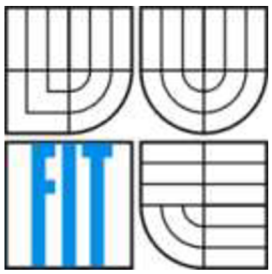
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

Fakulta informačních technologií
Faculty of Information Technology

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

Brno, 2016 Pavel Hamerník

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROCEDURÁLNÍ GENEROVÁNÍ MĚST

PROCEDURAL GENERATION OF CITIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL HAMERNÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. LUKÁŠ POLOK

BRNO 2016

Abstrakt

V této práci se zabývám zkoumáním metod pro procedurální generování terénu, měst a interiérů. Nejdříve jsou rozebrány nejčastější techniky procedurálního generování, poté použitelné techniky či algoritmy pro generování terénu a měst.

Pomocí těchto metod a technik se snažím vytvořit 3D aplikaci demonstrující generování světa, který obsahuje terén a město. Aplikace v závislosti na počátečním celém číslu dokáže generovat různé městské scény.

Abstract

My bachelor thesis covers studying methods of procedural generation of terrains, cities and interiors. First the most common techniques of procedural generation are discussed, then more useable techniques for procedural generation of terrain and cities.

With the assistance of these methods and practices i endevour to make a 3D application that simulates generation of world, which consists of terrain and city. Application generates different urban scenes based on input integer value.

Klíčová slova

procedurální generování, L-Systém, Perlinův šum, generování města,C++

Keywords

procedural generation, L-System, Perlin noise, city generation, C++

Citace

Hamerník Pavel: Procedurální generování měst, bakalářská práce, Brno, FIT VUT v Brně, 2016

Procedurální generování měst

Prohlášení

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

……
Pavel Hamerník
17.5.2016

Obsah

1 Úvod

Pojem procedurální generování se vztahuje ke generování obsahu za běhu aplikace. Často se tento pojem používá v počítačových aplikacích, herním a filmovém průmyslu. V těchto konkrétních oborech dokáže výrazně snížit náklady na výrobu. Vezměme si například herní průmysl, obvyklý způsob vytváření scény vyžaduje 3D designera, který stráví hodiny práce na vytváření geometrie dané scény. Dále je zapotřebí scénu opatřit texturami to je práce pro grafiky. Tento proces vývoje vyžaduje prostředky které by mohli být použity jinde pokud by scéna byla generována algoritmicky.

Ačkoli procedurální generování není použitelné kdekoli, existuje řada příkladů v nejrůznějších oblastech, kde procedurální generování dokázalo být efektivní. Použití se liší od procedurálně generovaných 3d objektů, přes procedurální syntézu zvuku, až po případy kde je procedurální generování použito na vytvoření veškerého obsahu (příklady jsou filmy jako Tron, Avatar, počítačové hry jako Spore nebo No Mans Sky).

Algoritmy procedurálního generování musí být navrženy tak, aby při stejných počátečních podmínkách vyprodukovali stejné výstupy.

2 Analýza problému

2.1 Typy procedurálního generování

Obor procedurálního generování je velice široký. Existuje řada metod a technik, některé se hodí více na daný problém než jiné. Níže jsou uvedeny nejčastěji používané procedurální metody a jejich stručný popis:

2.1.1 Fraktály

Přírodní objekty nelze lehce popsat konvečními geometrickými funkcemi, protože bývají nepravidelné, roztříštěné a vykazují složitost kterou nelze porovnat normální geometrií. Ale tyto objekty lze popsat pomocí matematického oboru nazývajícího se fraktální matematika. Základním principem fraktálu je že obsahují sobě-podobnost. To znamená, že jakkoli daný útvar pozorujeme, změnou měřítka či rozlišení, vidíme stále se opakující charakteristický tvar.
Jako každá procedurální metoda fraktálový tvar je definován algoritmem pro jeho vytvoření. V případě fraktálů se jedná o rekurzivní algoritmy. Každou další rekurzí tohoto algoritmu se produkuje detailnější verze počátečního tvaru. Sobě-podobnost je dosažena generováním stejných tvarů případně vzorů na čím dál menších měřítkách jak rekurze postupuje. Teoreticky není omezení kolik rekurzí se může provést a díky tomu fraktální tvar disponuje nekonečným detailem.

2.1.2 Formální gramatika

Gramatika se skládá z množiny přepisovacích pravidel, nad formální pomocí kterých se generují řetězce dle zadaného počátečního "symbolu". Proto si lze gramatiku představit jako generátor formálního jazyka. V souladu s procedurálním generováním lze gramatiku použít na generování jazyka, který pak představuje nějaký druh rozložení designu modelu. Příklady použití jsou tvarové gramatiky [2] používané pro vytvoření obrazů, či budov a L-Systémy dále v dalším odstavci.

2.1.3 L-Systémy

Lindermayerův systém nebo zkráceně L-systém je formální gramatika definovaná A.Lindermayerem jako matematická teorie biologického růstu. L-systémy byly původně určeny k studování replikace bakterií nebo růstu primitivních organizmů jako jsou řasy. Použití L-systému se v současné době hojně používá v počítačové grafice jako generování realistických modelů rostlin.

Hlavní koncept L-Systémů je přepisování. Přepisování je metoda pro definování složitých objektů přepisováním částí počátečního objektu pomocí pravidel pro jejich přepis.

2.2 Generování terénu

Terény mohou mít různé reprezentace. Hlavní rozdělení je na pravidelné a nepravidelné trojúhelníkové sítě. Pravidelné trojúhelníkové sítě mají výhodu protože je s nimi snazší práce a zaberou pravděpodobně méně místa na jejich uchování. Na druhou stranu nelze je použít na převisy v terénu.
Nejčastější reprezentace terénu bývá výškovou mapou (height map). Výšková mapa bývá zpravidla dvojrozměrné pole kde pro každý bod terénu na osách (X,Z) jsou uloženy hodnoty osy Y.

2.2.1 Metoda Diamond-Square

Algoritmus funguje na principu rozdělení oblasti na menší čtyři podoblasti dle středového bodu a náhodného upravování hodnot. Tento algoritmus je pak aplikován dokud každý z bodů nemá hodnotu. Algoritmus vyžaduje čtvercové pole hodnot o velikosti 2^N+1. Na začátku algoritmu jsou vygenerovány čtyři hodnoty každá v jednom rohu generované oblasti (pole). Prvním krokem je určení hodnot všech středových čtvercových oblastí, který se rovná průměru ohraničujících bodů plus náhodná odchylka. Poté se určí hodnoty středových bodů všech kosočtverců jejichž hodnota se vypočte jako předchozím kroku. Každou další iterací by se měla snižovat odchylka. V druhém kroku používající kosočtverec může nastat problém pokud se bod nachází na hranici pole. Tento problém lze řešit buď že se vezmou v potaz pouze tři body pro průměr a nebo se použije hodnota z druhé strany pole.

2.2.2 Perlinova šumové funkce

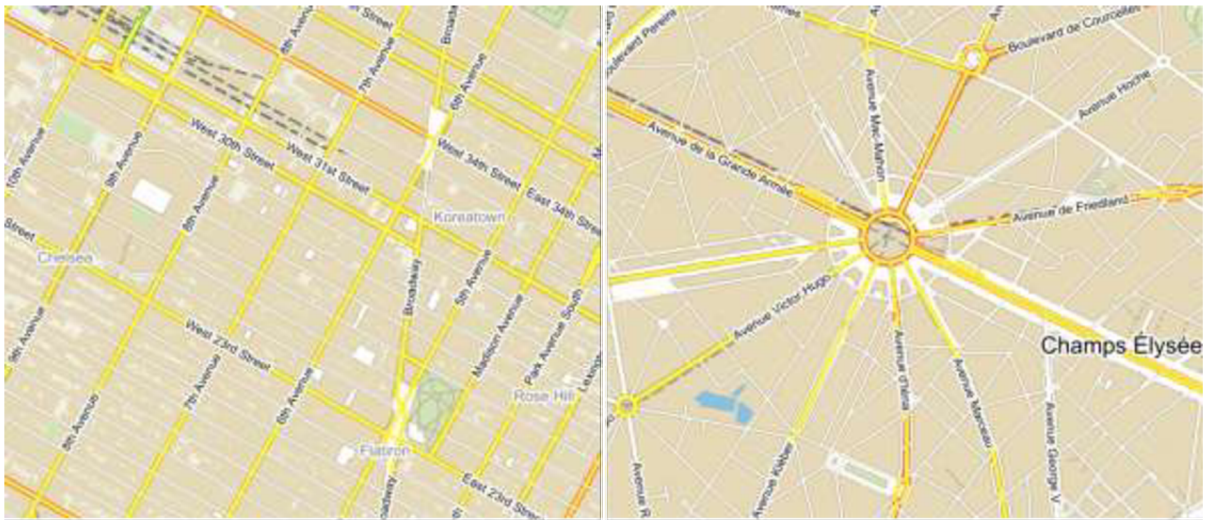
Perlinova funkce je součet několika šumových funkcí (oktáv) , které jsou založeny na pseudonáhodném generátoru čísel, který pro celá čísla vrací různé hodnoty. Šumová funkce vrací interpolovanou hodnotu z dvou výsledků, které nám vrátí psudonáhodný generátor pro celá čísla, mezi kterými se nachází vstup pro šumovou funkci. Jednotlivé šumové funkce se liší velikostí frekvence a amplitudy. Frekvence udává velikost intervalu, na kterém interpolujeme hodnoty generátoru. Amplituda je koeficient, kterým násobíme výslednou hodnotu funkce.

Při generování terénu perlinovou šumovou funkcí se nejčastěji používá kosinová interpolace. Šumová funcke s malou frekvencí a velkou amplitudou nám vytvoří základ s velkými zaoblenými kopci, ke které se pak přičítají další oktávy se zvyšující se frekvencí a snižující se amplitudou, které tyto kopce zvrásní. Perlinova šumová funkce je nejpoužívanější z metod, neboť různými součty různých oktáv můžeme dosáhnout rozličných terénů od skalnatých hor až po roviny s občasnými kopci. Perlinova funkce je periodická funkce, a tak můžeme výšku každého bodu vypočítat nezávisle na ostatních bodech. Proto můžeme vytvořit nekonečně velkou výškovou mapu, sice bude periodická, ale s dobře zvoleným generátorem periodu jen tak v terénu nepoznáme.

2.3 Generování města

Generování města lze rozdělit na několik fází: generování městských komunikací, generování stavebních pozemků a nakonec generování budov.

Silniční sítě jsou klíčovým aspektem města. Počet silnic a jejich uspořádání je dáno historickým vývojem města. Silniční sítě je obtížné zobecnit, protože jednotlivé silnice jsou mezi sebou různě propojeny a součástí velmi složitého systému. Když se podíváme na mapy měst lze pozorovat několik vzorů silničních sítí. Právě tyto vzory jsou klíčovým aspektem k utvoření procedurálního generátoru protože zahrnují strukturu silniční sítě. Existuje značné množství vzorů, které jsou v městech použity. Příkladem vzorů je například vzor "mřížky" zde jsou silnice pravidelné a mezi sebou navzájem kolmé připomínající vzhled mřížky. Konkrétní použité vzory v městech mohou být výsledkem historického vývoje, geografické polohy města, trendu plánování růstu, atd. Vzory měst mohou být zařazeny do několika kategorií. Moderní města USA jsou většinou uspořádány do mřížky. V některých Evropských městech (př. Paříž) je použit soustředný (radiální) vzor, který se většinou nachází okolo památek. Nicméně většina měst obsahuje více vzorů, kde různé vzory jsou v jiných částech města. Jednotlivé příklady vzorů můžeme pozorovat v Obrázek 2.1.



Obrázek 2.1: Příklad vzorů. Vlevo vzor mřížky v New Yorku. Vpravo soustředný vzor Paříž vítězný oblouk.

Budovy jsou objekty které je obtížné procedurálně generovat kvůli jejich individualitě. V moderních městech lze pozorovat velkou škálu budov které se mohou lišit jak jejich účelem nebo stavebním stylem. Účelů budov je mnoho a spolu s jejich geografické rozložení vytváří komplexní systém. Takový systém je obtížné modelovat, ale lze použít zjednodušené řešení. Budovy lze statisticky rozdělit do několika skupin jako jsou komerční, rezidenční a industriální. Takto definované skupiny pak lze modelovat jednoduchým mechanizmem. Vzhled budov a zejména jejich architektura či použitý materiál je často důsledkem architektonického stylu a kulturní vlivu dané doby. Protože by

takové budovy bylo obtížné modelovat je zapotřebí omezit složitost generátoru. Toho lze dosáhnout výrazným zjednodušením modelu budovy případně jeho aproximací.

V následujících sekcích budou popsány algoritmy pro generování sítě silnic a budov.

2.3.1 Algoritmy silničních sítí

Grid Layout

Algoritmus je velmi jednoduchý protože využívá dvojrozměrné pole, kde každý jeho prvek obsahuje náhodné číslo dle kterého pak generátor budov generuje danou budovu. Díky tomu že tento algoritmus používá celočíselné souřadnice je zapotřebí mu stanovit šířku bloku, kde se nachází budova, a silnice mezi bloky. Upravená verze tohoto systému pak může obsahovat sloučení několika bloků do jediného celku a díky tomu vytvoření větší budovy.

Tento přístup byl navržen na generování obrovských měst na dostupném hardwaru v roce 2003, aby pracoval v reálném čase. Ačkoli je tento systém jednoduchý na implementaci, tak nepřipomíná žádné z reálných měst a tudíž se výsledné město jeví velmi uměle.

L-Systém

L-systémy jsou od nepaměti užívané pro modelování přírodních jevů ale jsou i vhodné pro modelování měst díky jejich stručnosti, efektivnosti výpočtu. Použití L-systému pro generování silničních sítí je inspirováno díky podobnosti větvení u generování rostlin. L-systém může být navržen pro řadu dvojdimenzionálních vstupů jako například výšková mapa, hustota populace případně i jiné statistické mapy. Parametry dále mohou zahrnovat šířku silnic, úhel vyhlazování mezi hranami sítě, minimální délku silnice, atd.
Díky své schopnosti vytvářet reálně vypadající města byl algoritmus použit a detaily jeho implementace budou popsány v dalších sekcích.

Simulace založena na zástupcích města

Tento systém pracuje na bázi zástupců a pomocí nich se simuluje vývoj města. Zástupci jako jsou stavební úřad, stavební firmy (silnice,budovy), a mnoho dalších mezi sebou jednají a produktem je daná simulace. Silnice jsou tvořené ze segmentů které jsou sestaveny dle vzoru mřížky. Algoritmus povoluje odchylky od daného vzoru a je nastavitelný parametrem. Odchylka rovna nule vytváří síť silnic která připomíná jednotnou mřížku. Odchylka přibližující se hodnotě jedna naopak vytváří síť silnic připomínající přírodní strukturu. Propojenost jednotlivých segmentů lze nastavovat pomocí parametrů jako je např. šířka mřížky.
Algoritmus produkuje velmi reálné výsledky, dokáže efektivně přecházet mezi vzory silničních sítí. Použití toho přístupu je však velmi nepraktické, protože nevytváří pouze strukturu města ale také jeho

postupnou evoluci. Díky této složitosti je se vygenerování města velikosti malé vesnice velmi náročné a může zabrat řadově až desítky minut (nezahrnuje generování modelů pouze struktury).

Generování založené na šablonách

Vytvoření měst zde probíhá pomocí řady jednoduchých šablon a adaptivní šablony pro populaci. Hlavním konceptem tohoto návrhu je že požadovaná šablona silniční sítě je aplikována na geografickou mapu jako plán a silnice jsou pak deformovány dle lokálních omezení. Pro využití této metody je zapotřebí několik dvojdimenzionálních map. Nejdůležitější mapou je barevná mapa s informacemi o pevnině, vodě a vegetaci. Dále je zapotřebí černobílá mapa elevace. A při použití populační šablony je i zapotřebí mapa populační hustoty. Podle ní se určuje různé úrovně hustoty silniční sítě.
Silnice se od zadané šablony dokážou odklonit velice rychle, aby se vyhnuly překážkám jakou jsou vodní toky a také postupně zatáčí aby se vyhnuli obrovským skokům v nadmořských výškach. Použité šablony je možné nalézt v reálných městech avšak výsledky neodpovídají reálným městům jak z hlediska složitosti tak velikosti.

2.3.2 Generování budov

Geometrické primitivy

Vzhled budov se odvíjí od náhodného čísla které stanoví parametry jako je výška, šířka či počet podlaží. Použití náhodných čísel vedlejších budov může vyprodukovat podobně vypadající budovy. Architektura budovy je potom vytvořena kombinací základních geometrických objektů k vytvoření různých sekcí budov. Každá sekce budovy má potom jiný půdorys. Budova se tvoří shora dolů kdy v nejvyšším patře půdorys tvoří jeden či více základních geometrických objektů a směrem dolů se půdorys může obohatit o další geometrie.

Dělící gramatika

Dělící gramatika je založena na principu tvarové gramatiky. Tvarová gramatika je formální gramatika ale od L-systémů se liší tím že na rozdíl od písmen či symbolů využívá základní primitivní tvary. Přepisovací pravidla v tomto případě fungují že tvar či množina tvarů bude nahrazena jiným tvarem nebo množinou tvarů. Počáteční podmínkou je množina tvarů a připisování probíhá iterativně. Dělící gramatika je velmi realistická a může být navržena i tak, aby vytvořila různé typy architektur.

2.4 Pseudonáhodné interiéry

Generování založené na pravidlech

Přístup pomocí rozložení pomocí pravidel byl navržen T. Tutenelem [8]. Jejich metoda dovoluje vyřešit rozložení místností a také přidat objekty do scény v stejnou dobu. Z počátečního rozložení algoritmus nalezne vhodné umístění pro nové objekty dle zadané množiny pravidel. Relace mezi objekty může být stanovena buď implicitně nebo explicitně. Metoda využívá hierarchické bloky v procesu řešení, proto aby množina elementů které jsou řešeny byly považovány za jeden blok.

Stromová čtvercová mapa

Tato metoda je založena na práci Marsona [3]. Stromová mapa je efektivní a kompaktní způsob zobrazení prvků hierarchické struktury informace. Obecně stromové mapy rozdělují obsah na menší části které zastupují jednotlivé části hierarchie. Hlavním rozdílem stromových map a stromových čtvercových map je způsob jak je provedeno dělení. Účelem těchto map je udržování poměru stran generovaných obdélníků, definovaných jako maximum (šířka/výška, výška/šířka), co nejblíže hodnotě jedna.
Algoritmus dále funguje na principu stanovení obsahů v každé z tří části budovy. A poté použije stromovou čtvercovou mapu aby stanovil kde dané místnosti budou generovány. Toto vytvoří prvotní rozložení ze kterého se pak dle tabulky propojení odvíjejí ostatní možné mistnosti.

2.5 Existující řešení

2.5.1 CityGen

CityGen [9] využití algoritmů pro přirozený růst podobný k CityEngine. Hlavní silnice jsou vytvořeny tak aby určily hranice mezi sousedstvími nebo buňkami v rámci města. Vedlejší silnice obsluhují místní oblast mezi buňkami tím že poskytují přístup k hlavním sílnicím.

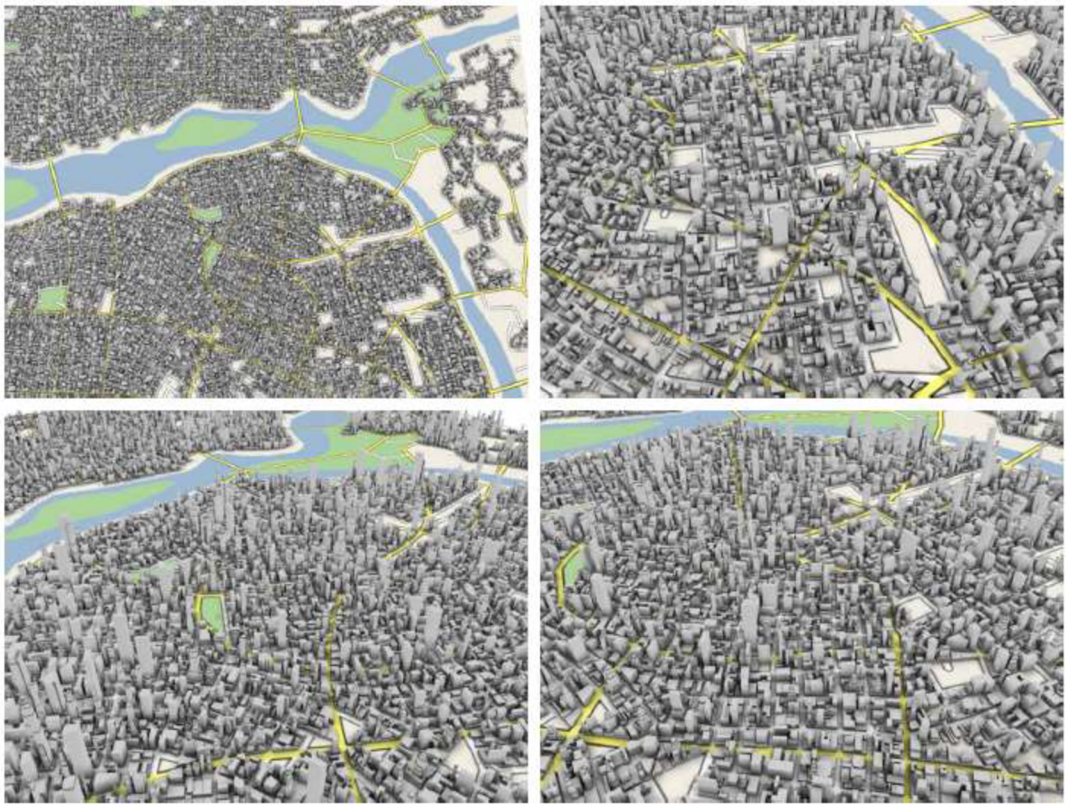
Obrázek 2.2: Ukázka generování CityGen převzato z [9]

2.5.2 CityEngine

Parish a Müller představily CityEngine v dokumentu nazvaném Procedural Modeling of Cities [4]. CityEngine sestává ze sady generátorů, které zahrnují generování silnic, generování budov, generování fasád budov které do sebe zapadají a tvoří tak zřetězené generování města. Hlavním použitým algoritmem pro generování jak silnic tak budov jsou zde L-systémy.

Pro generování silnic používají rozšířenou verzi L-systému nazvanou Self-sensitive L-Systems, která dokáže generovat sítě silnic takovým způsobem, že bere ohled na předchozí růst. Pro generování silnic je potřeba poskytnout parametry v podobě dvojdimenzionálních map. Parametry které vyžaduje CityEngine jsou elevace, vegetace a hraniční oblasti s řekami či jezery.

Při generování budov využívají různé fáze: vytvoření parcel budov, vytvoření geometrie budov a poté vygenerování otexturovaných stěn. Pro vytvoření parcel CityEngine využívá předcházejícího kroku generování silniční sítě. Nejprve získá bloky jež jsou oblasti ohraničené silnicemi. Poté se tyto bloky rozdělují na jednotlivé parcely. Každá parcela musí být přímo spojena se silnicí jinak je pro účely generátoru zahozena. Po stanovení parcel je poté generována geometrie budovu podle parametrického L-systému. CityEngine implementuje několik stylů budov jako jsou mrakodrapy, komerční budovy či rezidence a každý z těchto typů využívá jiné L-systémy.



Obrázek 2.3: Ukázka generování CityEngine převzato z [6]

3 Návrh řešení

Původní záměr bylo vytvořit knihovnu pro procedurální generování města poskytující možnost vytvoření scény z jakékoli aplikace.

3.1 Terén

Základem napodobení terénu 3D prostředí je výšková mapa. Hlavním záměrem aplikace bylo generování města, proto bylo potřeba zvolit takovou metodu generování, která příliš nezdrží zřetězené zpracování. Města se nenachází na visutých horách ale spíše v nížinách na rovném terénu. Díky tomu jsem pro implementaci zvolil Perlinův šum.

3.2 Silnice

Pro generování silnic jsem se rozhodl využít L-systémů protože poskytují střední cestu mezi rychlostí výpočtu a výsledkem připomínající realitu. Výpočet grafů silnic probíhá v 2D prostoru. Pro vytvoření silnice bylo zapotřebí stanovit gramatiku tak, aby brala v potaz již existující silnice. Systém pracuje se dvěma sadami pravidel které jsou globální cíle a lokální omezení. Silnice jsou vedeny dle daných globálních cílů, které reprezentují záměry jež může mít úřad pro další rozšíření. Potom jsou tyto silnice analyzovány dle lokálních omezení a upraveny aby splňovali podmínky světa a aktuální stav silniční sítě.

Globální cíle které systém využívá:

- Silniční úseky jsou vedeny podle vzoru sítě silnice nebo jejich kombinací.
- Silniční úseky jsou vedeny podél vrstevnic terénu.

Lokální omezení kterými se dále navrhované silnice upravují:

- Když se navrhovaný silniční úsek kříží s jiným, rozdělí se a v bodě propojení vytvoří křižovatku.
- Je-li koncový bod úseku v dosahu jiného bodu, tak se změní celý úsek aby směřoval do onoho bodu.
- Je-li úsek příliš krátký ale stále v dosahu teoretické křižovatky, je prodloužen do tohoto bodu.

Vzory silničních sítí byly navrženy pomocí tenzorových polí. Tenzorová pole jsou tenzorová veličina nad daným prostorem. Pro každý bod prostoru je tenzorovým polem je definován tenzor. Pro účely této aplikace je tensorové pole definované na dvojdimenzionálním prostoru. Každý z těchto tenzorů

má dva vlastní vektory které se nemění s jeho transformacemi. A právě toto jsou směrové vektory které hledáme pro posun v prostoru.

Definované základní vzory pomocí tenzorových polí:

Mřížka, je konstantní pro celé pole ale musí se poskytnout směrový vektor *v* který toto pole definuje.

$$T(p) = l * \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}$$

$$l = \sqrt{{v_x}^2 + {v_y}^2} \;\Big|\; \theta = atan\left(\frac{v_y}{v_x}\right)$$

Soustředný vzor, má svůj počáteční bod p_0 který je středem daného vzoru

$$T(p) = \begin{pmatrix} y^2 - x^2 & -2xy \\ -2xy & -(y^2 - x^2) \end{pmatrix}$$

$$x = x_p - x_0 \,|\, y = y_p - y_0$$

Kombinování vzorů je dosaženo zasazením všech definovaných tenzorových polí do prostoru při využití středového bodu jako u soustředného vzoru pro jednotlivá pole.

$$T(p) = \sum_i e^{-d|p-p_i|^2} T_i(p)$$

Tenzorová pole jsou významná pro manuální definování polí pro procedurální generátor avšak v této aplikaci jsou náhodně generována.

3.3 Parcely

Pro vytvoření parcel bylo použito rozdělení bloku podle orientovaného bounding boxu. Po určení orientovaného bounding boxu (dále jen OBB) bloku, definovaného mezi silnicemi, začíná jeho dělení. Dělení probíhá tak, že se OBB rozdělí na dva stejné díly podle středu delší strany a podle těchto dílů se rozdělí polygon určující blok uvnitř. Dělení je potom aplikováno na výsledné polygony rekurzivně dokud dále rozdělené polygony odpovídají podmínkám parcely. Tyto podmínky jsou parcela musí být z některé strany napojena k silnici a celkový obsah parcely musí být vyšší než stanovená hodnota. Využití tohoto přístupu má za výhodu, že je celkový výpočet parcel rychlý.

3.4 Budovy

Pro generování budov bylo zvoleno řešení podobné metodě s primitivy. Na generované parcele byl vymezen půdorys budovy půdorys budovy, tak aby měla budova správnou velikost a zároveň se stále vlezla do dané parcely. Bylo toho dosaženo náhodným výběrem bodů a kontrolami zda se stále nachází uvnitř patřičné parcely. Následně se vygenerovala geometrie dle zadaného počtu pater a vypočteného půdorysu.

4 Implementace

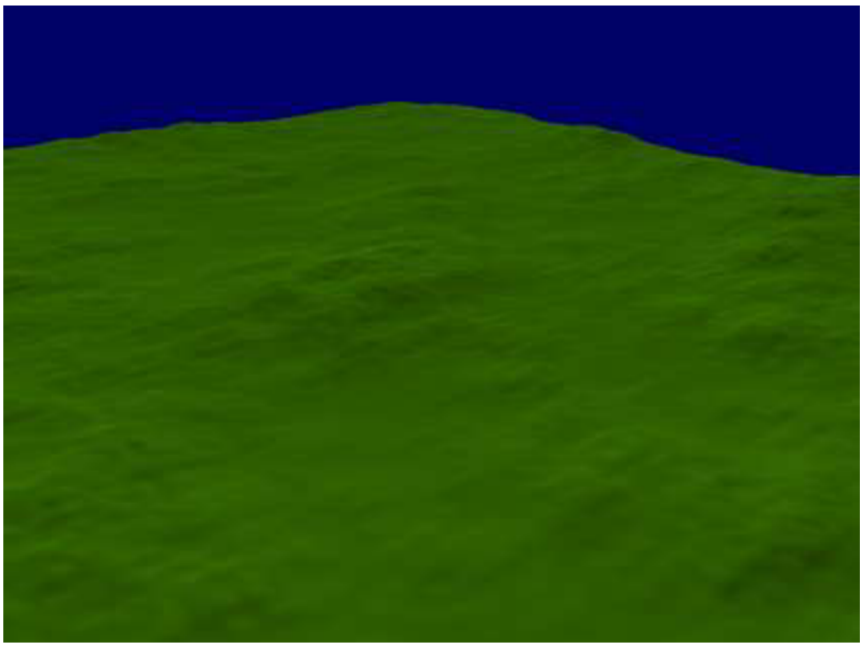
Pro implementaci vybraných algoritmů byl použit programovací jazyk C++ s použitím standardu C++11 a grafická knihovna OpenGL verze 3.3. Aplikace byla vyvíjena v prostředí Visual Studio 2015.

4.1 Terén

Pro algoritmus vygenerování terénu byla vytvořen třída zvaná Terrain. Tato třída poskytuje veškerou funkcionalitu týkající se generováním mapy a zároveň poskytuje rozhraní pro získávání hodnot výškové mapy. Pomocí konstruktoru *Terrain(float width, float length, long seed, float step)* se nastavují hlavní parametry terénu. První dva parametry určují velikost generované mapy. Třetí parametr je určuje počáteční nastavení ("náhodné semínko") pro generovaný terén. Poslední parametry určuje velikost dílčích čtverců trojúhelníkové sítě terénu. Dále lze pomocí metody setPerlinConfig(float amplitude, float frequency, int octaves) nastavit parametry algoritmu Perlinova šumu, více popsáno níže. Další význačnou metodou je *void build()* která generuje geometrii mapy.

Vytvoření terénu funguje jednoduše. Stačí vytvořit objekt třídy Terrain pomocí konstruktoru popsaného výše případně nastavit parametry dalšího generování pomocí setPerlinConfig(...) a dále zavolat metodu *build()* tohoto objektu. Metoda build je hlavní metoda pro generování geometrie terénu a zároveň vytvoří mezipaměť výškové mapy pro souřadnice s celými čísly. Využití mezipaměti má výhodu v rychlém získávání hodnot z výškové mapy.

Hodnoty výškové mapy lze získat metodami height(float x, float z) nebo height(glm::vec2 pos) obě tyto metody vrací stejný výsledek. Metoda funguje na bázi bilineární interpolace nejbližších celočíselných bodů k zadaným souřadnicím. Tato funkce pracuje s mezipamětí výškové mapy i bez ní. Pokud se dané souřadnice ještě nenachází v mezipaměti jsou do ni vloženy.



Obrázek 4.1: Výsledný vygenerovaný terén

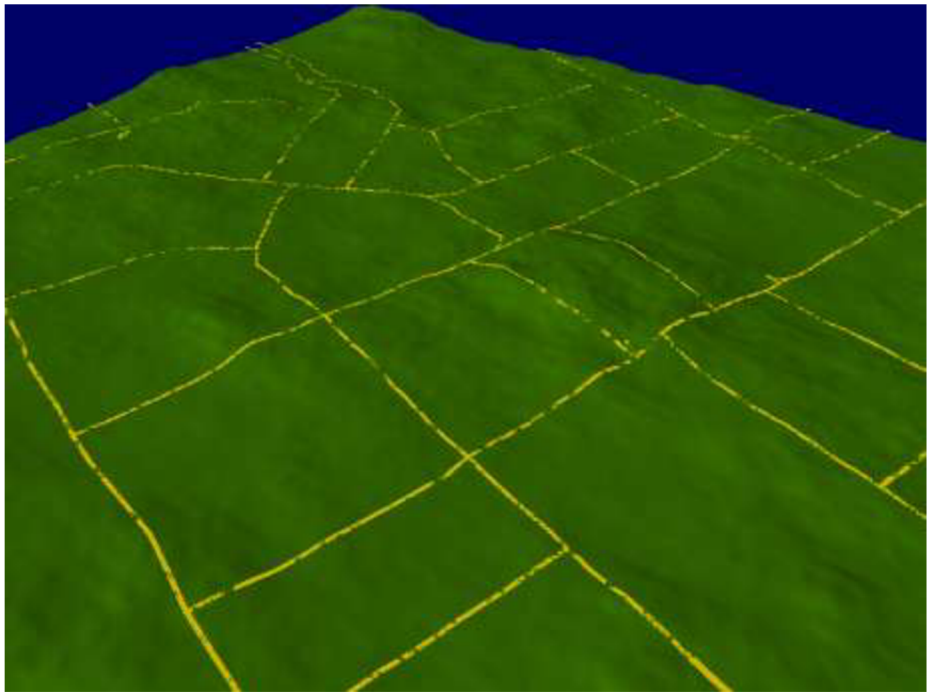
4.2 Silnice

Generování sítě silnic obstarává třída *RoadSystem2*. Pomocí konstruktoru lze nastavit náhodné semínko pro generátor počátečních bodů a využitý terén. Konstruktor náhodně určí v terénu centrum města a nastaví základní konfiguraci pro generování hlavních v vedlejších silnic. Tyto konfigurace lze nastavit i pomocí rozhraní *setMajorConfig(...)* nebo *setMinorConfig(...)*. Obě tyto funkce mají za parametry strukturu *Configuration*, která je vnitřní strukturou třídy *RoadSystem2*. Parametry této struktury jsou *mergeDistance* sloužící jako poloměr pro vyhledání bodu v okolí, *cosineAngle* jeho hodnotou je kosinus maximálního úhlu který má být brán v potaz při vyhledání bodu v okolí, *segmentLen* délka úseku silnice, *separation* vzdálenost mezi počátečními body a *roadWidth* které určuje šířku daného typu silnic. Dále pro vygenerování grafu silnic je zapotřebí zavolat metodu *build()*, která spustí generování podle výše zmíněných parametrů.

Generování metodou *build()* lze rozdělit na několik kroků. Prvním krokem je vygenerování počátečních bodů generátoru. Dalším krokem je vytvoření sítě hlavních silnic podle stanovených počátečních bodů. Poté jsou určeny regiony do kterých se budou generovat vedlejší silnice. Následně jsou pro každý z regionů určeny nezávisle počáteční body. A nakonec samotné generování vedlejších cest.

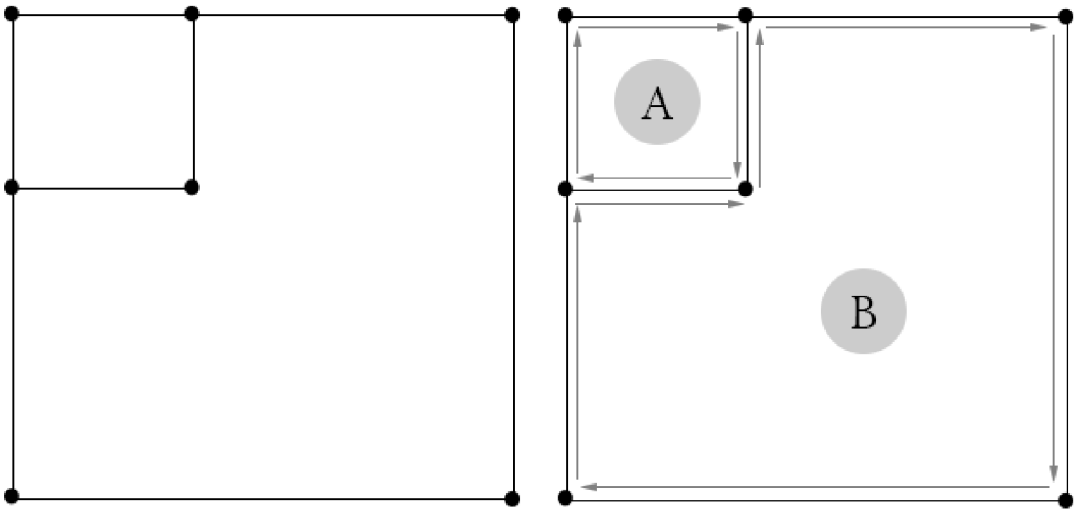
Vygenerování počátečních bodů se provádí pomocí metody *randomSeedsInBounds(...)* metoda má tři parametry: *bounds* určující obdélníkové ohraničení oblasti které určuje interval pro generované body, počet takto generovaných bodů *num* a odkaz na funkci *check* která kontroluje uživatelem definované vlastnosti. Příkladem využití parametru *check* je určení funkce která bude kontrolovat zda se bod nachází v určité výšce.

Až jsou připraveny body nastává první z hlavních částí algoritmu a to je generování grafu hlavních silnic. Získáme si nejvhodnější počáteční bod pomocí metody *nextSeed()*. Tato metoda zkontroluje zda daný bod neporušuje lokální podmínky. Poté je zavolána metoda *traceStreamline(..)* , která vrací křivku s vytvořenými úseky silnice. Tato metoda funguje na principu posouvání se z počátečního bodu k bodům navrhovaným globálními cíly a následného kontrolování lokálních podmínek, dokud není dosažen iterační limit, nebo je porušena podmínka návaznosti křivky. Pokračovat v křivce lze pouze pokud nebylo dosaženo degenerativního bodu globálních cílů nebo zda nově přidaný úsek se nenapojil do některého z bodů na křivce. Při procházení podél této křivky jsou zároveň s minimálním odstupem *separation* generovány nové počáteční body se směrem kolmým na směr procházení.



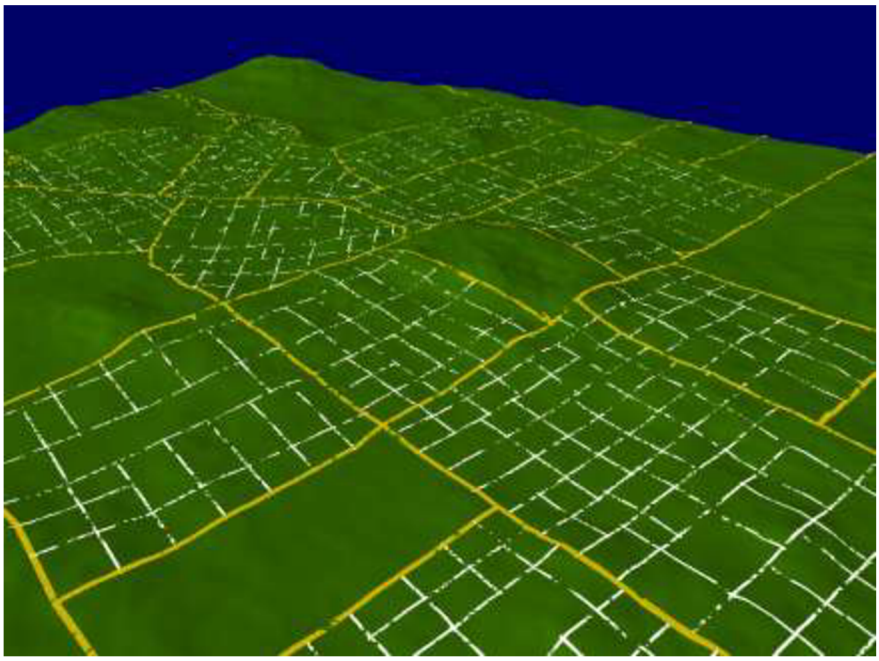
Obrázek 4.2: Demonstrace vytvoření hlavních silnic

Po vygenerování hlavních cest je třeba stanovit regiony. Regiony jsou polygony jejichž hrany jsou definovány úseky silnic z předchozího kroku. Každý úsek může být napojen právě na dva regiony (nejedná-li se o slepou cestu potom oba dva regiony jsou tentýž). Určení regionů probíhá pomocí třídy *RegionBuilder*, které pomocí konstruktoru předáme vygenerované úseky silnic a pomocí *getRegions()* získáme pole tříd Region. Regiony jsou generovány následujícím způsobem. Prochází se spojené úseky a postupně se přidávají do regionu. Až se narazí na křizovatku (tři a více úseků sdílející stejný bod) vybere se další úsek takový který svírá s aktuálním nejmenší úhel. Jakmile je smyčka uzavřena postupuje se na další úseky dokud se neprošlo každým úsekem dvakrát. Demonstraci práce algoritmu lze vidět na Obrázek 4.3. Následně jsou z regionu odstraněny všechny slepé cesty.



Obrázek 4.3: Demonstrace rozdělení na regiony. Vlevo graf ze kterého vycházíme. Vpravo výsledek který nám vygeneruje RegionBuilder.

Pro každý region jsou pak vygenerovány nové počáteční body podél jejich hrany zavolána funkce traceStreamline() s omezením na daný region.



Obrázek 4.4: Terén včetně hlavních (žlutě) a vedlejších silnic (bíle)

Tenzory pro funkci traceStreamline() jsou definovány v souboru Tensors.h. Kde všechny tenzory maji společného předka TensorField který definuje rozhraní pro práci s tenzory. Implementující třídy jsou GridTensorField (vzor mřížky), RadialTensorField (soustředný vzor), PolylineTensorField který vrací tenzor podél zadané křivky, HeightmapTensorField implementující vrstevnice, DecayTensorField který převádí tensorové pole do prostoru, AdditionTensorField definující sumu tensorů. Dále je zde třída EigenField, která dle poskytnutého tensorového pole vrací

vlastní vektory tensoru v zadaném bodě. Jedná se o metody *getMajorEigenVector* a *getMinorEigenVector*.

4.3 Parcely

Když jsou nyní silnice hotové je potřeba určit parcely pro budovy. Nejprve se vygenerují pomocí třídy *BlockGenerátor* z grafu silnic jednotlivé bloky. Generátor bloků funguje na stejném kódu jako již dříve zmíněný *RegionBuilder*. Generování bloků se spustí statickou metodou *createBlocks(edges)*. Metoda vyžaduje v pole všech silničních úseků v argumentu a výsledkem je pole všech bloků které graf obsahuje. Bloky jsou v tomto případě reprezentovány třídou *Block*.

Při generování bloku jsou brány v potaz šířky silnic a proto musí být získaný geometrický objekt ze všech stran ořezán o šířku dané silnice. Tohle zajišťuje konstruktor třídy *Block* očekávající dva parametry pole bodů definující polygon bloku a pole silničních úseků k nim přidruženým. Ořezávaní funguje posunováním bod po bodu okolo celého polygonu pro dvě úsečky definované ve vybraném bodě a hledáním průsečíků s posunutými původními úsečkami.

Po vygenerování všech bloků je zapotřebí parcely rozdělit. K tomu slouží třída *ParcelGenerator*. Hlavní metodou pro generování parcel je statická metoda *generateParcels(blocks,[config])*. Prvním parametrem metody jsou vygenerované bloky z přechozího kroku a druhým nepovinným parametrem je konfigurační struktura jež je definovaná uvnitř třídy *ParcelGenerator*.

Generování probíhá tak, že se z daného bloku udělá třída Parcel. Prvotní vytvoření každé z parcel nastaví každou z hran že je součástí originálu. Díky tomu lze jednoduše určit zda při rekurzivním dělení nepřišla parcela a přístup k cestě. Rozdělení parcely probíhá funkcí *splitByLine(..)* jejíž dva parametry jsou body definující úsečku. Funkce pak vrací pole rozdělených parcel. Abychom věděli, kde je nejlepší parcelu rozdělit je potřeba určit její orientovaný bounding box. Ten je určen postupnými rotacemi polygonu o malý úhel do té doby dokud není nalezen optimální bounding box. Tuto funkcionalitu zajišťuje metoda *bestBoundingBox()*. Parcela je podle něj dělena podle kratší z jeho os. Výsledné nové parcely jsou pak otestovány zdali splňují podmínky přístupu k cestě a minimální plochy. Pokud podmínky splňují je voláno dělení dále rekurzivně, pokud ne původní parcela je přidána do výstupní množiny.

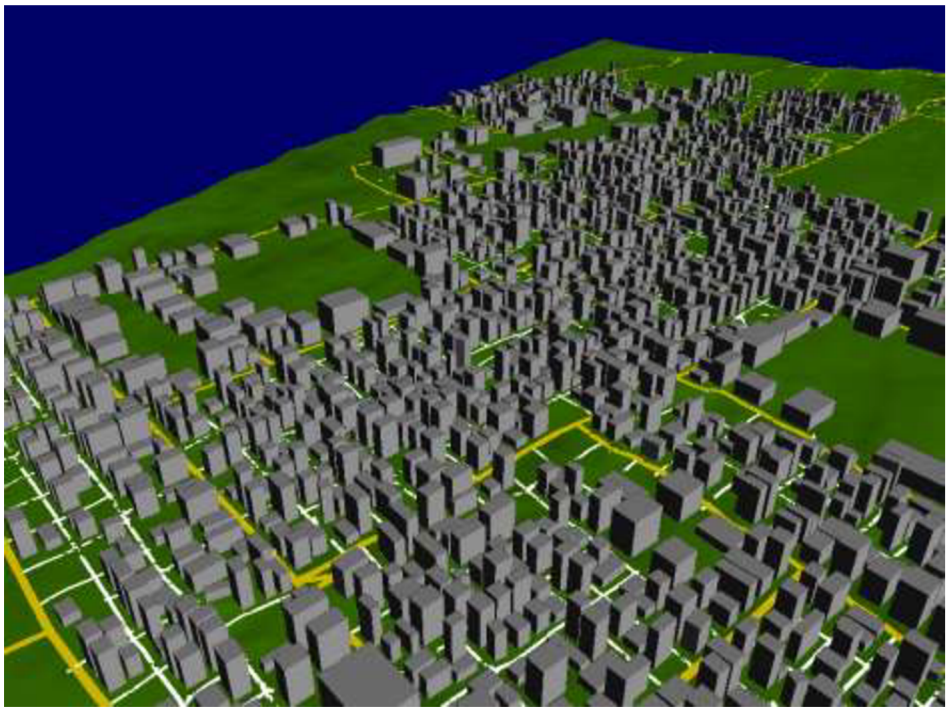
4.4 Budovy

Budovy jsou generovány třídou *BuildingSystem*. Konstruktoru této třídy se předají parcely vygenerované v předchozím kroku, náhodné číslo představující semínko a vygenerovaný terén. Metodou *build()* se spustí generování budov. Pro každou budovu jsou stanoveny náhodné parametry šířka (z pohledu při generování budovy je to potom hloubka) a počet pater. A je zavolán konstruktor

třídy *Building*, ve které je přímo zabudováno generování geometrie objektu. Generátoru jednotlivé budovy jsou předány parametry parcela, počet pater, barva (nevyužito), výška nejvyššího bodu pozemku a hloubka generované budovy.

Generování budovy probíhá nejdříve stanovením která ze stran pozemku je přilehlá k silnici a zároveň nejdelší. Pokud je splněna podmínka minimální velikosti budovy generování pokračuje dál jinak se nastaví příznak invalid a budova nebude započítána do výsledné množiny hodnot. Dle středu a náhodné šířky se stanoví šířka geometrie a dále se otestuje zda se nachází v rozmezí pozemku. Nyní když jsou určeny dva počáteční body objektu ze strany od silnice se stanoví hloubka. Hloubka je zadána jako náhodný parametr avšak obdélníkový půdorys se nemusí někdy vlézt do vymezeného pozemku. Proto je hloubka snižována dokud není dosaženo buď výsledku který se do pozemku vleze nebo selhání. Dále se dle počtu pater generuje obdélníková geometrie objektu.

Jelikož není praktické aby v OpenGL, pro každou budovu byl generován vlastní buffer a shader bylo zapotřebí všechny geometrie objektů sjednotit do jedné v rámci třídy *BuildingSystem*. V metodě *prepare()*, jež slouží k připravení OpenGL bufferů, se potom výsledné geometrie slučují do jednoho velkého celku.



Obrázek 4.5: Výsledné vygenerované město

5 Závěr

Cílem této práce bylo seznámit čtenáře s principy procedurálního generování světa skládajícího se z terénu, města a interiérů. Použití různých procedurálních technik může vést k různým výsledkům od nerealistického ale velmi rychlého přístupu až po simulace trvající desítky minut. Záleží zvláště na druhu aplikace, pro kterou se snažíme daný algoritmus implementovat.

V rámci implementace jsem čtenáře seznámil s principy generování scény s terénem a městem a jejich vzájemné zřetězené zpracování při generování. Výsledná aplikace generuje věrohodná rozložení měst s možností pohybu kamerou v rámci scény. Program na základě celočíselného vstupu dokáže vygenerovat různé scény. Takové scény se pak liší v rozložení města, terénu s různými výškovými mapami a tak dále.

Možné rozšíření algoritmů se nabízí v algoritmu generování sítě silnic a jeho úpravě, aby generování poblíž center měst. Dále možností složitější geometrie budov a namísto náhodně generovaných budov přizpůsobit budovy okolnímu sousedství.
Pro herní účely by se algoritmy generování museli více optimalizovat a pravděpodobně i zrychlit pomocí vláken, aby se čas vygenerování celého světa snížil.

Literatura

[1] KELLY, George; MCCABE, Hugh. A survey of procedural techniques for city generation. *Institute of Technology Blanchardstown Journal*, 2006, 14: 87-130.

[2] MÜLLER, Pascal, et al. Procedural modeling of buildings. *Acm Transactions On Graphics (Tog)*, 2006, 25.3: 614-623.

[3] MARSON, Fernando; MUSSE, Soraia Raupp. Automatic real-time generation of floor plans based on squarified treemaps algorithm. *International Journal of Computer Games Technology*, 2010, 2010: 7.

[4] PARISH, Yoav IH; MÜLLER, Pascal. Procedural modeling of cities. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001. p. 301-308.

[5] Ken's Academy Award. *Ken's Academy Award.* [online]. [cit. 2016-05-10]. Dostupné z: *https://mrl.nyu.edu/~perlin/doc/oscar.html*

[6] CHEN, Guoning, et al. *Interactive Procedural Street Modeling* [online]. [cit. 2016-05-16]. Dostupné z: http://www.sci.utah.edu/~chengu/street_sig08/street_sig08.pdf

[7] ELIAS, Hugo. *Perlin Noise* [online]. [cit. 2016-05-16]. Dostupné z: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

[8] TUTENEL, Tim; SMELIK, Ruben M.. *Rule - based layout solving and its application to procedural interior generation* [online]. [cit. 2016-05-14]. Dostupné z: https://graphics.tudelft.nl/Publications-new/2009/TSBD09a/TBSD09aRB.pdf

[9] *Citygen: Procedural city generation* [online]. 2008 [cit. 2016-05-13]. Dostupné z: http://www.citygen.net/

Příloha A

Obsah CD

Přiložené CD obsahuje zdrojové kódy a soubory aplikace, spustitelnou verzi aplikace ve formátu .exe, bakalářkous práci ve formátu .pdf i se zdrojovým souborem a soubory s návodem k aplikaci.

./ProceduralCityGeneration - složka obsahující veškeré zdrojové soubory
./ProceduralCityGeneration/ProceduralCityGeneration.sln - projekt ve Visual Studiu 2015
./Dokumentace - složka obsahující bakalářskou práci jak v .doc tak v .pdf
./readme.txt - návod pro ovládání aplikace