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 GENEROVÁNÍ MĚST

PROCEDURAL GENERATION OF CITIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **KRISTIÁN BÍLÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ STARKA,**
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: **Bílý Kristián**
Program: Informační technologie
Název: **Procedurální generování měst**
Procedural Generation of Cities
Kategorie: Počítačová grafika

Zadání:

1. Nastudujte prodecurální generování, zaměřte se na generování měst. Nastudujte potřebné knihovny a nástroje.
2. Navrhněte procedurální generátor města.
3. Implementujte procedurální generátor města.
4. Vytvořte plakát nebo video demonstrující práci.

Literatura:

- Po dohodě s vedoucím

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

Vedoucí práce: **Starka Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 9. května 2019

Abstrakt

Bakalářská práce se zabývá procedurálním generováním měst. Jejím cílem je navrhnout, implementovat a otestovat aplikaci schopnou vygenerovat město v uživateli zvoleném terénu. Práce využivá L-Systémy pro generování sítě cest. Klade důraz na pestrost modelů budov. Výsledná aplikace je schopna generovat město a patřičně reagovat na uživatelské vstupy, jenž je základní struktura města a terén.

Abstract

Bachelor thesis is about procedural city generation. It's goal is to design, implement and test aplication, that is capable of city generation in user's terrain. The work uses L-Systems for generating road network. It also focuses on variation of building models. Final aplication is capable of city generation. Aplication properly reacts to users input. Users input is basic city structure and terrain.

Klíčová slova

procedurální generování, procedurální generování měst, procedurální generování budov, C++, OpenGL, L-Systémy

Keywords

procedural generation, procedural city generation, procedural building generation, C++, OpenGL, L-Systems

Citace

BÍLÝ, Kristián. *Procedurální generování měst.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Starka,

Procedurální generování měst

Prohlášení

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

.....
Kristián Bílý
14. května 2019

Poděkování

Rád bych poděkoval panu Ing. Tomašovi Starkovi za vedení a odbornou pomoc při vzniku této práce.

Obsah

Kapitola 1

Úvod

Procedurální generování nachazí čím dál větší uplatnění v herním a filmovém průmyslu. Je to způsobeno hlavně zlepšujícím se hardwarem. Ten totiž umožňuje vytvářet lepší a detailnější modely či efekty. To ale zapříčiňuje zvyšující se cenu těchto projektů. Dobrý procedurální generátor je schopen vygenerovat výsledek, dle potřeb uživatele, a tak podstatně snížit cenu projektů.

Cílem této bakalářské práce je procedurálně vygenerovat město. To by mělo byt co nejpestřejší, ale zároveň by si mělo udržet určitou reálnost. Cílem je tedy naimplementovat generátor, který bude schopný procedurálně generovat město. V reálném životě byla stavba měst ovlivněna terénem, ať už šlo o hory, nebo řeky. Sítě pozemních komunací, ačkoliv se drží různých vzorů, nebývají stejné. Architektura budov ve větších mestěch bývá velmi různorodá. Generátor by tedy měl být schopen pracovat, do jisté míry, s terénem. Sítě cest by měly tvořit náhodné vzory. Důležitá je také schopnost generátoru vygenerovat dostatečnou pestrost modelů budov. Není tedy žádoucí, aby se ve městě opakovaly identické modely.

Kapitola 2

Teorie

Tato kapitola nabízí teoretické znalosti, které byly využity při tvorbě generátoru. Stěžejní znalosti pro tuto bakalářskou práci jsou především informace o generování náhodných čísel (sekce 2.1) a L-systémech (sekce 2.3).

2.1 Generování náhodných čísel

Generování náhodných čísel je využito v různých oblastech, ať už jde o informatiku, matematiku, či fyziku. Generátor náhodných čísel může být buď výpočetní, nebo je fyzické zařízení.

V případě fyzického zařízení, náhodných čisel lze dosáhnout například měřením šumu. Bylo také sestrojeno zařízení, které pořizovalo snímky lávové lampy, zaručující „skutečnou náhodu“.

Výpočetní generátor získává náhodná čísla pomocí početních úkonů. V tomto případě už nejde o nepředvídatelná náhodná čísla, ale čísla s určitou pravidelností. Jde tedy jen o dojem náhodnosti. Tato čísla se nazývají pseudonáhodná čísla.

2.1.1 Kongruentní generátor

Lineární kongruentní generátor je jeden z nejpoužívanějších výpočetních generátorů. Pierre L'Ecuyer napsal práci [7], ve které je generátor popsán. Generuje tedy pseudonáhodná čísla. Je popsán rovnící:

$$x_{i+1} = (ax_i + c) \, mod \, m \tag{2.1}$$

kde a, c a m jsou zvolené konstanty. Počáteční x_0 se nazývá „náhodné semínko“ (anglicky „random seed“). Výsledná náhodná čísla jsou v intervalu $0 < x_i < m$. Po m náhodných číslech se čísla začnou opakovat, dosáhlo se tedy periody generátoru. Plnou periodu generátor nabyde, když:

- c a m jsou nesoudělná čísla, to znamená, že mají pouze jednoho společného dělitele, tím je číslo 1
- $a - 1$ je dělitelné všemi prvočíselnými faktory m
- $a - 1$ je násobek 4, pokud m je násobek 4

2.1.2 Xorshift

Xorshift [9] je třída jednoduchých náhodných generátorů. Jde o výpočetní generátory. Patří pod skupinu LFSR (anglická zkratka „Linear Feedback Shift Register“), využívající bitové operace. Čísla získává opakovaným použitím operace exkluzivní disjunkce (jinak „exkluzivní or“ či „XOR“). Operaci XOR používá na svojí bitově posunutou verzi.

Xorshift patří mezi nejrychlejší výpočetní generátory. Běžná rychlost generátorů je asi 200 miliónů náhodných čísel za sekundu. Použití tří Xorshift operací při jednom volání nabídne až $2^{128} - 1$ náhodných 32-bitových čísel. Potřeba je jen čtyř konstant x, y, z, w. Perioda pro Xorshift je $2^k - 1$ kde $k = 32, 64, 96, 128, 160, 192$.

Xorshift vymyslel George Marsaglia. Napsal také článek *Xorshift RNGs* [9], ze kterého byly čerpány informace pro popis výše.

2.1.3 Mersenne twister

Mersenne twister [10] patří mezi jeden z nejpoužívanějších generátorů pseudonáhodných čísel. S tímto generátorem přišli Makoto Matsumoto a Takuji Nishimura. Generátor vznikl především jako řešení nedostatků starších výpočetních generátorů.

Už podle názvu „Mersenne twister“ je zřejmé, že má spojitost s Mersennovým prvočíslem. Spojitost s tímto prvočíslem má právě perioda generátoru. Ta nabývá hodnoty $2^{19937} - 1$. Standardní implementace Mersenne twisteru, MT19937, generuje 32-bitová čísla. Existuje také verze implementace, která generuje 64-bitová čísla. Ta je nazvána MT19937-64.

Generátor má své výhody, ale i nevýhody. Výhodou je velká perioda ($2^{19937} - 1$). Také prošel různými testy, jako například *Diehard tests*, nebo většinu testů *TestU01*. Nevýhodou je zatížení RAM, potřebuje totiž kolem 2,5kB. Také v případě, že se uloží sekvence 624 vygenerovaných čísel, lzee pak předpovědět všechna ostatní čísla.

2.2 Procedurální generování

Se zlepšujícím se hardwarem se rozšiřují i možnosti v mnoha oblastech, ať už jde o herní průmysl, filmy nebo grafická dema. Nese to s sebou i značnou nevýhodu. Větší světy ve hrách, nebo lepší efekty ve filmech či demech, stojí více času designerů, a to znamená i více peněz. Procedurální generovaní toto dokáže, do jisté míry, řešit.

Procedurální generování generuje data automaticky. Není tak potřeba trávit čas manuální tvorbou. Tím se ušetří čas i peníze při realizaci projektu. Obrovskou výhodou procedurálního generování je fakt, že se dá ve výše uvedených oblastech vygenerovat skoro cokoliv. Ať už to jsou mapy, vegetace, 3D modely, efekty, zvuky či textury. Může se do generování zavést náhodnost, ale také nemusí. Při tvorbě filmu není potřebné, aby efekt vypadal vždy jinak. Ale při generování mapy pro hru, jde o velmi užitečnou vlastnost. Hraje-li hráč pokaždé na jiné mapě, má tak ze hry pokaždé trochu jiný zážitek.

Procedurální generování se využívá už mnoho let. V herním průmyslu je velmi oblíbené. Celá série hry Civilization využívá procedurální generování pro tvobu mapy. V sérii Diablo je generování využito jak pro mapu, tak i pro atributy většiny předmětů. Dalším příkladem využití procedurálního generování je filmová trilogie Pán Prstenů. V této trilogii se generovaly bitvy obrovských armád.

2.3 L-systémy

L-systémy byly vyvinuty biologem Aristidem Lindenmayerem. Původně sloužily pro modelování růstu rostlin. Postupně se vyvinuly na složitější rostlinné a jiné fraktální struktury. Lindenmayer také sepsal práci na toto téma. Kniha se jmenuje *The algorithmic beauty of plants* [8] a z ní bylo čerpáno pro definici L-systémů a to včetně přepisovacího, parametrického i stochastického.

2.3.1 Přepisovací L-systémy

Základním typem L-systému je D0L-systém. Jde o deterministický bezkontextový L-systém. Definovat ho lze:

Definice 1 *L-systém G je trojice $G = (V, \omega, P)$ kde*

- *V je abeceda, tedy neprázdná množina symbolů. Každý symbol abecedy je obvykle reprezentován písmenem.*
- *$\omega \in V^+$ je axiom, definující počáteční stav L-systému*
- *$P \subset V \times V^*$ je konečná množina pravidel. Přepisovací pravidlo se zapisuje ve tvaru $a \to \omega$, kde $a \in V$ a $\omega \in V^*$, definuje tedy přepisování symbolů*

L-systém se tvoří pomocí iterací. V každé iteraci se paralelně přepíší všechny symboly aplikováním pravidel.

Příkladem L-systému může být třeba Fibonacciho posloupnost.

2.3.2 Želví grafika

Pod pojmem „želví grafika“ si lze představit želvu, která při chůzi zanechává stopu. Každý symbol L-systému je povel pro želvu. Například symbol F může pro želvu znamenat krok vpřed, zatímco smybol + otočení o určitý úhel. Aby bylo možné určit polohu a směr, kterým se má želva vydat, lze želvu vyjádřit jako trojici (x, y, α), kde x a y jsou souřadnice a α je úhel, na který je želva otočená.

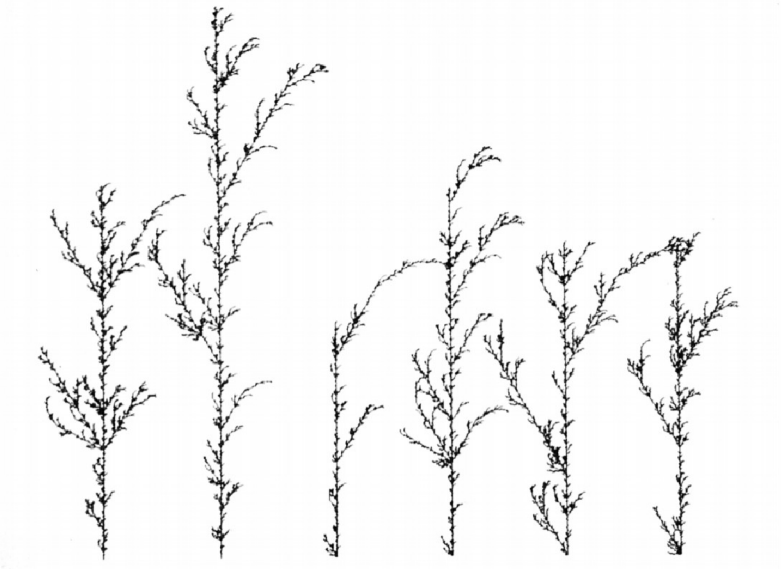
2.3.3 Stochastické L-systémy

Přepisovací L-systémy jsou deterministické. Z toho vyplývá, že n-tá iterace bude mít vždy stejný výsledek. V mnoha případech je ale žádoucí, aby byl výsledek proměnlivý. Toho se dá dosáhnout přidáním určité náhody. Tak lze díky jednomu L-systému vygenerovat pokaždé jinou strukturu (například strom, nebo v případě procedurálního generování města, budovu).

Jednodušší variantou je přidat náhodu do interpretace. Tím zůstane systém stejný, pouze se mění interpretace příkazů pro pomyslnou želvu. Nahodně se tak dá určovat délka kroku želvy, či změna jejího otočení.

Druhou metodou je porušení determinismu. Každý symbol už proto nemá právě jedno pravidlo. Pravidlům, u kterých se má projevit náhoda, je přidána váha. Váha určuje šanci, jakou bude pravidlo vybráno.

Ukázka stochastického L-systému je na obrázku 2.1. Je na něm ukázáno šest rostlin vygenerovaných za použití stochastismu. Rostliny vypadají jako různé vzorky stejného druhu.



Obrázek 2.1: Na obrázku je šest rostlin vygenerovaných pomocí stochastického L-systému. Obrázek je převzaný z *The algorithmic beauty of plants* [8].

2.3.4 Parametrické L-systémy

L-systém, jehož symboly jsou rozšířené o parametry, se nazývá Parametrický L-systém. Parametr bývá většinou reálné číslo, ale může to být i aritmetický výraz, proměnná nebo dokonce funkce. Hodnota parametru je vyhodnocena ve chvíli, kdy je potřeba. Symbol může obsahovat konečný počet těchto parametrů. Parametry se zapisují do kulatých závorek za symbol, přičemž parametry se nepovažují za symboly L-systému.

Příklad lze uvést s pomyslnou želvou zanechávající stopu. Želva se může pohnout jen o přesně danou vzdálenost nebo se otočit jen o přesně daný úhel. V parametrickém L-systému lze vzdálenost nebo úhel specifikovat za běhu, právě pomocí parametrů. Další příklad je fraktál H-strom, jenž lze vidět na obrázku 2.2 nebo třeba list růže, který je na obrázku 2.3.

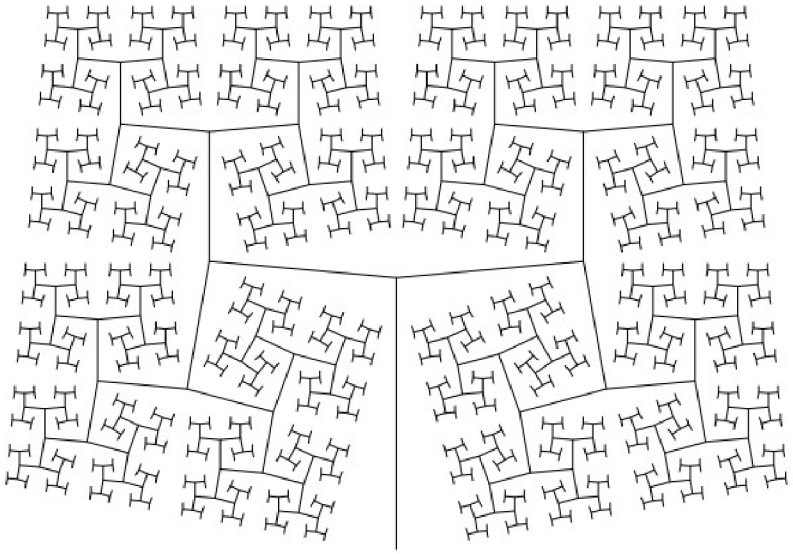
2.4 Existující řešení

Kapitola ukazuje již existující řešení s obdobnou tématikou této práci. Jelikož procedurální generování měst není nic nového, obdobných řešení existuje celá řada.

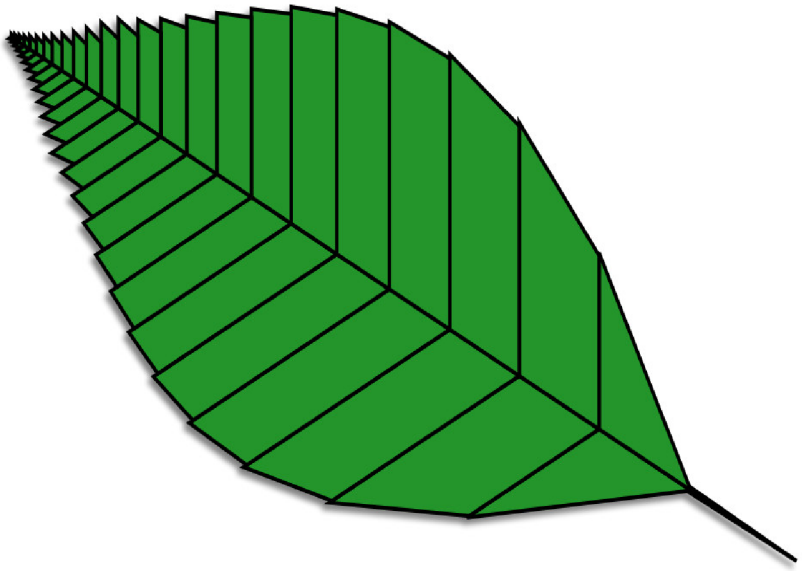
Undiscovered Worlds [4]

Jak se neustále zlepšuje hardware, je možné mít počítačové hry se stále většímy světy. Tím ale roste i cena a čas pro jejich zrealizování. Tento generátor vznikl jako snaha snížit čas i cenu tím, že svět vygeneruje procedurálně.

Je schopný vygenerovat město obrovských rozměrů. Cesty tvoří klasický mřížkový vzor. Modely budov připomínají mrakodrapy. Pro tvoření půdorysů budov je použita metoda, při které se polygony skládájí přes sebe [11]. Ve výsledném městě může být až 4×10^{18} unikátních budov. Architektura města je inspirována městem Melbourne, což lze vidět na obrázku 2.4.



Obrázek 2.2: Na obrázku je H-strom, který lze vytvořit paramerickým L-systémem. Obrázek je převzaný z *The algorithmic beauty of plants* [8].



Obrázek 2.3: Na obrázku je list růže, který lze vytvořit parametrickým L-systémem. Obrázek je převzaný z *The algorithmic beauty of plants* [8].

CityEngine [12]

Praci zhotovili Parish a Müller. Díky uživatelským vstupním datům je generátor schopný vygenerovat město na míru. Například při správných datech může být výsledkem úctihodná kopie Manhattanu.

Generátor obratně pracuje s výškovou mapou, mapou vody i mapou hustoty zalidnění. Dokáže pracovat s různými vzory pro generování cest. To vše je uživatelský vstup.

Obrázek 2.4: Ukázka z Undiscovored Worlds [4].

Pro generování cest je využito parametrického L-systému. Systém je definován devíti pravidly. Budovy jsou generovány pomoci stochastických parametrických L-systémů. Jsou definovány tři typy budov, a to mrakodrapy, komerční budovy a residenční boduvy. Textury budov jsou také generovány procedurálně.

Práce Yoava I H Parish a Pascal Můllera se stala inspirací pro tuto bakalářskou práci. Ukázku výsledného města lze videt na obrázku 2.5

Obrázek 2.5: Ukázka ze CityEngine [12].

CityGen [15]

Práce vznikla jako snaha ušetřit čas a prostředky vývojářum, především v herním průmyslu. S prací přišli George Kelly a Hugh McCabe. Je značně ispirována prací Parishe a Müllera[12].

Stejně jako práce, jenž byl CityGen ispirován, generování probíha za pomocí L-systémů. Generování probíha ve třech fázích, generování cest, generování bloků a generování budov.

Rozdílem výše zmiňované práce a generátorem CityGen je ten, že CityGen nepracuje s geografickými daty poskytnutými uživatelem. Jako vstupní parametr pro generování je množina bodů. Tyto body určují křižovatky, které budou spojovat hlavní cesty města. Mezi hlavními cestami se vytvoří menší cesty (ulice), které pak vyplní město.

CityBuilder [6]

Generátor je prací Thomase Lechnera, Bena Watsona a dalších. Generátor nemá za cíl napodobení již existujících měst, ale vygenerovat náhodná města různých kultur, která jsou reálným velmi podobná. Podobnost reálným městům zaručuje především zaměření se na postupný růst města. Klade také velký důraz na prostředí, ve kterém se město generuje.

Protože generátor klade velký důraz na prostředí (tedy terén), nejdůležitějším vstupním paramterem je právě terén. Je však použito i několik dalších vstupních parametrů pro definování města. Jsou připravovýny i parametry zahrnující kulturu, kterou je město ovlivněno, či doba, do které je město zasazeno.

Není zde využito L-systémů. Využívá sadu agentů pro generování sítě cest i budov. Implementovány jsou dva typy budov (rezidenční a komerční), ačkoliv jsou připravovány další.

Kapitola 3

Návrh

Tato kapitola v první řadě definuje cíle a zaměření projektu. V druhé řadě pojednává o návrhu procesu generování, včetně všech jeho částí.

3.1 Definice cílů

Město je velmi rozsáhlý pojem. Existuje spousta dílčích pojmů, které město definují, nebo s městem silně souvisí. Například terén odjakživa ovlivňoval města v nejrůznějších aspektech. Ať už to bylo umístění města, kdy se města stavěla u řek nebo u hor. Struktura města je terénem také ovlivněna, protože bylo vždy jednodušší, aby cesty vedly okolo hor, než přes ně. Historický růst měst definoval strukturu pozemních komunikací spoustě měst. Kultura zase značně ovlivňuje především architekturu města. Dokonce ekonomická stránka ovlivňuje město. Bohatší města mají více možností jak se rozrůstat. Je mnoho dalších aspektů, které města definují, nebo do jisté míry ovlivňují.

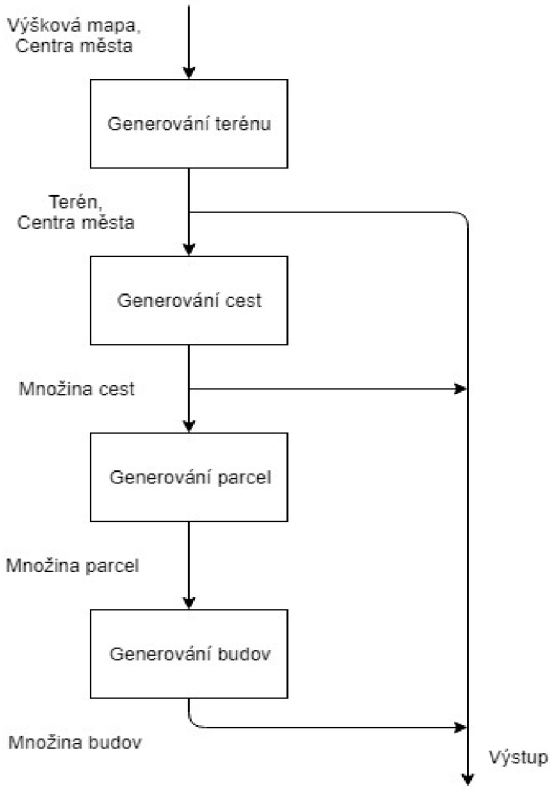
Pokud by generátor měst měl být schopen detailně řešit všechny výše uvedené aspekty, stal by se z něj obrovský projekt. Zatím neexistuje práce, která by toto svedla. Všechny dosavadní existující generátory se vždy detailně zaměřují pouze na několik určitých aspektů a zbylé buď to ignorují, nebo řeší jen zjednodušeně. Některé zasazují města do co nejvíce reálné podoby terénu. Jiné zase řeší především historický růst měst. Pak jsou tu takové, co mají detailně implementované generování budov. V sekci 2.4 je popsaných několik již existujících generátorů. Každý z nich je zaměřen na odlišné aspekty. Například *CityBuildier* [6] je zaměřen hlavně na historický růst města a reakce pozemní komunikace na terén. *CityEngine* [12] je práce rozsáhlejší, řeší spoustu aspektů poměrně detailně. Ovšem nebere v potaz například historický růst či kulturu města.

Je tedy velmi důležité, se zamyslet a definovat, jakých cílů by vlastně generátor měl dosáhnout. Generátor, jenž tato bakalářská práce popisuje, generuje město tak, aby dbal na náhodnost a rozmanitost, ale udržel si i určitou úroveň reálnosti. Generátor využívá zjednodušený terén (sekce 3.3) což městu dodává jistou reálnost. Generuje jeden reálný vzor cest, jenž je nejnáhodnější existující vzor (vzory jsou popsány níže, sekce 3.4). Na tvorbu parcel (sekce 3.5) využívá jednodušší ověřené algoritmy, ale přesto jsou parcely dostatečně podobné realitě. Na budovy (sekce 3.6) se klade největší důraz, jak v rozmanitosti, náhodnosti tak i reálnosti. Důležitou složkou je i interakce s uživatelem, aby se město do jisté míry podobalo jeho představám.

3.2 Architektura

Celý proces generování terénu a města lze rozdělit do čtyř fází. První fází je generování terénu. Druhou je generování pozemní komunikace. Třetí tvoření parcel. Ve čtvrté fázi se pak generují budovy. Každá fáze ja závislá na té předchozí. Diagram fází lze vidět na obrázku 3.1. Na obrázku lze vidět hierarchické propojení jednotlivých fází, jejich vstupy a výstupy.

Dělení procesu generování do fází je poměrně bežná věc. Všechny popsané, již existující generátory ze sekce 2.4, generují město ve fázích. Fáze však nemusí být vždy stejné, ať už do počtu, nebo do jejich obsahu.



Obrázek 3.1: Obrázek ukazuje fáze generování města. Popisuje také vstupy a výstupy každé fáze i celkového generování.

3.3 Terén

Jak už bylo okrajově zmíněno výše, terén měl na města vždy velký vliv. V minulosti ovlivňoval jak umístění města, tak i jeho strukturu. Města vybudovaná u řek měla výhodu v transportu a obchodu. Města vybudovaná u kopce či hory měla strategickou výhodu při obraně. Příkladem mohou být velká Evropská města. Paříž vybudovaná u řeky Seiny, Praha u řeky Vltavy nebo Londýn u řeky Temže.

Pokud je tedy terén součástí generátoru, je potřeba získat výškovou mapu, aby se terén mohl vygenerovat. Tady je opět potřeba se rozhodnout, zda-li se bude výšková mapa terénu procedurálně generovat, nebo importovat. Pro vygenerování výškové mapy se dá využít

šum. Jednou z vhodných variant je *Perlinův šum* [5]. Je rychlý, potřebuje málo paměti a poskytuje poměrně dobrý výstup. Pracuje s n-dimenzionálními poli. Hodnoty každé buňky pole jsou získány interpolací hodnot okolních buněk a jejich vzdálenosti. Každá buňka obsahuje vektor, který právě ovlivňuje okolní buňky. Každá buňka je pak pixel výškové mapy. Další variantou je například *algoritmus Diamond Square* [3].

Pokud je vhodnější, aby si uživatel volil terén sám, imporotvání výškové mapy je ideální. *CityEngine* [12] pracuje s reálnými výškovými mapami. Jednodušší variantou je importování šedotónového obrázku. Hodnota pixelu obrázku je pak hodnota ve výškové mapě na stejné pozici.

Tato bakalářská práce není primárně zaměřená na práci s terénem, avšak bere v potaz důležitost terénu. Generátor tedy pracuje se zjednodušenou verzí. Terén má pouze tři výšky. První výška představuje **vodní plochu**, druhá pak **rovinu** a třetí **hory**. Město se generuje ve výšce **rovina** a je omezováno ostatními výškami. Touto realizací je terén podstatně zjednodušen, zároveň si výsledné město udržuje určitou úroveň reálnosti. Pro vytvoření terénu využívá šedotónový obrázek jako výškovou mapu.

3.4 Pozemní komunikace

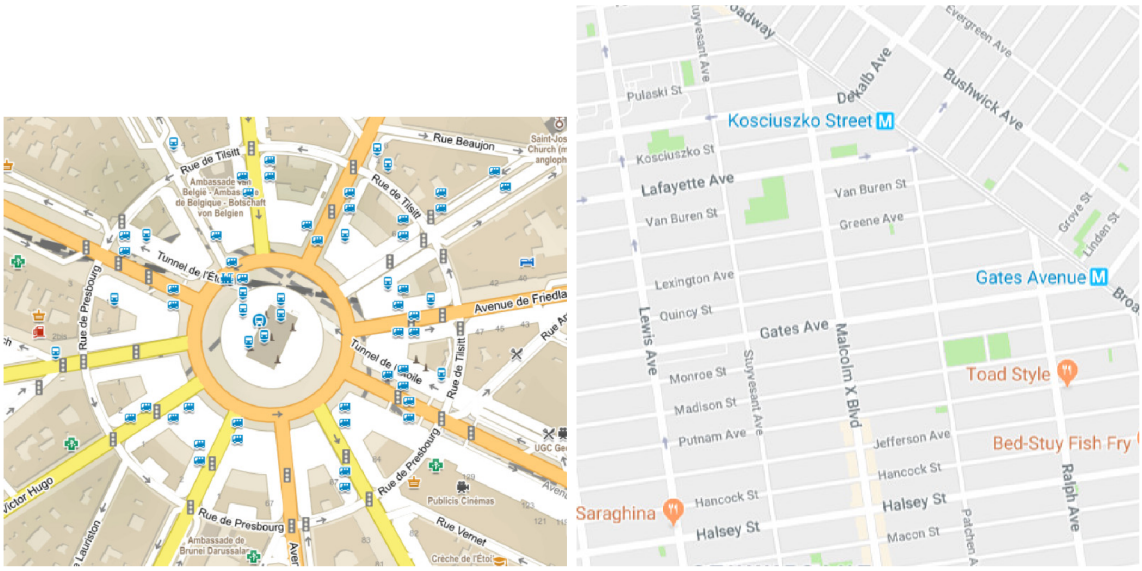
Cesty jsou velmi důležitou složkou města. Ovlivňují mnoho aspektů, především však jeho strukturu. V realitě jsou cesty stavěny v určitých vzorech. Pozorováním skutečných měst si lze všimnout, že vzory se často střídají i v rámci jednoho města. To je způsobeno historickým růstem měst. Tato problematika je popsána Parishem a Müllerem [12]. Tito pánové shrnuli veškeré vzory cest do čtyř typů:

- **Klasický vzor** - Toto je nejzákladnější vzor. Neobsahuje žádná pravidla, která by přesně definovala jeho vzhled. Vyskytuje se především ve starších částech měst. Tento vzor lze najít například v Praze.
- **Obdelníkový vzor** - V tomto vzoru cesty tvoří pravidelné obdelníkové bloky. Cesty tedy navzájem svírají pravý úhel. Tento vzor se vyskytuje převážně v nových městských částech, nebo v moderních velkoměstech, například v New Yorku.
- **Kruhový vzor** - Cesty jsou uspořádány do soustředných kružnic. Tento vzor lze zahlédnout v historických částech měst, nebo okolo významných míst, například náměstí. Vzor se vyskytuje například v Paříži.
- **Vzor San Francisco** - Tento vzor je použit převážně v městech, kde jsou velké rozdíly v nadmořské výšce. Cesty se táhnou tak, aby neměly příliš velké stoupání nebo klesání. Cesty, s velkým rozdílem v nadmořské výšce, jsou spojeny menšími cestami. Jak už název napovídá, tento vzor se vyskytuje například v San Franciscu.

Na obrázku 3.2 lze vidět ukázku **kruhového vzoru** nalevo a **obdelníkového vzoru** napravo.

Ačkoli vzor použitý v této bakalářské práci připomíná **vzor klasický**, cesty jdou generovány pravoúhle. Některé části města pak postrádají jakoukoli pravidelnost, jiné mají cesty tvořící pravoúhlé polygony. Není to však dané definováním vzorů v určitých oblastech, ale napojováním a lámaním cest. Typy cest jsou popsány níže.

Ukázka, jak může vypadat síť cest, lze vidět na obrázku 3.3. Je na něm zřetelně vidět, jak určité části postrádají jakoukoliv pravidelnost, zatímco v jiných tvoří cesty pravoúhlé polygony.



Obrázek 3.2: Obrázek nalevo ukazuje **Kruhový vzor** cest. Obrázek napravo ukazuje **Obdelníkový vzor**. Obrázky jsou převzány z [*Google maps*].

Obrázek 3.3: Na obrázku je ukázka vygenerované pozemní komunikace, která má **vzor klasický**, nicméně některé části jsou pravoúhlé.

Centra

Práce Parishe a Müllera [12] popisuje i hustotu zalidnění. Hustota zalidnění zásadně ovlivňuje strukturu města. V oblastech s velkou hustotou jsou cesty frekventovanější a je větší tendence stavět větší budovy. V oblastech s nízkou hustotou je naopak hustota cest velmi malá a budovy jsou spíše nízké. V oblastech, kde není žádná populační hustota, se ulice vůbec negenerují.

Hustota zalidnění je reprezentována jako množina center zalidnění. Centrum zalidnění je kružnice s určitou váhou. Se zvyšující se vzdáleností od středu kružnice se váha snižuje. Váha slouží pro navádění cest, které se snaží stáčet k nejvyšší váze hustoty v blízkosti. Cesty pak jednotlivé centra spojují a v jejích dosahu generují ulice a budovy.

Centra v této bakalářské práci jsou podobná, mají však dvě zásadní odlišnosti. V první řadě centra nemají váhy. Jejich rádius slouží pouze jako hranice pro některé typy cest. Druhá zásadní odlišnost je, že není jeden typ center, ale jsou dva.

- **Primární centrum** - Centrum, ze kterého se začínají generovat cesty. Existuje právě jedno primární centrum. Pro vygenerování města je nutné, aby primární centrum byl zadáno.
- **Sekundární centrum** - Centra, jenž jsou napojena k **primárnímu centru** pomocí pozemní komunikace. Existují především pro definování tvaru města. Nemusí být zadané žádné, v tom případě pozemní komunikace nemá co spojovat a tak jsou cesty v **primárním centru** natočeny náhodně.

Typy cest

V reálných městech existuje mnoho typů cest - dálnice, rychlostní cesty, silnice, ulice, polní cesty a další. Liší se ve více aspektech, nicméně nejvíce relativní odlišností pro tuto bakalářskou práci je, jejich velikost. Podle Müllera a Parishe [12] lze však zobecnit všechny tyto typy cest do dvou typů - hlavní a vedlejší cesty. Hlavní cesty spojují centra zalidnění a drží se vzorů cest. Vedlejší cesty pak vyplňují místo vzniklé mezi hlavními cestami a vzorů se nedrží.

V bakalářské práci se velikosti cest nerozlišují. Jsou však zavedeny tři typy cest. Každý typ má jinou úlohu a společně tvoří strukturu města. Typy se generují postupně.

- **Hlavní cesty** - Hlavní cesty slouží pro spojení center a jsou to jediné cesty, které vedou i ven z města. Je to první typ cest, který se generuje. Začínají v primárním centru a generují se ve směru sekundárních center. Vedou přímo k sekundárním centrům. Jakmile centrem projdou, vygenerují jednu odbočku a začnou se náhodně natáčet.
- **Silnice** - Silnice jsou druhé v pořadí při generování pozemní komunikace. Jsou generovány z hlavních cest a nachází se pouze v dosahu center. Generují se rovně, ale občas se lomí o určitý úhel. Neumí generovat odbočky.
- **Ulice** - Posledním typem jsou ulice. Ty vyplňují většinu města. Generují se pouze ze silnic a také se mohou nacházet pouze v dosahu center. Vedou pouze rovně. Odbočky se generují velmi často.

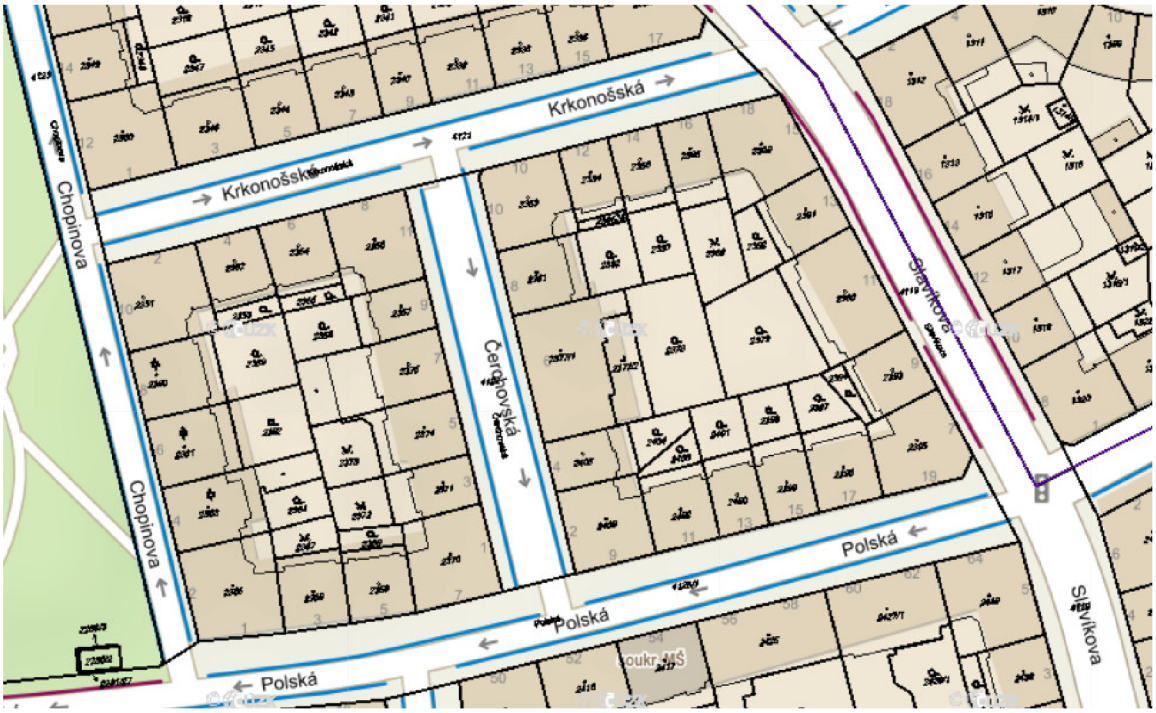
3.5 Parcely

Další fází je tvoření míst pro umístění budov. Již vytvořená pozemní komunikace tvoří množinu bloků. Jako blok si lze představit polygon, jehož hrany tvoří právě cesty. Bloky ale bývají často příliš velké na to, aby se do něj vygenerovala pouze jedna budova. Proto se musí blok rozdělit na menší územní celky, tedy parcely. V této etapě tedy probíhá tvoření bloků a následné jejich rozdělení na parcely.

3.5.1 Rozdělení města

Reálná města jsou dělená do bloků. Bloky jsou pak dále děleny na parcely. Pro představu se lze podívat na obrázek 3.4. Obrázek je převzat z katastrálních map. Bílé tlusté čáry značí cesty. Lze si všimnout, že cesty tvoří výše zmíněné polygony, tedy bloky. Ty jsou vybarveny hnědou barvou. Dále se blok dělí na výše zmíněné parcely, které jsou ohraničeny černými čarami.

Na obrázku 3.4 si taky lze všimnout, že ne všechny parcely mají přístup k cestě. Ty jsou pak často přikupovány k jiným parcelám, nebo se musí řešit přístup k cestě přes jiné parcely. V této bakalářské práci se však parcely bez přístupu k cestě neřeší a zůstavají prázdné.



Obrázek 3.4: Ukázka parcel z katastrálních map. Tlustá bílá čára znázorňuje cesty. Blok je vybarven hnědou barvou, parcely jsou pak ohraničeny černě.

3.5.2 Tvoření parcel

Jak už bylo zmíněno, prvním krokem pro tvoření parcel je získání bloků. Bloky se pak dělí na parcely. Tento proces je popsán v práci *Procedural generation of parcels in urban modeling* [14]. Je v nich uvedeno několik algoritmů, jak blok rozdělit.

Tvorba bloků

Tvorba bloků je inspirována algoritmem pro řešení labyrintů. Algoritmus se nazývá *Wall follower* [13], nebo také *Left-hand rule*. Pro pochopení algoritmu lze uvést příklad. Existuje člověk stojící na křížovatce. Půjde rovně, dokud nenarazí na další křižovatku. Na ní zahne doleva. Pokud tyto dvě akce bude opakovat, nakonec můžou nastat dva výsledky. Buď příjde opět na původní křižovatku, nebo skončí ve slepé uličce. V případě, že naráží na původní

křižovatku, obešel celý jeden blok. Pokud se tento člověk postaví do každé křižovatky ve městě a tento proces zopakuje pro každou cestu, která se v ní kříží, tak obejde všechny bloky ve městě. Pokud ale narazí na slepou uličku, na počáteční křižovatku se touto metodou už nedostane. V takovém případě se blok ignoruje.

Tvorba parcel

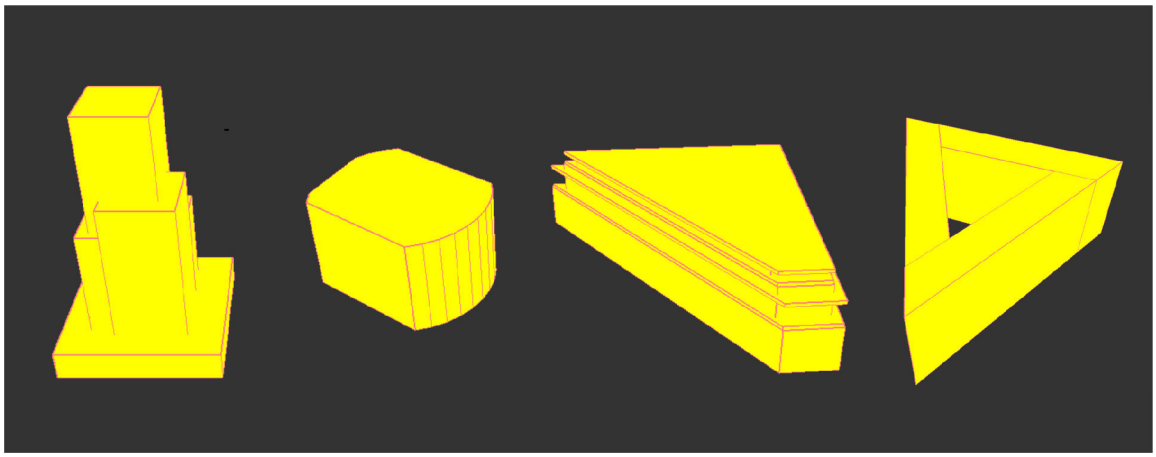
Práce *Procedural generation of parcels in urban modeling* [14] popisuje několik algoritmů, jak blok rozdělit. Dělení bloků lze docílit pomocí orientovaného bounding boxu [14], přímkové kostry [1] či Voronojové teselace [2]. Jak práce popisuje, nejideálnější je využití všech algoritmů pro získání co nejreálnějších parcel. V této bakalářské práci však byl využit pouze orientovaný bounding box, který poskytuje dostatečnou reálnost i náhodnost pro tento projekt.

3.6 Budovy

Poslední etapou generování je generování budov. Budovy tvoří hlavní vizuální složku města. Architektura města je ovlivněna spoustou vlivů. Příkladem vlivu ovlivňující architekturu, může být kultura. Když se srovná architektura Prahy a architektura Rijádu v Saudské Arábii, na první pohled je rozdíl zřetelný. Budovy však nejsou stejné ani v rámci jedné kultury. Jen v Praze existuje nespočet budov, které si nejsou vůbec podobné. A přesně tato rozmanitost v rámci jedné kultury je žádoucí v této bakalářské práci.

Budovy se generují na parcelách. Na každé parcele může být právě jedna budova. Pro zaručení rozmanitosti města je potřeba, aby modely budov byly co nejpestřejší. Pestrosti lze dosáhnout například přiřazováním náhodné výšky každému generovanému modelu, nebo mu náhodně přiřazovat natočení. Ale pro ještě větší rozmanitost byly implementovány čtyři algoritmy pro generování modelů budov. Architektura těchto budov vychází převážně z amerických velkoměst, jako New York. Tyto algoritmy vznikly inspirací práce *Pixel City* [16]. Každý algoritmus generuje naprosto odlišný typ modelů budov. Ukázku každého typu lze vidět na obrázku 3.5. Algoritmy jsou schopny generovat tyto typy budov:

- **Mrakodrapy** - Budovy připomínají styl mrakodrapů. Jsou velmi rozmanité ve svém tvaru i rozměrech.
- **Kancelářské budovy** - Jediný typ budovy s kruhovým motivem. Představují velké, prosklené budovy většinou sloužící pro kanceláře. Tento typ neposkytuje zas až tak velkou rozmanitost.
- **Trojúhelníkové budovy** - Svojí vizuální stránku převzaly od luxusních hotelů především z Austrálie. Jsou to budovy s trojúhelníkovým půdorysem. Typ modelu byl hlavně zaveden kvůli trojúhelníkovým parcelám, které mají moc ostrý úhel, takže se do nich nevleze žádný jiný typ. Jde o nejméně rozmanitý typ.
- **Klasické budovy** - Klasické budovy by měly představovat všechny ostatní budovy. Ať už to jsou historické, nebo budovy ve městě moc nevýrazné. Tento typ není inspirován žádným konkrétním typem budov ve skutečnosti. Jsou ale také velmi rozmanité ve svém tvaru i rozměrech a doplňují panorama vygenerovaného města.



Obrázek 3.5: Na obrázku je ukázka každého typu budov. První budova (zleva) je typu mrakodrap, druhá typu kancelářská budova, třetí je klasická budova a čtvrtá pak typu trojúhelníková budova.

Kapitola 4

Implementace

4.1 Terén

Výšková mapa je získána importem šedotónového obrázku. Terén je implementován jako pravidelná mřížka. Počet bodů mřížky je roven počtu pixelů ve výškové mapě. Vzdálenost mezi body je určena konstantou. Velikost terénu (mapy) je tedy přímo úměrná velikosti šedotónového obrázku poskytnutého uživatelem.

Výšky

Je běžné, že výška každého bodu mřížky je rovna hodnotě pixelu výškové mapy, který má stejný index. Mřížka tedy má rozptyl výšek $< 0, 255 >$. Také je možné použít záporné výšky, interval tedy potom je $< -128, 128 >$. Výšky často bývají upravené konstantou nebo funkcí, aby výsledné výšky měly přijatelnější hodnoty pro daný projekt, takže jsou rozsahy odlišné od výše uvedených.

Jak už bylo zmíněno v návrhu (sekce 3.3), výšky jsou limitovány pouze na tři stupně. První stupeň je **hora**, která má výšku $128 \times KONSTANTA$. Dále je **rovina**, která má výšku 0. Posledním stupněm je **vodní plocha**, ta má výšku $-128 \times KONSTANTA$. Výšky jsou násobeny konstantou proto, aby byl větší rozptyl jednotlivých stupňů a **hory** tak byly vyšší a **vodní plochy** hlubší. Ukázka mřížky terénu je na obrázku 4.1.

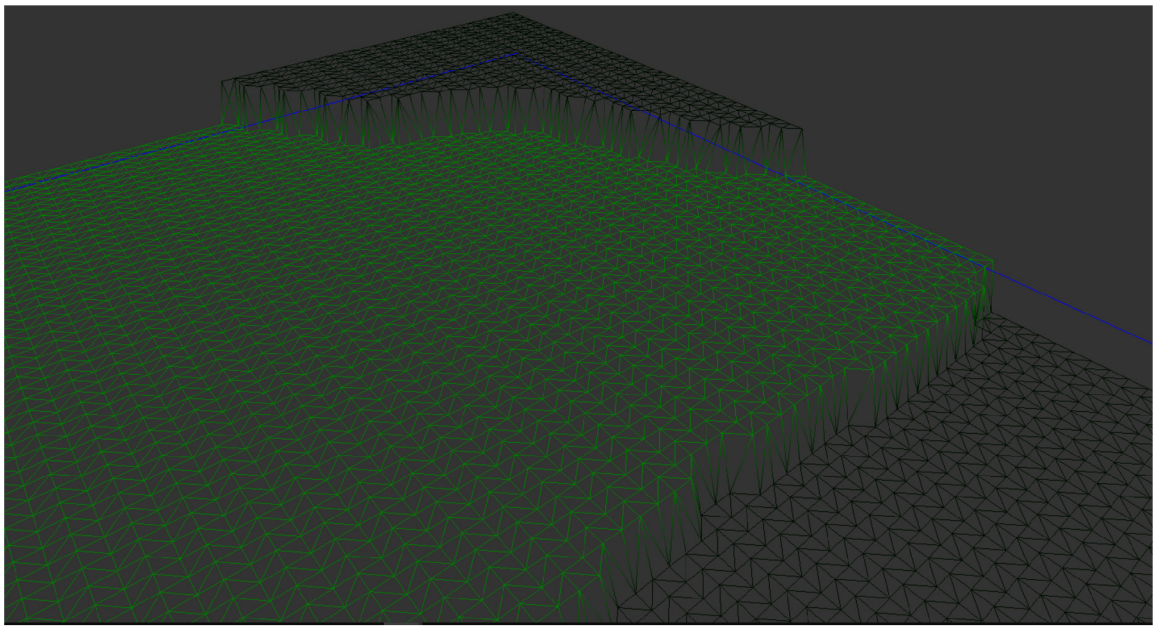
Pro lepší vizuální efekt je implementována i vodní hladina. Ta je reprezentována obdelníkem vyplněným modrou barvou. Tento obdelník má rozměr terénu a je výškově posazen o něco níže, než je **rovina**. Ukázka kompletního terénu je na obrázku 4.2.

4.2 Pozemní komunikace

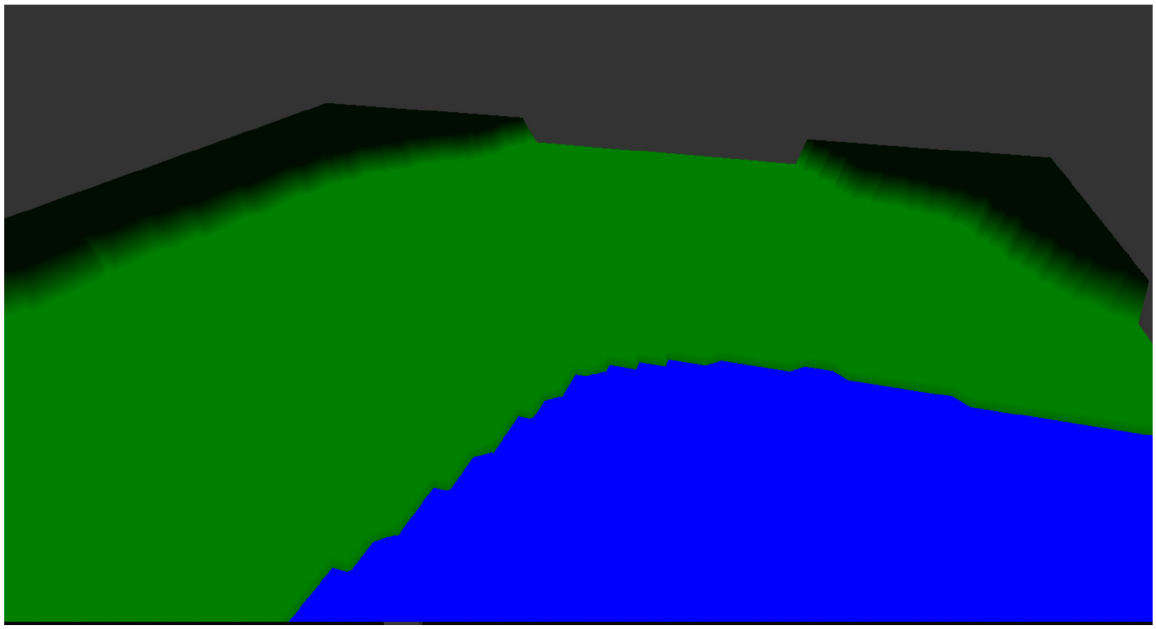
Generování pozemní komunikace je implementováno jako stochastický parametrický L-systém (popsán v sekci 2.3), jehož výstup je množina segmentů. Segmenty jsou popsány níže.

Generování každého segmentu probíhá ve dvou fázích. První fází je samotné vygenerování segmentu pomocí třídy funkcí **GlobalGoals**. Druhou fází je upravení segmentu okolním podmínkám pomocí třídy funkcí **LocalConstraints**. Obě sady funkcí jsou popsány níže.

Způsob generování pozemní komunikace je inspirováno prací *Procedural modeling of cities* [12] včetně segmentů, funkcí **GlobalGoals** a **LocalConstraints**.



Obrázek 4.1: Ukázka mřížky terénu. Mřížka se zbarvuje do černa v případě změny výšky.



Obrázek 4.2: Ukázka hotového terénu včetně vodní hladiny. Terén se zbarvuje do černa v případě změny výšky, tedy i pod vodní hladinou.

Segment

Segment je krátká úsečka, která nese různé atributy. Segmenty na sebe polohově navazují a tak vizuálně tvoří plynulou cestu. Segmenty jsou navzájem provázané, to znamená, že každy segment má odkaz na předchozí segment a na následující segment. Segment také obsahuje reference na odbočky, které ze segmentu vedou. Provázáním se zřetelně ulehčí

a urychlí algoritmy, které se segmenty pracují, například tvoření parcel, které je popsané v sekci 4.3.

LocalConstraints

V těchto funkcích se kontroluje vztah segmentu vůči blízkému okolí. Snaží se najít vhodné hodnoty pro segment (například polohu koncového bodu) a následně segment podle vhodných hodnot upravit. Funkce vrací status segmentu. Statusy mohou být **OK**, **MODIFIED** nebo **WRONG**. Status **OK** vrací v případě, že segment nemusel být upraven, a to znamená, že se na jeho základě můžou vygenerovat další segmenty. Pokud dojde k úpravě segmentu, díky které už segment nemůže být předlohou pro další segment, **LocalConstraints** vrací status **MODIFIED**. Existuje ještě třetí možnost. Funkce nebyly schopné najít vhoné hodnoty pro segment, takže bude muset být odstraněn. V tom případě je použitá návratová hodnota **WRONG**.

Funkce **LocalConstraints** řeší především:

- V případě, že segment zasahuje mimo mapu, je modifikován, aby končil na hraně terénu.
- V případě průsečíku s jiným segmentem je modifikován, aby končil v bodě dotyku.
- V případě, že je v dosahu segmentu konec jiného segmentu, je segment modifikován, aby byly oba segmenty napojeny.

V druhém a třetím případě však vzniká problém. Segment může mít průsečík s více segmenty najednou. V tom případě se použije nejbližší. To samé platí i v případě, že v dosahu končí více segmentů.

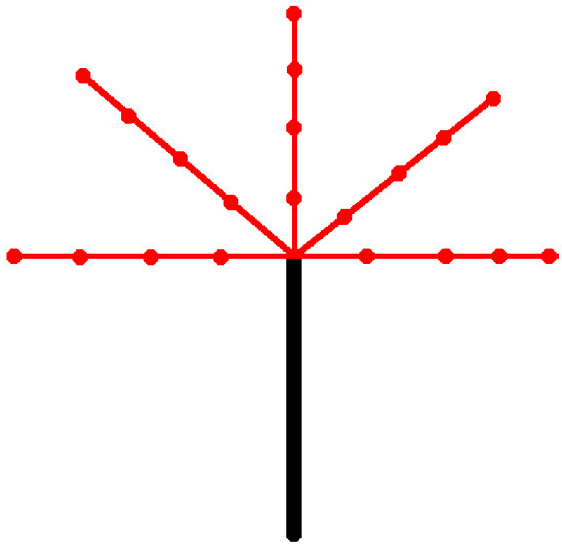
V druhém a třetím případě se na sebe napojují segmenty z různých směrů, nebo může být v jednom bodě napojených segmentů více něž dva. Z toho důvodu se v takové situaci vytvoří **křižovatka**. **Křižovatka** je bod, který ale obsahuje reference na všechny segmenty, které se v tomto místě protínají. **Křižovatky** jsou nezbytné pro algoritmus pro vyhledávání parcel, popsaný v sekci 4.3.

LocalConstraints také řeší polohu segmentu vůči terénu.

Detekce terénu

Detekce terénu probíhá díky „paprskům“. Jde o úsečky, které začínájí v koncovém bodě segmentu. První úsečka (paprsek) svírá se segmentem 90°, každá další svírá o 45° více, až do 180°. Ve čtvrtce, půlce, tričtvrtině a na konci paprsku se vypočítá výška a kontroluje se, zda-li neleží jinde, než v **rovině**. Pro představu, jak algoritmus funguje, lze kouknout na obrázek 4.3.

V kapitole 4.1 bylo zmíněno, že cesty se často táhnou podél řek a hor. Stejně tak byly implementované pravidla pro **hlavní cesty**. V případě, že je detekována v blízkosti segmentu a v jeho směru hora, či vodní plocha, segment vygeneruje odbočky v obou směrech a další segment už negeneruje. Všechny segmenty, které pak vychází z obou odboček, se drží hory, či vodní plochy. Pokud na horu, nebo vodní plochu narazí **silnice** nebo **ulice**, přestanou se dál generovat. Negenerují ani odbočky.



Obrázek 4.3: Na obrázku lze vidět způsob detekce terénu. Černá čára je segment, červené čáry jsou „paprsky“. Červené body v paprscích jsou místa, kde se vypočítává výška terénu.

GlobalGoals

V těchto funkcích probíhá samotné generování segmentů. Nově generovaný segment je generován na základě předchozího, již existujícího segmentu. Vstupem do těchto funkcí je tedy již existující segment. Podle typu cesty, kterou segment reprezentuje a podle příznaků, které segment obsahuje se rozhodne, jaký nově vygenerovaný segment bude.

Lze uvést příklad. Vstupním segmentem je segment typu **Hlavní cesta**, táhnoucí se podél řeky. Nově vygenerovaný segment tedy bude také typu **Hlavní cesta** a bude mu předán příznak, aby se táhnul podél řeky.

Funkce **GlobalGoals** také řeší generování odboček. Odbočky se generují náhodně. To však s sebou nese jeden velký problém. Když by se to nechalo pouze na pseudonáhodě, která je popsána v sekci 2.1 (v jazyce C++ na funcki `rand()`), docházelo by ke generování nerovnoměrně rozprostřených odboček. Jinými slovy, pokud šance na vygenerování odbočky by byla 25%, tak by se stejně odbočka nevygenerovala každý čtvrtý segment. Docházelo by k situacím, kdy by se odbočka nevygenerovala deset segmentů a poté u tří segmentů po sobě ano a to ve stejném směru. To by mělo za následek velmi špatnou strukturu města. Proto byla zavedena zvyšující se šance. První segment má tedy šanci na generování odbočky 15%. V případě, že odbočku nevygeneruje, další má 25%, další 35% a tak dále. V případě, že se odbočka vygeneruje, šance pro další segment se resetuje na počátečních 15%. Všechny hodnoty v tomto odstavci byly použity ilustračně a v aplikaci se liší.

Generují se zde i axiomy. Ty pro vygenerování předchozí segment nepotřebují, potřebují však **primární centrum**.

Generování

Algoritmus 1 zjednodušeně popisuje způsob generování segmentů. Je zde použita fronta *Fronta*, která uchovává vygenerované segmenty, které ale ještě neprošly modifikací vzhledem

Algorithm 1 Pseudokód generování cest

```
Fronta.add(vytvorAxiomy())
vyčisti Segmenty
while Fronta není prázdná do
    S = Fronta.pop()
    status = LocalConstraints(S)
    if status != NEVHODNÝ then
        if status == OK then
            Fronta.push(GlobalGoals(S))
        end if
        Segmenty.push(S)
    else
        zahození S
    end if
end while
```

k okolí, tedy před využitím *LocalConstraints*. Dalé existuje struktura uchovávající již hotové segmenty. Ta je nazvána *Segmenty*.

Jako první se vytvoří axiomy a vloží se do *Fronty*. Algoritmus posléze přejde do cyklu. Cyklus končí v případě, že *Fronta* je prázdná.

Z *Fronta* se vytáhne první segment *S*. Musí se zjistit, zda-li je segment *S* validní vůči okolí. Takže je vstupem do funkcí *LocalConstraints* a ty pak vrátí jeho status. Pokud status ukazuje, že je segment *S* nevhodný, je zahozen. Pokud ukazuje, že byl modifikován, může být uložen do hotových segmentů, tedy do *Segmenty*. Pokud ale status ukazuje na skutečnost, že je segment *S* v pořádku, ještě před jeho uložení do *Segmenty* se musí vygenerovat jeho navazující segment, popřípadě i odbočka. To znamená, že se použijí funkce *GlobalGoals*, kde vstupní segment bude právě segment *S*.

4.3 Parcely

Tato sekce popisuje implementaci tvoření parcel a bloků, jejichž popis je v návrhu (sekce 3.5). Nejdříve popíše generování bloků a vysvětlí konkávnost a konvexnost těchto bloků. Následně je popsána implementace a využití orientovaného bounding boxu pro dělení bloků na parcely.

Tvoření bloků

S vytvářením bloků velmi pomůže vzájemné propojení segmentů tvořící cesty. Jelikož každý segment zná svého předchůdce a následovníky, lze se jednoduše pohybovat po segmentech a tvořit bloky.

Algoritmus 2 ukazuje velmi zjednodušený pseudokód, který je použit pro vytvoření bloků. Reálný kód je podstatně složitější, jelikož se zde vyskytují problémy, více křižovatek na jednom segmentu nebo změna směru segmentů.

Iteruje se přes všechny křižovatky, které při generování pozemní komunikace vznikly. Algoritmus 2 se pak provede pro každý segment, každé křizovatky.

Algoritmus 2 popisuje průchod přes segmenty. Končí v případě, že koncový bod aktuálního segmentu je shodný s počatečním bodem počátečního segmentu. Aktuálním segmentem

Algorithm 2 Pseudokód vytváření bloků

```
PrvniSegment = dalsiSegmentZKrizovatky
AktualniSegment = PrvniSegment
BodyParcely.clear()
while PrvniSegment.pocatecniBod != AktualniSegment.koncovyBod do
    while jakakolivKrizovatka neleží na AktualniSegment do
        BodyParcely.add(AktualniSegment.koncovyBod)
        if AktualniSegment.dalsiSegment == NULL then return NULL
        end if
        AktualniSegment = AktualniSegment.dalsiSegment
    end while
    K = krizovatkaLeziciNaAS
    AktualniSegment = K.nejlevejsiNapojenySegment
end while
```

se stává následující segment aktuálního segmentu, dokud se nenarazí na křižovatku, nebo následující segment neexistuje. V případě, že se narazí na křižovatku, zkontrolují se všechny její segmenty a vybere se ten, který s aktuálním segmentem svírá nejmenší úhel (je „nejvíce vlevo“). Při každé změně aktuálního segmentu se uloží jeho počateční bod, protože je to bod definující blok. Výsledkem je pak množina bodů, které definují blok.

V algoritmu nastávají další problémy. Do křižovatky jsou segmenty napojeny z různých směrů, takže se musí při každém výběru segmentu z křižovatky kontrolovat směr, který se při průchodu může měnit. Také může nastat situace, kdy neexistuje další segment aktuálního segmentu. V tom případě blok nikdy nebude uzavřen a tak se odstraňuje. Na segmentu může být více křižovatek. V tomto případě se vybere ta nejbližší prvnímu bodu segmentu.

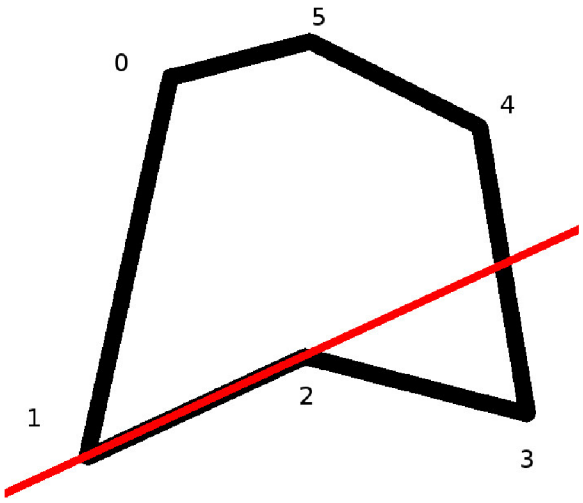
Konkávnost a konvexnost bloků

Blok je ve své podstatě polygon. Městské cesty jsou nepravidelné, takže vytvořené bloky bývají často konkávní polygony. Konkávní polygon se také dá dělit, například pomocí Voronojové teselace [2], nebo i pomocí „orientovaného bounding boxu“, který je popsán níže, na parcely. Avšak práce s konvexními polygony je jednodušší. Konkávní blok se tedy nejdříve rozdělí na několik konvexních. Tím se zjednodušší algoritmy pro dělení, navíc se už tímto úkonem velikost bloku redukuje.

Polygon si lze představit jako seřazenou množinu bodů. Pokud se u některého bodu objeví konkávnost (tedy úhel, svírající předchozí bod, aktuální bod a následující bod, je větší než 180°), polygon je dělen přímkou, která prochází bodem, u kterého se prokázala konkávnost a předchozím bodem.

Na obrázku 4.4 je ukázáno, jak dělení vypadá. Konkávnost byla prokázána v bodě 2 (protože body 1, 2 a 3 svírají úhel větší jak 180°). Polygon je tedy dělen přímkou (červená čára), která prochází bodem 2 a bodem předchozím, tedy bodem 1.

Toto dělení je velmi jednoduché i rychlé na výpočet. Může však blok rozdělit nevhodně, především v případě, že bude mít mnoho nevhodně umístěných konkávních bodů. Ale vzhledem k povaze pozemní komunikace v této práci, tato situace měla nastat jen zřídka.



Obrázek 4.4: Ukázka dělení konkávního polygonu na dva konvexní. Černá čára představuje polygon, číslovky číslují vrcholy polygonu a červená čára je přímka, která dělí polygon.

Využití Bounding boxu

Bouding box je obdelník, ve kterém se nachazí polygon. V praxi se často používá pro detekci kolize. Bouding box lze vidět na obrázku 4.5. Využití je také popsané v práci zabývající se tvorbou parcel [14].

Dělení probíha rekurzivně. Nejdříve je bounding box vypočítán pro celý blok. Vždy je natočený podle prvních dvou bodů bloku. To lze vidět i na obrázku 4.5 Následně se tento bounding box rozdělí na polovinu a tím i blok. Pro každou polovinu bloku se vypočítá nový bounding box a ten se opět rozdělí. Toto probíhá, dokud obsah bloku neklesne pod minimální hodnotu, nebo některá ze stran bounding boxu není příliš krátká. Výsledkem jsou parcely pro stavbu budov. Ukázka tohoto algoritmu je na obrázku 4.6.

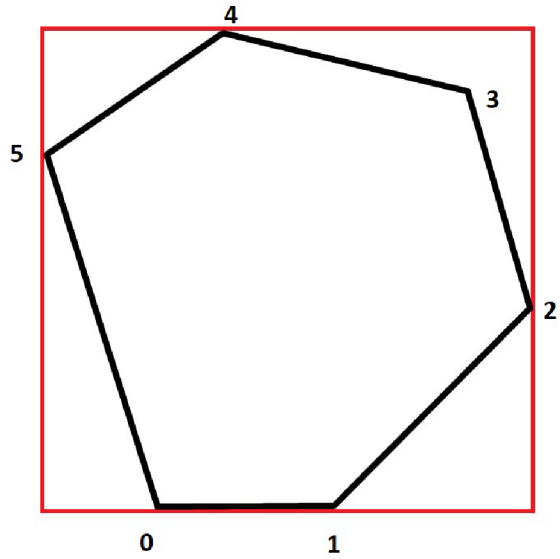
4.4 Budovy

Generování probíhá ve dvou fázích. Nejdříve se vygeneruje půdorys budovy. Každý typ budovy má svojí metodu, jak půdorys vygenerovat. Druhá fáze je vygenerování stěn a střechy na základě půdorysu. Myšlenka tohoto generování je převzata z *Procedural modeling of buildings* [11] a *Procedural Modeling of Cities* [12].

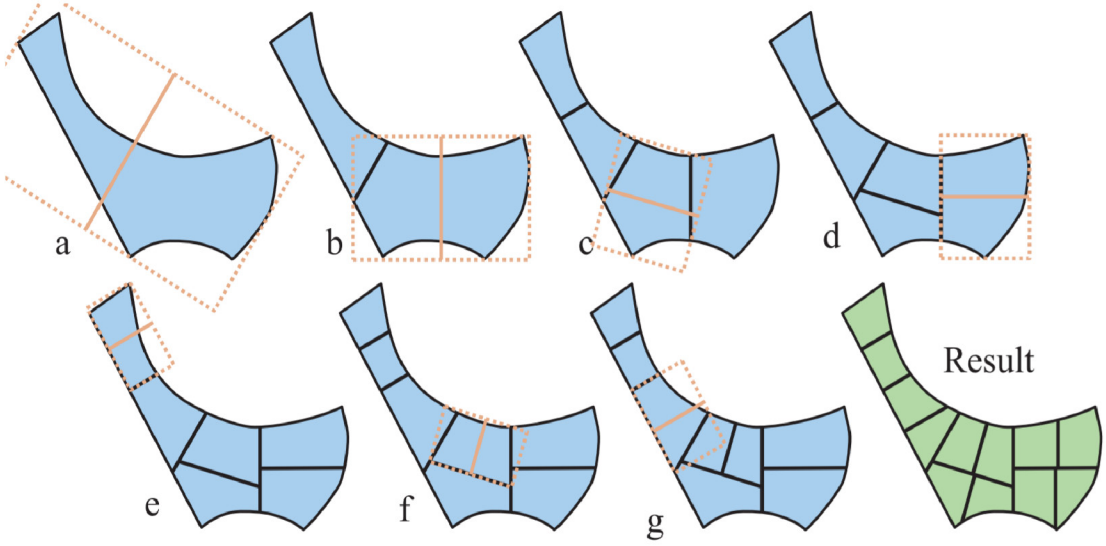
Mrakodrapy

Mrakodrapy se skládají ze tří částí, podstava, jádro a rozšíření. Jak už bylo zmíněno, první fází je generování půdorysu. To znamená, že se vygenerují půdorysy všech tří částí a až poté se generují stěny a střecha.

První na řadě je podstava. Ta má vždy tvar čtverce, který má vždy velikost největšího čtverce, který se do parcely vejde. Slouží také jako hranice pro generování zbytku budovy. Další na řadě je jádro. To má tvar čtverce nebo obdelníku. Vygeneruje se tak, že střed podstavy musí náležet v jádru. Posledním krokem je generování rozšíření, kterých může



Obrázek 4.5: Ukázka bounding boxu. Černá čára je polygon a červená čára znázorňuje jeho bounding box.

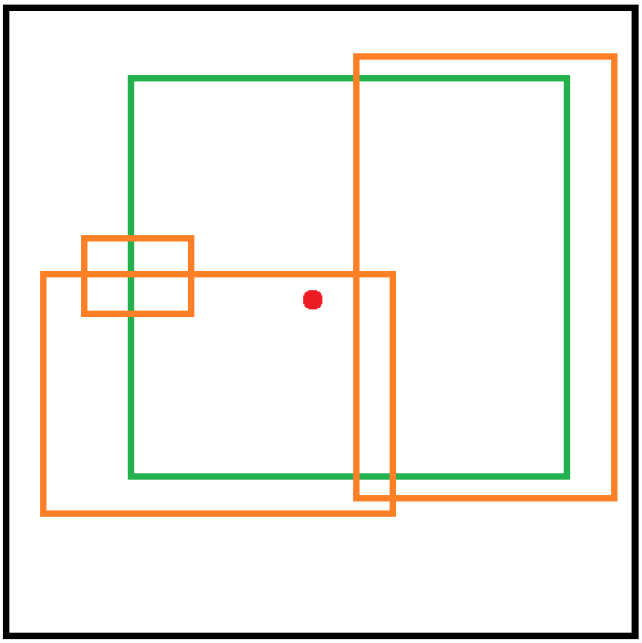


Obrázek 4.6: Ukázka dělení bloků pomocí bounding boxu. Obrázek je převzaný z *Procedural generation of parcels in urban modeling* [14].

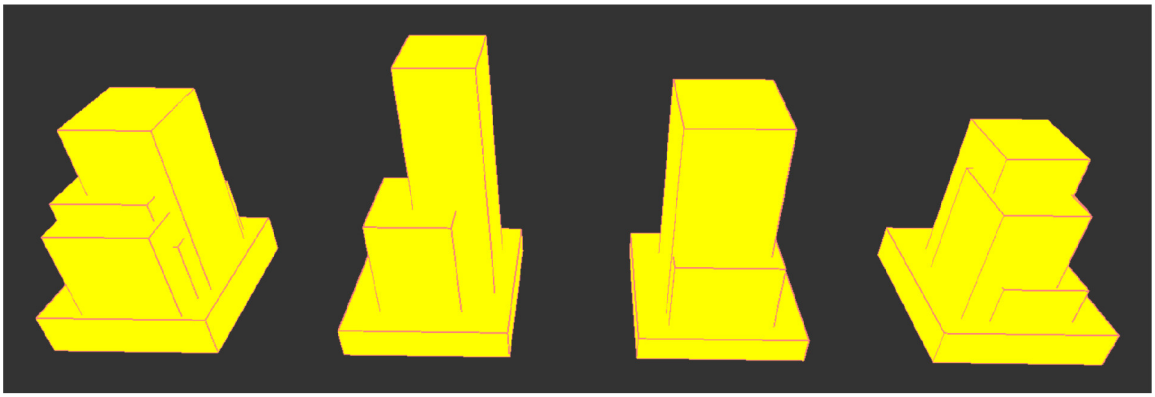
být až tři. Generování rozšíření funguje na podobném principu jak generování jádra. Hlavní rozdíl je v tom, že střed podstavy nemusí být nutně vně rozšíření. Rozšíření však musí být v určité maximální vzdálenosti od středu. Tim nevznikne situace, že by rozšíření neprotínalo jádro. Rozdílem je také větší rozmanitost ve velikostech rozšíření. Výsledný půdorys lze vidět na obrázku 4.7 a hotové modely na obrázku 4.8.

Tento algoritmus využívá hned několik prvků náhody. Ať už jde o rozměry podstavy, jádra či rozšířením, nebo o počet rozšíření a jejich vzdálenost od středu. Je tedy velmi malá

pravděpodobnost, že by mohly být dva modely identické, nebo že by se tak vůbec mohly jevit.



Obrázek 4.7: Na obrázku lze vidět půdorys budovy typu **mrakodrap**. Černý čtverec znázorňuje podstavu a červený bod jeho střed. Zelený obdelník jádro a oranžové obdelníky zase rozšíření.



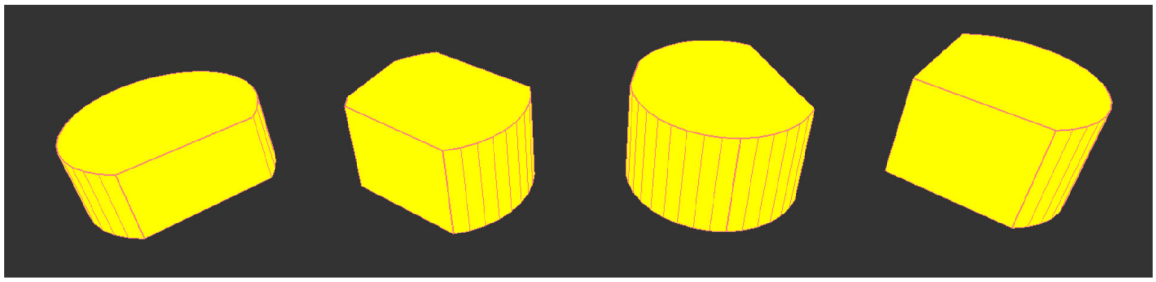
Obrázek 4.8: Obrázek obsahuje čtyři modely budov typu **mrakodrap**.

Kancelářské budovy

Kancelářské budovy jsou jediným typem, který má kruhovou tématiku. Skládá se pouze z jedné části a tou je jádro.

Poloměr kancelářské budovy je stejný, jako má největší možná kružnice vně polygonu, který tvoří parcelu. Po obvodu největší možné kružnice se tvoří body s rozdílem 10°. Bodů na kružnici tedy bude 36. Při generování každého bodu je šance, že se přeskočí 90°, tedy

devět bodu. Toto přeskočení se však může uplatnit maximálně 3×, jinak by vznikla budova ve tvaru krychle. Ukázka modelů tohoto typů je na obrázku 4.9.

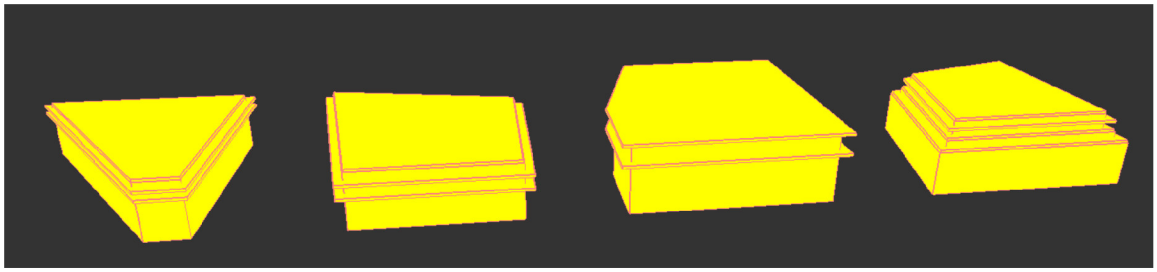


Obrázek 4.9: Obrázek obsahuje čtyři modely budov typu **kancelářská budova**.

Klasické budovy

Klasické budovy se skládají z několika jader. Jádra se generují na sebe. Půdorys prvního (nejspodnějšího) jádra kopíruje tvar parcely, jen je zmenšený o 10%. Každé další jádro kopíruje tvar prvního jádra, může však zůstat se stejnou velikostí, ale může se také zvětšit či zmenšit. Ukázka těchto budov je na obrázku 4.10.

Modely klasických budov se liší rozměry i počtem pater. Není tedy příliš velká pravděpodobnost, že by byly dva modely identické.

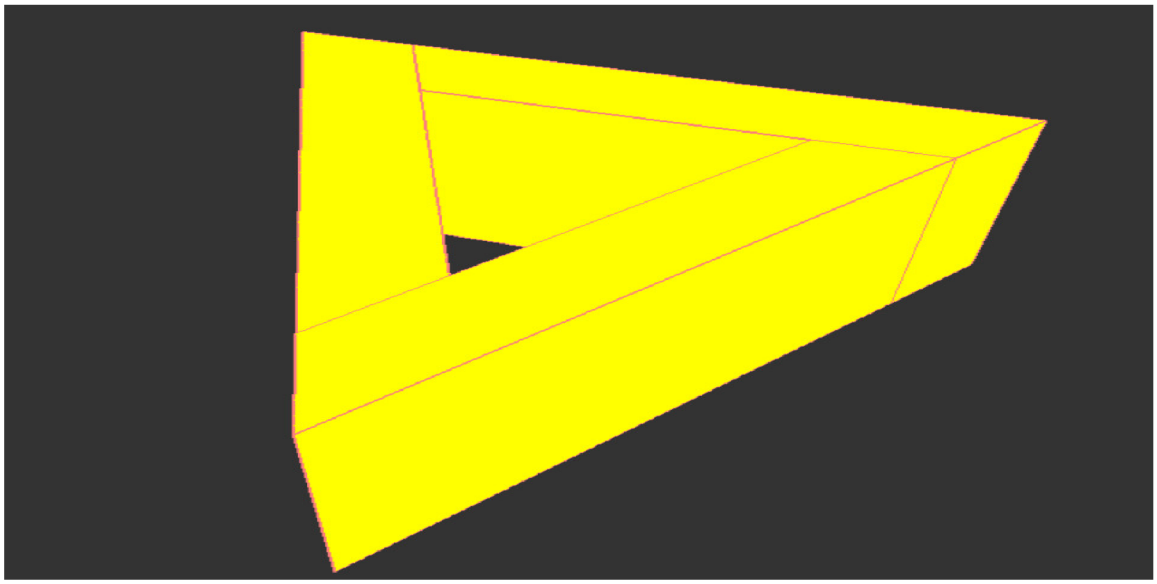


Obrázek 4.10: Obrázek obsahuje čtyři modely budov typu **klasická budova**.

Trojúhelníkové budovy

Trojúhelníkové budovy se tedy mohou generovat pouze na trojúhelníkových parcelách. Trojúhelníková budova má tři jádra. Všechna tři jádra dohromady tvoří jednu trojúhelníkovou budovu. Půdorys kopíruje tvar parcely, jen je zmenšený o 10% a má uprostřed díru. Díra také kopíruje tvar parcely, jen je zmenšená daleko více. Ukázka takové budovy je na obrázku 4.11.

Nevýhodou toho algoritmu je fakt, že jediným prvkem náhody, který je při generování využit, jsou rozměry. Takže i když budovy nebudou identické, na pohled se tak jevit mohou.



Obrázek 4.11: Obrázek obsahuje model budovy typu **trojúhelníková budova**.

Kapitola 5

Testování

V této kapitole jsou ukázky výstupů generátoru. Ke každému výstupu je i ukázán vstup, který tento výstup umožnil. Je vždy ukázán šedotónový obrázek a souřadnice použitých center. Dále je vidět vygenerovaný terén, síť pozemní komunikace a výsledné panorama města. Ke každému příkladu je poskytnut i součet budov tvořící město.

Město na ostrově

Toto město je posazeno na ostrově. Ostrov je obehnán vodní plochou a z části i horou. Na ukázce lze vidět, jak bude město vypadat v případě, že bude vygenerováno na terénu, který je ze všech stran ohraničen vodní plochou či horou. Šedotónový obrázek představující výškovou mapu je na obrázku 5.1. Je zde ukázán i terén z ptačí perspektivy. Na obrázku 5.2 je ukázána pozemní komunikace. Lze si všimnout, že cesty obešly celý ostrov a to včetně horské části. Obrázek 5.3 ukazuje panorama města.

- Rozměr šedotónového obrázku je 90×90, rozměr terénu je tedy 21600×21600.
- Ve městě je 1750 budov.

Město u vody

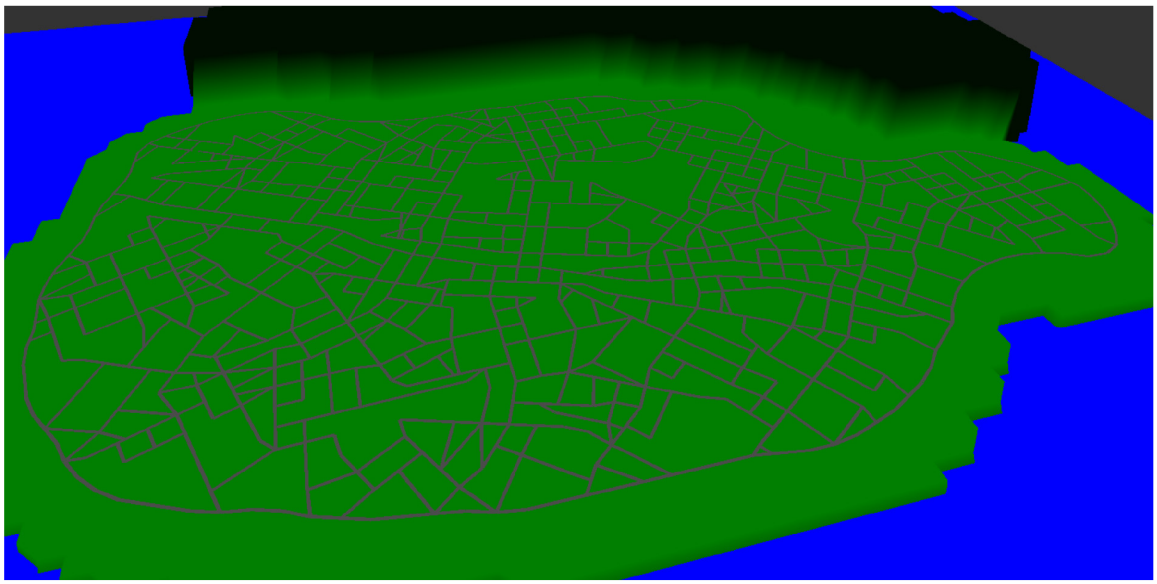
Město je posazeno u vody. Příklad ukazuje, jak může vypadat město, které je posazené u vody v neuzavřeném prostranství. Šedotónový obrázek lze vidět na obrázku 5.4, na kterém je i terén z ptačí perspektivy. Na obrázku 5.5 jde vidět pozemní komunikace. Obrázek 5.6 ukazuje panorama města. Pro lepší orientaci v síti pozemní komunikace jsou viditelná i centra, zaznačená růžovou tečkou.

- Rozměr šedotónového obrázku je 100×100, rozměr terénu je tedy 24000×24000.
- Ve městě je 2298 budov.

Město u dvou pahorků

Město je posazeno na terén, který je z jedné strany ohraničen řekou a horou. Z druhé strany pak jen horou. Uprostřed terénu jsou dva pahorky. Na tomto příkladu lze vidět, jak vypadá

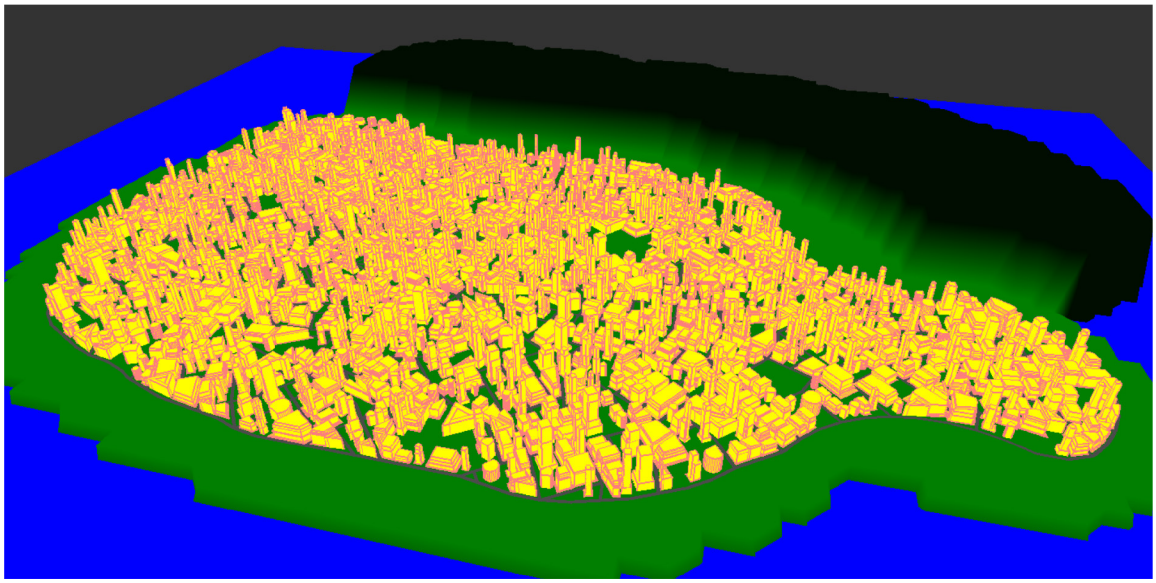
Obrázek 5.1: Obrázek nalevo ukazuje šedotónový obrázek představující výškovou mapu pro terén, který je na pravém obrazků. Šedotónový obrázek má rozměry 100×100 a terén 24000×24000.



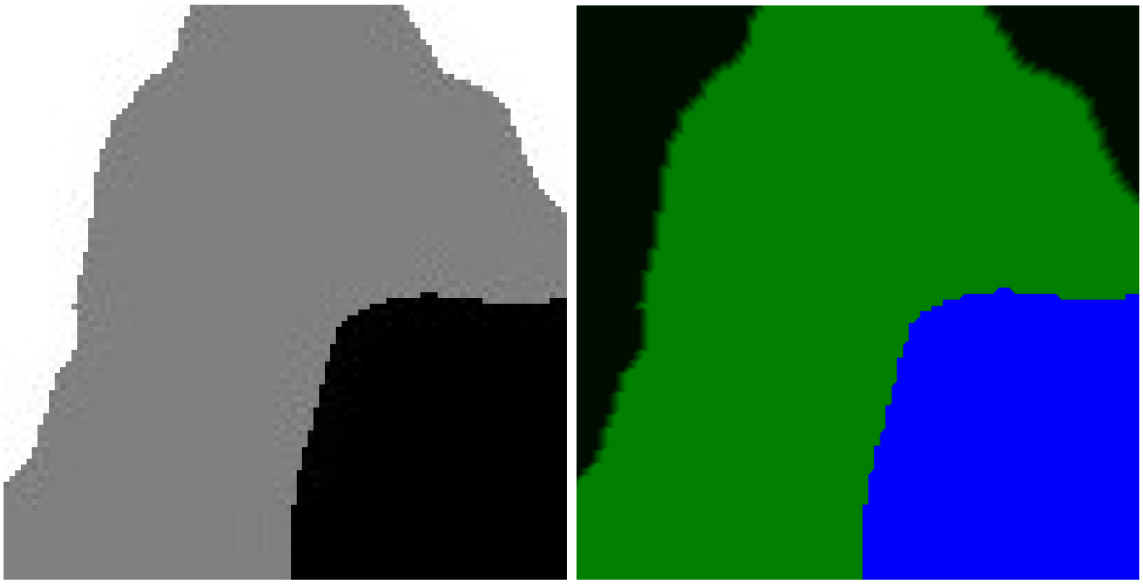
Obrázek 5.2: Obrázek ukazuje vytvořenou síť pozemní komunikace pro město vygenerované na ostrově.

město velých rozměrů, které se rozléhá po většině mapy. Na obrázku 5.7 je vidět šedotónový obrázek a terén z ptačí perspektivy. Cesty jsou vidět na obrázku 5.8. Výsledné panorama je na obrázku 5.9. Protože jde o obrovské město, tak aby bylo lépe vidět rozmístění center, jsou zaznačeny růžovou tečkou.

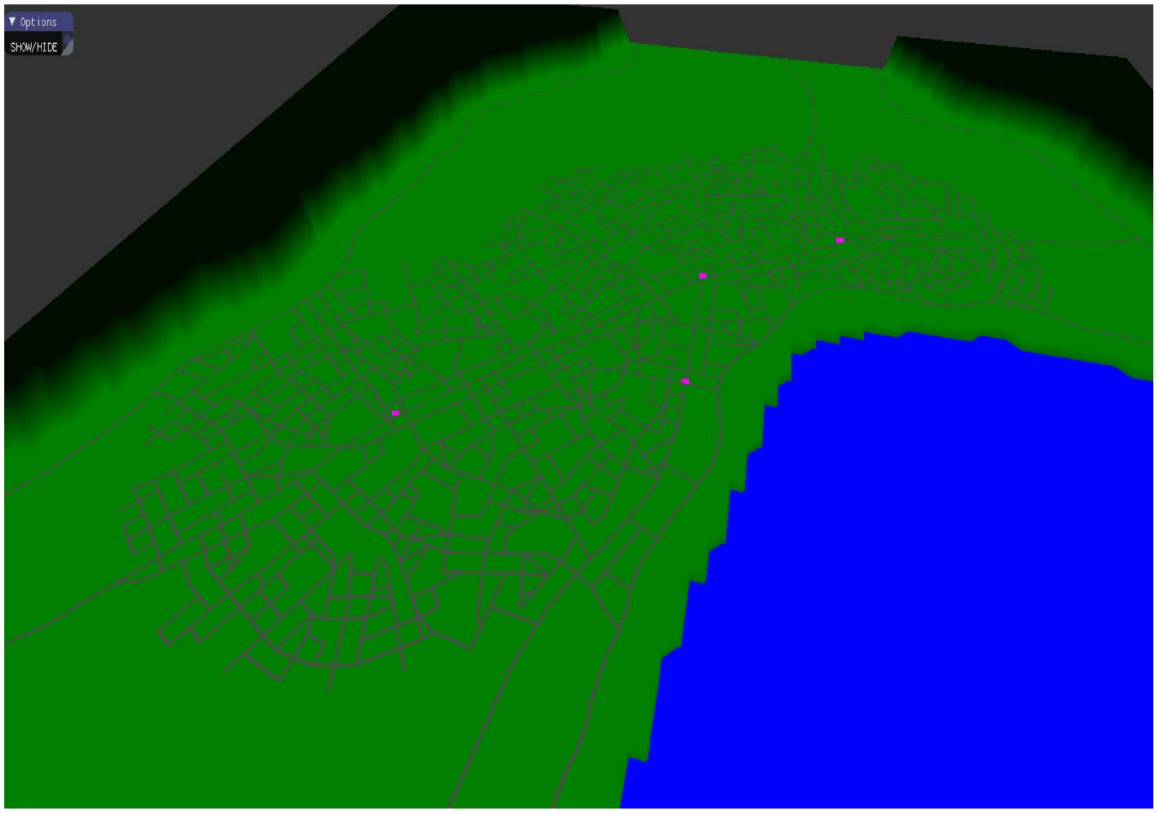
- Rozměr šedotónového obrázku je 100×100, rozměr terénu je tedy 24000×24000.
- Ve městě je 6353 budov.



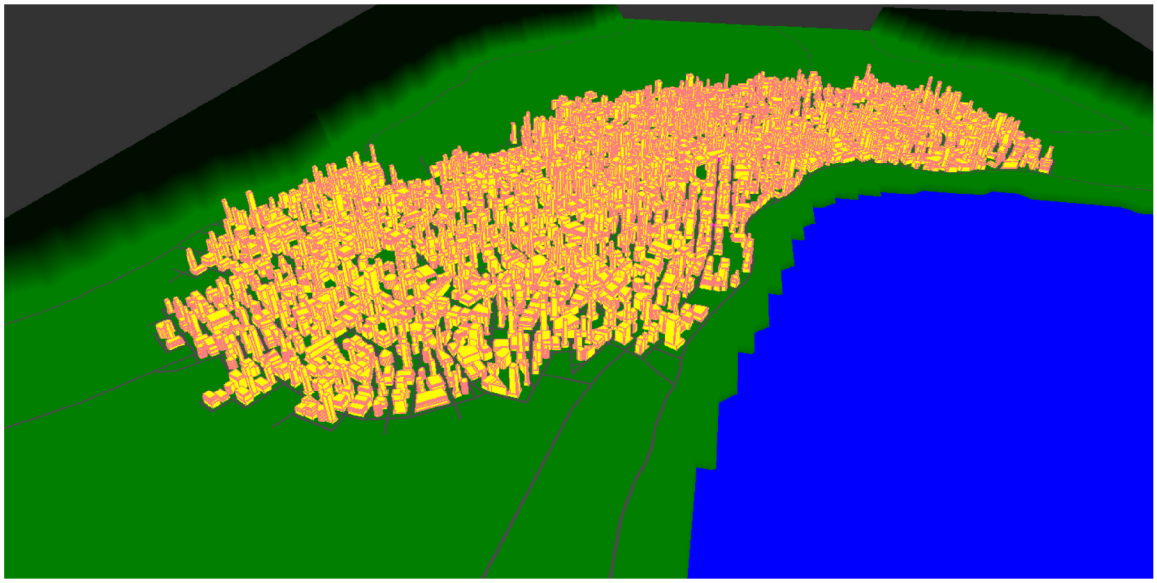
Obrázek 5.3: Obrázek ukazuje panorama hotového města vygenerovaného na ostrově.



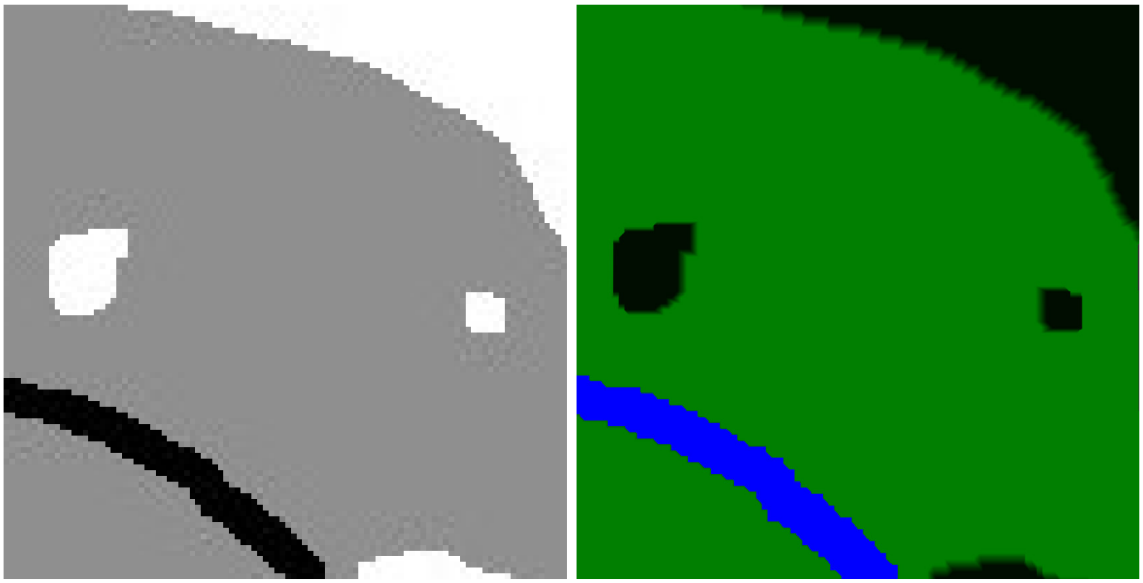
Obrázek 5.4: Obrázek nalevo ukazuje šedotónový obrázek představující výškovou mapu pro terén, který je na pravém obrazků. Šedotónový obrázek má rozměry 100×100 a terén 24000×24000.



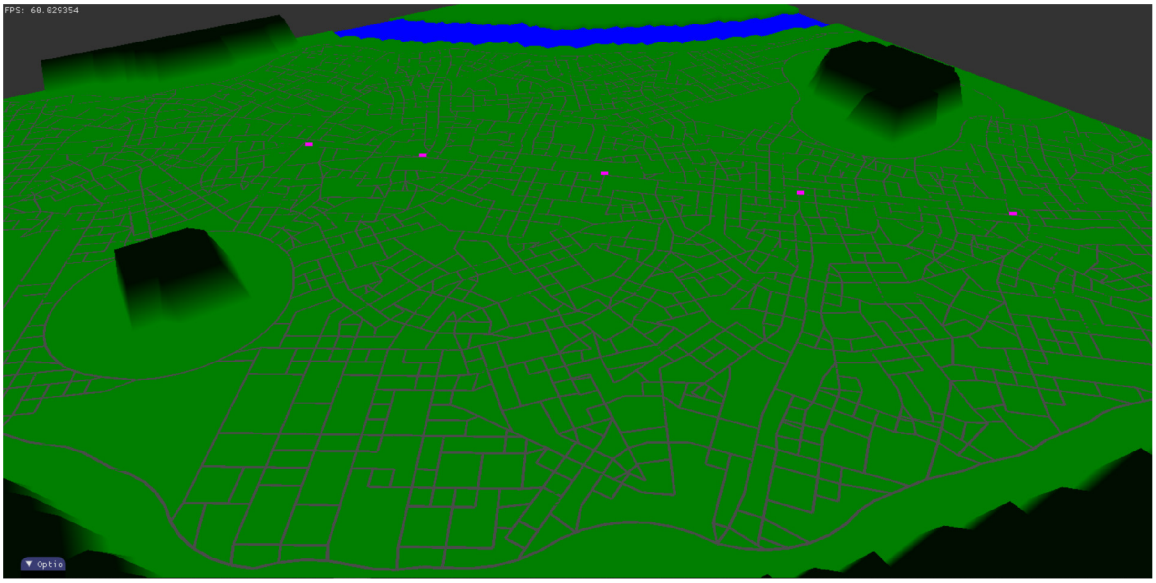
Obrázek 5.5: Obrázek ukazuje vytvořenou síť pozemní komunikace pro město vygenerované u vody ve volném prostranství. Růžové tečky znázorňují centra.



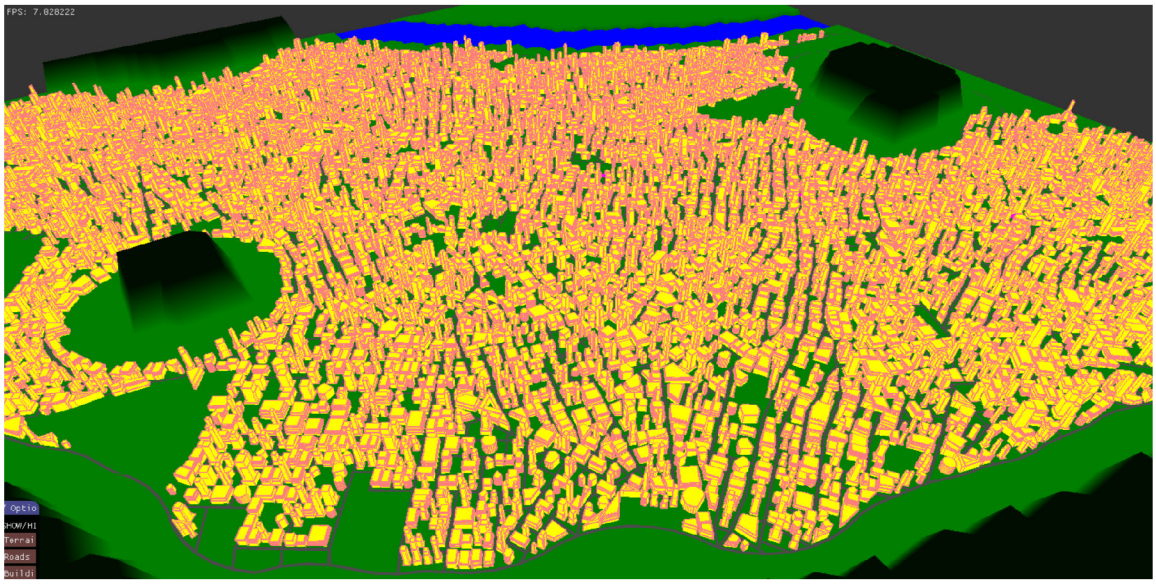
Obrázek 5.6: Obrázek ukazuje panorama hotového města vygenerovaného u vody ve volném prostranství.



Obrázek 5.7: Obrázek nalevo ukazuje šedotónový obrázek představující výškovou mapu pro terén, který je na pravém obrazků. Šedotónový obrázek má rozměry 100×100 a terén 24000×24000.



Obrázek 5.8: Obrázek ukazuje vytvořenou síť pozemní komunikace pro město vygenerované na terénu se dvěma pahorky a maximálním rozsahem primárního centra. Růžové tečky ukazují polohu zadaných center.



Obrázek 5.9: Obrázek ukazuje panorama hotového města vygenerovaného na terénu se dvěma pahorky a maximálním rozsahem primárního i sekundárních center.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo prostudovat problematiku procedurálního generování měst. Dále navrhnout a implementovat generátor, který bude schopný generovat město na základě těchto poznatků. Generování města je však velmi široký pojem, proto bylo nutné zadání specifikovat. Generátor má tedy generovat město s vysokou náhodností a pestrostí, zejména u budov. Zároveň se ale musí do jisté míry držet reálných atributů města. Implementován je i terén. Ten je však zjednodušený. Existují pouze tři výšky terénu. Pro generování pozemní komunikace využívá stochastického parametrického L-systému. Generování parcel probíhá za pomocí orientovaného bounding boxu. Jsou implementovány čtyři algoritmy generující budovy. Uživatelským vstupem je šedotónový obrázek jako výškova mapa pro generování terénu. Další uživatelský vstup je množina center, které určují hrubou strukturu města.

Zadání se splnit podařilo. Výsledné město se drží reálných aspektů města. Cesty reagují na hory a vodní plochy. Je využit reálný vzor pozemní komunikace. Použitý vzor je nejnáhodnější z existujících vzorů. Typy budov jsou inspirovány skutečnými. Implementovány jsou čtyři typy, to městu dodává pestrost a zároveň náhodnost. Ačkoli se parcely tvoří s dostatečnou náhodností, občas se některé parcely vytvoří nevhodně a nevyužije se tak efektivně celý prostor bloku. Generátor zvládá generovat města jak velkých roměrů, tak i malých. V obou případech se však architektonicky drží tématiky velkoměst. Techniky generování, použité v této bakalářské práci, jsou inspirovány již existujícími, které jsou použity v obdobných pracích.

Možností jak pokračovat v této práci je celá řada. Protože terén v této práci má pouze tři výšky, prvním z možných rozšíření je rožíření terénu na celých 256 výšek. Rozšířit se dá také pozemní komunikace. Zavedení lepší reakce na terén (například zavedením mostů), by reálnost města vzrostla. Dají se také zavést další existující vzory cest. Tvorbě parcel by se mohlo implenetovat několik dalších algoritmů, které by společně mohly vyřešit dosavadní nedostatky parcel. Rozšíření se ale také dá brát z jiného pohledu. Do města by se daly přidat i animace, například pohybující se automobily. Dalším rozšířením může být i export modelů z vygenerovaného města. To značně rozšíří uplatnitelnost aplikace.

Literatura

[1] Aichholzer, O.; Alberts, D.; Gärtner, B.; aj.: *A Novel Type of Skeleton for Polygons.* ročník 12, 1995: str. 752–761.

[2] Du, Q.; Faber, V.; Gunzburger, M.: *Centroidal Voronoi Tessellations: Applications and Algorithms*, ročník 41. SIAM Review, 1999, s. 637–676.

[3] Fournier, A.; Fussell, D.; Carpenter, L.: *Computer rendering of stochastic models.* ročník 25, 1982: s. 371–384.

[4] Greuter, S.; Parker, J.; Stewart, N.; aj.: *Undiscovered Worlds – Towards a Framework for Real-Time Procedural World Generation.* 2003.

[5] Hugo, E.: *Perlin Noise.* [Online; navštíveno 02.04.2019].
URL https://web.archive.org/web/20160510013854/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

[6] Lechner, T.; Watson, B.; Wilensky, U.; aj.: *Procedural City Modeling.* 2004.

[7] L'Ecuyer, P.: *Tables of linear congruential generators of different sizes and good lattice structure.* ročník 68, 1999: s. 249–260.

[8] Lindenmayer, A.; Prusinkiewicz, P.: *The Algorithmic Beauty of Plants.* Springer-Verlag Berlin, 1996, ISBN 0-387-94676-4.

[9] Marsaglia, G.: *Xorshift RNGs. Journal of Statistical Software*, ročník 8, 2003.

[10] Matsumoto, M.; Nishimura, T.: *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. TOMACS - Special issue on uniform random number generation*, ročník 8, 1998: s. 3–30.

[11] Müller, P.; Wonka, P.; Haegler, S.; aj.: *Procedural generation of parcels in urban modeling*, ročník 25. Computer Graphics Forum, 2006, ISBN 1-59593-364-6, s. 614–623.

[12] Parish, Y. I. H.; Müller, P.: *Procedural modeling of cities.* Proceedings of the 28th annual conference on Computer graphics and interactive techniques, 2001, ISBN 1-58113-374-X, s. 301–308.

[13] Sharma, K.; Munshi, C.: *A Comprehensive and Comparative Study Of Maze-Solving Techniques by Implementing Graph Theory.* ročník 17, 2015.

[14] Vanegas, C. A.; Kelly, T.; Weber, B.; aj.: *Procedural generation of parcels in urban modeling*, ročník 31. Computer Graphics Forum, 2012, s. 681–690.

[15] Whelan, G.; Kelly, G.; McCabe, H.: *Roll your own City*. 2008: s. 534–535.

[16] Young, S.: *Pixel City*. [Online; navštíveno 02.04.2019].
URL http://www.shamusyoung.com/twentysidedtale/?p=2940

Příloha A

Obsah CD

Na přiloženém paměťovém médiu je možné najít následující materiály:

- **./cityORG/** - složka, ve které jsou zdrojové soubory aplikace, včetně knihoven potřebných pro spuštění
- **./dokumentace/latex/** - složka, ve které je technická zpráva napsaná v jazyce LaTeX.
- **./dokumentace/pdf/** - složka, ve které je technická zpráva ve formátu PDF.
- **./README.txt** - soubor, ve kterém je návod, jak aplikace přeložit a spustit. Také v něm lze najít ovládání a několik tipů.