je proveden výpočet kolizí s vybranými segmenty. Tento krok je možné vidět na obrázku 3.4(b).

   Pokud je kolizí nalezeno více, je uložena pouze kolize s nejnižší vzdáleností od zdrojového bodu (pozice, ze které byl proveden krok protažení segmentu).

3. **Oprava a uložení** nalezených kolizí.

   Nejnovější silnice, tj. silnice na nejvyšší úrovni (výchozí silnice jsou na nulté úrovni, jim podřízené silnice jsou na první úrovni, atd.), je zkrácena do bodu kolize a je ukončena. Výsledek tohoto kroku je možné vidět na obrázku 3.4(c).

   Nalezené kolize jsou uloženy k oběma silnicím — tvoří informaci o křižovatce.



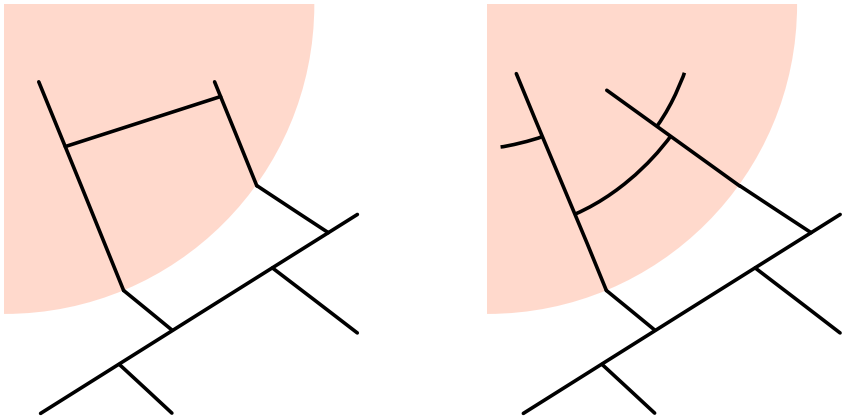
(a) „Protažení“ segmentů (b) Hledání kolizí (c) Odstranění chybných segmentů

Obrázek 3.4: Postup generování rozložení silnic

Na obrázku (a) je možné vidět protažení existujících segmentů silnic nehledě na následné kolize. Kolize vzniklé touto akcí jsou následně vyhledány a jsou určeny chybné segmenty — obrázek 3.4(b). Chybné segmenty jsou nakonec odstraněny a odpovídající cesty ukončeny — obrázek 3.4(c).

3.2.3 Modifikační zóny, úprava vlastností

Modifikační zóny jsou možností jak dočasně, popř. permanentně, upravit vlastnosti segmentu či celé silnice. Tyto zóny mohou být pro jednoduchost reprezentovány např. kruhem — snadné zjištění, zda segment do zóny zasahuje či ne. Případy, kdy je zóna příliš malá a ani počáteční ani koncový bod segmentu nejsou v mezích kruhu a přesto jím segment prochází nejsou brány v potaz.



Obrázek 3.5: Ilustrace chování modifikačních zón

Ukázka principu chování modifikačních zón. Na obrázcích je vždy jedná zóna, která upravuje směr úseček rozložení cest při jejich generování.

3.2.4 Vznik nových silnic, podřízené silnice

Vytváření nových silnic by mělo být částečně náhodně ovlivnitelné (drobná změna směru, rychlosti aj.). Avšak primárně by se vytváření nových silnic mělo řídit vzdáleností, resp. vzdáleností od poslední nově vytvořené silnice. Tato specifikace umožní hrubý předpoklad rozměrů později vytvořených parcel.

Velikost vygenerovaného města může být omezena například nastavením maximální úrovně zanoření silnic, maximální vzdálenosti od určitého bodu, aj.

3.3 Generování parcel

Parcela je objekt popisující místo, kde bude později možné generovat modely budov či vkládat jiné objekty. Tento objekt je popsán libovolným počtem okrajových bodů, které vymezují hranice parcely.

Jako parcela by měl být chápán prostor, který je ze všech stran uzavřený segmenty silnic. Pro jednoduché tvary může být parcela specifikována např. pouze čtyřmi, ovšem alespoň, třemi body. Tyto body mohou být vytvořeny z pozic křižovatek, začátků/konců silnic či jednotlivých segmentů.

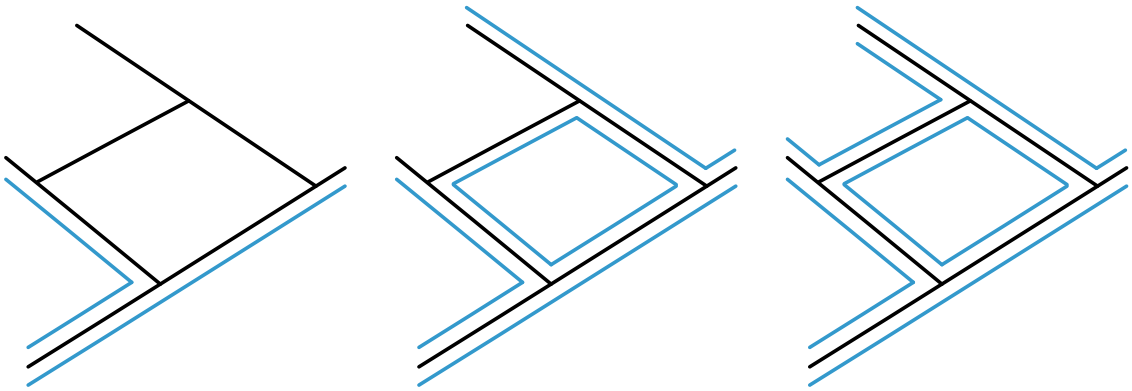
Jedna parcela nereprezentuje místo pouze pro jeden objekt — parcelu by mělo být možné rozdělit na další menší části (to může být komplikované u složitých ohraničení), které budou později podstoupeny jiným objektům ke zužitkování (vytvoření/vložení modelů, ...).

3.3.1 Procházení rozložení silnic, vytváření parcel

Jediným zdrojem informací, nutný k implementaci této funkcionality, je datová strukturu reprezentující rozložení silnic (3.2). Na základě průchodu touto strukturou mohou být parcely poměrně snadno sestaveny (ovšem záleží na konkrétní implementaci dané struktury).

Prvotním uzlem pro začátek generování by měly být počátky výchozích silnic (3.2.1), je ovšem možné začít prakticky kdekoliv, díky navržené funkcionalitě struktury. Pro každou stranu (pravá, levá) pomyslné silnice bude vytvořena jedna nová parcela. Při návštěvě každého uzlu bude existující parcele přidán nový bod. V případě, že na některé ze stran není aktivní parcela je vytvořena a je jí předán tento bod.

Průchod postupně pokračuje ve všech směrech uložených v aktuálním uzlu, vždy pouze pro odpovídající parcelu (pravá parcela na křižovatce pokračuje vždy doprava, levá doleva). Ve chvíli, kdy průchod dorazí na slepý uzel (ve zvoleném směru nepokračuje žádný segment) je aktivní parcela ukončena. Stejně tak při pokusu o vložení již existujícího hraničního bodu dojde k ukončení parcely. Tímto postupným průchodem jsou vytvořeny hranice všech parcel. Označením příslušných uzlů navštívenými je možné předejít vícenásobným průchodům a vytváření duplicitních parcel.



Obrázek 3.6: Generování parcel pro budovy

Ukázka postupného generování parcel mezi několika úsečkami reprezentující budoucí silnice. Parcely jsou odsazeny od krajů — tím uvolňují prostor pro modely silnic.

3.3.2 Dělení parcel

Dělení vytvořených parcel je nutná záležitost, budova je málo kdy osamocena a obklopena silnicemi. Je tedy nutné prvotní parcelu chápat spíše jako blok parcel. Rozdělení parcely na menší části je možné realizovat například následujícím způsobem:

1. **Vytvoření** čtvercové/obdélníkové **obálky** hranic parcely.

   Tento krok je nutný pouze v případě, kdy samotné hranice parcely již nejsou čtvercového či obdélníkového formátu. Pro složitější hranice parcel může jít o komplikovanou záležitost.

2. **Rozdělení obálky** další čtverce/obdélníky.

   Zde mohou být použity různé proměnné určující např. hustotu nově vzniklých parcel.

3. **Vyřazení či úprava objektů**, které se nenacházejí v původních mezích parcely.

   Nově vzniklé objekty, které do původních hranic parcely zasahují pouze částečně nebo vůbec jsou buď upraveny tak, aby se nacházely v platných mezích nebo jsou úplně vyřazeny a nepoužity.

3.4 Generování modelů

Sestavování modelů je jedním z posledních kroků generování. V této kapitole bude popsána zvolená implementace vytváření modelů budov a později silnic.

3.4.1 Modely budov

Modelů budov v této chvíli již může být vytvořena celá škála — modely s eliptickým nebo kruhovým základem, kvádrové budovy či jejich kombinace atd. V tomto ohledu snadným, ovšem velmi dobře vypadajícím řešením jsou právě modely sestavené z několika kvádrů. Před začátkem generování modelu budovy je ovšem důležité provést ještě jeden proces — normalizace terénu parcely.

Normalizací terénu je myšleno jeho vyrovnání. První možností vyrovnání je skutečně výběr odpovídajících vertexů nacházejících se uvnitř parcely a následné zprůměrování jejich výšek popřípadě nastavení nové hodnoty získané jiným způsobem. Druhou možností pak zůstává vytvoření „základu“ modelu, objektu, na kterém bude model budovy postaven.

Výpočet výšky modelu

Určování výšek budov pouze na základě pseudonáhodného generátoru je sice funkční, ne ovšem velmi realistické, zvláště pak, když jsou meyzi modely dovoleny velké výškové rozdíly. Realističtějšího rozložení výšek je možné dosáhnout za použití Perlinova šumu. Za kombinace obou přístupů je možné dosáhnout náhodných ovšem konzistentních výsledků. Inspirace rozložení výšek modelů budov může čerpána například z obrázku 3.7 na straně 26.

Perlinův šum je možné použít jako ukazatel maximální výšky v daném prostoru a následně může být použito pseudonáhodného generátoru k výběru výsledné výšky modelu.

Sestavení modelu

Postup sestavení modelu, který bude v této části popsán se bude zaměřovat především na modely budov sestavené z více částí. Zároveň tyto budovy pasují spíše do většího města, protože většinou vypadají jako výškové budovy. Pro jiný typ města bude pravděpodobně nutné radikálně upravit parametry případně i postup.

Vytvoření jednoho „stavebního bloku“ modelu může probíhat ve třech krocích:

1. **Vytvoření** základního **2D tvaru** bloku.

   V tom to kroku je v prostoru parcely vytvořen základní dvourozměrný tvar modelu.

2. Náhodný **posun či deformace** tvaru.

   Vytvořený tvar nyní může být zvětšen, zmenšen, posunut či jinak deformován. Pořadí a parametry těchto operací mohou být přesně určeny, případně mohou být ovlivněny nebo úplně řízeny náhodnými jevy.

   Tento krok může být také rovnou proveden v rámci vytvoření tvaru.

3. **Vytvoření 3D** bloku ze specifikací základního tvaru.

   V posledním kroku je ze základního dvourozměrného tvaru vytvořen třírozměrný objekt. Nejsnazším způsobem je duplikace vrcholů základního tvaru a změna jejich výšky v prostoru.

   Může také dojít k dalším posunům či deformacím.

Stavební bloky popsané výše je možné použít v náhodném nebo předem daném počtu s různými specifickými nastaveními k sestavení finálního modelu budovy.

Výběr barevného tónu

Při zobrazení velkého počtu podobných modelů může snadno dojít k pocitu monotónnosti. Vygenerování náhodně zbarvených modelů pomůže tento problém vyřešit. Vhodným výběrem barev jsou různé jemné odstíny oranžové, modré i bílo–šedé až černé, jak je možné vidět na obrázku 3.7 níže.

Obrázek 3.7: Fotografie výškových budov v New York City, převzato[1]

Na fotografii je možné vidět mnoho různě sytých odstínů oranžové, modré nebo šedé. Dále je také možné vidět podobnosti ve výškách budov a to především na pozadí — existují oblasti s všeobecně vyššími budovami a také oblasti s nízkými výškově konzistenními budovami.

K výběru náhodného odstínu je pravděpodobně nejlepší využít barevného modelu HSL, resp. HSV. V takovém případě je ovšem nutné provést výpočet výsledné barvy v modelu RGB.

Vytvořený odstín je možné snadno aplikovat na povrch modelu v shaderu na grafické kartě. Kromě drobné modifikace shaderu, vygenerování odstínu a jeho předání nejsou nutné žádné další změny.

3.4.2 Modely silnic

Modely silnic by měly pozičně kopírovat vytvořené rozložení silnic z kapitoly 3.2 a rozdíly ve výškách terénu. Tyto objekty tvoří prostor mezi jednotlivými parcelami pro budovy.

Vytvoření modelu spočívá ve výpočtu pozic bodů, které se nacházejí v určité vzdálenosti na obou stranách segmentů rozložení silnic.

3.5 Textury

Textury jsou velmi důležitým vizuálním prvkem, které mohou vytvořeným modelům dodat značnou dávku realističnosti, pro implementaci funkcionality nástroje však nejsou klíčové.

[1] https://www.pexels.com/photo/466685/

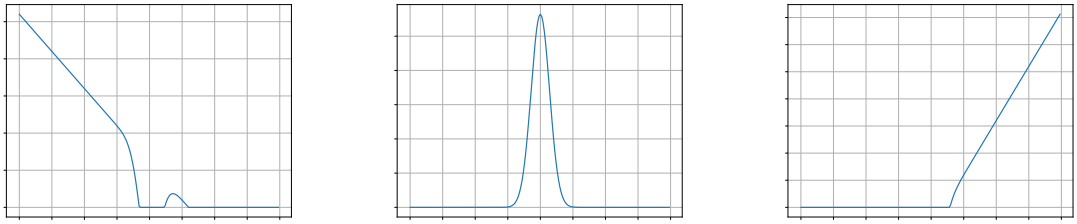
3.5.1 Terén

Cílem je na model nanést celkem tři textury — hlínu, trávu a kámen. Jejich intenzita bude v tomto případě založena na výšce vrcholu, ale je ovšem možné vybírat texturu například na základě sklonu terénu (za pomoci normál), či jiného principu.

Textury by měly být míchány tak, že

- *hlína* bude zobrazena v prohlubních terénu a při přechodu rovného terénu do kopce,
- *travnatá plocha* bude zobrazena pouze na rovném terénu (výška ≈ 0),
- a *kámen* bude zobrazen na kopcích.

Na obrázku 3.8 jsou pak znázorněny funkce, které popisují intenzitu jednotlivých textur dle specifikací uvedených výše. Vyhodnocení intenzity textur může být provedeno jednou — při generování vrcholů terénu, popřípadě mohou intenzity být vyhodnocovány každý snímek při vykreslování grafickou kartou. Pokud jsou intenzity vypočteny při generování vrcholů, je následně obtížnější kdykoliv a téměř jakkoliv upravovat terén (intenzity upravených vrcholů je nutné přepočítat). Výpočet intenzit textur v shaderu má dopad na rychlost vykreslování.



(a) Intenzita textury hlíny (b) Intenzita textury trávy (c) Intenzita textury kamene

Obrázek 3.8: Funkce intenzity textur

Funkce intenzity jednotlivých textur v závislosti na výšce vrcholu. Uprostřed grafu pomyslná nula, nalevo záporné hodnoty (prohlubně), napravo kladné hodnoty (kopce).

K nanesení textur by mělo být použit princip triplanárního mapování (sekce 2.5.4). Tato technika zamezí případným chybám v mapování textur na model terénu.

3.5.2 Budovy

Modely budov budou, stejně jako terén, používat více než jednu texturu. Pro stěny budov může být použita prakticky jakákoliv textura betonu, ovšem je vhodné texturu zvolit relativně světlou — to umožní snadné a viditelné aplikování barevných tónů, které byly zmíněny v sekci 4.6.2.

Neodmyslitelnou částí výškových budov jsou velká skleněná okna. K zobrazení oken je možné použít nejrůznější existující textury, avšak v rámci tohoto návrhu bude zvolen jiný postup — procedurální vygenerování prostorů oken, resp. generování *specular map* (textura nesoucí informaci informaci o odrazivosti povrchu, dále pouze specular map) ve tvaru oken.

Principiálně generování a používání specular mapy spočívá pouze v umístění bílých ploch na texturu o určitých rozměrech a jejím namapování na stěny objektu.

Využitím vlastností cubemapy (tato situace předpokládá použití skyboxu dle návrhu popsáného v sekci 3.5.3 níže) lze odraz prostředí ve specular mapě zajistit velice snadno.

3.5.3 Skybox

Skybox je speciální textura složená z šesti obrázků obsahující „pozadí“ scény. Jeho použití je velice výhodné nejen proto, že scéně dodá značnou dávku realističnosti, ale především kvůli užitečným vlastnostem dovolující snadné vytváření odrazů (například v oknech).

Další informace o cubemap texturách v sekci 2.5.3 na straně 13.

Kapitola 4

Implementace

Tato kapitola rozebírá konkrétní implementaci tohoto nástroje, která byla v rámci této práce vytvořena v jazyce C++ s důrazem na multiplatformnost řešení (Windows, Linux).

4.1 Použité knihovny

ArgumentViewer[1] Implementuje třídy pro snadné zpracování a manipulaci s přepínači z příkazové řádky při spuštění programu. Dále poskytuje možnost přehledného výpisu programem přijímaných přepínačů.

BasicCamera[2] Zajišťuje implementaci tříd reprezentující kamery scény. Třídy na základě své pozice a rotace v prostoru poskytují snadný výpočet transformačních matic používané v shaderech pro správné vykreslení scény.

FreeImage, FreeImagePlus[3] Implementuje funkce a třídy pro snadné načítání, ukládání, zpracování a modifikaci obrázků v tradičních formátech jako například JPEG, PNG, BMP, TIFF aj.

geGL[4] Knihovna poskytuje funkce a pomocné třídy pro snadnější práci s knihovnou OpenGL v jazyce C++. Poskytnuté třídy pak dovolují použití objektově orientovaného přístupu pro komunikaci prostřednictvím knihovny OpenGL.

glm[5] Knihovna OpenGL Mathematics (glm) je matematická knihovna pro grafické programování v jazyce C++ založena na specifikaci jazyka pro programování shaderů v OpenGL (GLSL).

SDL2[6] Multiplatformní knihovna poskytující nízkoúrovňový přístup ke správě zvuku, přístupu ke grafickému hardware a vstupu z klávesnice, myši či dalších zařízení.

SDL2CPP[7] Poskytuje funkce a třídy pro snadnou práci s knihovnou SDL2.

[1] https://github.com/dormon/ArgumentViewer
[2] https://github.com/dormon/BasicCamera
[3] http://freeimage.sourceforge.net/
[4] https://github.com/dormon/geGL
[5] https://glm.g-truc.net/
[6] https://www.libsdl.org/
[7] https://github.com/dormon/SDL2CPP

Vars[8] Implementuje funkce a třídy pro správu proměnných napříč celým programem. Dále také umožňuje snadnou registraci změn jednotlivých proměnných, což umožňuje další optimalizace implementace.

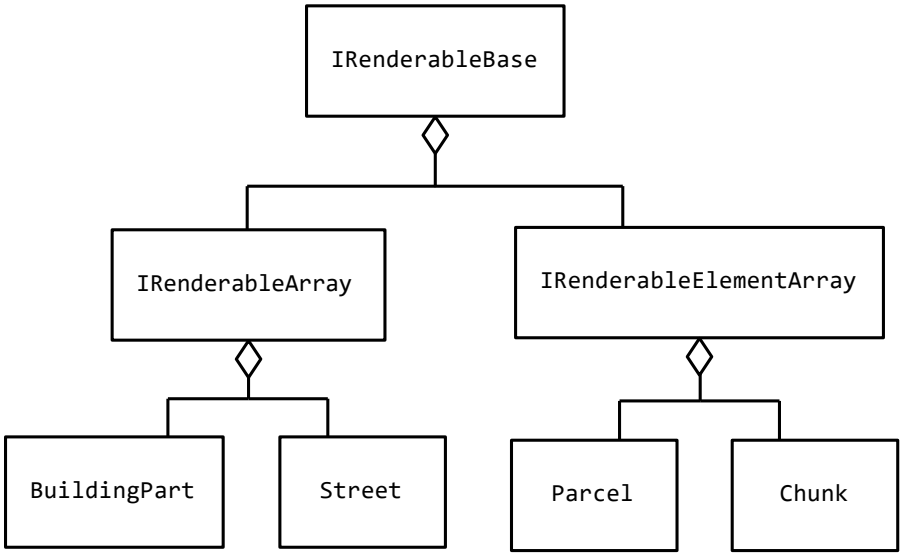
4.2 Zobrazení

Proces zobrazení konkrétního objektu lze v této implementaci rozdělit na dvě části — výběr objektu, resp. aktivace odpovídajícího *bufferu* (úložiště dat, dále pouze buffer), a vlastní rasterizace *shaderem* (program běžící na GPU vykreslující objekt, dále pouze shader).

Volba objektu je řízena programem běžícím na CPU, přesněji odesíláním instrukcí grafické kartě prostřednictvím knihovny geGL, resp. OpenGL. Vykreslení zprostředkují programy běžící na grafické kartě — shadery.

4.2.1 Přepínání bufferů

Implementované třídy pro volbu objektů, resp. manipulaci s datovými buffery objektů, využívají dědičnosti jazyka C++. Třídy jednotlivých zobrazitelných objektů (`Terrain::Chunk`, `Infrastructure::Street` aj.) vždy dědí od třídy specifikující způsob vykreslení objektu vizte Obrázek 4.1.



Obrázek 4.1: Diagram tříd zobrazitelných objektů

Bázovou třídou pro všechny zobrazitelné objekty je `IRenderableBase`. Třídy `IRenderableArray` a `IRenderableElementArray` pak určují, jakým způsobem se jej třída pro vykreslování pokusí zobrazit.

Tohoto řešení za pomoci parametrického polymorfismu využívá třída `Application::Renderer`. Ta na základě typu vstupních parametrů volí instrukce odesílané na grafickou kartu a efektivně tak zapouzdřuje odesílání instrukcí. Tím dovoluje snadnou manipulaci se zobrazitelnými objekty při jejich vykreslování, vizte Zdrojový kód 1.

[8] https://github.com/dormon/Vars

```
for (const auto& building : buildings)
        for (const auto& part : building->parts)
                renderer->Render(part);
```

Vykreslovací smyčka modelů budov, třída `Application::Renderer` efektivně zapouzdřuje logiku potřebnou k vykreslení objektů.

Zdrojový kód 1: Ukázka kódu pro vykreslení budov

Toto řešení sice umožňuje velmi snadnou manipulaci s objekty, není však zdaleka optimálním. Bázové třídy, ze kterých vycházejí všechny třídy zobrazitelných objektů, vždy využívají vlastních bufferů — při velkém množství objektů vzniká také velké množství bufferů. To vede k nutnosti častěji zasílat instrukce grafické kartě, což proces vykreslování samozřejmě zpomaluje.

Optimální řešení je takové, které vyžaduje zaslání minimálního počtu instrukcí grafické kartě k vykreslení co největšího počtu objektů. Nejlépe všech objektů stejného typu.

4.2.2 Vykreslení

Implementovaný nástroj používá k vykreslování 7 různých shaderů. Tři jsou určeny běžnému vykreslování různých objektů — implementují různé zobrazovací metody či osvětlovací modely. Další tři jsou určeny k analýze stavu (vizualizace normálových vektorů či *wireframe* [běžně používaný termín pro zobrazování primitiv modelů, dále pouze wireframe]) a jeden slouží k vykreslení skyboxu (o implementaci skyboxu více v kapitole 4.7.3 na straně 41). Většina z nich (i některé shadery určené k analýze) využívá Phongova osvětlovacího modelu (2.1.2).

Jednotlivé typy shaderů (vertex, fragment atd.) jsou v rámci jednoho programu vždy ukládány do jednoho souboru. Jejich načtení a uchování za běhu programu umožňuje `Application::ShaderLoader`.

Dále třída `Application::ShaderManager` zprostředkovává snadné zavedení, validaci a přepínání shaderů. Mimo zasílání odpovídajících instrukcí prostřednictvím knihovny geGL, resp. OpenGL, zajišťuje i nastavení správných uniformních proměnných u jednotlivých shaderů při jejich přepínání na grafické kartě. Jednotlivé programy se v konfiguraci používaných uniformních proměnných mohou lišit.

4.3 Terén

Generovaní jednotlivých částí terénu probíhá ve třech fázích. V první fázi jsou vytvořeny vrcholy terénu v prostoru. Dále následuje propojení odpovídajících vrcholů — vytvoření primitiv. Posledním krokem je výpočet normálových vektorů jednotlivých vrcholů.

Proces tvorby terénu využívá několika tříd a funkcí. Třída `Terrain::Map` implementuje správu a přístup k jednotlivých částem terénu. Ty jsou v programu reprezentovány třídou `Terrain::Chunk` a nesou data určité podmnožiny vrcholů terénu.

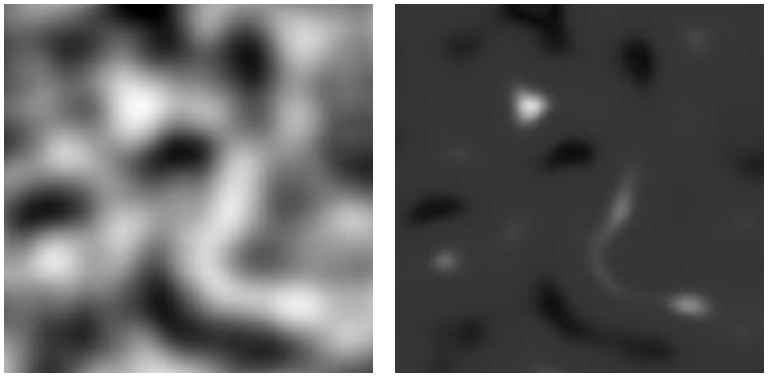
Klíčovou roli v celém programu ovšem hraje výšková mapa, reprezentovaná třídou `Terrain::HeightMap` — ta je abstrakcí výpočtu výšky pro libovolné body v prostoru.

4.3.1 Výšková mapa

Funkce pro výpočet výšky bodu v prostoru přijímá jediný vstupní parametr — jeho souřadnice. Ovšem dále také využívá řadu dalších, při spuštění specifikovaných, proměnných modifikujících chování generátoru Perlinova šumu. Všechny výše zmíněné vstupy jsou použity pro výpočet Perlinova šumu s oktávami (2.3.2), jehož hodnota slouží jako základ pro získání výsledné výšky v bode vstupních souřadnic.

Před použitím hodnot Perlinova šumu je provedena jeho normalizace. Při spuštění je vypočteno určité množství vzorků, z těchto hodnot je následně vybráno maximum a minimum. Pomocí těchto hodnot pak probíhá normalizace všech dalších generovaných čísel. Toto řešení není optimální, protože pokud počáteční počet vzorků není extrémně vysoký může snadno dojít k vygenerování vyššího čísla než je registrované existující maximum. Aktualizace hodnot minima a maxima v průběhu generování by způsobila nenávaznost jednotlivých částí terénu.

Po normalizaci je výsledná hodnota použita jako vstup mapovací funkce určující konečný tvar terénu. Obrázek 4.2(a) zobrazuje normalizované vzorky přemapované na hodnoty 0-255 a použité jako barva jednotlivých pixelů. Na obrázku 4.2(b) je vidět výsledná hodnota výšky po namapování vzorků na výškovou funkci.



(a) Zdrojový Perlinův šum (b) Výsledek po namapování

Obrázek 4.2: Výšková mapa

Obrázek (a) zobrazuje výstup funkce pro výpočet Perlinova šumu. Na obrázku (b) je pak možné vidět výsledek namapování zdrojového šumu na výškovou funkci.

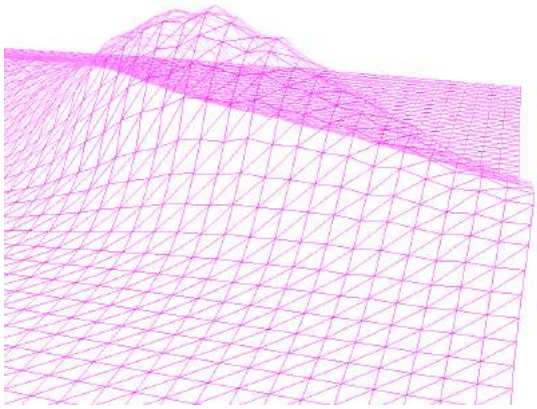
Implementace využívá skutečnosti, že použitý generátor pseudonáhodných čísel generuje při stejné inicializační hodnotě vždy stejnou posloupnost čísel — to dovoluje použití proměnné pro seed k zajištění vytvoření stejného terénu.

4.3.2 Generování primitiv

Implementované řešení se shoduje s jeho návrhem v kapitole 3.1.2.

Vrcholy terénu jsou vytvořeny v pravidelné mřížce, za pomoci dvou jednoduchých `for` cyklů, které iterují přes přednastavené počty výsledných vrcholů a vypočítávají jejich souřadnice. Po výpočtu souřadnic jednotlivých vrcholů je následně proveden výpočet výšky na daných souřadnicích (volání funkcí ze třídy `Terrain::HeightMap`). Vlastnosti mřížky mohou být ovlivněny několika proměnnými — detail, rozměry částí terénu aj.

Výsledné primitivy modelu je možné vidět na obrázku 4.3.

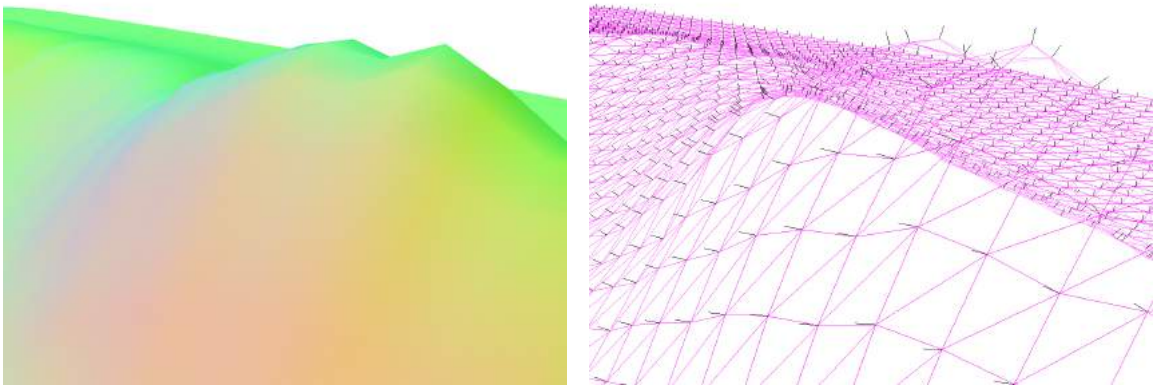


Obrázek 4.3: Vygenerované primitivy terénu

Zobrazení vytvořených primitiv modelu terénu speciálním shaderem.

4.3.3 Výpočet normál

Skutečná implementace kopíruje řešení navržené v kapitole 3.1.4. Na obrázku 4.4 je pak možné vidět vizualizaci výsledků řešení.



(a) Mapování hodnot normál na RGB spektrum (b) Vizualizace normál speciálním shaderem

Obrázek 4.4: Výsledné hodnoty normál zobrazené speciálními shadery

Na obrázku (a) je zobrazení provedeno mapováním hodnot normálových vektorů na hodnoty barvy fragmentů. Výsledkem jsou různé barvy různých intenzit v přímé závislosti na směru a velikosti normálového vektoru. Obrázek (b) zobrazuje normály speciálním shaderem jako jednoduchou černou úsečku z pozice vrcholu — ta indikuje jak velikost vektoru, tak i jeho směr.

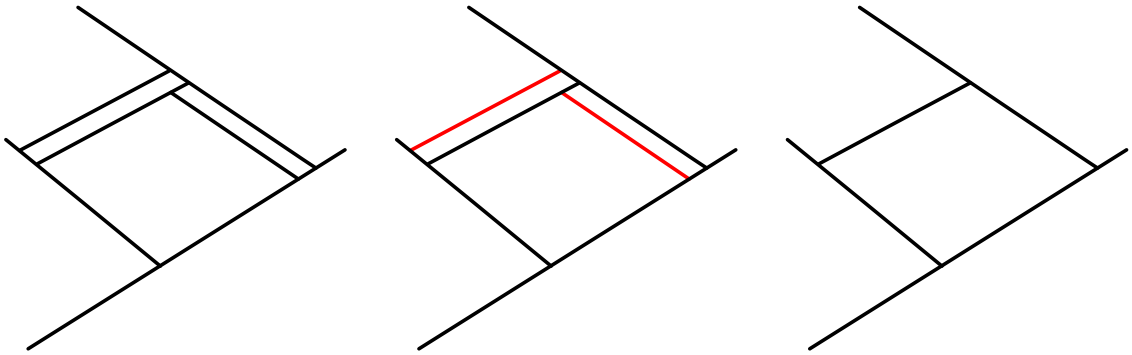
4.4 Rozložení silnic

Rozložení silnic je klíčovým elementem implementace tohoto nástroje od původního návrhu uvedeného v kapitole 3.2 se však liší. Struktura umožňující snadné procházení rozložení po jednotlivých křižovatkách nebyla součástí prvotního návrhu a její implementace nebyla později dokončena.

Tato část ve své implementaci využívá celou řadu tříd: `Infrastructure::StreetMap` a `Infrastructure::Street` jsou použity k reprezentaci vytvořených výsledků. Dále jsou použity třídy `Terrain::HeightMap`, ta je použita k samotnému řízení generování — rozložení musí respektovat existující terén, a `Terrain::Map`, tato třída reprezentuje existující mapu a umožňuje zjistit, kdy silnice opustila prostor vytvořené mapy.

Postup generování zůstává identický dle návrhu uvedeném v kapitole zmíněné výše. Kolizí vzniklé křižovatky jsou ovšem ukládány do seznamu ve třídách reprezentující celé silnice, struktury, které lze snadno procházet nejsou vytvářeny. Namísto toho vždy dochází k vyhledávání křižovatek v dříve zmíněných seznamech.

Při postupném generování silnic začalo v této implementaci docházet k chybám ve výsledném rozložení (ty je možné vidět na obrázku 4.5(a)). Tento problém byl odstraněn post–procesem — nejdříve jsou vypočteny vzdálenosti jednotlivých silnic. Silnice, které mají mezi sebou příliš krátkou vzdálenost a jsou rovnoběžné, případně svírají příliš ostrý úhel, jsou zvoleny (Obrázek 4.5(b)) a odstraněny (Obrázek 4.5(c)).



(a) Výsledek generování (b) Výběr chybných silnic (c) Odstranění vybraných silnic

Obrázek 4.5: Průběh odstranění chybných segmentů rozložení silnic

Na obrázku (a) je možné vidět existující implementační chybu generování. Vyhledání chybně vygenerovaných segmentů je vidět na obrázku (b), jejich následné odstranění pak na obrázku (c).

Generování rozložení silnic využívá upravenou implementaci quadtree (popsáno dále v následující kapitole 4.4.1). Jakékoliv delší generování by bez použití této optimalizační strategie nebylo možné (časová náročnost s přibývajícími ulicemi exponenciálně roste).

4.4.1 Quadtree

Implementovaná verze quadtree, která je použita pro segmenty rozložení silnic, je založena na mírně upraveném algoritmu pro ukládání a vyhledávání objektů. Algoritmus využívá „obalový“ objekt segmentu silnice, který zapouzdřuje různé operace pro kontrolu vzájemných pozic s objekty stejného typu. Klíčovým rozdílem oproti implementaci quadtree pro body je možnost uložení segmentu do uzlu stromu i v případě, kdy daný uzel má podřízené uzly.

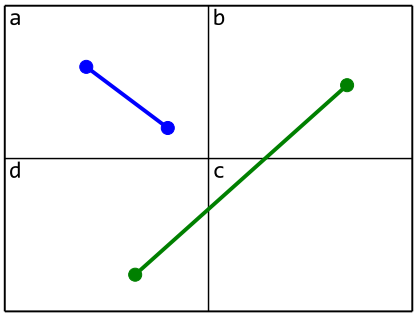
Obalovým objektem je konkrétně myšlena třída `Utils::RectBounds`, která prakticky obsahuje částečnou implementaci třídy `System.Drawing.Rectangle`z jazyka C#. Ta repre-

zentuje vnější hranice segmentů přímky (maximální prostor, který může segment v prostoru zabírat).

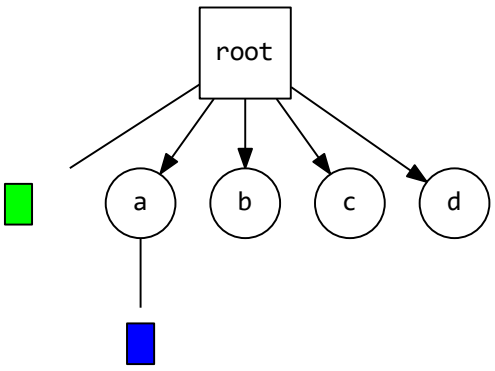
Vložení objektu začíná na nejvyšší úrovni — kořenovým uzlem. Pokud je možné hranice objektu plně obsáhnout v některém z podřízených uzlů, je proveden pokus o vložení objektu do daného uzlu. V případě, že daný uzel žádné podřízené uzly nemá a jeho velikost přesahuje minimální možnou velikost uzlu, jsou mu přesně čtyři podřízené uzly vytvořeny (prostor je rozdělen na další čtyři části). Tato akce probíhá, dokud není buď možné objekt plně obsáhnout některým z uzlů, popřípadě do chvíle, kdy nemohou být vytvářeny další podřízené uzly (protože by došlo k porušení podmínky jejich minimální velikosti). Když není možné objekt uložit do žádného uzlu na určité úrovni, je daný objekt uložen o úroveň výše. Znázornění dělení prostoru a ukládání objektů v datové struktuře quadtree je možné vidět na obrázku 4.4.1.

Implementace je limitována výchozí velikostí kořenového uzlu, jehož prostor je dělen pro podřízené uzly. Pokud jsou vkládané objekty mimo rozměry kořenového uzlu do do něj přesto vloženy — to může vést ke zpomalení celé implementace při špatně zvolených výchozích rozměrech. Stejná situace může nastat i při vkládání příliš velkých objektů, které sice jsou v mezích uzlů, ale jsou tak velké, že musí být uloženy vysoko ve stromové struktuře. Tím pro velkou část volání dochází k nutnosti zpracovávat velké množství objektů v nadřazených uzlech.

Vyhledávání v uzlech probíhá obdobným způsobem. Nejdříve je nalezen uzel na nejnižší úrovni kam se může objekt dostat, zde je vytvořen výběr všech objektů, které tento uzel obsahuje a které jsou také obsaženy v uzlech na nižších úrovních. Odtud pak dochází k postupnému vracení se do vyšších úrovní, kde se k výběru přidávají také objekty z vyšších úrovní.



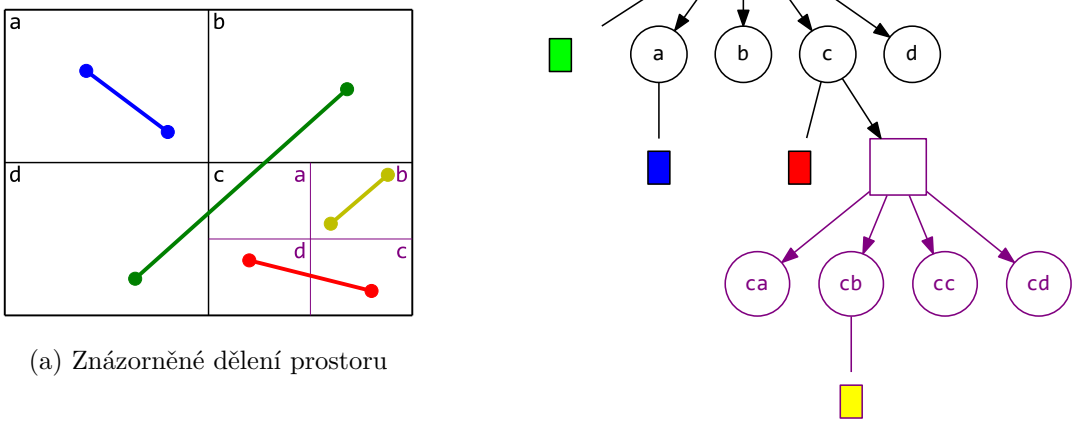
(a) Znázorněné dělení prostoru



(b) Struktura quadtree

Obrázek 4.6: Ukázka dělení prostoru s dvěma úsečkami pomocí quadtree

Na obrázku (a) je zobrazen prostor se dvěma úsečkami a pomyslným rozdělením tohoto prostoru pomocí této implementace quadtree. Vzniklá datová struktura, po vložení těchto tvou úseček, je zobrazena na obrázku (b).



(a) Znázorněné dělení prostoru

(b) Struktura quadtree

Obrázek 4.7: Ukázka dělení prostoru se čtyřmi úsečkami pomocí quadtree

Na obrázku (a) je zobrazen prostor se čtyřmi úsečkami a pomyslným rozdělením tohoto prostoru pomocí této implementace quadtree. Vzniklá datová struktura, po vložení těchto čtyřech úseček, je zobrazena na obrázku (b).

Tato implementace byla založena na principech již existující implementace[9] v jazyce C#.

4.5 Parcely

Parcely slouží jako místo pro umístění objektů. Tato implementace tedy rozlišuje dva typy parcel: parcely pro modely budov a parcely pro modely silnic.

Algoritmy pro generování parcel využívají výhradně třídy reprezentující rozložení silnic, tedy `Infrastructure::Street`, a struktury s nimi spojené (křižovatky aj.).

4.5.1 Parcely pro budovy

Definují prostor, který je možné buď dělit na další parcely nebo využít k „výstavbě“ modelů budov. Z důvodu chybného návrhu interakcí rozložení silnic se skutečná implementace liší od návrhu v kapitole 3.2.

Jako vstupní bod generování jsou určeny výchozí silnice (3.2.1). V jejich směru jsou na obou jejich stranách vytvořeny první hraniční body nových parcel. Další body jsou vytvořeny na první nejbližší křižovatce dané silnice (ovšem pouze pro parcelu na straně křižovatky). Algoritmus se v tuto chvíli zanoří o úroveň níže a stejným principem pokračuje v hledání dalších bodů dané parcely. Tento postup je ukončen v případě, kdy v požadovaném směru není nalezena další křižovatka (jako poslední bod hranic parcely je zvolen koncový

[9] https://www.codeproject.com/Articles/30535/quadtree

bod úsečky rozložení) nebo když je registrován pokus o znovu–vložení počátečního bodu hranic parcely (v tomto případě je již registrována smyčka a parcela je kompletní).

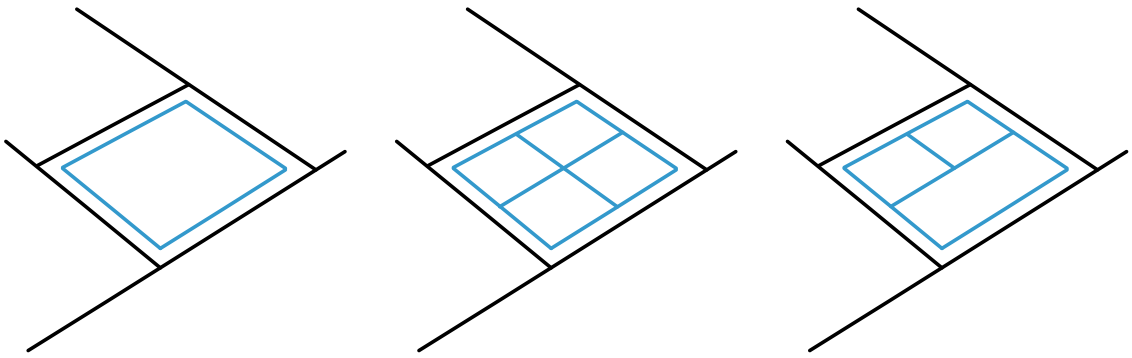
Po dokončení parcely algoritmus pokračuje vytvářením dalších parcel a v pokračování parcel nedokončených na původní silnici. Následně se postupně zanořuje i do podřízených silnic čímž postupně zpracuje všechny existující křižovatky a vytvoří všechny možné parcely.

V průběhu zpracovávání křižovatek si algoritmus vede seznam již zpracovaných křižovatek s informací pro kterou stranu byly použity. Tím je schopen zamezit vytváření parcel využívajících duplicitních křižovatek.

Dělení parcel

Rozdělení parcel na menší části dovoluje vytvoření zástavby s nastavitelnou hustotou. To dovoluje radikálně zvýšit počet vytvořených modelů v rámci daného města.

Dále se také provádí náhodné spojování vzniklých částí parcely. Díky této možnosti není vždy vynucen stejný poměr stran parcel — to přispívá k různorodosti staveb.



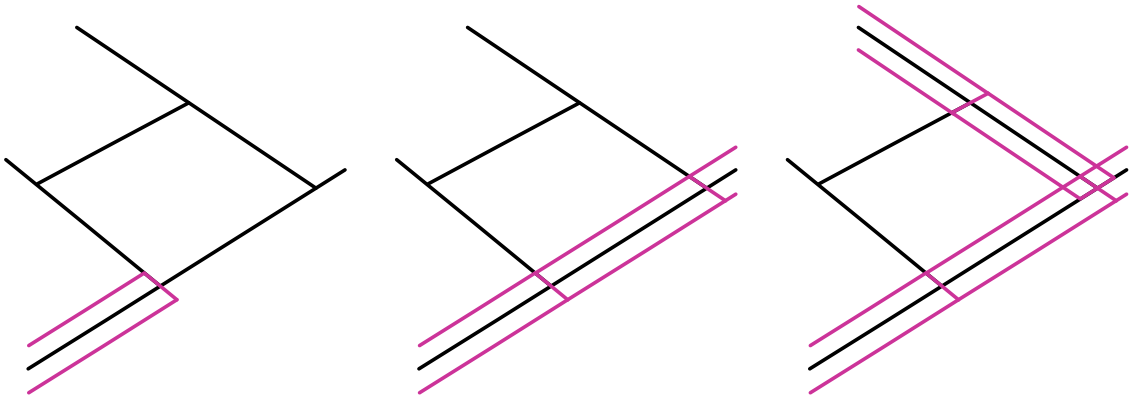
(a) Vstupní existující parcela (b) Rozdělení parcely na části (c) Náhodné spojení částí

Obrázek 4.8: Ilustrace dělení vytvořených parcel na menší části

Prvním krokem dělení je výběr existující parcely — obrázek 4.8(a). Obrázek 4.8(b) ilustruje rozdělení zvolené parcely na menší části. Posledním krokem je možnost náhodného spojení některých vytvořených částí parcely, tu je možné vidět na obrázku 4.8(c).

4.5.2 Parcely pro silnice

Tento typ parcel je složen pouze z několika bodů, které jsou vygenerovány po segmentech mezi dvojicemi křižovatek (popřípadě začátkem/koncem rozložení silnic) v nastavené vzdálenosti od úseček definující rozložení (4.4, 3.2). To umožňuje nastavovat jejich výslednou šířku mezi parcelami.



Obrázek 4.9: Ilustrace krokového generování parcel pro silnice

4.6 Budovy

Tento nástroj se zaměřuje na vizualizaci především novodobé metropole. Cílem je tedy vytvářet především moderní výškové budovy v hustější zástavbě.

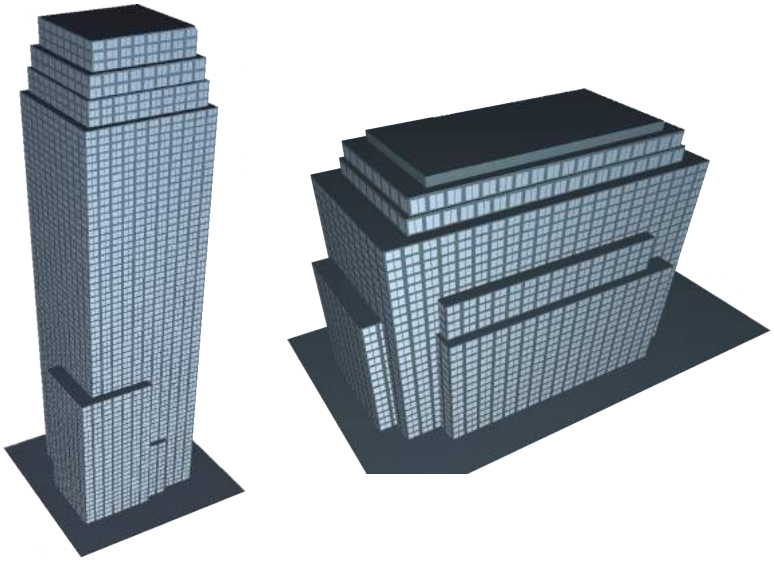
Generování využívá vytvořených parcel k určení prostoru, ve kterém bude model vytvořen. Dále také třídy `Infrastructure::Building` a `Infrastructure::BuildingPart`, které ukládají vytvořené modely. V neposlední řadě využívá generátor také třídy `Terrain::HeightMap` k práci s výškou terénu v místech parcely.

4.6.1 Generování tvaru

Generování modelů budov probíhá dle návrhu z kapitoly 3.4.1. Implementace však vytváří modely budov pouze z kvádrů, které se hodí právě pro modely moderních výškových budov. K úpravě výšky terénu nedochází — terén je vyrovnán „základovým blokem“, který svou základnou kopíruje terén avšak shora poskytuje modelu rovnou plochu.

Nejdříve jsou vytvořeny základní dvourozměrné reprezentace budoucích kvádrů. Ty jsou následně náhodně přesunuty v rámci hranicí parcel. Posledním krokem je duplikace podstavy a určení výšky.

Kombinací několika těchto bloků je možné dosáhnout poměrně vysoké vzhledové důvěryhodnosti výsledné budovy. Některé vytvořené modely budov jsou zobrazeny na obrázcích 4.10.

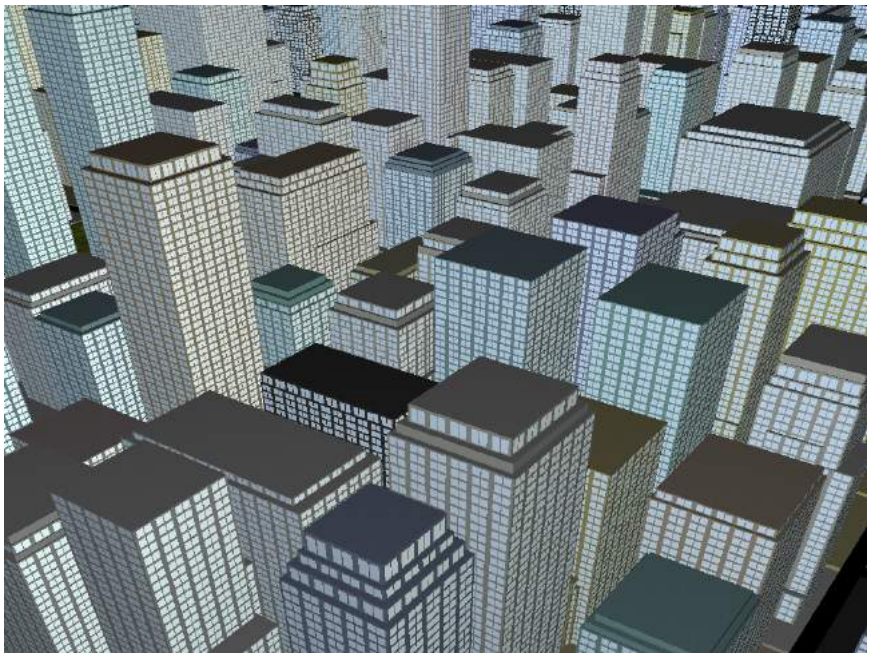


Obrázek 4.10: Vytvořené modely budov

Ukázka finálních budov vytvořených implementovanou technikou generování.

4.6.2 Náhodný barevný tón

Vygenerování náhodného barevného odstínu probíhá velice jednoduše. Nejdříve je zvoleno, pro které kanály a jaké hodnoty RGB spektra budou generovány — to neprobíhá náhodně, v programu jsou stanoveny tři různé odstíny, které je možné v různých intenzitách vygenerovat (oranžový, modrý a černý). Následně je náhodně vygenerováno číslo z předem zvolených intervalů. Výslednou implementaci je možné vidět na obrázku 4.11.



Obrázek 4.11: Barevné odstíny na budovách

Ukázka textur po aplikaci náhodně vygenerovaných barevných tónů pro jednotlivé budovy.

Intervaly RGB hodnot je nutné zvolit vhodně, protože při různých kombinacích náhodných čísel může dojít k vytvoření problematických barev (zvláštní odstín, příliš velká sytost aj.). Aplikace vytvořeného barevného odstínu je pak v GLSL extrémně stnadné: `diffuseColor *= v_textureTint;` (proměnná `diffuseColor` reprezentuje získanou barvu textury, `v_textureTint` obsahuje vygenerovaný odstín).

4.7 Textury

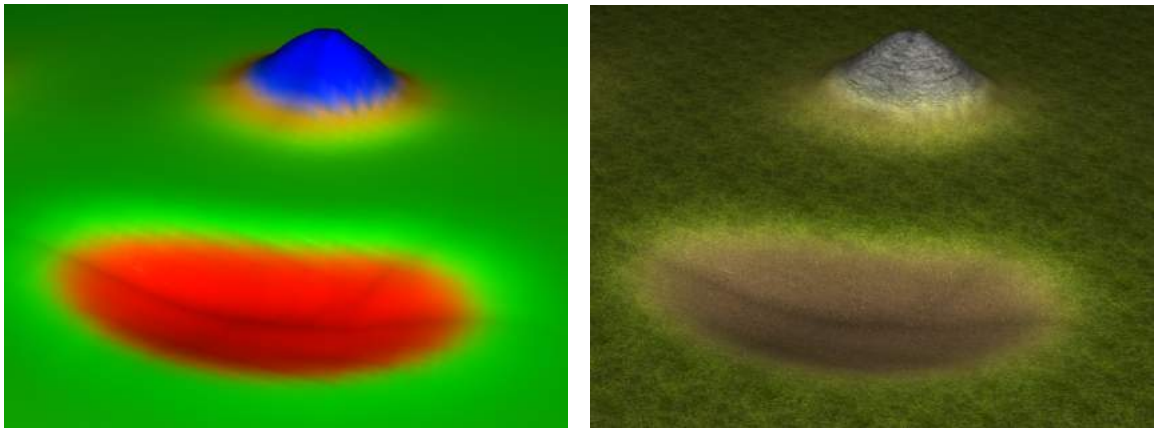
Implementovaný nástroj využívá mimo standardních textur i další, např. cubemap či specular. Jejich použití v programu bude popsáno dále v této kapitole.

Všechny textury použité v implementaci práce byly, kromě textur pro skybox, zakoupeny a staženy z obchodu Textures.com[10]. Textury pro skybox byly použity z balíčku textur, který je zdarma dostupný v obchodě Unity Asset Store[11].

4.7.1 Terén

Na terén jsou dle návrhu nanášeny celkem tři textury — hlína, tráva a kámen. Aplikace textur na povrch modelu terénu využívá principu triplanárního mapování (2.5.4) k zamezení chybného zobrazení.

Pro výpočet intenzit textur tato implementace využívá tří matematických rovnic, jejichž průběh hrubě kopíruje průběhy funkcí prezentované v kapitole 3.5.1, resp. na obrázku 3.8. Shader poté vypočítává odpovídající koordináty a provádí výsledné mapování textur na terén. Jednotlivé části terénu určeny odpovídajícím texturám mohou být vidět na obrázku 4.12(a). Výsledné nanesení textur je na obrázku 4.12(b).



(a) Části terénu určené odlišným texturám (b) Výsledné mapování textur

Obrázek 4.12: Vizualizace mapování textur na model terénu

Na obrázku (a) jsou zvýrazněny části terénu, které jsou určené jednotlivým texturám. Červená odpovídá textuře hlíny, zelená trávě a modrá textuře kamene. Další barvy vznikají kombinací výše zmíněných barev. Obrázek (b) prezentuje výsledek konečného mapování vybraných textur na zvýrazněná místa.

[10] https://www.textures.com
[11] https://assetstore.unity.com/packages/2d/103633

4.7.2 Budovy

Na povrch modelů budov jsou nanášeny dvě „podkladové“ textury — textura tmavšího betonu (nanášena na střechy, základy) a světlejší textura, která je určena pro stěny modelu. Tyto textury jsou na povrch nanášeny čistě prostřednictvím shaderů.

Budovy dále využívají spekulární textury k určení odrazivých míst na budovách. Tato textura je předem vygenerována a namapována na primitivy modelu. Při nanášení je možné určovat hustotu nanesené spekulární textury — hustota rozmístění oken na budovách.

Spekulární textury využívá shader k zobrazení odrazu skyboxu. Při pohledu na odrazivý prostor shora však dochází k problému — shader zobrazuje odraz skyboxu. Tento problém je možné vyřešit principem převzatým z triplanárního mapování textur. Za určitého úhlu odraženého paprsku dochází k zobrazení jiné barvy, případně odrazu z jiné cubemap textury.

Výsledné nanesení textur je možné vidět na obrázku 4.13.

Obrázek 4.13: Detail výsledného povrchu budovy

Na obrázku je vidět detail použité textury povrchu budovy, spolu s aplikovanou spekulární mapou a odrazem skyboxu. Na modelu budovy v pozadí lze vidět i aplikovaný černý odstín (4.6.2).

4.7.3 Skybox

Textura skyboxu je mapována na povrch krychle a je proto sestavena z šesti oddělených obrázků. Ty jsou v konkrétním pořadí načteny jednoduchým `for` cyklem ze souboru a nahrány na GPU. Vrcholy krychle jsou staticky uloženy v kódu aplikace (nedochází k jejich výpočtu) protože je není nutné měnit.

Při samotném vykreslování je nutné dočasně zakázat zápis do *depth bufferu* (hloubkový buffer snímku, na základě jeho obsahu GPU vyhodnocuje, zda budou jednotlivé primitivy vykreslovány). To způsobí, že krychle bude vždy zobrazena za všemi ostatními objekty na scéně. Posledním nutným krokem je úprava transformační matice — strukturu matice z knihovny glm lze snadno přetypovat na matici nižšího řádu, odstranit tak část prostorového posunu a zamezit tím tak „opuštění“ vnitřku krychle.

Výslednou implementaci lze vidět na panoramatickém snímku na obrázku 4.14

Obrázek 4.14: Ukázka použití skyboxu

Panoramatický snímek zobrazené textury nebe (skyboxu) složený z několika menších snímků.

Kapitola 5

Závěr

V rámci této práce byla úspěšně vytvořena implementace nástroje pro generování měst včetně okolního terénu, silnic i budov a náležitých detailů. Program také dále využívá několika textur pro zlepšení vzhledu vygenerovaných modelů i okolního prostředí. Při spuštění nástroje je možné specifikovat několik proměnných, které slouží k řízení algoritmů programu. Vygenerované modely terénu a budov je možné bez textur exportovat a použít ve všech oblíbených programech pro práci s 3D grafikou.

Export modelů provádí pouze export pozičních informací a normál objektů. Tento modul by mohl být dále vylepšen i optimalizován. I přes to, že samotný terén není přímo tématem této práce by mohla být zvýšena jeho realističnost — toho by bylo možné dosáhnout změnou implementace zpracování výsledky Perlinova šumu. Na konzistentní výšku terénu jsou ovšem vázány další implementační záležitosti. Dále by mohly být implementovány modifikační zóny zmíněné v kapitole návrhu. Tím by mohlo být dosaženo zajímavějších rozložení cest pro město. V neposlední řadě by bylo možné implementovat vylepšený generátor, který by uměl generovat více typů budov.

Díky této práci jsem měl možnost nahlédnout na počítačovou grafiku z úplně jiného úhlu, než doposud. Nejen, že jsem získal mnoho zkušeností s 3D grafikou, použitým programovacím jazykem, dalšími používanými technologiemi či postupy, dostalo se mi tím i unikátního pohledu na kombinaci všech těchto principů. Téma bych si vybral znovu, tentokrát bych ovšem kladl větší důraz na testování implementace a lepší počáteční návrh.

Literatura

[1] Biagioli, A.: Understanding Perlin Noise. 2014, [Online; navštíveno 13. října 2018]. URL https://flafla2.github.io/2014/08/09/perlinnoise.html

[2] Ebert, D. S.; Musgrave, F. K.; Peachey, D.; aj.: *Texturing and Modeling, Second Edition: A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 1998, ISBN 080166361X.

[3] Green, D.: *Procedural Content Generation for C++ Game Development*. Packt Publishing, 2016, ISBN 9781785886355.

[4] Johnson, N.: Nick's Blog. 2009, [Online; navštíveno 27. října 2018]. URL http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves

[5] Korn, O.; Lee, N.: *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*. Springer, 2017, ISBN 9783319530888.

[6] Lague, S.: Procedural Landmass Generation. 2017, [Online; navštíveno 12. října 2018]. URL https://github.com/SebLague/Procedural-Landmass-Generation

[7] Nevala, E.: Introduction to Octrees. 2014, [Online; navštíveno 29. října 2018]. URL https://www.gamedev.net/articles/programming/general-and-gameplay-programming/introduction-to-octrees-r3529/

[8] Owens, B.: Use Tri-Planar Texture Mapping for Better Terrain. 2014, [Online; navštíveno 6. května 2019]. URL https://gamedevelopment.tutsplus.com/articles/gamedev-13821

[9] Patel, A.: Making maps with noise functions. 2015, [Online; navštíveno 13. října 2018]. URL https://www.redblobgames.com/maps/terrain-from-noise/

[10] Polack, T.: *Focus On 3D Terrain Programming (Game Development)*. Course Technology PTR, 2002, ISBN 1592000282.

[11] Rost, R. J.; Licea-Kane, B.; Ginsburg, D.; aj.: *OpenGL Shading Language*. Addison-Wesley Professional, 2009, ISBN 9780321669223.

[12] Shaker, N.; Togelius, J.; Nelson, M. J.: *Procedural Content Generation in Games (Computational Synthesis and Creative Systems)*. Springer, 2016, ISBN 9783319427164.

[13] Stevens, J.: Lighting Models In Unity. 2016, [Online; navštíveno 29. září 2018]. URL https://www.jordanstevenstechart.com/lighting-models

Příloha A

Obsah CD

3rd-party/ Zdrojové kódy třetích stran.

bin/ Binární soubory aplikace a zkompilované knihovny třetích stran.

build/ Soubory projektu, podklady pro překlad.

cmake/ Konfigurační soubory pro CMake.

docs/ Složka obsahující krátké prezentační video a další materiály technické dokumentace.

- **thesis/** Zdrojové i zkompilované soubory technické dokumentace práce pro LaTeX.
- **thesis-plots/** Zdrojové soubory pro generování některých grafů a obrázků použitých v dokumentaci.

include/ Soubory externích knihoven.

lib/ Soubory externích knihoven.

output/ Složka pro soubory exportované implementovaným nástrojem.

res/ Složka s texturami a zdrojovými soubory používaných shaderů.

share/ Soubory externích knihoven.

src/proceduralCity/ Zdrojové kódy implementovaného nástroje.

Příloha B

Ovládání programu

Pohyb kamery

W , S , A , D	Pohyb kamery dopředu, dozadu, doprava a doleva
Spacebar , C	Pohyb kamery nahoru, dolu
+ SHIFT	Zrychlení pohybu

Generování objektů

X	Generování rozložení silnic
P	Vyhledání chybně vygenerovaných rozložení cest
O + P	Odstranění nalezených chybně vygenerovaných rozložení cest
K	Vytvoření všech možných parcel
H	Odstranění všech vygenerovaných parcel
J	Čištění nevhodných parcel
B	Vygenerování modelu jedné budovy
SHIFT + B	Vygenerování modelu jedné budovy každý snímek
O + B	Odstranění všech modelů budov
N	Vygenerování modelu jednoho segmentu silnice
SHIFT + N	Vygenerování modelu jednoho segmentu silnice každý snímek
O + N	Odstranění všech modelů silnic

Další

CTRL + T	Přepnutí režimu celé obrazovky
R	Přepnutí aktivního shaderu
I	Spuštění/Ukončení záznamu
F	Nový klíčový bod pro automatický pohyb kamery
O + F	Odstranění všech klíčových bodů pro automatický pohyb kamery
G	Spuštění automatického pohybu kamery po přednastavené cestě
CTRL + G	Spuštění automatického pohybu kamery se záznamem
(CTRL +) + + F	(Rychlé) Zvýšení délky záznamu
(CTRL +) − + F	(Rychlé) Snížení délky záznamu