VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

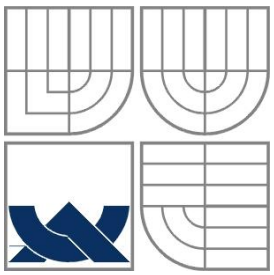
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROCEDURÁLNÍ GENEROVÁNÍ TOPOLOGIE MĚSTA

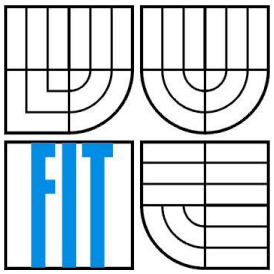
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ ZVĚŘINA
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PROCEDURÁLNÍ GENEROVÁNÍ TOPOLOGIE MĚSTA

PROCEDURAL GENERATION OF CITY TOPOLOGY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ ZVĚŘINA
AUTHOR

VEDOUCÍ PRÁCE — ING. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá problematikou procedurálního modelování. Konkrétně se jedná o procedurální generování topologie města, využívající fraktální geometrii, nahrazovací gramatiky v podobě L-systémů, knihovnu OpenGL pro zobrazení scény s detailnějším popisem způsobů texturování, osvětlení a dalších grafických efektů.

Abstract

This bachelor's thesis is about problematique of procedural modeling. It especially tatgets procedural generation of city topology, using fractal geometry, rewriting gramatics using L-systems, OpenGL library for displaying scene and detailed description of approaches in texturing, lighting and other graphics effects.

Klíčová slova

Procedurální modelování, procedurální generování, fraktální geometrie, L-systémy, OpenGL, osvětlení, mlha

Keywords

Procedural modelling, procedura generation, fractal geometry, L-systém, OpenGL, lighting, fog

Citace

Zvěřina Lukáš: Procedurální generování topologie města, bakalářská práce, Brno, FIT VUT v Brně, 2010

Procedurální generování topologie města

Prohlášení

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

……
Lukáš Zvěřina
17. května 2010

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu Ing. Aleši Láníkovi za vedení a vynaloženou motivaci při tvorbě této práce.

Obsah

1 Úvod

V dnešní době rychle se vyvíjejících informačních technologií, zvyšování kapacity operačních pamětí, vzrůstajících výkonů procesorů, grafických akcelerátorů a ostatních počítačových komponent, ale také díky klesajícím cenám tohoto hardwaru, vzniká a rozvíjí se mnoho samostatných vědních disciplín. Jednou z nich je počítačová grafika obsahující nemalou kapitolu procedurální modelování (generování), která se zabývá vytvářením náhodných objektů pomocí určitého algoritmu. Tyto objekty bývají většinou přírodního rázu, protože příroda je také postavena na náhodě. Uplatnění nachází v oblasti simulátorů, počítačových her, filmovém průmyslu apod.

Současný uživatel osobního počítače se již nespokojí s pouhými konzolovými aplikacemi a ovládáním systému z příkazového řádku, ale vyžaduje plně grafické prostředí a svobodné ovládání systému pomocí klikání myši na ikony. Hlavním cílem programátora by tedy mělo být vytvoření programu, který by měl co nejmenší nároky na uživatele, ale současně na něj nepůsobil dojmem, že je příliš omezován. Proto by měl obsahovat přívětivé uživatelské rozhraní s dostatečným množstvím nastavitelných parametrů při co možná nejjednodušším ovládání.

Tato práce, si klade za cíl, navrhnou a implementovat program, který je přívětivý pro uživatele a zabývá se problematikou procedurálního modelování, konkrétně generováním topologie města a jeho zobrazením na obrazovku.

Následující kapitola patří do teoretické části práce. Popisuje metody a techniky procedurálního modelování. Rozebírá způsoby, jak lze získat trojrozměrný počítačový model objektů. Vysvětluje metody fraktální geometrie, s ní spojený pojem fraktál a jeho různé varianty, a přepisovací gramatiky v podobě L-systémů.

Třetí kapitola se zabývá knihovnou OpenGL, která se používá pro vykreslování scén na obrazovku. Zahrnuje způsoby texturování v OpenGL, rozebírá osvětlení a skladbu světla ze tří základních složek a jak s těmito složkami pracuje Phongův osvětlovací model. Zmínka je zde také o možných způsobech stínování těles a grafických efektech, jako je např. mlha.

Další čtyři kapitoly tvoří implementační část. Jsou v nich postupně popsány metody generování povrchu, jeho možné použití a nastavení. Dále je rozebrán způsob generování silniční infrastruktury, která tvoří základní plochu pro generování budov popsané v šesté kapitole. Poslední kapitola patřící do implementační části pojednává o možnostech generování stromů.

Práci uzavírá osmá kapitola závěr, kde je zhodnoceno splnění zadání, možné pokračování projektu, můj vlastní názor na celou bakalářskou práci a zkušenosti, které mi práce dala.

2 Procedurální modelování

Informace v této kapitole jsou volně převzaty z [1]. Existují tři možné způsoby jak získat trojrozměrný počítačový model nějakého objektu:

- pořízením trojrozměrného snímku
- iterativním modelováním
- procedurálním modelováním

Trojrozměrný snímek je možné získat např. naskenováním objektu prostorovým skenerem nebo vyfotografováním objektu a z fotografií rekonstruovat výsledný obraz. U této metody se však vyskytují jistá omezení, jako je velikost snímaného objektu nebo jeho členitost. Velice těžko se tímto způsobem například získá snímek stromu včetně jednotlivých listů a větví.

Další možností je iterativní modelování, kdy je celý objekt vytvořen animátorem pomocí myši a klávesnice. Jde však o velice náročnou a časově zdlouhavou práci, je také nezbytný určitý talent animátora. Na druhou stranu ale má autor povědomí o tom, co vytváří, a jsou mu známy veškeré detaily práce.

Poslední možností je získat model pomocí vygenerování algoritmem. Tato metoda se nazývá procedurální modelování a dále bude podrobněji popsána. Hlavní úlohou je automatické generování objektů, které jsou vizuálně podobné objektům okolního světa, nebo které se podobně chovají.

Procedurální modelování můžeme rozdělit na tři hlavní oblasti. Fraktální geometrie poskytuje algoritmy pro generování krajin, hor, mraků, kamenů, korálů, atd. Druhou oblastí jsou algoritmy vycházející z gramatik. Hlavním zástupcem jsou L-systémy používané pro generování rostlin a stromů. Systémy částic tvoří poslední oblast a používají se zejména ke generování hejn ptáků, padajících míčů, explozí, simulaci dýmu, ohně, atd.

2.1 Fraktální geometrie

Člověkem vytvořené předměty se vyznačují jistou geometrickou přesností. Například stůl se skládá z ploché desky tvaru kvádru a čtyř stejných kuželů tvořící nohy. V přírodě se ale takové předměty běžně příliš nevyskytují, například mrak na obloze nelze popsat žádnými geometrickými útvary. Právě pro tyto účely vznikla v šedesátých letech minulého století fraktální geometrie, která je někdy označovaná jako „morfologie amorfního“. V klasické Euklidovské geometrii se popisují objekty rovnicemi a invariance transformacemi, jako je posunutí nebo otočení. Naopak fraktální geometrie se zabývá invariancí ke změně měřítka a používá algoritmy, nejčastěji rekurzivní.

Ve fraktální geometrii se nelze obejít bez soběpodobnosti, což vlastně představuje invarianci ke změně měřítka. Soběpodobnou strukturu je možné sestavit z libovolného počtu zmenšených kopií originálu. Například úsečku můžeme rozložit na libovolný počet menších a z nich složit původní.

2.1.1 Fraktální dimenze

Pro objasnění fraktální dimenze je nutné nejdříve vysvětlit pojem topologická dimenze, což lze nejlépe na příkladech. Bod má topologickou dimenzi nula, úsečka jedna, čtverec dvě a krychle tři, dimenze zde tedy v podstatě určuje rozměr objektu. V topologické dimenzi číslo jedna je mírou délka, ve druhé je jednotkou obsah a ve třetí objem. Například pro výpočet obvodu K velmi členitého objektu je vhodné jej rozdělit na N částí o délce l. Tyto hodnoty dosadíme do vzorce $K = N \cdot l$. Při opakování experimentu s jinou hodnotou l a tudíž i N se výsledky K od sebe do jisté míry liší. To je způsobeno zvolenou velikostí měřítka: čím menší l, tím přesnější výsledná hodnota. Pro měřítko

blížící se nule se však K blíží nekonečnu. Aby tedy K mělo smysluplnou hodnotu, je nutné upravit vztah pro výpočet K:

$$K = N \cdot l^D , \tag{2.1}$$

Kde D značí fraktální dimenzi. Pro méně členité objekty je fraktální dimenze D rovna topologické.

objekt	odhad fraktální dimenze
pobřeží	1,26
povrch mozku člověka	2,76
neerodované skály	2,2 - 2,3
obvod 2D průmětu mraku	1,33

Tabulka 2.1 Odhad fraktální dimenze některých přírodních útvarů

Výpočet fraktální dimenze D je závislý na N transformacích a jejich měřítku S, určen je rovnicí:

$$D = \frac{\log N}{\log 1/S} \tag{2.2}$$

a např. pro fraktál tvořený úsečkami platí:

$$S = \frac{1}{N} \tag{2.3}$$

protože pro n-násobné rozdělení je měřítko n-násobně větší. Pro fraktál vytvořený ze čtverců platí:

$$S = \frac{1}{\sqrt{N}} \tag{2.4}$$

neboť při n-násobném zvětšení se plocha jednotkového čtverce zvětší na druhou.

2.1.2 Fraktál

Přesná definice fraktálu je obtížná a existuje jich více. Fraktál je množina, jejíž fraktální dimenze je ostře větší nežli její dimenze topologická [2]. Obecně tedy lze říci, že za fraktál je považován každý obtížně měřitelný objekt s velkou členitostí. Dle [3]: „Fraktál je členitý objekt, jehož struktura je dána opakováním určitého tvaru v různých velikostech“.

Matematicky je popsán množinou vzniklou sloučením nekonečně mnoha kopií sama sebe:

$$A = \bigcup_{i=1}^{\infty} \varphi_i(A) \tag{2.5}$$

Fraktál jako soběpodobnou množinu lze podle druhu soběpodobnosti rozdělit na fraktály deterministické (přesně podobné), které využívají transformace a nepoužívají náhodná čísla, a nedeterministické (statisticky soběpodobné), nazývané jako stochastické neboli náhodné. S těmito druhými se v počítačové grafice pracuje nejčastěji, protože využívají náhodná čísla a jejich výsledky se nejvíce podobají pravé přírodě.

2.1.3 Deterministické fraktály

Generování deterministických fraktálů je velice jednoduché, využívají rekurzivní postupy a transformace posunutí, otáčení a změnu měřítka. Nejčastěji se využívají k vytvoření složitějších povrchů a objemů, které lze potom použít k testování výkonu CPU nebo testování rychlosti sledování paprsku.

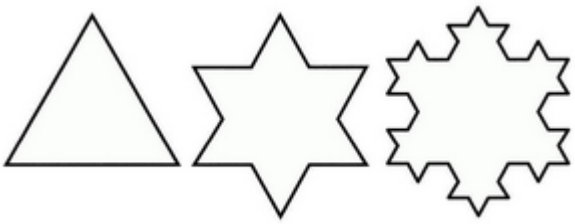
Nejjednodušším fraktálem je Cantorovo[1] diskontinuum. Tato metoda je založena na rozdělení úsečky na tři části, z nichž se prostřední vyjme a na zbylé dvě je aplikován rekurzivní algoritmus. Počet opakování původní množiny je N = 2 a množina se zmenšuje na třetiny (S = 3). Po dosazení do vzorce pro výpočet fraktální dimenze vznikne výsledná rovnice:

$$D = \log 2 / \log 3 \approx 0.63 \tag{2.6}$$

což odpovídá dimenzi mezi dimenzí bodu nula a úsečky jedna.

Obrázek 2.1 Prvních pět iterací Cantorova diskontinua, zdroj [1]

Kochova sněhová vločka byla popsána v roce 1904 Helge von Kochem[2] a je dalším příkladem deterministických fraktálů. Jako základ algoritmu je brán rovnostranný trojúhelník, u kterého se vyjme prostřední třetina každé strany. Nad vzniklou dírou se zobrazí nový trojúhelník s délkou strany rovnající se třetině původní.



Obrázek 2.2 První dvě iterace Kochovi vločky, zdroj [1]

2.1.4 Nedeterministické fraktály

V přírodě se vyskytují jevy a objekty, které jsou velmi často ovlivněny náhodnými procesy. Matematický formalismus, který umožňuje tyto jevy popsat, se řídí stochastickými fraktály, při jejichž generování se uplatňují náhodná čísla. Lze s nimi modelovat zemský povrch, hory, mraky, apod.

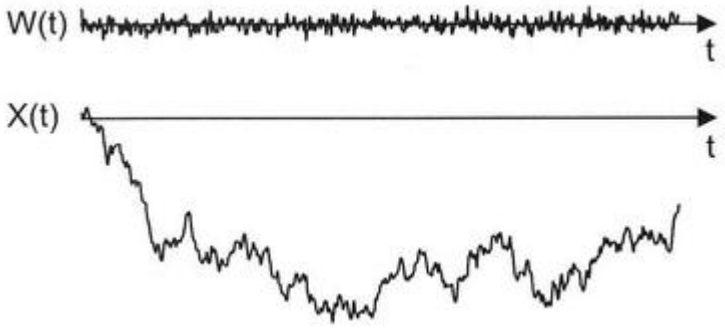
Základem náhodných fraktálů je tzv. Brownův pohyb. V roce 1827 Brown[3] pozoroval chaotický pohyb částeček pylu na vodní hladině. Pohyb způsobují nárazy molekul vody na malá pylová zrníčka. Proto se dnes používá k simulaci přírodních objektů v počítačové grafice.

[1] Georg Ferdinand Ludwig Philipp Cantor (1845 – 1918) německý matematik a logik

[2] Niels Fabian Helge von Koch (1870 – 1924) švédský matematik

[3] Robert Brown (1773 – 1858) skotský botanik

Obrázek 2.3 je graf funkce jednorozměrných náhodných pulsů W(t) s normalizovaným gaussovským rozložením N(0, 1). Tyto pulsy působí v kolmém směru na vodorovně se pohybující bod a jednorozměrný Brownův pohyb je jejich kumulativním součtem.



Obrázek 2.3 Jednorozměrný Brownův pohyb, zdroj [1]

Další metodou Brownova pohybu je náhodné přesouvání středního bodu. Metoda je aplikována na úsečku, která je rekurzivně rozdělena v půlce a její střed je posunut o náhodné číslo. Tímto způsobem vznikne lomená čára. Existují ji složitější varianty této metody aplikované např. na čtverec.

Nejjednodušší metodou je metoda náhodných poruch. Ta je aplikována na dvojrozměrnou matici, v níž je každý bod reprezentován výškou. V dostatečně dlouhém kroku se matice dělí přímkou na dvě poloviny, přičemž jedna polovina je navýšena o náhodné číslo a druhá o toto číslo snížena. Alternativou může být použití kružnice místo přímky, kdy se vše uvnitř kružnice sníží a vně zvýší.

Poslední zmiňovanou metodou je metoda superpozice s využitím goniometrických funkcí. Princip superpozice vychází z elektrotechniky, v níž se kombinuje účinek dvou či více veličin [4]. Např. součtem několika goniometrických funkcí sinus s náhodnými periodami a amplitudami vznikne nová unikátní funkce vhodná např. pro generování terénu nebo kopců.

2.2 L-systémy

L-systémy (zvané též Lindenmayerovy[4]) jsou matematické formalizmy, které vycházejí ze systémů paralelního přepisování řetězců. Principem těchto systémů je přepisování řetězců (posloupnosti symbolů) podle pravidel definované gramatiky. Jednotlivé symboly mají svůj vlastní geometrický význam, např. otočení, posunutí, generování objektu, kreslení čáry, apod.

Hlavním využitím L-systémů je především simulace růstu rostlin, ale lze je také využít ke generování městské infrastruktury, modelování říčních toků, křivek definovaných dělícími schématy. Můžeme je rozdělit na jednodušší dL-systémy a otevřené L-systémy.

2.2.1 dL-systémy

Základním typem L-systémů je deterministický a bezkontextový, v některých literárních zdrojích nazývaný též jako D0L-systém. Nulou v názvu je udáno, že se jedná o tzv. bezkontextový přepis. Determinismus znamená, že pro každý symbol existuje maximálně jedno pravidlo [5] .

Definice 2.1: *Deterministický bezkontextový L-systém je uspořádaná trojice*

$$G = \langle V, P, S \rangle,$$

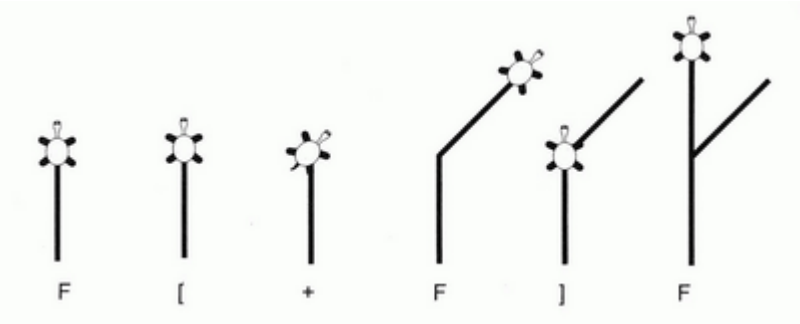
[4] Aristid Lindenmeyer (1925 – 1989) maďarský biolog

kde

- *V je konečná abeceda symbolů A, B, ...,*
- *P je konečná množina pravidel tvaru* $A \to B; A \subset V; B \subset V^*$
- *S je axiom: neprázdná posloupnost symbolů pro kterou platí* $S \in V^+$

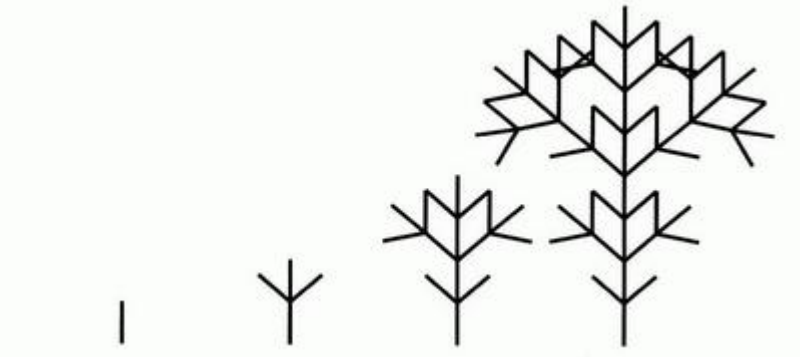
Ke geometrické interpretaci posloupnosti symbolů se využívá tzv. želví grafika, kde želva znamená pomyslné kreslící zařízení. Želva se umístí do výchozího bodu tak, aby orientace byla totožná s vektory os. Poté čte posloupnost symbolů, které reprezentují určitý příkaz. Nejprve je tedy nutné vytvořit definice jednotlivých symbolů abecedy V.

Přidáním tzv. závorkových L-systémů je docíleno uložení aktuálního stavu želvy na zásobník (nejčastěji používaná závorka [pro uložení). Později lze tento stav obnovit vyjmutím ze zásobníků (opačná závorka] pro vyjmutí). Celou situaci demonstruje obr. interpretací řetězce F[+F]F, kde F znamená posun vpřed o krok d, znaménka + nebo – rotaci o předem daný úhel a závorky [] již výše zmíněnou práci se zásobníkem.



Obrázek 2.4 Interpretace řetězce F[+F]F, zdroj [1]

Obsah uzavřených závorek tvoří samostatnou část objektu. Díky tomu závorkové L-systémy dokáží jednoduše generovat rozvětvené struktury, jako jsou rostliny, keře nebo stromy.



Obrázek 2.5 Větvička vytvořená dL-systémem, zdroj [1]

2.2.2 Otevřené L-systémy

Nedeterministické kontextové parametrické L-systémy, zkráceně nazývané jako otevřené, byly především navrženy k simulaci růstu rostlin. Jejich výhodou je otevřenost, která poskytuje možnost obousměrné integrace s okolím. Tímto je umožněn přenos biologických signálů od kořenů k listům a zpět. Rostlina např. poskytuje informace okolí o své pozici a rozložení nebo o množství látek, které produkuje. Naopak např. půda jí poskytuje údaje o úrodnosti, reaguje na výskyt škůdců, množství dopadajících slunečních paprsků, atd.

3 OpenGL

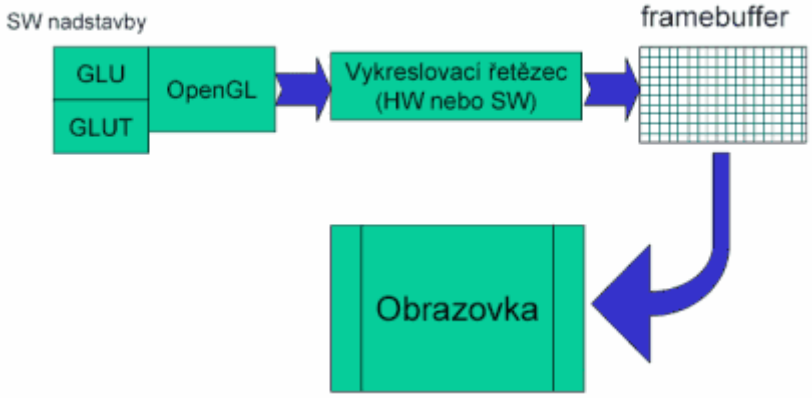
Informace v této kapitole byly volně převzaty z [6] a [7]. „Knihovna OpenGL (Open Graphics Library) byla navržena firmou SGI (Silicon Graphics Inc.) jako aplikační programové rozhraní (Application Programming Interface – API) k akcelerovaným grafickým kartám resp. celým grafickým subsystémům“ [6]. Hlavním cílem bylo vytvořit knihovnu, která by fungovala na různých typech grafických akcelerátorů a také v případě, že se na platformě žádný akcelerátor nevyskytuje. Pro tuto situaci byla vyvinuta softwarová simulace. Dnes ji lze použít na různých platformách, jak na unixových systémech včetně Linuxu, tak i na Microsoft Windows a dalších.

Kvůli nezávislosti na operačním systému, grafických ovladačích a správcích oken neobsahuje žádné prostředky pro tvorbu uživatelského rozhraní GUI (Graphical User Interface), zpracování událostí nebo funkce pro práci s okny, např. vytváření, změna velikosti, uzavírání okna. V těchto případech lze použít nástavbovou knihovnu OpenGL, nejčastěji GLUT (OpenGL Utility Toolkit). Pro dosažení maximální nezávislosti používá také vlastní datové typy, jako GLint, GLfloat, nebo GLdouble, protože hodnoty klasický typů int, float apod. se mohou na různých platformách lišit.

Pro volnou ruku programátora jsou knihovny OpenGL tvořeny tak, aby je bylo možné použít v různých programovacích jazycích. Hlavičkové soubory s novými datovými typy, konstantami a sadou funkcí s vlastním rozhraním jsou primárně psané pro jazyky C a C++. Pro ostatní programovací jazyky bývají většinou automaticky vytvořeny z hlavičkových souborů jazyk C.

V hlavičkových souborech je také deklarováno deset základních grafických primitiv pro vykreslování obrazců. Patří mezi ně izolovaný bod, úsečka zadaná dvěma koncovými body, řetězec úseček (polyčára), smyčka vytvořená z úseček (uzavřená polyčára), trojúhelník, trs trojúhelníků, pás trojúhelníků, rovinný čtyřúhelník, pás rovinných čtyřúhelníků a rovinný konvexní polygon. Z těchto primitiv lze složit jakékoliv složitější těleso.

Každému řídícímu bodu jednotlivých primitiv lze nastavit parametry, jako je barva, šířka, průhlednost, a lze s nimi provádět transformace posunutí, rotace atd. OpenGL se chová jako stavový automat a díky této vlastnosti zůstávají jednotlivé parametry zachovány, dokud je nezměníme. Např. před samotným vykreslováním nastavíme barvu, která zůstane zachována pro všechny vyobrazené objekty. Výhoda tkví v tom, že tak lze snížit počet zadávaných příkazů a změnou jednoho je možné změnit vlastnosti celé scény. Vykreslovaná primitiva mohou být osvětlena nebo pokryta texturou. Dále je možné celou vykreslovanou scénu „skrýt“ v mlze.



Obrázek 3.1 Princip vykreslování objektů v OpenGL na obrazovku, zdroj [6]

Vykreslování scény se provádí procedurálně – voláním funkcí OpenGL vzniká rastrový obrázek uložený v tzv. framebufferu, kde je každému pixelu přiřazena barva, alfa kanál, hloubka

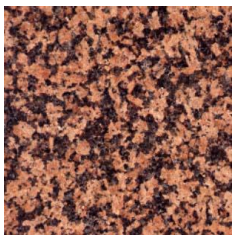
a popřípadě další parametry. Z framebufferu jsou tyto jednotlivé pixely čteny a následně zobrazeny na obrazovce, jak ukazuje obrázek 3.1.

Při spuštění stejného kódu na různých platformách nebo grafických akcelerátorech není zaručeno, že budou přesně identické. Některé pixely se mohou mírně lišit barvou, což může být způsobeno různou přesností reprezentace čísel na grafických kartách nebo jinou bitovou hloubkou uloženou v Z-bufferu (paměti hloubky). Celkové barevné a geometrické podání scény však zůstává zachováno.

3.1 Texturování v OpenGL

Nanášení textur neboli texturování znamená pokrytí povrchu zobrazovaných těles různými obrazci, přičemž se nijak nemění jejich geometrické vlastnosti. Nejčastěji se setkáváme s plošnými texturami (dvojdimenzionálními), ale existují i jednodimenzionální a trojdimenzionální (objemové) textury.

Textury můžeme získat dvojím způsobem. Z klasického rastrového obrázku např. nakreslením, naskenováním nebo vyfocením, nebo za pomoci algoritmu nejčastěji založeného na fraktálních metodách, čímž získáme tzv. procedurální texturu. Procedurální textury mají tu výhodu, že mohou vzniknout již před samotným vykreslováním nebo až za běhu programu, kdy se mohou přizpůsobit vykreslovanému objektu např. rozměry nebo orientací plošky. Tyto textury ale OpenGL přímo nepodporuje a musí se implementovat ručně.



Obrázek 3.2 Ukázka procedurální textury

Textury se kromě klasického obarvení povrchu využívají hlavně tam, kde je třeba simulovat složitější strukturované povrchy. Např. při zobrazení domu lze vykreslit fasádu a na ni jednotlivá okna, při vykreslování celého města však vznikne příliš mnoho objektů a vykreslování se tak zpomalí. Tomu je možné předejít vytvořením textury s celou stranou domu, viz obrázek 3.3.

Obrázek 3.3 Rastrová textura domu

V OpenGL lze zvolit různé filtrace textur, režimy mapování textur na plošky, multitexturování a další grafické efekty.

Velkou výhodou je, že se v OpenGL nevyskytuje příliš mnoho omezení. Pozor je ale třeba dát hlavně na velikost textur, kdy každá strana musí být násobkem mocniny dvou, např. 128x128 pixelů nebo 128x256 pixelů. 123x125 není podporováno a program takovou texturu nenačte.

3.2 Osvětlení

Pro výpočet barvy objektů ve scéně se v OpenGL používá tzv. Phongův[5] osvětlovací model. Hlavní výhodou Phongova modelu je, že využívá fyzikální a optické vlastnosti reálného světa tak, aby byl výpočet co nejrychlejší a současně se podobal co nejvíce realitě. Světlo, které na objekt dopadá a následně se od něj odráží, se skládá ze tří složek.

Ambientní složka (ambient) představuje okolní světlo, které není přímo vyzářeno z žádného světelného zdroje. Vzniká odrazem od okolních předmětů a to i několikanásobným, není tedy patrné, odkud přesně vychází. V OpenGL se toto světlo považuje za všesměrové, neboť scénu osvětluje ze všech stran stejně, nezávisle na poloze zdrojových světel. Použitím pouze ambientního světla dochází ke ztrátě 3D efektu a objekt se jeví jako plošný, viz obrázek 3.4.

Obrázek 3.4 Ambientní složka světla, zdroj [6]

Difúzní složka (diffuse) vyjadřuje světlo dopadající na objekt z konkrétního světelného zdroje, které se od tohoto objektu odráží do všech směrů se stejnou intenzitou. Tímto způsobem lze vytvářet matné materiály. V OpenGL je diffuse upravena tak, že výsledná barva je vytvořena pouze za pomoci polohy objektu a zdroje světla. Poloha pozorovatele na ni nemá vliv, což zjednodušuje a urychluje výpočty.

Obrázek 3.5 Difúzní složka světla, zdroj [6]

Poslední složkou jsou odlesky (specural). Paprsek světla se v okamžiku jeho dopadu odráží dle zákona dopadu a odrazu, nikdy se však neodrazí ideálně a dochází tak k rozptylu. „Phongův osvětlovací model tento fenomén modeluje tak, že se stanoví koeficient změny intenzity odraženého světla podle úhlu odklonu počítaného odraženého paprsku od paprsku ideálně odraženého. Vektory těchto dvou paprsků se vynásobí a výsledek se umocní zadaným koeficientem“ [6]. Obrázek 3.6 ukazuje čajovou konvičku pouze s odlesky.

Obrázek 3.6 Ukázka odlesků, zdroj [6]

[5] Bui Tuong Phong (1942 - 1975) vietnamský badatel a průkopník počítačové grafiky

Pro vykreslení reálních těles se téměř vždy používá kombinace jednotlivých složek světla. Obrázek 3.7 představuje situaci, na které jsou použity všechny tři složky, a proto se nejvíce podobá realitě.

Obrázek 3.7 Čajová konvička s použitím všech tří složek světla, zdroj [6]

3.2.1 Výpočet Phongova osvětlovacího modelu

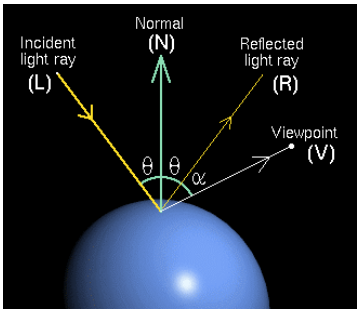
Celková intenzita světla I se vypočítá z parametrů zdrojového světla a nastavení materiálu objektu, a to zvlášť pro každou barevnou složku RGB (red, green, blue). Ve skutečnosti se tedy počítají tři intenzity I_{RED}, I_{GREEN}, I_{BLUE} a každému barevnému kanálu světla i materiálu lze nastavit jinou hodnotu.

$$I = c_a I_a + c_d I_d (NL) + c_s I_s (VR)^n \tag{3.1}$$

kde

- I představuje celkovou intenzitu světla
- I_a je intenzita ambientní složky
- I_d je intenzita difúzní složky
- I_s je intenzita odlesků
- c_a je koeficient materiálu pro ambient, může být v rozmezí 0 až 1, kdy nula znamená nulovou odrazivost, jednička naopak úplnou odrazivost (platí i pro zbylé koeficienty)
- c_d je koeficient materiálu pro difúzní složku
- c_s je koeficient pro odlesky
- N je normála k povrchu tělesa
- L je vektor světelného paprsku
- V je vektor z povrchu tělesa k pozorovateli
- R je vektor ideálně odraženého paprsku
- n je exponent udávající míru lesklosti

Význam součinu vektorů NL znázorňuje obrázek 3.8. Pokud jsou tyto vektory lineárně závislé (rovnoběžné), světelný zdroj je přímo nad osvětleným tělesem a intenzita je tedy maximální.



Obrázek 3.8 Vektory použité v Phongově osvětlovacím modelu, zdroj [6]

3.2.2 Stínování

OpenGL rozlišuje dva základní typy stínování: konstantní a Gouraudovo. Existuje i Phongovo stínování, které ale OpenGL přímo nepodporuje a je nutno jej naprogramovat ručně.

Konstantní stínování je jednoduší a také rychlejší. Princip stínování spočívá ve výpočtu barvy pro celou plošku (nejčastěji trojúhelník) a v následném vykreslení plošky touto barvou. Metoda je nevýhodná zejména při stínování oblých těles, kdy jsou jasně patrné přechody mezi jednotlivými ploškami.

Viditelnost hran se částečně snaží eliminovat Gouraudovo stínování. Barva se zde nepočítá pro celou plošku, ale pro jednotlivé vrcholy. Ploška je poté vyplněna barevným přechodem mezi nimi. Gouraudova metoda také není dokonalá, protože barva se počítá pouze na vrcholech a světelné podmínky na ploškách jsou ignorovány. Toto může být patrné při vykreslování velkých ploch, na kterých jsou viditelné absence odlesků, a nemá tedy smysl pro ně odlesky používat.

3.3 Mlha

Jeden z grafických efektů, které OpenGL podporuje, je mlha (fog). Mlha funguje na principu míchání barev, tzn. objekty nejblíže k pozorovateli (kameře) mají svou původní barvu zadanou např. funkcí glColor*(), texturou nebo materiálem. Naopak objekty vzdalující se od pozorovatele se postupně mísí s barvou mlhy. Nejvzdálenější předměty už nejsou viditelné a přebírají barvu mlhy.

Obrázek 3.9 Ukázka obrázku zahaleného v mlze, zdroj [6]

Hustota mlhy závisí na již zmíněné vzdálenosti od pozorovatele, použité funkci pro výpočet mlhy a nastavením jejich parametrů. V OpenGL jsou implementovány tři funkce pro výpočet:

- **lineární funkce** – je nejjednodušší, ale nejméně odpovídá realitě, vypočítá se $f = (end - z)/(end - start)$, kde *start* a *end* určují rozsah pro zobrazení mlhy a z představuje vzdálenost objektu
- **exponenciální funkce se záporným koeficientem** – nejpoužívanější, náročnější, ale výsledky více odpovídají realitě, vzorec pro výpočet $f = e^{-density*z}$, kde *density* je hustota mlhy a z opět značí vzdálenost objektu
- **exponenciální se záporným koeficientem na druhou** – nejsložitější, nejvěrnější realitě, pomalý výpočet, intenzita roste se čtvercem vzdálenosti, $f = e^{-(density*z)^2}$

Vzhledem k tomu, že výpočet se neprovádí pro jednotlivé vrcholy jako u stínování, ale počítá se pro každý fragment, je dnes tento efekt implementován přímo na grafických akcelerátorech. Využívá se např. ve hrách pro rychlejší vykreslování, kdy se nemusí zobrazit celá scéna, ale pouze určitá vzdálenost od pozorovatele.

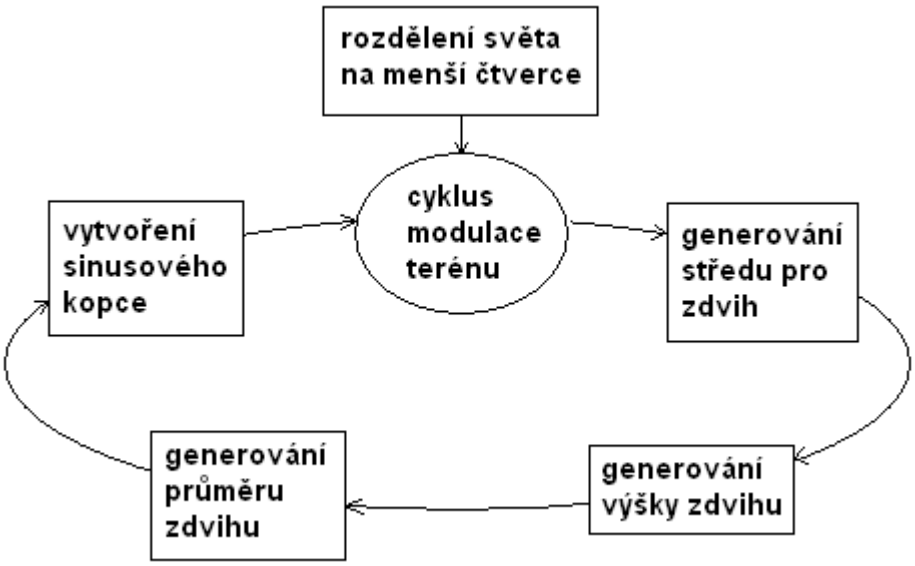
4 Generování terénu

Jako základ pro budoucí městskou zástavbu je nutné mít na čem stavět, proto mým prvním krokem bylo vymodelování světa s deformací terénu. Existuje mnoho algoritmů pro deformaci, např. algoritmus *midpoint* nebo perlin *noise*, já si ale vybral poněkud méně známý způsob, a to metodu superpozice s použitím modulace goniometrických funkcí, konkrétně funkce sinus.

4.1 Implementace terénu

Podle zvolené velikosti světa ve tvaru kolmého čtyřúhelníku je celá plocha pomocí funkce `generate_surface` rozdělena na menší čtverce, které se uloží do matice. Tímto vzniká plochá čtvercová síť, o kterou se dále stará funkce `generate_height`.

Ta je volána cyklem for, počet provedení cyklů si může navolit sám uživatel. Doporučuji používat okolo 50 cyklů, kde je zvlnění terénu dosti patrné. Funkce potřebuje čtyři parametry: pozici středu kopce (souřadnice x a y vrcholů čtverců uložených v matici), maximální výšku kopce zvolenou uživatelem (opět doporučuji používat hodnoty v rozmezí 5 až 15m) a průměr kopce odvozený z velikosti světa. Obrázek 4.1 znázorňuje celý cyklus ve zjednodušené formě.



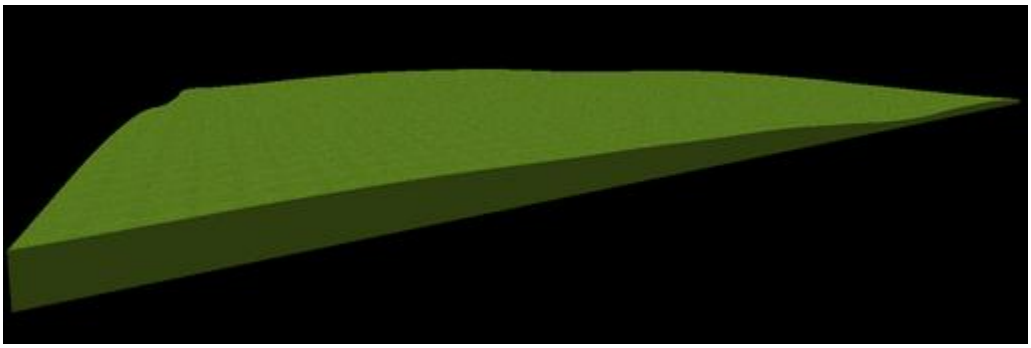
Obrázek 4.1 Zjednodušené schéma generování terénu

Poté se pracuje pouze se čtvercem o délce strany průměru kopce. Uloží se současná výška bodů a následně se zvednou body na čtverci podle funkce sinus ve vertikálním i horizontálním směru. Tímto je ale dosaženo dvounásobku zadané výšky kopce a půdorys netvoří kruh. Proto je nutno projít celou plochu znovu, snížit výšku o polovinu a body vně kruhu vrátit na výšku původní.

4.2 Použití a vliv nastavení

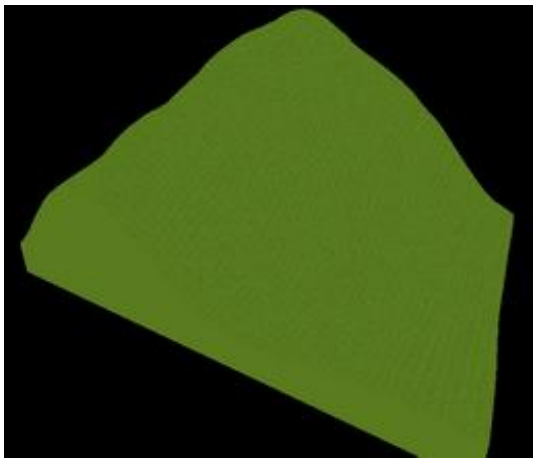
U terénu má uživatel možnost nastavit již výše zmíněné dva parametry, maximální výšku sinusového kopce a počet generovaných kopců. Jejich hodnoty mají velký vliv na budoucí podobu celého města. Je třeba si uvědomit, že zobrazené město má v základním nastavení rozměry 2x2 kilometry, proto by měly námi zvolené hodnoty odpovídat realitě.

Zde je vygenerovaný povrch podle základních parametrů, dle mého názoru pro město s běžnou zástavbou zcela odpovídající. Obsahuje padesát kopců s maximální výškou pět metrů.

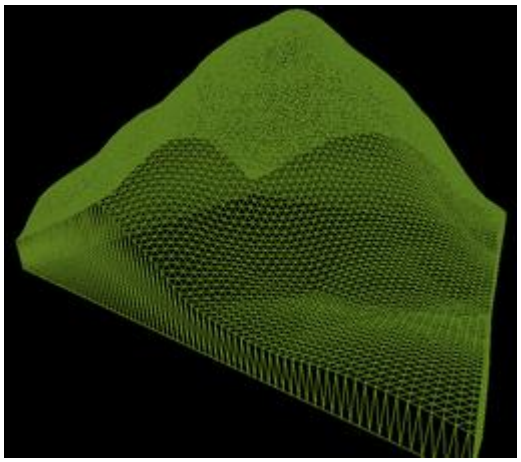


Obrázek 4.2 Vygenerovaný povrch s 50ti kopci a výškou 5m

Menší výšku než pět metrů nemá smysl použít, protože zvlnění pak není příliš znatelné. Další ukázka je opět s padesáti cykly, ale s maximální výškou kopce padesát metrů. Detaily jsou lépe patrné na drátovém modelu. Toto nastavení není vhodné pro budování domů ani silnic, ale bylo by např. možné je použít pro zjednodušené zobrazení větších hor.

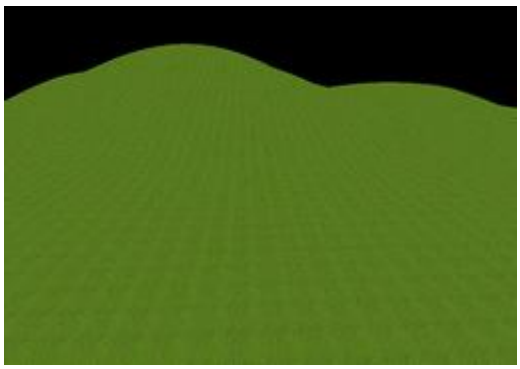


Obrázek 4.3 Povrch s 50ti kopci a výškou 50m

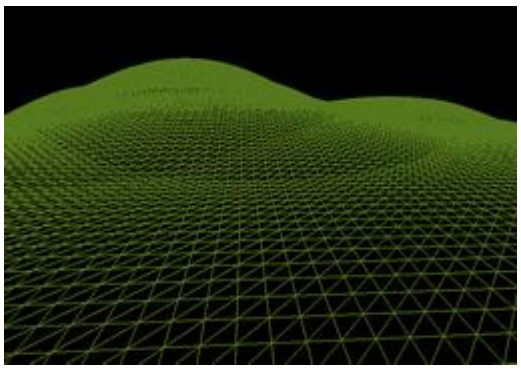


Obrázek 4.4 Drátěný model obrázku 7.3

Poslední ukázka je s pěti cykly a maximální výškou padesát metrů. Pro lepší viditelnost opět přidávám drátěný model. Jak je vidět, pět cyklů nedostačuje, protože nedokáží pokrýt celou plochu - pokud ovšem nechceme generovat např. nížiny s občasným kopcem.



Obrázek 4.5 Povrch s 5ti kopci a výškou 50m



Obrázek 4.6 Drátěný model obrázku 7.5

5 Generování silnic

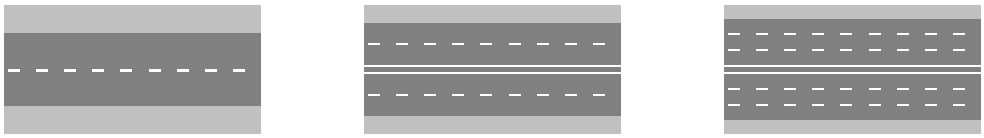
Po úspěšném vygenerování terénu přicházejí na řadu silnice. Ty hrají důležitou roli v dalším dělení města na menší části (sektory). Celý systém je navržen tak, že se generují rovnoběžné vertikální a horizontální silnice a následně v jejich průsečících křižovatky. Výsledkem je šachovnice s různě velkými poli.

5.1 Implementace silnic

Základem je funkce `generate_line`, která potřebuje dva parametry, a to počet vertikálních a horizontálních silnic, které se mají generovat. Počet je odvozen od velikosti světa a počtu kvadrantů na sektor (viz kapitola 6.1 Sektory a kvadranty) tak, aby byly dodrženy určité rozestupy mezi cestami. Uživatel ale může tyto parametry ovlivnit změnou množství silnic v procentech, kdy 100% je základní hodnota a odpovídá asi 50 metrovým rozestupům na kvadrant mezi silnicemi.

Funkce spočítá maximální rozestupy a z této hodnoty náhodně určí skutečné mezery mezi cestami. Minimem je šířka silnice, aby nedocházelo k překrytí. V případě základního nastavení (100% silnic a dva kvadranty na sektor) jsou rozestupy v intervalu mezi šířkou silnice a sty metry.

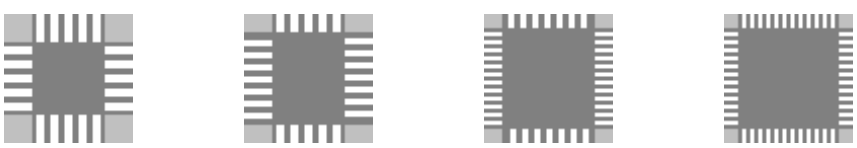
K dispozici jsou tři druhy silnic, jednoproudová, dvouproudová a tříproudová. Každá má navíc po stranách ještě pruh znázorňující chodník. Generátor vždy vybere jednu z možností. Funkce tyto parametry uloží do struktury `line` a ty zapíše jako vektory do paměti.



Obrázek 5.1 Ukázka textur silnic (zleva) jednoproudová, dvouproudová, tříproudová

Jelikož není terén rovný, ale kopcovitý, musí jej silnice kopírovat. Nejjednodušší způsob je silnice, které byly doposud celistvé a táhly se od začátku po konec světa, rozdělit na menší části (stejné jako u funkce `generate_surface` viz kapitola 4.1). To zajišťují funkce `generate_line_v` (pro vertikální) a `generate_line_h` (pro horizontální). Vytvoří matici bodů, ze kterých potom lze sestavit jednotlivé silnice. Navíc je přidán i výškový bod odvozený z výšky terénu pod silnicí, aby jej tak silnice kopírovala. Vzhledem k rychlosti vykreslování scény jsem byl nucen zvolit větší rastr dělení silnic, a proto se může stát, že při výrazně kopcovitém terénu občas povrch přes silnici prosvítá.

O určení středu křižovatek se stará funkce `generate_intersections`, která projde jednotlivé vektory silnic a nalezne jejich průsečíky. S nimi dále pracuje funkce `generete_crossroads`. Zjistí, kolikaproudové silnice se v daném průsečíku kříží a podle nich spočítá rozměry křižovatky. Ty také musí kopírovat terén, takže pro každý roh je dopočítána jeho výška. Stejně jako u silnic může kvůli velkému rastru terén prosvítat. Nakonec je podle typu silnic vybrána správná textura a celá scéna vykreslena.



Obrázek 5.2 Textury křižovatek podle silnic jednoproudová/jednoproudová (1/1), 1/2, 2/3, 3/3

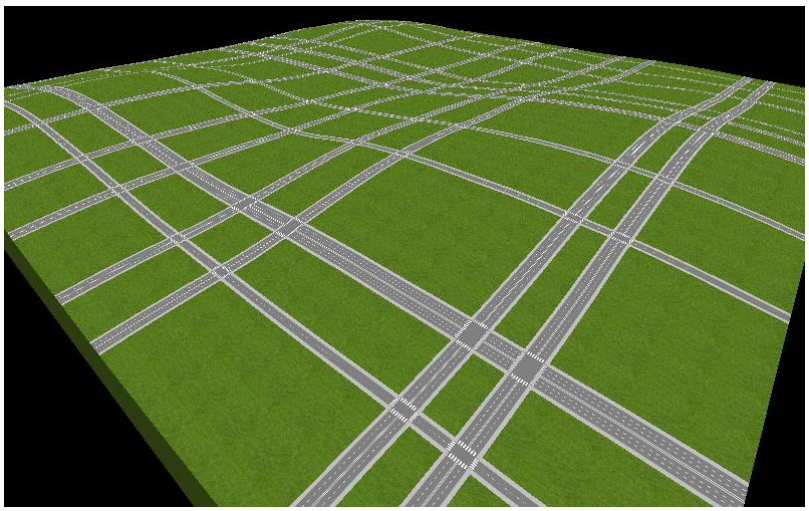
Přikládám ještě zjednodušené schéma generování systému silnic, na němž je celý proces zřetelnější.

Obrázek 5.3 Schéma generování systému silnic

5.2 Použití a vliv nastavení

V předcházející kapitole byly zmíněny dva parametry, které může uživatel nastavit. Jedná se o množství silnic v procentech pro vertikální a horizontální zobrazení. Obecně doporučují tyto hodnoty neměnit a nechat je na 100%, ale pro větší volnost jsou zpřístupněny. Lze volit rozmezí od 0%, kdy se nezobrazí žádné silnice, do 200%, kdy jich je dvojnásob. Musíme ale počítat s tím (viz následující kapitola 6.1), že se budovy generují do prostoru mezi silnice, a proto je třeba volit i vhodné nastavení počtu kvadrantů na sektor.

Obrázek 5.4 zobrazuje standardní nastavení, tedy 100% silnic, dva kvadranty na sektor v každém směru. Pouze velikost světa je zmenšena na 1000 x 1000 metrů, kvůli lepší názornosti. Jak je vidět, mezi silnicemi je dostatek plochy pro generování budov.



Obrázek 5.4 Silnice s terénem při standardním nastavení

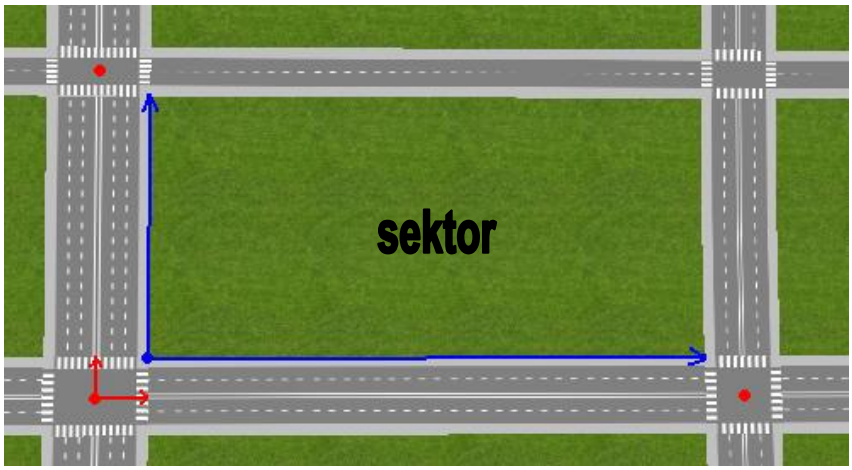
Rozsah nastavení poskytuje poměrně velký prostor pro generování dostatečného množství kombinací, je tedy možné experimentovat. Mohou se tak ale vyskytnout kombinace, které se příliš realitě nepodobají.

6 Budovy

Vzhledem k tomu, že je nyní povrch a silniční systém kompletní, může na řadu přijít generování nejpodstatnější části města, budov. V předcházejících kapitolách jsem použil mnou definované pojmy „sektor“ a „kvadrant“, proto je zde podrobněji rozeberu. Dále popíšu strukturu a princip generování budov a jejich možné nastavení.

6.1 Sektory a kvadranty

Místo mezi dvěma vertikálními a dvěma horizontálními silnicemi tvoří prostor, který jsem nazval sektorem. Je to základní stavební parcela, z níž se dále vychází. O sektory se stará funkce `generate_sectors` a jejím úkolem je zjistit pozici počátku a velikost daného sektoru. Pozice se jednoduše získá z průsečíku spodní horizontální a levé vertikální silnice a přičtením poloviny jejich šířky. Pro určení velikosti se vychází také z průsečíků, ale je potřeba ještě získat dva z rohů sektoru a tentokrát odečíst šířku silnic. Tuto situaci popisuje obrázek 6.1, červené tečky znázorňují potřebné průsečíky pro výpočet a červené šipky vzdálenost, kterou je třeba přičíst, popřípadě odečíst. Modrá tečka znázorňuje počátek sektoru a z ní vycházející šipky velikost sektoru.

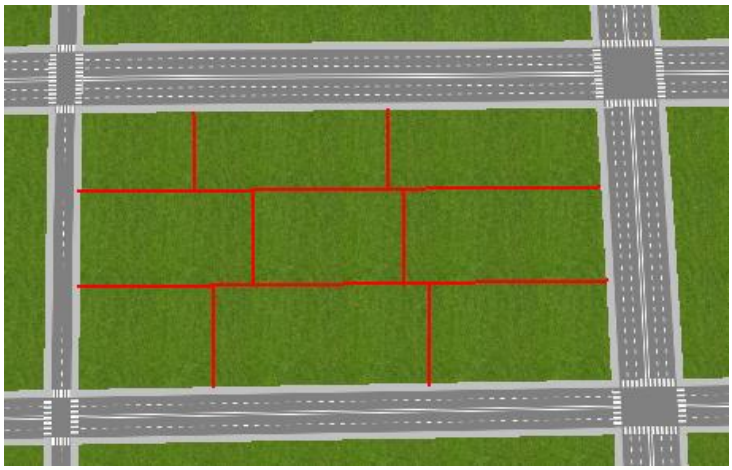


Obrázek 6.1 Znázornění sektoru a jeho parametry

Rozdělením sektoru na menší části vzniknou kvadranty, které ohraničují prostory pro budovy. Počet těchto kvadrantů si může uživatel navolit sám, implicitně jsou nastaveny dva kvadranty vertikálně a dva horizontálně, celkem tedy vzniknou čtyři kvadranty na sektor. Toto nastavení jsem zvolit proto, aby každá budova ležela v jednom rohu sektoru. Lze samozřejmě zvolit i jiná nastavení, a to od jednoho do pěti kvadrantů v každém směru, což odpovídá minimálně jednomu a maximálně pětadvaceti kvadrantům na sektor. Tímto počtem je zaručena dostatečná variabilita, hranici jsem zvolil proto, že při vyšším počtu kvadrantů město nevypadalo dobře.

Celý algoritmus dělení sektorů je následující: funkce `generate_quadrants` načte pozici a velikost sektoru. Podle počtu generovaných kvadrantů určí podílovou velikost a v cyklu generuje náhodné délky kvadrantů. Tato velikost je rovna 50 – 100% podílové velikosti. Poslednímu kvadrantu se již délka negeneruje - připadne na něj zbytek délky sektoru. Cykly jsou dva pro oba směry, proto žádný kvadrant není stejný, pokud se náhodou nevygenerují dvě stejná čísla.

Obrázek 6.2 představuje zobrazení principu dělení sektorů na kvadranty při použití jiného nastavení než implicitního, konkrétně na 3 x 3 kvadranty. Červené čáry značí hranici „pozemku“ jednotlivých budov.

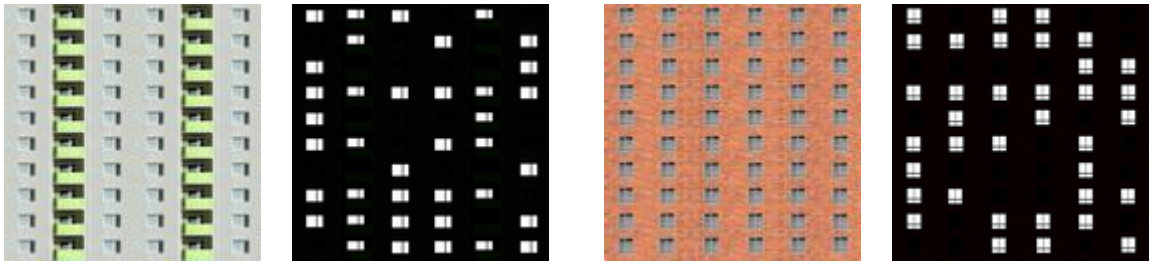


Obrázek 6.2 Princip dělení sektorů na kvadranty

6.2 Generování budov

Generování budov je nejdůležitější a nejzajímavější část celého projektu. Jak jsem již zmínil v kapitole 6.1, základní stavební plochu pro budovy tvoří kvadrant. Na každém z nich může vyrůst až pět odlišných bloků budov tvořící jeden velký komplex.

Hlavní funkcí pro generování je `generate_building`, která prochází jednotlivé kvadranty a zjišťuje jejich velikost. Pokud je plocha příliš malá, nastaví příznak pro volnou plochu a pokračuje dalším kvadrantem. Když zde je plocha dostatečná, náhodně určí, z kolika bloků se bude komplex skládat, a také mu vybere jednu z nabízených textur a jejich noční variantu. Budovy se skládají z místností konstantní velikosti, proto je nutno určit, kolik se jich na daný kvadrant může maximálně vejít. Pokud zůstane na kvadrantu ještě volné místo, vygeneruje se z něj náhodné posunutí počátku budovy.

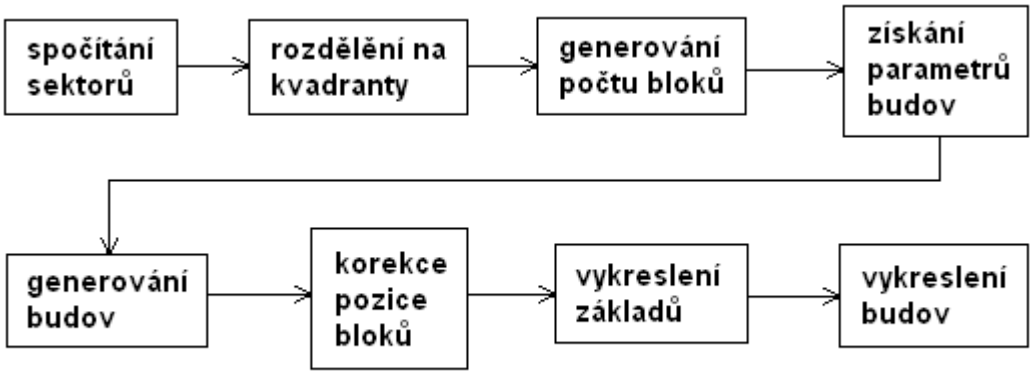


Obrázek 6.3 Ukázka použitých textu budov a jejich noční varianty

Následně se procházejí jednotlivé bloky. Uživatel si může předem navolit až pět druhů budov. Každé z nich nastaví procentuální pravděpodobnost, že se vygeneruje právě tato varianta, a zvolí, kolik pater bude mít. Lze nastavit minimální a maximální počet. Tímto je možné například získat město pouze s nízkými domy a minimem výškových budov, nebo město plné mrakodrapů.

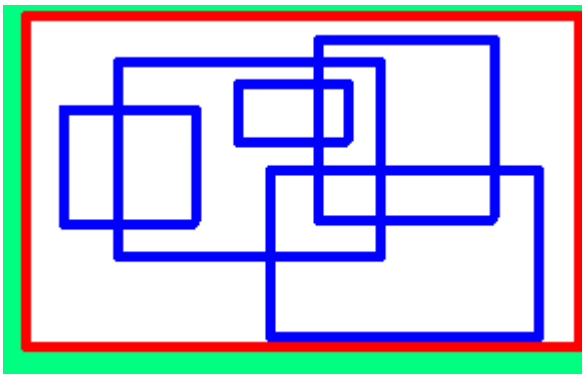
Podle typu budovy se všechny parametry předají funkci `generate`, která určí rozměry a výšku bloku. Pokud nejsou rozměry maximální, znovu provede náhodný posun počátku, aby všechny bloky nezačínaly v jednom bodě. Nakonec spočítá výšku terénu v každém rohu bloku a nejvyšší zapíše spolu s ostatními parametry do struktury `house`, kterou předá zpět funkci `generate_building`.

Po vygenerování posledního bloku porovná, který z nich je položen nejvýše. Touto hodnotou přepíše výšky ostatních bloků, aby všechna patra byla v jedné úrovni a aby některé neleželo částečně pod povrchem. Tato drobnost způsobuje značné zpomalení vykreslování, protože se každá budova musí vykreslovat dvakrát. Poprvé se vykreslí část pod nejvyšším bodem terénu, tedy jakési základy, a podruhé část budovy nad povrchem s patry. Nakonec je provedena korekce v osách x a y, kdy je podle čísla bloku každý mírně posunut, aby při rotaci scény nemohlo docházet k problikávání stěn bloků se stejnou pozicí.



Obrázek 6.4 Schéma algoritmu generování budov

Celou situaci rozmístění lépe znázorňuje obrázek 6.5. Zelená barva značí rozměry kvadrantu. Červený obdélník vyznačuje plochu s maximálním možným počtem bytů konstantní šířky a je náhodně umístěn na zelené ploše. Modré obrazce představují bloky budovy, které mají náhodně určený počet bytů a jsou náhodně posunuty. Bloky se také mohou různě křížit a překrývat.



Obrázek 6.5 Příklad rozmístění bloků na kvadrantu

6.3 Možnosti a ukázky nastavení

Pro shrnutí zopakuji možnosti nastavení budov. Lze generovat až pět typů, kdy každému lze navolit pravděpodobnost, s jakou se bude konkrétní typ vyskytovat. Také je možné stanovit počet pater v rozsahu od minimálního po maximální.

Základní nastavení představuje obrázek 6.6, kde má téměř polovina budov maximálně pět pater a představuje tedy běžné domy. Třetinu tvoří výškové domy s maximálně patnácti patry a zbytek je rozdělen mezi mrakodrapy, kde ty nejvyšší představují pouze 5% podíl budov ve městě. Parametry základního nastavení jsou následující:

- Typ 1: výskyt 40%, min pater 3, max pater 5

- Typ 2: výskyt 35%, min pater 5, max pater 15
- Typ 3: výskyt 10%, min pater 10, max pater 20
- Typ 4: výskyt 10%, min pater 20, max pater 30
- Typ 5: výskyt 5%, min pater 30, max pater 40

Obrázek 6.6 Ukázka budov se základním nastavením

Zde je ukázka města se sto procenty budov pouze v rozmezí jedno až pět pater. Přestože se tato varianta na první pohled zdá nepoužitelná, lze ji využít například pro generování malé vesničky.

Obrázek 6.7 Ukázka města s nízkými domy

Možné je tedy měnit nastavení pouze u výšky budov, protože to je jediný parametr, který neovlivňuje okolí. Navíc ostatních parametrů je tolik, že by byl editor příliš rozsáhlý a nepřehledný.

7 Stromy

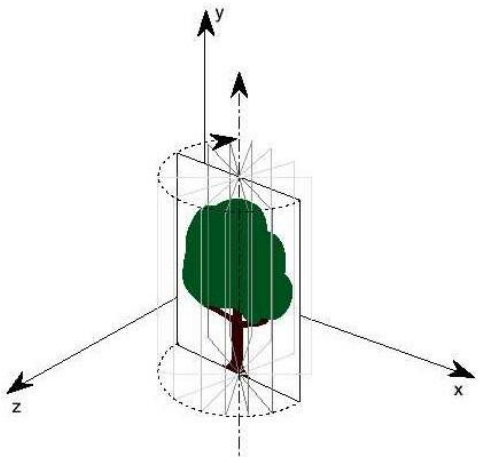
Ve městě by podle mého názoru neměla chybět zeleň, proto mě napadlo projekt rozšířit a přidat stromy. Byla to také příležitost, jak si vyzkoušet další variantu procedurálního generování s použitím L-systémů. Zpočátku jsem zkoušel vsadit stromy do prostředí tak, aby každý z nich byl unikátní. To se ale v důsledku pomalého vykreslování nezdařilo. Strom se skládá z jednotlivých větví (úseček), kterých obsahuje až několik tisíc, například při sto stromech se mělo současně vykreslit okolo milionu vertexů, což je poněkud komplikované. Zvolil jsem tedy nakonec jinou metodu a vytvořil tak další malý program, který generuje textury korun stromů. Tyto textury lze po malé úpravě aplikovat v hlavním projektu.

7.1 Vložení stromů do prostředí města

Protože jsem nechtěl generovat celý les, ale pouze oživit prostředí města, stromy se vyskytují jen na prázdných kvadrantech. Tedy na takových, kde v důsledku malé plochy kvadrantu nestojí žádná budova, a je zde proto nastaven příznak prázdné plochy. (viz. 6.2 Generování budov).

Algoritmus prochází jednotlivé kvadranty a zjišťuje, zda jsou volné nebo ne. Pokud ano, načte zbylé parametry kvadrantu a testuje, je-li dostatečně velký alespoň pro jeden strom. Když kvadrant podmínku splní, je náhodně vybrán bod, na kterém bude strom stát. Aby každý strom nebyl úplně stejný, generátor náhodných čísel vygeneruje výšku stromu a tyto parametry uloží.

Vzhledem k rychlosti probíhá vykreslování tak, že nejprve se vykreslí kmen. Ten se skládá ze dvou stejných textur s motivem dřeva, které leží v osách x a y a protínají se uprostřed, čímž tvoří kříž. Na stejném principu se vykresluje i koruna, kde je přidána ještě jedna textura v ose z, aby při pohledu shora objekt vypadal trojrozměrně.



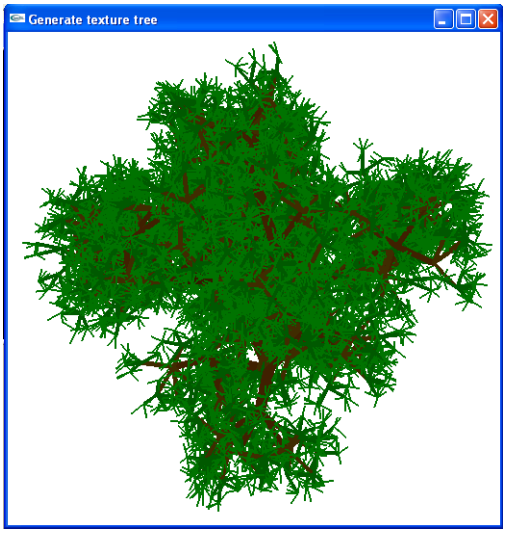
Obrázek 7.1 Princip billbordingu, zdroj [6]

Díky přidání alfa kanálu do textury koruny se může uplatnit blending (míchání barev), který zajistí, aby se zobrazily pouze větve a okolní prostředí textury bylo průhledné. U detailního pohledu na výsledný strom může být postřehnutelné prolínání textur. Při běžném přiblížení ale nejsou tyto detaily patrné a objekt se jeví jako plně trojrozměrný, i když je tvořen pouze pěti plochami. Celý vykreslovací algoritmus se tak zrychlil více než tisícinásobně.

7.2 Tvorba textur stromů

Texturu koruny stromu vytvoří program *gentree*. Tento program je pouze doplňkový, není potřeba při běžném používání hlavního programu. Textury už z něj byly vygenerovány a použity ve městě. Ponechávám jej tedy jako ukázku nebo jako možnost vygenerovat si vlastní textury.

Po spuštění se zobrazí okno s vygenerovaným stromem. Protože je vytvořen náhodně, nemusí ležet přesně uprostřed okna. Lze jej však uchopením levého tlačítka myši přesouvat. Pro přiblížení slouží klávesa „i“ nebo stisk prostředního tlačítka myši. Pro oddálení je to klávesa „o“ nebo opět prostřední tlačítko. Po vycentrování a správném přiblížení lze výtvor uložit stiskem klávesy „s“. Ve složce projektu se objeví nový soubor „*pixmap.tga*“. Textura má rozměry 512 x 512 pixelů, ale jen 24 bitovou hloubku barev. Při použití v projektu by se však zobrazovala i bílá místa okolo stromu, proto je nutno obrázek nejdříve upravit a přidat mu alfa kanál. Mohu doporučit např. program GIMP. Aby textura nezabírala zbytečně moc místa, lze ji zmenšit na polovinu. Při této velikosti je ještě zachováno dost detailů.



Obrázek 7.2 Okno gentree s vycentrovanou texturou stromu

Obrázek se vytvoří rozrůstáním větví z počátečního bodu. Je navoleno, kolik má v každém koncovém bodě vyrůst nových větví a kolikrát se má tento cyklus opakovat neboli zanořit. Konkrétně jsou nastavena čtyři dělení a šest zanoření. Funkce `generate_tree` vezme výchozí pozici (0, 0, 0), z níž vygeneruje body ležící v maximální vzdálenosti (`width`). Generátor pro stanovení pozice bodu využívá také úhel, aby strom rostl do výšky. Takto funkce vygeneruje větve se stejným počátkem podle zvoleného počtu dělení. Nové body jsou výchozí pozicí pro rekurzivní volání funkce `generate_tree` s menší vzdáleností `width` a zmenšeným zanořením o jedničku. Funkce se ukončí v okamžiku, kdy se zanoření rovná nule.

Při vykreslování se jednotlivé počáteční body propojí úsečkami, kterým je podle hloubky zanoření nastavena šířka a barva. Hlavní větve jsou nejsilnější a postupně se zužují tak, jak je tomu ve skutečnosti. Barva je použita pro rozlišení velikosti větví a nabytí dojmu reálného stromu. Využívají se odstíny od tmavě hnědé po světle zelenou.

8 Závěr

Dle zadání bylo mým úkolem prostudovat metody pro generování topologie městských ulic, tvaru a rozmístění budov a deformaci terénu. V programu jsou obsaženy tři druhy silnic s chodníky včetně křižovatek, které na ně navazují. Budovy se tvoří náhodně, lze navolit až pět různých typů, lišících se od sebe výškou a různým otexturováním. Pro deformaci terénu jsem použil metodu superpozice, takže výsledné zvlnění povrchu je také dostatečné.

Každé město by mělo mít alespoň trošku zeleně, a proto jsem do zástavby implementoval stromy, jejichž koruny jsou tvořeny pomocí L-systémů. V programu lze zapnout noční mód a pro zpestření je možné u denní i noční varianty města zapnou mlhu, která scéně dodá tu pravou atmosféru. Za zmínku také stojí, že okolo města je tzv. skybox, který u denního režimu simuluje mraky a Slunce, takže při pohledu na město proti Slunci jsou patrné stíny na objektech, a při nočním režimu simuluje hvězdnou oblohu.

Následujícím bodem zadání bylo nastavení parametrů pro generování a vhodný způsob jejich navolení pomocí GUI aplikace. Tuto část jsem implementoval v podobě editoru, kde lze navolit jednotlivé parametry pro silnice, budovy, terén a mlhu. Přímo z editoru je také možné spustit hlavní program.

Dále jsem uvažoval, jak v případném dalším pokračování projekt vylepšit. Zamýšlel jsem optimalizovat průběh vykreslování, protože v současné verzi může jeho rychlost působit slabším grafickým akcelerátorům problémy. Bylo by také možné navrhnout nové typy budov a více druhů textur tak, aby bylo na první pohled patrné, o jakou budovu se jedná, a dále vylepšit jejich zobrazovaní, např. pomocí lepšího filtrování. Poté přidat do města život v podobě jezdících aut s umělou inteligencí. Poslední věcí, kterou jsem měl v plánu, byly pohybující se mraky generované metodou perlin noise.

Nevím, zda by se mi podařilo všechny vyjmenované operace implementovat a jak by si program vedl po výkonnostní stránce, ale i současná verze mi dala hodně nových zkušeností a poznatků. Pochopil jsem metody procedurálního modelování, kdy nejproblémovější částí bylo generování stromů L-systémy. Nejsložitější bylo dosáhnout stromů, které by se podobaly realitě, a poté jejich vykreslení. Protože obsahovaly velké množství vertexů, vykreslení bylo příliš pomalé a vyžadovalo optimalizaci. Z toho důvodu vznikl další program generující koruny stromů jako textury, čímž jejich vykreslení mnohonásobně urychlil. Prohloubil jsem si také poznatky o OpenGL, které bylo pro mě doposud velkou neznámou. Jako příklad mohu uvést zobrazování scény, práci s kamerou, osvětlení, texturování, grafické efekty, jako je např. mlha, a mnoho dalšího. V neposlední řadě mě nadchla tvorba a programování GUI aplikace, díky níž jsem se naučil základům práce s Qt Creatorem. Mým prvotním záměrem bylo vytvořit program, který by všechny parametry generoval náhodně sám, uživatel by jej nemohl nijak ovlivnit. Změnil jsem však názor, protože by se tak můj výtvor mohl velice rychle stát nezajímavým a nudným.

Literatura

[1] J. Žára, B. Beneš, J. Sochor, P. Felkel: Moderní *počítačová grafika*. Computer press, Brno, 2004. ISBN 80-251-0454-0

[2] B. B. Mandelbrot: *The Fractal geometry of Nature*. W. H. Freeman, San Francisco, 1982.

[3] S. D. Ebert, F. K. Musgrave, D. R. Peachey, K. Perlin, S. Worley: *Texturing & Modeling A Procedura Approach.* Morgan Kaufmann Publisher Inc, 2001

[4] Wikipedie: *Superpozice (elektrotechnika)*. [online], [cit. 17.05.10] <http://cs.wikipedia.org/wiki/Superpozice_(elektrotechnika)>

[5] M. Dušek. Bakalářská práce: *Modelování rostlin pomocí L-systémů*. [online], Praha ČVUT: 2006. [cit. 17.05.10]. <http://www.cgg.cvut.cz/members/sloup/pages/theses/DusekMartin2006.pdf>

[6] P. Tišnovský: *Seriál grafická knihovna OpenGL*. [online]. 2003. [cit. 17.05.10] <http://www.root.cz/serialy/graficka-knihovna-opengl/>

[7] R. S. Wright: *OpenGL SuperBible*. Waite Group Press, 1999.

Seznam obrázků

Seznam příloh

A Manuál

Celý projekt se skládá ze tří částí. Hlavní část tvoří program *gencity* a pro nastavení jeho parametrů slouží program *editor*. Poslední část *gentree* je doplňková a je o ní zmínka v kapitole 7.2.

A.1 Překlad pro Windows

- Překlad *gencity*
  - Zdrojové soubory se nachází na přiloženém CD ve složce Zdrojové soubory\gencity
  - Program byl vytvořen ve vývojovém prostředí NetBeans IDE 6.7.1, proto jej lze využít pro překlad (lze využít i jiné vývojové prostředí)
  - Po spuštění NetBeans IDE zvolte záložku *File – New Project* – jako typ projektu zvolte *C/C++ Application* – zvolte název nového projektu a stiskněte *Finish*
  - Objeví se okno s novým projektem, rozklikněte jej a stiskem pravého tlačítka myši na *Headers Files* zvolte *Add Exist Item…*, zde přidejte všechny hlavičkové soubory s koncovkou h
  - Stejným způsobem přidejte soubory s koncovkou cpp do *Source Files*
  - Je třeba nastavit parametry pro překladač, klikněte pravým tlačítkem myši na projekt a zvolte *Properties*
  - V záložce *C++ Compiler* u položky *Include Directories* zadejte cestu ke knihovně glut (lze využít složku glut-3.7.6-bin přiloženou na CD\Zdrojové soubory)
  - V záložce *Linker* u položky *Additional Library Directories* zadejte opět cestu ke knihovně glut a k položky *Libraries* přidejte cestu ke knihovnám glu32, opengl32 a glut32
  - Projekt by měl jít v tuto chvíli přeložit v *release* verzi např. stiskem klávesy F6

- Překlad *editoru*
  - Zdrojové soubory se nachází na přiloženém CD ve složce Zdrojové soubory\editor
  - Program byl vytvořen ve vývojovém prostředí Qt Creator 1.3.1, pro jistější přeložení je mít třeba nainstalováno Qt SDK: Complete Development Environment ze stránek http://qt.nokia.com/downloads
  - V Qt Creatoru zvolíme *Create New Project – Qt4 Gui Application* – zadáme jméno a projekt dokončíme
  - Ve složce nového projektu nahradíme soubory ze složky CD\Zdrojové soubory\editor
  - Nyní by měl jít projekt přeložit v *release* verzi stiskem na *Run*

- Překlad gentree
    - Překlad probíhá stejným způsobem jako u gencity pouze se změnou zdrojových souborů ze složky CD\Zdrojové soubory\gentree

A.2 Překlad pro linux

- Překlad gencity
    - Program vyžaduje mít nainstalované knihovny OpenGL a GLUT
    - Samotný překlad probíhá zadáním příkazu *make* ve složce CD\Zdrojové soubory\gencity

- Překlad editoru
    - Překlad je totožný jako ve Windows

- Překlad gentree
    - Program vyžaduje mít nainstalované knihovny OpenGL a GLUT
    - Samotný překlad probíhá zadáním příkazu *make* ve složce CD\Zdrojové soubory\gentree

A.3 Spuštění ve Windows

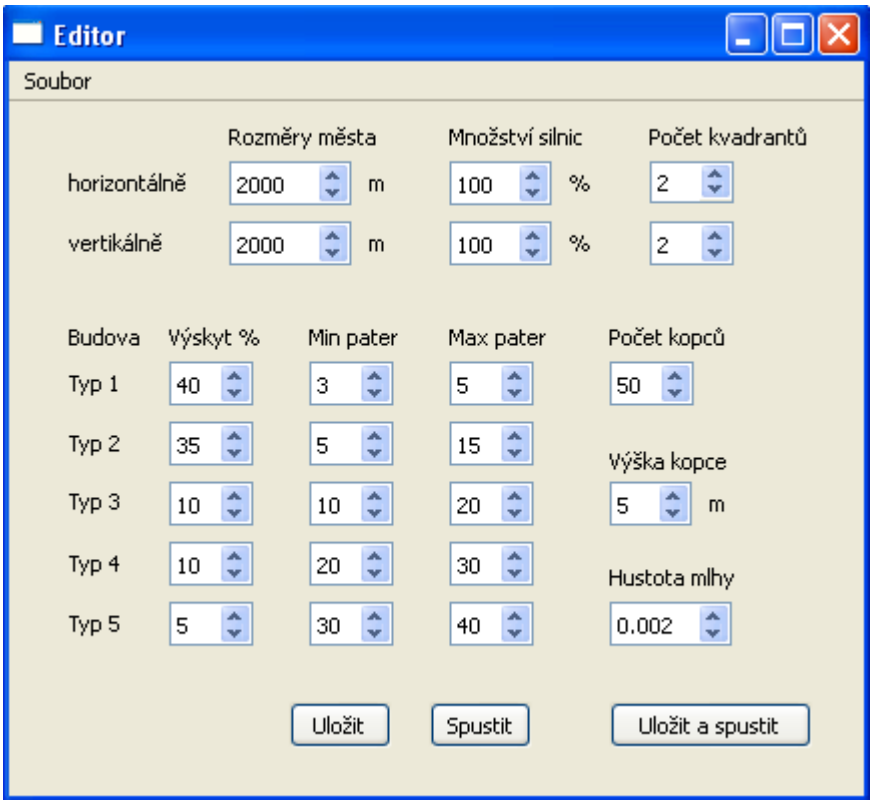
- Přeložené soubory *gencity.exe* a *editor.exe* z jednotlivých projektů zkopírujte do jedné složky
- Soubor *gencity.exe* vyžaduje složku *texture* ze složky CD\Spustitelné soubory\WIN a knihovnu *glut32.dll*, proto je také přikopírujte
- Soubor *editor.exe* vyžaduje knihovny *libgcc_s_dw2-1.dll, mingwm10.dll, QtCore4.dll* a *QtGui4.dll* nacházející se opět ve složce CD\Spustitelné soubory\WIN
- Nebo lze celý přeložený projekt spustit ze složky CD\Spustitelné soubory\WIN

A.4 Spuštění v Linuxu

- Přeložené soubory *gencity* a *editor* z jednotlivých projektů zkopírujte do jedné složky
- Soubor *gencity* vyžaduje složku *texture* ze složky CD\Spustitelné soubory\LINUX
- Nebo lze celý přeložený projekt spustit ze složky CD\Spustitelné soubory\LINUX

A.5 Použití a ovládání

- Spuštěním souboru *editor.exe* ve Windows nebo souboru *editor* v Linuxu se zobrazí následující okno:

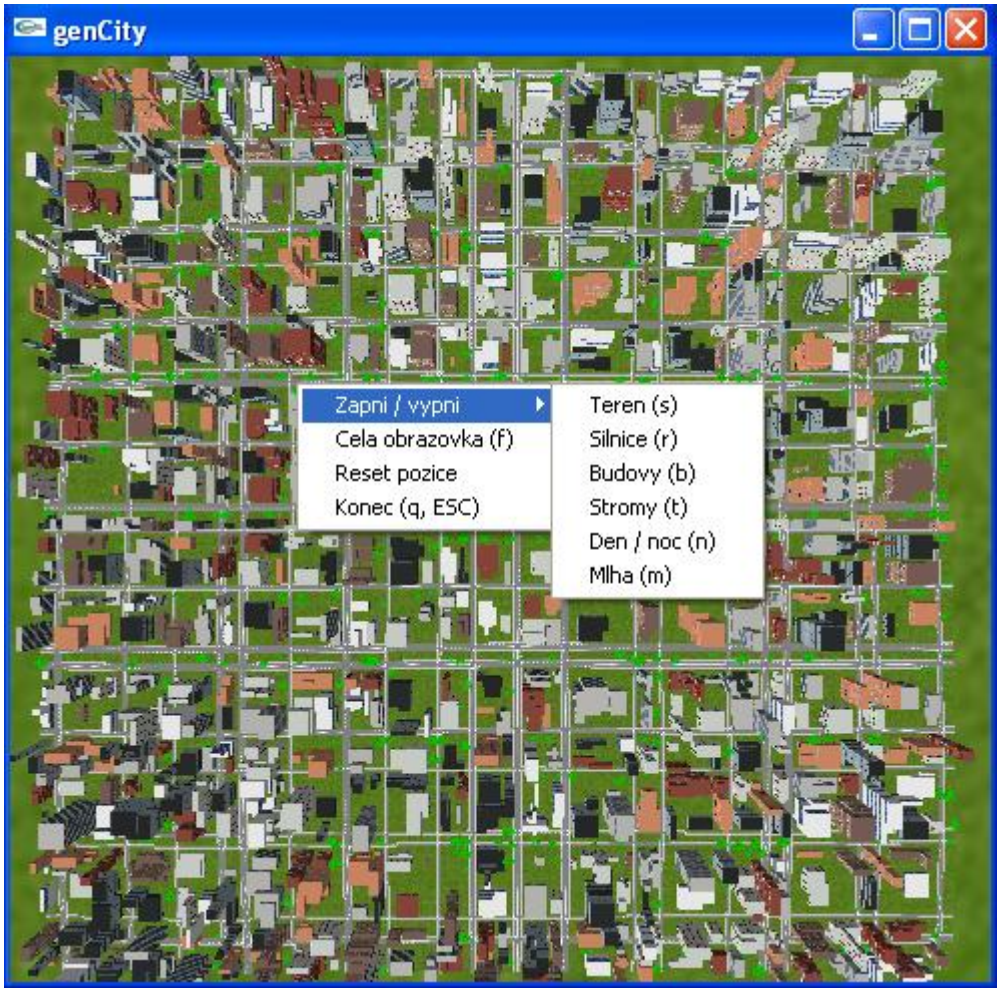


Okno editoru nastavení

- Parametry rozměry města, množství silnic a počet kvadrantů lze zvolit zvlášť pro horizontální a vertikální směr
- **Rozměr města** udává, na jak velké ploše se bude město generovat, lze nastavit rozmezí od 1000m do 4000m, po 100m krocích, pokud je zapsána jiná hodnota, program ji zaokrouhlí na stovky metrů
- **Množství silnic** udává procentuální počet silnic ve městě, hodnota je odvozena od velikosti města a počtu kvadrantů, rozsah je od 0% (nevykreslí se žádné silnice) do 200% (vykreslí se dvojnásobný počet oproti základní hodnotě)
- **Počet kvadrantů** udává, kolik budov se může maximálně vejít na prostor mezi silnicemi a lze nastavit od 1 po 5
- **Počet kopců** představuje množství generovaných kopců ve městě, může jich být až 500 nebo taky žádný
- **Výška kopce** udává maximální výšku jednoho kopce, pokud na stejném místě už nějaký kopec leží, výšky se sčítají, rozsah hodnot 1m – 99m

- **Hustota mlhy** je exponent pro funkci vytvářející mlhu, čím je číslo větší, tím je mlha hustější, rozsah 0,001 – 0,5
- Zbylé parametry slouží pro nastavení budov, lze nastavit až pět typů
- **Výskyt %** značí, s jakou pravděpodobností se daný typ vygeneruje, lze zadat od 0% po 100%, program si hlídá, aby celková hodnota výskytů nepřekročila 100%, proto před zvýšením jednoho typu se musí nejprve snížit jiný typ, pokud na nastavení není součet hodnot 100%, zbylé procenta se přičtou k typu jedna
- **Min pater** a **Max pater** udává rozmezí počtu pater, které bude použito pro generování daného typu budovy, program upravuje hodnoty min a max tak, aby max byla vždy alespoň o jedničku větší než min
- Tlačítko **Uložit** uloží navolené nastavení do souboru
- Tlačítko **Spustit** spustí program gencity bez uložení nastavení a použije poslední známou konfiguraci
- Tlačítko **Uložit a spustit** kombinuje funkci dvou předchozích tlačítek

➢ Po spuštění programu *gencity* se spustí okno s vygenerovaným oknem města

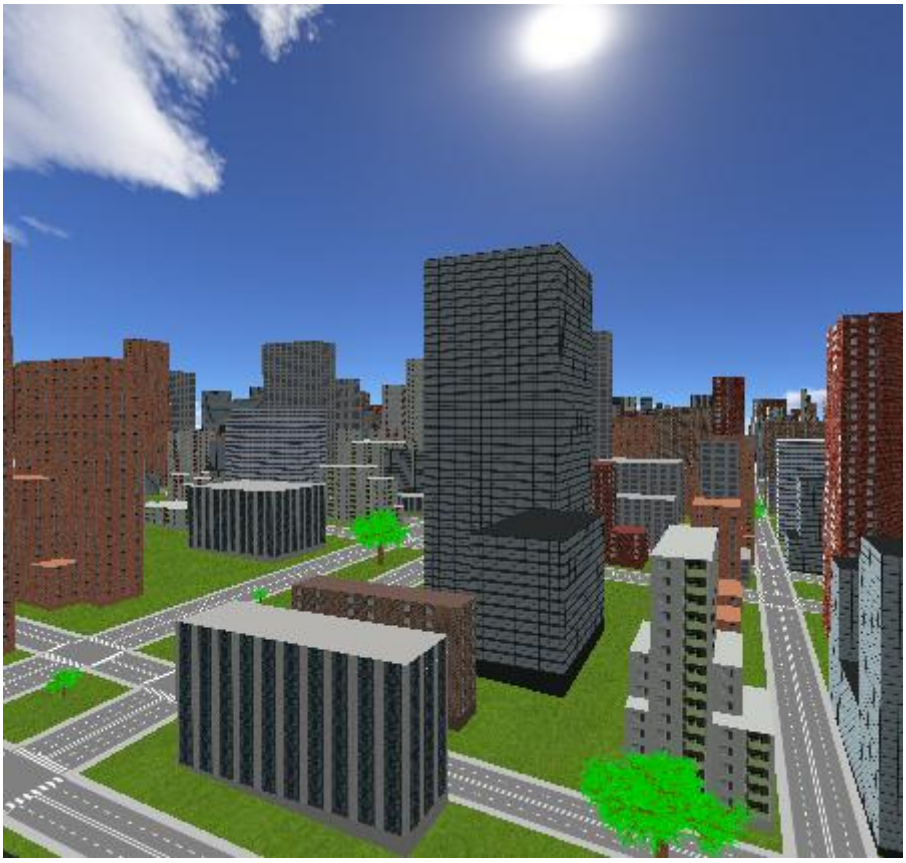


Okno s vygenerovaným městem a zapnutým menu

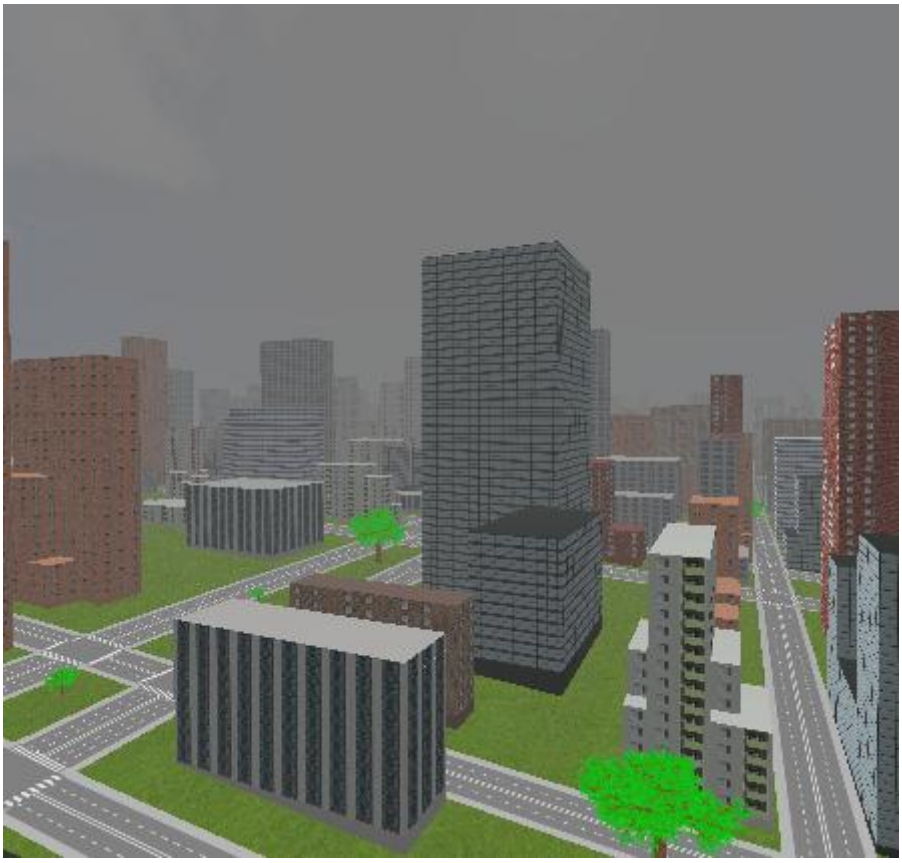
- K rotaci kamery slouží levé tlačítko myši a táhnutí myší daným směrem
- K posunu kamery slouží levé tlačítko se stisknutou klávesou **Ctrl**
- Pro přibližování a oddalování lze použít prostřední tlačítko myši nebo klávesy **i** (přiblížení) a **o** (oddálení)
- Stisknutí pravého tlačítka je vyvoláno menu programu, první položka menu Zapni / vypni zapíná a vypíná objekty na scéně, lze využít i klávesové zkratky:
    - Terén klávesa **s** nebo **S**
    - Silnice klávesa **r** nebo **R**
    - Budovy klávesa **b** nebo **B**
    - Stromy klávesa **t** nebo **T**
    - Přepínání mezi dnem a nocí klávesa **n** nebo **N**
    - Mlha klávesa **m** nebo **M**
- Druhá položka menu přepíná mód okna mezi zobrazením přes celou obrazovku a zobrazením v okně, klávesové zkratky jsou **f** nebo **F**
- **Reset pozice** nastaví kameru do výchozí pozice, jaká byla při spuštění programu
- Poslední položka menu **Konec** ukončí daný program, zkratky **q**, **Q** a **ESC**
- Dále klávesou **a** nebo **A** se spustí animace suplující video, demonstující práci
- Klávesou **x** nebo **X** lze zapnout tzv. QUICK_MODE, který vypíná některé objekty a vykreslování se tak urychlí

A.6 Ukázky výstupu programu

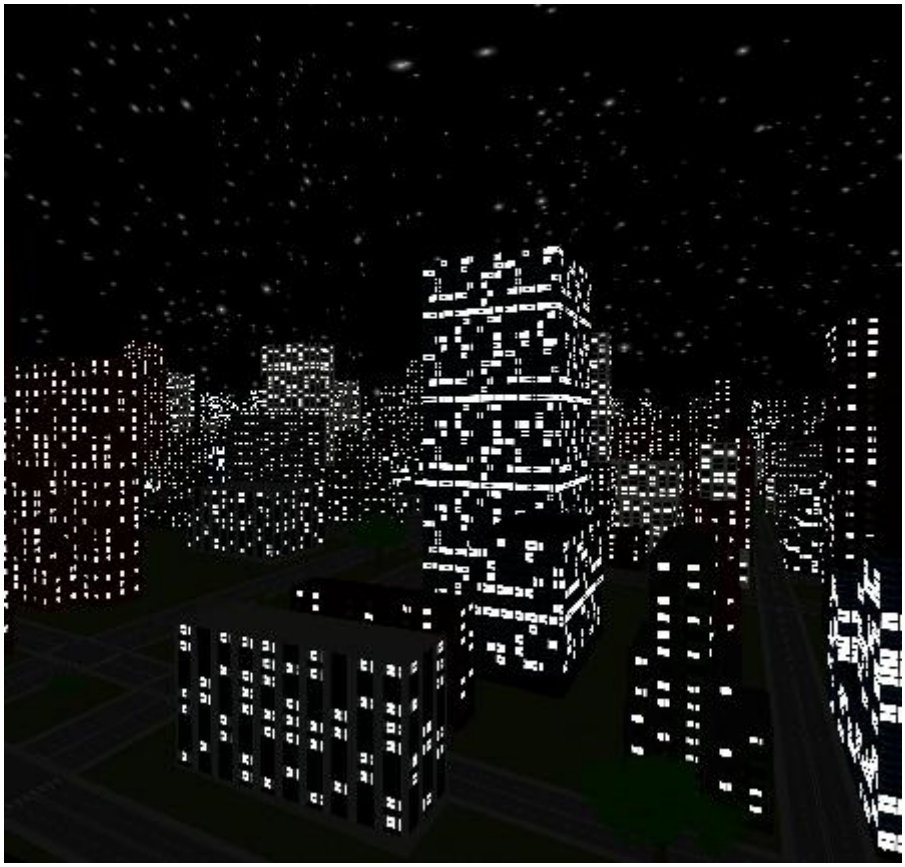
Osvětlená část města, pohled od Slunce



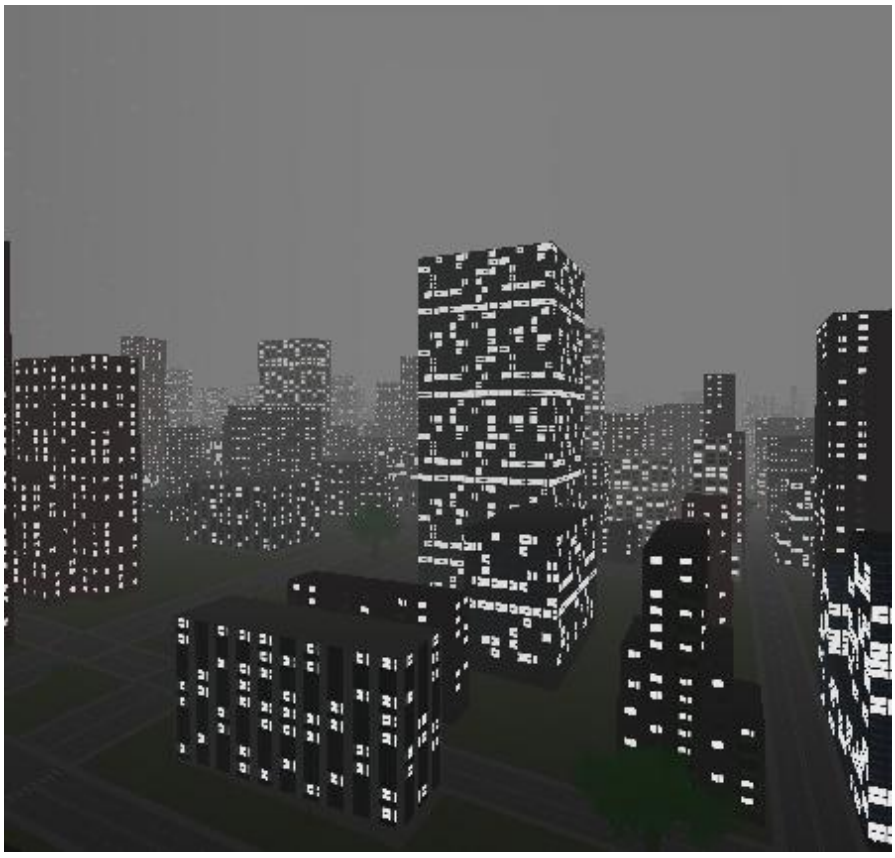
Pohled proti Slunci, na budovách je vidět stín



Denní město zahalené mlhou



Noční město s pohledem na jasnou oblohu



Noční město zahalené mlhou

B Obsah CD

- Spustitelné soubory
  - LINUX
  - WIN
- Zdrojové soubory
  - editor
  - gencity
  - gentree
  - glut-3.7.6-bin
- Plakát.pdf
- Technická zpráva.doc
- Technická zpráva.pdf