

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNĚ GENEROVANÉ MĚSTO

DIPLOMOVÁ PRÁCE

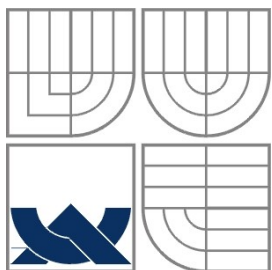
MASTER'S THESIS

AUTOR PRÁCE

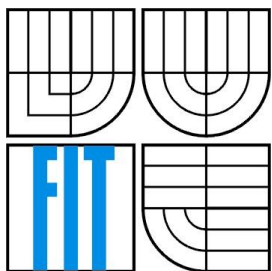
AUTHOR

Bc. PETR PANÁČEK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROCEDURÁLNĚ GENEROVANÉ MĚSTO

PROCEDURALLY GENERATED CITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR PANÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RUDOLF KAJAN

BRNO 2011

Abstrakt

Tato práce se zabývá problematikou procedurálně generovaného města. Jsou zde rozebrány jednotlivé kroky vytváření města. Těmito kroky jsou generování silnic, hledání minimálních cyklů uvnitř grafu tvořeného sítí silnic, dělení stavebních pozemků a generování budov. Silnice i budovy jsou generovány pomocí L-systému. Generování probíhá na základě vstupních map, jako jsou výšková mapa, mapa hustoty zalidnění a mapa vodních ploch. Navržené postupy jsou poté použity jako základ pro implementaci aplikace generující město.

Abstract

This paper deals with problem of procedurally generated city. There are described steps of creation of city. These steps are: road generation, extraction of minimal cycles in graph, division of lots and generation of buildings. Road and buildings are generated by L-system. Our system generate a city from input images, such as height map, map of population density and map of water areas. Proposed approaches are used for implementation of application for generation of city.

Klíčová slova

L-systém, gramatika, přepisovací pravidla, levá strana přepisovacího pravidla, pravá strana přepisovacího pravidla, vzory silnic, minimální cyklus (kružnice), extrakce cyklů, extrakce cest, dělení stavebních pozemků, obalové těleso, instancování

Keywords

L-system, grammar, production rules, predecessor, successor, road patterns, minimal cycle, cycles extraction, filaments extraction, lot subdivision, bounding box, instancing

Citace

Petr Panáček: Procedurálně generované město, Brno, FIT VUT v Brně, 2011

Procedurálně generované město

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Rudolfa Kájana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Panáček
1. května 2011

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Rudolfu Kajanovi za poskytnutí odborných informací, rad a materiálů.

© Petr Panáček, 2011

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah.....	1
1 Úvod.....	3
2 L-systémy.....	4
2.1 Želví grafika.....	4
2.2 Závorkové L-systémy.....	6
2.3 Stochastické L-systémy.....	7
2.4 Kontextové L-systémy.....	8
2.5 Podmíněná přepisovací pravidla.....	8
2.6 Parametrické L-systémy.....	8
3 Generování silnic.....	10
3.1 L-systém pro generování silnic.....	10
3.2 Globální cíle.....	12
3.3 Lokální omezení.....	13
3.4 Vyhlažování silnic.....	15
4 Hledání cyklů v grafu.....	18
4.1 Nalezení minimálního cyklu.....	18
4.2 Nalezení potřebných hran.....	19
4.3 Odstraňování hran.....	21
4.4 Extrakce cest.....	22
4.5 Extrakce cyklů.....	22
4.6 Heap (hromada).....	22
5 Generování budov.....	26
5.1 Dělení stavebních pozemků.....	26
5.2 L-systém pro generování budov.....	28
5.3 Triangulace konkávního polygonu.....	34
6 Analytická geometrie.....	38
6.1 Průsečík dvou úseček.....	38
6.2 Průsečík kružnice a úsečky.....	40
7 Implementace.....	41
7.1 XNA Framework.....	41
7.2 Struktura programu.....	45
8 Závěr.....	48
Literatura.....	49

Příloha A.....	51
Příloha B.....	52
B.1 Ovládání aplikace.....	52

1 Úvod

Tato práce se zabývá problémem procedurálního generování grafického modelu města. Pro takovéto město se nabízí nejrozumnější oblasti využití. Příkladem mohou být výzkumné účely, můžeme město použít při plánování výstavby města, nebo pro vytváření prostředí pro simulace. Hlavní využití se ale nachází především v zábavním průmyslu, v oblasti počítačových her a filmů.

Model města je ovšem velmi rozsáhlá a komplexní záležitost. Ruční vytváření takového města vyžaduje spoustu práce, a to nejen při samotném modelování, ale i při jeho návrhu. Systém pro generování měst se tedy snaží ušetřit práci vynaloženou v této oblasti. Přestože takto vygenerované město nebude dosahovat kvality ručně vytvořeného modelu, v určitých případech pro nás může být výsledek přijatelný.

Systém pro generování měst popsáný v tomto projektu předpokládá jako vstup sadu obrázků ovlivňujících vytváření města. Těmito obrázky jsou výšková mapa, mapa hustoty zalidnění a mapa vodních ploch. Vstupní obrázky budou zadávány uživatelem. Jejich úkolem je především ovlivnit rozložení silnic, ale najdou využití i při vytváření budov, kde se hustota zalidnění projeví na výsledné velikosti budov.

Systém pro generování silnic a budov je navržen jako L-systém. Obecné informace o problematice L-systémů jsou popsány v kapitole 2. Jednotlivé kroky generování silnic můžeme najít v kapitole 3. Jakmile jsou vygenerovány jednotlivé cesty, je nutné nalézt uzavřené smyčky uvnitř grafu tvořeného sítí silnic. Touto problematikou se zabývá kapitola 4. Smyčky uvnitř grafu tvoří prostory pro generování budov. Popis generování budov najdeme v kapitole 5. V kapitole 6 jsou rozebrány některé problémy analytické geometrie řešené při implementaci aplikace. Kapitola 7 se zabývá samotnou implementací aplikace pro generování města.

2 L-systémy

Abychom lépe porozuměli dalšímu textu, musíme si ze všeho nejdříve vysvětlit pojem L-systém, jelikož právě L-systém bude použit jako základ pro generování silnic a budov.

L-systém, nebo také Lindenmayerův systém je paralelní přepisovací systém používaný především při modelování růstu stromů, keřů a dalších přírodních útvarů. Ve své nejjednodušší podobě jsou tyto systémy definovány pomocí bezkontextových nebo regulárních přepisovacích gramatik. Podstatou gramatik je přepisování řetězců podle určitých pravidel, tato pravidla mohou být buď předem dána, nebo se mění v průběhu generování. Každý symbol v řetězci má zpravidla geometrický význam.

Gramatika L-systému je velmi podobná gramatice, jak ji známe z teoretické informatiky. Definice gramatiky L-systému je následující:

Gramatika L-systému je trojice $G = (V, \omega, P)$, kde

1. V je množina proměnných. Jedná se o symboly, které mohou být nahrazeny. Nerozlišují se non-terminální a terminální symboly.
2. ω je počáteční symbol z množiny V , definuje počáteční hodnotu L-systému.
3. P je množina přepisovacích pravidel (production rules). Přepisovací pravidlo se skládá ze dvou řetězců, kterými jsou předchůdce (predecessor) a následník (successor).

Přepisovací pravidla jsou aplikována iterativně od počátečního symbolu. Jedná se o paralelní přepisovací systém. To znamená, že při každé iteraci je přepsán každý symbol v řetězci, pokud pro něj existuje přepisovací pravidlo. Toto je významný rozdíl mezi gramatikami L-systémů a gramatikami používanými pro formální jazyky.

Pravidlo zapisujeme ve tvaru:

$$id: predecessor \rightarrow successor,$$

kde id je unikátní identifikátor pravidla, $predecessor \in V$ představuje symbol, který může být nahrazen pravou stranou (*successor*).

2.1 Želví grafika

K interpretaci řetězce vzniklého použitím gramatik se typicky používá tzv. želví grafika, ve které želva odpovídá kreslicímu zařízení, pomocí něhož vzniká obraz složený z úseček. Želva je definována svým stavem, přičemž stav se skládá ze dvou složek: polohy a orientace. Želva poté čte sekvenci

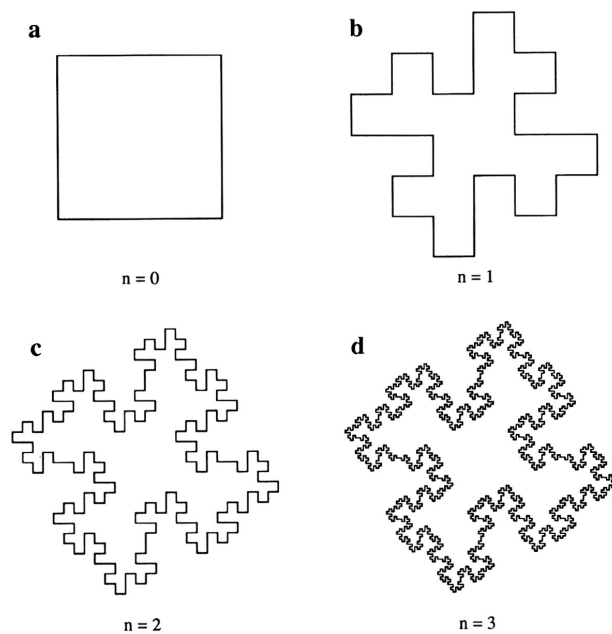
symbolů a interpretuje je jako příkazy. Nejjednodušší symboly pro interpretaci mohou být následující:

- F posun želvy dopředu spolu s kreslením čáry
- + natočení želvy doprava o jistý úhel
- natočení želvy doleva o jistý úhel

Příkladem jednoduchého L-systému využívajícího tyto symboly potom může být:

ω : F – F – F – F

p: F $\rightarrow$ F – F + F + F F – F – F + F



*Obrázek 2.1: Obrazce generované gramatikou
L-systému (zdroj obrázku [8])*

Na obrázku 2.1 vidíme obrazce vzniklé použitím dané gramatiky. Obrazec *a* vzniká interpretací počátečních hodnot L-systému. Obrazec *b* vzniká interpretací L-systému po jedné iteraci. Na obrazci *c* vidíme interpretaci po dvou iteracích, na obrazci *d* po třech iteracích.

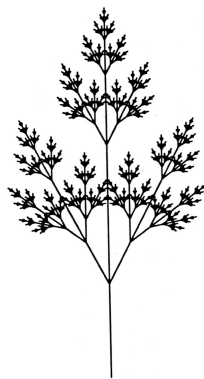
2.2 Závorkové L-systémy

L-systém popsaný výše nemohl vytvářet rozvětřující se struktury. Nebylo to možné z toho důvodu, že želva, která L-systém vykreslovala, neměla žádnou paměť do které by ukládala parametry, jako jsou pozice a orientace. Řešení tohoto problému spočívá v zavedení zásobníku, do kterého se ukládá stav želvy. Stav želvy může být ze zásobníku kdykoliv vybrán. Při výběru stavu se želva vrátí na uložené místo a změní svoji orientaci patřičným směrem.

Spolu se zavedením zásobníku musí být také rozšířena gramatika L-systému. Jsou přidány dva terminální symboly (symboly, které není možné prepisovat). První je popsán znakem „[“ a představuje uložení stavu želvy na zásobník. Druhým znakem je „]“ a odpovídá získání stavu želvy ze zásobníku, tedy posunu na uloženou pozici spolu s natočením. Všimněme si, že při zápisu se v gramatice mezi znaky „[“ a „]“ vkládá sekvence příkazů, které neovlivňují výsledný stav želvy. Po interpretaci řetězce mezi závorkami se želva vrátí do původní pozice. Obsah závorek tedy můžeme svým způsobem chápat jako podprogram. L-systémy rozšířené o tyto symboly se nazývají závorkové L-systémy.

Příkladem závorkového L-systému může být:

ω : A
p1: A $\rightarrow$ F [+A] [-A] F A
p2: F $\rightarrow$ F F



*Obrázek 2.2: Obrazec generovaný
závorkovým L-systémem
(zdroj obrázku [8])*

2.3 Stochastické L-systémy

U deterministických L-systému je jednoznačně definováno, které pravidlo se vybere a jak bude vypadat n -tá iterace výpočtu při opakovaném použití L-systému. Pokud chceme do systému vnést prvek náhody, použijeme stochastické L-systémy. Nejjednodušší možností přidání náhodnosti do systému je použití náhodné délky kroku a natočení. Gramatika přitom zůstává nezměněna.

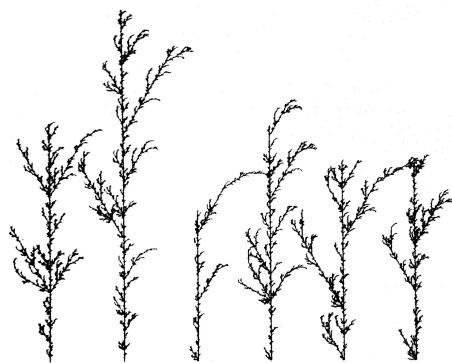
Druhým případem stochastičnosti je použití náhodného výběru přepisovacího pravidla, přičemž pravidlu je přiřazena pravděpodobnost, s jakou má být vybráno. Pravděpodobnost vybrání daného pravidla je ve výsledku dána vahou daného pravidla vydělenou součtem vah všech pravidel, která mohou být vybrána. Pokud váhy nejsou uvedeny a přesto se jedná o stochastický systém, všechna pravidla mají stejnou váhu.

Pravidlo stochastického L-systému se zapisuje ve tvaru:

id: predecessor $\rightarrow$ successor : prob,

kde *prob* představuje pravděpodobnost s jakou je pravidlo vybráno. Příkladem stochastického L-systému může být:

ω : F
p1: F $\rightarrow$ F [+F] F [-F] F : 0.33
p2: F $\rightarrow$ F [+F] F : 0.33
p3: F $\rightarrow$ F [-F] F : 0.34



Obrázek 2.3: Obrázek generovaný
stochastickým L-systémem
(zdroj obrázku [8])

2.4 Kontextové L-systémy

Kontextové L-systémy specifikují, v jakém kontextu musí být přepisovaný symbol, tedy jaký symbol musí být na levé, případně pravé straně od přepisovaného symbolu.

Pravidlo kontextového L-systému se zapisuje ve tvaru:

$$\text{id: } lc < \text{predecessor} > rc \rightarrow \text{successor},$$

kde lc je levý kontext, neboli výčet symbolů, které musí být vlevo od přepisovaného symbolu, rc je pravý kontext, tedy výčet symbolů, které musí být vpravo od přepisovaného symbolu. Pravidlo kontextového L-systému může mít levý kontext, pravý kontext nebo oba kontexty. Příkladem kontextového L-systému může být:

ω : BAAAAAAAA

p1: $B < A \rightarrow B$

p2: $B \rightarrow A$

2.5 Podmíněná přepisovací pravidla

Podmíněná přepisovací pravidla jsou rozšířena o podmínku, která musí být splněna, aby bylo pravidlo použito. Pokud podmínka není splněna, hledá se další pravidlo. Pokud žádné pravidlo nesplňuje podmínku, je symbol zkopírován do další iterace.

Podmíněné přepisovací pravidlo má tvar:

$$\text{id: } \text{predecessor} : \text{cond} \rightarrow \text{successor},$$

kde cond je podmínka, která musí být splněna, aby bylo pravidlo použito.

2.6 Parametrické L-systémy

Parametrické L-systémy jsou rozšířením L-systémů o parametry symbolu. Takovéto symboly jsou často označovány jako moduly. Každý symbol může mít libovolný konečný počet parametrů. Parametrem je obvykle reálné číslo. Může se jednat i o proměnnou, funkci nebo aritmetický výraz. Parametrické symboly mají závorkovou syntaxi. Parametry jsou tedy umístěny uvnitř závorek. Jednotlivé parametry se oddělují čárkou.

Příkladem parametrického L-systému může být:

ω : $B(2) A(4,4)$

p1: $B(x) : x > 0 \rightarrow B(x - 1)$

p2: $B(x) : x \leq 0 \rightarrow C$

p3: $A(x,y) : y > 3 \rightarrow B(x) A(x/y, 0)$

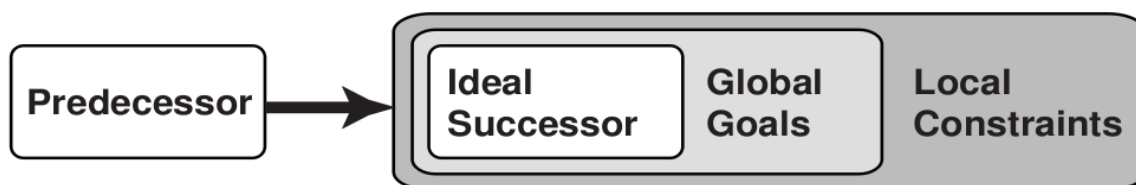
p4: $A(x,y) : y \leq 3 \rightarrow A(2 \cdot x, x+y)$

3 Generování silnic

V této části se budeme zabývat principem generování silnic. Rozebereme si způsob generování pomocí L-systému. Zaměříme se na nastavování počátečních hodnot a omezujících parametrů při generování silnic a popíšeme si postupy, pomocí kterých můžeme zjemnit průběh silnice, aby lépe zapadala do terénu vytvořeného z výškové mapy.

3.1 L-systém pro generování silnic

L-systém použitý pro tento projekt vychází z práce [1]. Během generování se využívá sada předem daných přepisovacích pravidel. Jedná se o parametrický L-systém, jednotlivými symboly jsou tedy moduly obsahující parametry. Pokud bychom chtěli navrhnout gramatiku pro generování silnic běžným způsobem, bylo by zapotřebí vytvořit komplexní systém pravidel s velkým množstvím parametrů a podmínek. S použitím každého omezení by musela být přidávána nová přepisovací pravidla a systém by velmi rychle narostl. Místo toho jsou hodnoty parametrů v pravidlech skryty a generování probíhá následovně. V prvním kroku je vytvářena obecná šablona (ideal successor). Na ni je následně použita sada globálních cílů a nakonec lokálních omezení, které upravují dané parametry.



Obrázek 3.1: Úprava přepisovacího pravidla (zdroj obrázku [1])

Pro vytváření silnic je použita pouze následující sada pravidel:

$$\begin{aligned}
 \omega: & R(0, \text{initialRuleAttr}) ?I(\text{initRoadAttr}, \text{UNASSIGNED}) \\
 p1: & R(\text{del}, \text{ruleAttr}) : \text{del} < 0 \rightarrow \varepsilon \\
 p2: & R(\text{del}, \text{ruleAttr}) > ?I(\text{roadAttr}, \text{state}) : \text{state} == \text{SUCCEED} \\
 & \quad \{ \text{globalGoals}(\text{ruleAttr}, \text{roadAttr}) \text{ vytvoří parametry} \\
 & \quad \text{pro: } p\text{Del}[0-2], p\text{RuleAttr}[0-2], p\text{RoadAttr}[0-2] \} \\
 & \quad \rightarrow B(p\text{Del}[1], p\text{RuleAttr}[1], p\text{RoadAttr}[1]), \\
 & \quad \quad B(p\text{Del}[2], p\text{RuleAttr}[2], p\text{RoadAttr}[2]), \\
 & \quad \quad R(p\text{Del}[0], p\text{RuleAttr}[0]) ?I(p\text{RoadAttr}[0], \text{UNASSIGNED}) \\
 p3: & R(\text{del}, \text{ruleAttr}) > ?I(\text{roadAttr}, \text{state}) : \text{state} == \text{FAILED} \rightarrow \varepsilon
 \end{aligned}$$

$p4: B(\text{del}, \text{ruleAttr}, \text{roadAttr}) : \text{del} > 0$
 $\rightarrow B(\text{del}-1, \text{ruleAttr}, \text{roadAttr})$
 $p5: B(\text{del}, \text{ruleAttr}, \text{roadAttr}) : \text{del} == 0$
 $\rightarrow [R(\text{del}, \text{ruleAttr}) ?I(\text{roadAttr}, \text{UNASSIGNED})]$
 $p6: B(\text{del}, \text{ruleAttr}, \text{roadAttr}) : \text{del} < 0 \rightarrow \varepsilon$
 $p7: R(\text{del}, \text{ruleAttr}) < ?I(\text{roadAttr}, \text{state}) : \text{del} < 0 \rightarrow \varepsilon$
 $p8: ?I(\text{roadAttr}, \text{state}) : \text{state} == \text{UNASSIGNED}$
 $\{\text{localConstraints}(\text{roadAttr}) \text{ nastavuje parametry pro:}$
 $\text{state, roadAttr}\}$
 $\rightarrow ?I(\text{roadAttr}, \text{state})$
 $p9: ?I(\text{roadAttr}, \text{state}) : \text{state} != \text{UNASSIGNED} \rightarrow \varepsilon$

ω nastavuje počáteční hodnoty modulů R a $?I$. Tyto dva moduly společně představují jeden úsek silnice. Nastavení modulů musí odpovídat globálním cílům, nesmíme tedy počáteční silnici umístit na vodní plochu. Modul R nese informaci o použitých pravidlech při vytváření silnice. Těmito pravidly je myšlen styl, kterým se silnice řídí při vytváření nových částí (více v kapitole 3.2). Informace o pravidlech jsou předávány přes parametr ruleAttr . Dalším parametrem je proměnná del , která v případě tohoto modulu nese informaci o tom, zda se má modul smazat. Modul R slouží kromě jiného také k vytváření nové větve silnic.

Modul $?I$ nese informace o parametrech silnice, jako je pozice, úhel apod. Všimněme si otazníku u daného modulu. Ten nám značí, že se jedná o modul typu dotaz, který se ptá na absolutní hodnotu pozice a natočení želvy. Tato informace je pro nás důležitá, protože potřebujeme znát parametry silnice pro aplikaci globálních cílů a lokálních omezení. Údaje o silnici jsou předávány přes parametr roadAttr . Druhým parametrem je state . Tento parametr udává stav, ve kterém se silnice nachází. Hodnota parametru může být změněna pouze pomocí lokálních omezení, a to na hodnotu *SUCCEED* (úspěch), nebo *FAILED* (neúspěch). Před tímto nastavením je hodnota vždy *UNASSIGNED*. Hlavní úlohou modulu $?I$ je samotné vytváření daného úseku silnice (vlození silnice do systému).

První tři pravidla jsou určena pro modul R . Pravidlo $p2$ slouží pro vytvoření nové větve silnic. V tomto pravidle dochází k přepsání modulu R . Modul $?I$ je zde pouze pro zjištění stavu silnice a jejích parametrů. Pravidlo nám také říká, že modul $?I$ by měl být napravo od modulu R . Pokud byla předchozí silnice úspěšně vytvořena ($\text{state} == \text{SUCCEED}$), potom vzniká nová větev. Parametry této větve jsou nastaveny pomocí globálních cílů. Větev se skládá ze třech částí. První částí jsou moduly R a $?I$, ty tradičně představují jednu silnici. Tato silnice se přibližně drží ve směru silnice předchozí. Pomocí globálních cílů může být nastaven parametr del na zápornou hodnotu. V takovém případě jsou moduly R a $?I$ v příštím cyklu odstraněny pravidlem $p1$ a $p7$. Druhou částí zmíněné větve jsou dva moduly B . Tyto moduly nesou informaci o dalších dvou silnicích. Jedna z nich představuje

odklon vpravo a druhá vlevo od předchozí silnice. Parametr *del* nyní nese informaci nejen o možném smazání daného modulu, ale také o zpoždění, s jakým bude modul *B* převeden na silnici v podobě modulů *R* a *?I*. Pokud by k takovému zpoždění nedocházelo, mohlo by se stát, že vedlejší silnice zablokuje cestu silnici hlavní.

Pravidla 4 až 6 jsou určena pro modul *B*. Pravidlo *p4* využívá informace o zpoždění a pouze snižuje hodnotu *del*. Jakmile je hodnota *del* rovna nule, vytvoří se namísto modulu *B* dvojice *R* a *?I*, tvořící silnici. Pravidlem *p6* je modul *B* odstraněn v případě, že globální cíle nastavili hodnotu *del* na zápornou.

Pravidla 7 až 9 jsou určena pro modul *?I*. Pravidlo *p8* kontroluje a upravuje parametry silnice pomocí lokálních omezení. Pokud je hodnota *state* nastavena na *FAILED*, jsou odstraněny moduly *R* a *?I* pomocí pravidel *p3* a *p9*. Pravidlo *p9* odstraní modul *?I* i v situaci, kdy je *state* nastaven na *SUCCEED*. V takovém případě dojde k finálnímu vytvoření dané silnice a v seznamu modulů už tato informace není potřeba. Pro modul *R* znamená nastavení *state* na *SUCCEED* to, že může vytvořit novou větev silnic.

3.2 Globální cíle

Hlavním úlohou globálních cílů je nastavit počáteční hodnoty jednotlivých modulů. Systém bere v úvahu hodnoty ze vstupních obrazů a pravidla pro generování silnic (*ruleAttr*), a podle nich nastavuje jednotlivé atributy modulů.

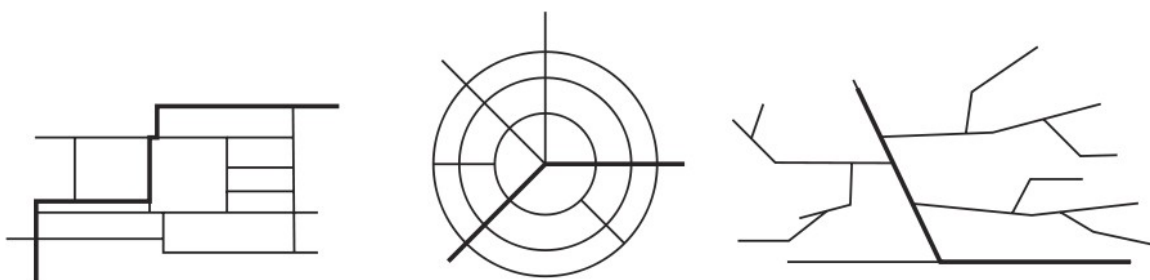
Hustota zalidnění

Důležitým vstupním parametrem, který ovlivní globální rozvržení silnic, je informace o hustotě zalidnění. Zatímco dálnice budou tvořit spojení mezi hustě zalidněnými oblastmi, ostatní silnice povedou do obydlených oblastí a budou umožňovat přístup k dálnicím. Každá z dálnic se při vytváření rozhoduje, kterým směrem její cesta povede. Dálnice vytváří několik potenciálních cest směřujících pod určitým úhlem. Podél každé z nich je počítána hustota zalidnění zjišťovaná z příslušného obrázku. Při výpočtu je pro nás ovšem podstatnější hustota zalidnění ve větší vzdálenosti od počátku generovaného úseku dálnice. Proto je daná hodnota vynásobena vzdáleností od počátku. Tato vzdálenosti je brána jako číslo od 0 do 1, přičemž 0 znamená začátek dálnice a 1 znamená její konec. Tento výpočet se provede pro každý bod na přímce představující dálnici a všechny hodnoty se sečtou. Úsek dálnice s největší hodnotou bude považován za nejlepší a vybere se pro vykreslení.

Vzory silnic

Dalším parametrem určujícím vstupní hodnoty pro rozvržení silnic je vzor silnic. Tento parametr určuje globální rozvržení silnic, které je typické pro dané město, nebo část města. Příklady vzorů silnic mohou být následující:

- *Základní vzor*: Nejjednodušší možný vzor. Všechny cesty takového vzoru sledují hustotu zalidnění. Bývá označován jako přirozený růst silniční sítě. Všechny ostatní vzory vycházejí z tohoto vzoru s omezením pro úhly větvení silnic a délky segmentů.
- *Rastrový vzor*: Tento vzor se řídí lokálním nebo globálním úhlem a maximální délkou bloku. Jedná se o nejčastější vzor v městských oblastech, kde silnice tvoří pravoúhlé bloky.
- *Radiální vzor*: Dálnice sleduje radiální cestu od jistého středového bodu.
- *San Francisco*: Hlavní silnice jsou vedeny s nejmenší možnou změnou výšky. Silnice v různých výškových polohách jsou poté spojeny menšími přímými cestami. Tento vzor bývá použit v oblastech s proměnnou výškou terénu.



Obrázek 3.2: Vzory silnic: rastrový, radiální, základní (zdroj obrázku [1])

3.3 Lokální omezení

Lokální omezení upravuje parametry nastavené globálními cíli takovým způsobem, aby odpovídali podmínkám prostředí na lokální úrovni. Pokud není nalezena žádná cesta, která by vyhovovala omezením, je hodnota modulu reprezentujícího cestu nastavena na FAILED. Daný modul je následně odstraněn z řetězce modulů.

Funkce řešící lokální omezení se skládá ze dvou hlavních částí:

- kontrola, zda se silnice nachází v povolené oblasti (nezasahuje do vodních ploch apod.)
- hledání průniku s jinou silnicí, případně hledání křižovatky, která je v požadované vzdálenosti od koncového bodu silnice

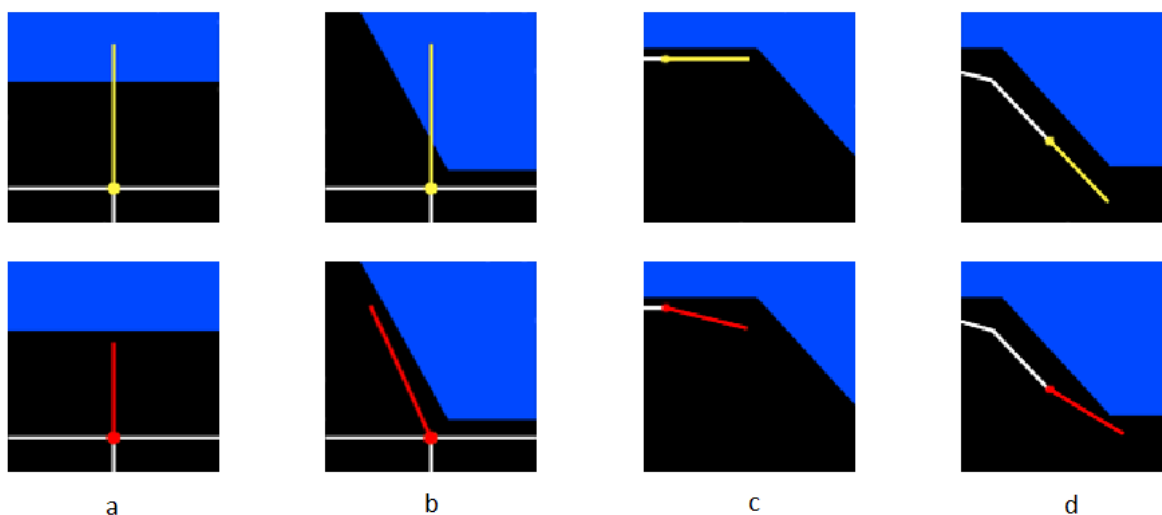
Přístup do zakázané oblasti

Pokud se ukáže, že se silnice nachází v zakázané oblasti, existuje několik způsobů jak se s tímto problémem vypořádat.

Nejjednodušším řešením je zkrátit silnici na takovou délku, aby do oblasti nezasahovala. Tato možnost je ideální v případě, že silnice zasahuje do zakázané oblasti pouze malou částí. V případě, že

jsme příliš blízko zakázané oblasti a zkracování již nemá smysl, je nutné změnit úhel, pod kterým bude silnice vykreslena. Pro vykreslování vybereme silnici, která se svým úhlem nejvíce blíží původně zamýšlenému směru. Úhel by také neměl přesáhnout určité maximum dané pravidlem (vzorem silnice), kterým se silnice při vytváření řídí. Pokud není možné vytvořit silnici v daném rozmezí úhlu, je silnice odstraněna. V případě dálnic je možné překročit zakázanou oblast s určitým omezením na délku silnice uvnitř zakázané oblasti. Takováto dálnice potom může tvořit most přes vodní plochu.

Silnice ovšem často vedou podél pobřeží. Abychom je tedy usměrnili touto cestou, potřebujeme přidat ještě další pravidla měnící úhel silnice. Pokud je za koncovým bodem silnice voda, natočíme silnici mírně směrem od pobřeží, přestože silnice nezasahuje do vodní plochy. Tento případ je znázorněn na obrázku 3.3.1 c. Druhé pravidlo zjišťuje, zda je v blízkosti silnice (po její levé nebo pravé straně) voda. Pokud ano, natočí se silnice ještě více k ní. Na obrázku 3.3.1 d můžeme vidět, jak tento případ vypadá.



Obrázek 3.3.1: Přístup do zakázané oblasti (zdroj obrázku [8])

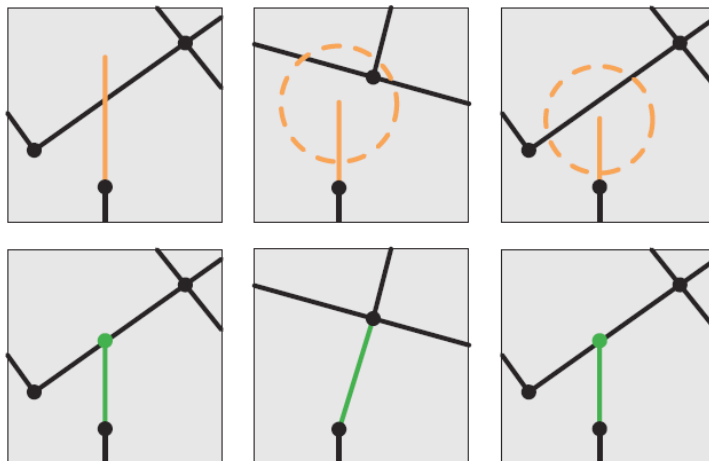
Napojování silnic

V této části se budeme zabývat uzavřenými smyčkami ve struktuře silnic. Se slepými cestami se v případě silnic můžeme setkat spíše výjimečně. Většinou jsou cesty napojeny na jiné cesty, nebo se otáčejí a napojují sami na sebe. Výsledné silnice tedy netvoří stromovou strukturu, ale spíše síť silnic.

Uzavřené smyčky jsou vytvářeny právě za pomoci lokálních omezení. Nejdříve se zjišťuje, zda se aktuální silnice protíná s některou z již vytvořených silnic. Pokud ano, je aktuální silnice zkrácena a je zaznamenán nový uzel představující křižovatku. Jestliže k protnutí nedojde, je hledána silnice v určité vzdálenosti od koncového bodu aktuální silnice. Primárně se hledá již existující křižovatka,

na kterou by bylo možné se připojit. Pokud je v blízkosti pouze silnice bez křižovatky, dojde k prodloužení aktuální silnice a vytvoření nového spojení.

Poté, co jsou na silnici aplikována lokální omezení, nic nebrání konečnému vykreslení silnice a zanesení této informace do struktury silnic.



Obrázek 3.3.2: Napojování silnic (zdroj obrázku [1])

3.4 Vyhlažování silnic

Silnice, jak je vygenerována v této fázi, může v případě určitého typu a pro určité umístění působit příliš hrubým dojmem. Jedná se především o dálnice umístěné v hornatém terénu, kde jsou dva vzdálenější body spojeny pouze jednou čarou. U takového typu silnic je potřeba zjemnit jejich krok v závislosti na výškových rozdílech podél cesty. Jinými slovy chceme převzorkovat původní úsek silnice.

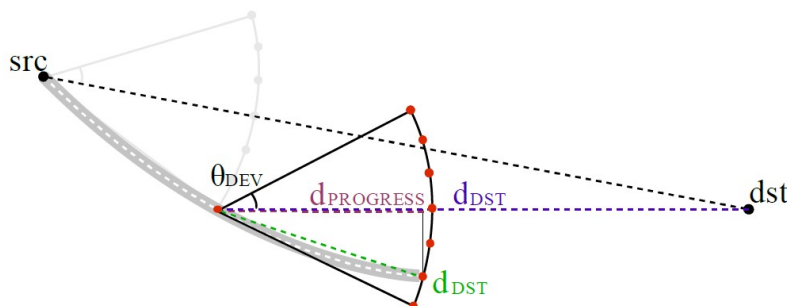
Přístup pro převzorkování silnice vychází z práce [3]. Nová silnice je kreslena z počátečního bodu původního úseku silnice v předem daných intervalech s cílem dostat se do koncového bodu původního úseku silnice.

Parametry pro vytváření silnice jsou následující:

- d_{SAMPLE} : velikost kroku
- n_{SAMPLE} : počet vzorků
- θ_{DEV} : úhel vychýlení

Kontrolní bod pro kreslení silnice je vždy posunut o d_{SAMPLE} od předchozího bodu a je vychýlen maximálně o θ_{DEV} od směru daného cílovým bodem. V každém kroku je nabídnuta sada kontrolních bodů. Jejich počet je dán hodnotou n_{SAMPLE} a jejich rozmístění je dáno úhlem $2\theta_{DEV}$ (viz obrázek 3.4.1).

Jeden z těchto bodů je vždy vybrán jako další uzel výsledné silnice. Vykreslování silnice je ukončeno ve chvíli, kdy se kontrolní bod nachází ve vzdálenosti d_{SNAP} od cílového bodu.



Obrázek 3.4.1: Průběh vykreslování silnice (zdroj obrázku [3])

Pro výběr kontrolních bodů existují různé strategie. Každá z nich má za následek vytvoření odlišného průběhu dané silnice.

- *Strategie minimální výšky:* Jedná se o nejzákladnější strategii. Je vybírán kontrolní bod s nejmenší výškovou hodnotou. Výsledná silnice svým tvarem připomíná tok řeky.
- *Strategie minimálního výškového rozdílu:* Tato strategie se snaží vyhnout prudkým změnám výšky. Hledá tedy nejmírnější cestu směrem k cíli. Problém ovšem může nastat, pokud je mezi počátečním a koncovým bodem silnice výrazný výškový rozdíl. V takovém případě se silnice stále může vyhýbat koncovému bodu. Z toho důvodu je počet kroků omezen. Ve výsledku pak ale dostaneme hladký průběh silnice s ostrým zalomením při napojování na koncový bod.
- *Strategie rovnoměrného výškového rozdílu:* Tato strategie se dívá dopředu a rozhoduje se na základě aktuálního kroku. Cílem je, aby výškový rozdíl mezi sousedními body byl vždy stejným podílem celkového rozdílu mezi počátečním a koncovým bodem. Výběr nového kontrolního bodu se řídí následujícím pravidlem:

$$\text{MinAbs}(\text{elevation}_{STEP} / d_{PROGRESS} - \text{elvation}_{DEST} / d_{DST}),$$

kde elevation_{STEP} představuje výškový rozdíl současného kroku, elvation_{DEST} představuje výškový rozdíl mezi současným bodem a cílovým bodem, $d_{PROGRESS}$ je délka současného kroku a d_{DST} odpovídá vzdálenosti k cíli.



Obrázek 3.4.2: Cesty s různým výběrem kontrolního bodu, modrá – strategie minimální výšky, červená – strategie minimálního výškového rozdílu, zelená – strategie rovnoměrného výškového rozdílu (zdroj obrázku [3])

4 Hledání cyklů v grafu

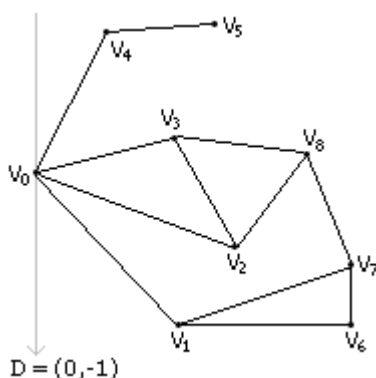
Dalším krokem při vytváření města je stavba budov. Abychom ale mohli stavět budovy, je potřeba určit místa, na která můžeme budovy umístit. Pro stavbu budov budou nejvhodnější uzavřené cykly uvnitř grafu tvořeného sítí silnic. Přesněji řečeno nás budou zajímat minimální cykly grafu. Algoritmus pro detekci minimálních cyklů se nazývá Minimum Cycle Basis. Existuje mnoho druhů tohoto algoritmu odvíjející se od typu grafu, uspořádání uzlů apod. Postup navržený pro tento projekt vychází z práce Davida Eberlyho [4].

Jednotlivé kroky algoritmu jsou následující. Ze všeho nejdříve se jednotlivé uzly seřadí podle osy x od nejmenších po největší. Cykly jsou poté extrahovány právě v tomto pořadí. Jakmile je cyklus extrahován, jsou odstraněny hrany, které nejsou dále potřeba. Tím je zabráněno opakované extrakci stejných cyklů.

4.1 Nalezení minimálního cyklu

Nyní si popíšeme hledání cyklu grafu. Pro popis využijeme graf z obrázku 4.1. Při hledání budeme začínat bodem V_0 . Můžeme si všimnout, že hrana (V_0, V_1) může být pouze součástí jednoho cyklu. Při průchodu hranou z bodu V_0 do bodu V_1 vidíme, že na pravé straně této hrany se nachází vnější část cyklu, a zároveň se zde nenachází žádný jiný cyklus. Naopak na levé straně hrany se nachází minimální cyklus tvořený hranou (V_0, V_1) . Zamysleme se tedy, čím je hrana (V_0, V_1) výjimečná.

Pokud si jednotlivé hrany představíme jako ručičky hodin se středem v bodě V_0 , je hrana (V_0, V_1) nejvíce natočena po směru hodinových ručiček. Toto natočení se bere vzhledem k předchozí hraně. Pokud předchozí hrana neexistuje, použije se vertikální hrana $D = (0, -1)$.



Obrázek 4.1: Příklad grafu
(zdroj obrázku [4])

První hranou minimálního cyklu je tedy (V_0, V_1) . Víme, že cyklus leží nalevo od této hrany. Další hrany, které budeme procházet, budou tedy ty, které jsou nejvíce natočeny proti směru

hodinových ručiček vzhledem k předchozí hraně. Poté co se průchodem dostaneme k uzlu V_0 , je nalezen minimální cyklus.

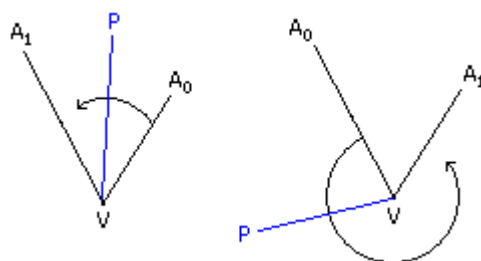
4.2 Nalezení potřebných hran

V této části si popíšeme, jakým způsobem zjistit, které hrany jsou natočeny nejvíce po směru hodinových ručiček a proti směru hodinových ručiček.

Pro testování vztahů mezi hranami budeme používat vektorový součin dvou vektorů. Součin vektorů je pro vektory $u = (u_1, u_2, u_3)$, $v = (v_1, v_2, v_3)$ definován následovně:

$$u \times v = (u_2 v_3 - u_3 v_2, u_3 v_1 - u_1 v_3, u_1 v_2 - u_2 v_1) \quad (1)$$

Na obrázku 4.2.1 máme následující 2D vektory: $D_0 = A_0 - V$, $D_1 = A_1 - V$, $D = P - V$. Pokud je převedeme na 3D vektory, dostaneme $E_0 = (D_0, 0)$, $E_1 = (D_1, 0)$, $E = (D, 0)$. Jedná se tedy o vektory s hodnotou z rovnou nule.



Obrázek 4.2.1: Zjišťování vztahu mezi vektory (zdroj obrázku [4])

Pokud chceme zjistit, zda se vektor D nachází mezi vektory D_0 a D_1 , musíme vypočítat vektorový součin $E \times E_1$ a $E_0 \times E$. Vzhledem k tomu, že je hodnota z rovna nule, je výsledek součinů následující:

$$E \times E_1 = (0, 0, D \cdot D_1^\perp), \quad E_0 \times E = (0, 0, D_0 \cdot D^\perp), \quad (2)$$

kde $(x, y)^\perp = (y, -x)$

Jestliže zjišťujeme, zda se nacházíme mezi dvěma vektory, je potřeba se rozhodnout, který prostor považujeme za místo mezi vektory. V tomto případě se bude jednat o prostor, který představuje vnější část potenciálního cyklu grafu. Při zjišťování polohy vektoru musíme rozlišovat, zda je cyklus v místě uzlu V konvexní či konkávní. Pro tyto dva případy se vyhodnocení výsledku vektorového součinu bude lišit.

Pro konkávní cyklus se nacházíme mezi vektory, pokud:

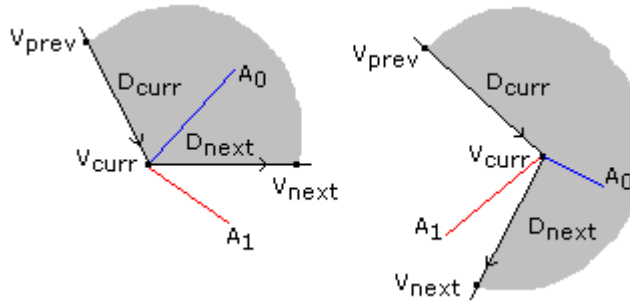
$$D_0 \cdot D^\perp > 0 \wedge D \cdot D_1^\perp > 0 \quad (3)$$

Pro konvexní cyklus se nacházíme mezi vektory, pokud:

$$D_1 \cdot D^\perp < 0 \vee D \cdot D_0^\perp < 0 \quad (4)$$

Největší natočení po směru hodinových ručiček

Při průchodu grafem si označíme předchozí uzel jako V_{prev} , současný uzel jako V_{curr} a následující uzel jako V_{next} . Vektor představující předchozí hranu definujeme jako $D_{prev} = V_{curr} - V_{prev}$. Vektor představující současnou hranu definujeme jako $D_{curr} = V_{next} - V_{curr}$. Jako následující uzel je vybrán sousední uzel, který není předchozím uzlem. Současný uzel sousedí také s dalšími uzly, které by se mohly stát následujícím uzlem.



Obrázek 4.2.2: Hledání následujícího uzlu
(zdroj obrázku [4])

Obrázek 4.2.2 vlevo ukazuje příklad výběru následujícího uzlu pro případ, že je cyklus konvexní v místě současného uzlu. Na obrázku můžeme vidět dva uzly, které musí být zpracovány, jsou jimi uzly A_0 a A_1 . Uzel A_0 je zamítnut, protože hrana k tomuto uzlu je umístěna proti směru hodinových ručiček od současné hrany. Následujícím uzlem se stane uzel A_1 , protože právě jeho hrana se nachází po směru hodinových ručiček od současné hrany. Tuto skutečnost zjistíme podle toho, že se hrana $D = A_1 - V_{curr}$ nachází mezi vektorem $V_{prev} - V_{curr} = -D_{curr}$ a $V_{next} - V_{curr} = D_{next}$. Pro výpočet použijeme rovnici (4):

$$D_{next} \cdot D^\perp < 0 \vee D \cdot (-D_{curr})^\perp < 0 \quad (5)$$

nebo ekvivalentně:

$$D_{curr} \cdot D^\perp < 0 \vee D_{next} \cdot D^\perp < 0 \quad (6)$$

Obrázek 4.2.2 vpravo ukazuje příklad výběru následujícího uzlu pro případ, že je cyklus konkávní v místě současného uzlu. Opět je vybrán uzel A_1 , jehož hrana je umístěna po směru

hodinových ručiček od současné hrany. Vektor $D = A_l - V_{curr}$ se nachází mezi vektory $-D_{curr}$ a D_{next} . Pro otestování této skutečnosti použijeme rovnici (3):

$$-D_{curr} \cdot D^\perp > 0 \wedge D \cdot D_{next}^\perp > 0 \quad (7)$$

nebo ekvivalentně:

$$D_{curr} \cdot D^\perp < 0 \wedge D_{next} \cdot D^\perp < 0 \quad (8)$$

Největší natočení proti směru hodinových ručiček

Přístup je velmi podobný hledání největšího natočení po směru hodinových ručiček. Pro obrázek 4.2.2 vlevo se kandidátem pro následující uzel stává A_0 . Podmínka pro zvolení uzlu A_0 za následovníka je:

$$D_{curr} \cdot D^\perp > 0 \wedge D_{next} \cdot D^\perp > 0 \quad (9)$$

Pro obrázek 4.2.2 vpravo je kandidátem pro následující uzel opět A_0 . Podmínka pro zvolení uzlu A_0 za následovníka je:

$$D_{curr} \cdot D^\perp > 0 \vee D_{next} \cdot D^\perp > 0 \quad (10)$$

4.3 Odstraňování hran

Při hledání minimálních cyklů je vhodné, abychom odstranili již nalezené cykly, které by ovlivňovali další zpracování grafu. Vzhledem k tomu, že je hrana (V_0, V_1) nejvíce natočena po směru hodinových ručiček, víme, že je použita pouze jedním cyklem. Může tedy být bezpečně odebrána. Hrana (V_7, V_8) by také mohla být odebrána, ale v tomto případě si nemůžeme být jisti tím, že není součástí jiného cyklu.

Primitiva, která budeme odstraňovat z grafu nejsou pouze cykly, ale i posloupnosti hran bez jakéhokoliv větvení. Takováto posloupnost se podle teorie grafů označuje jako cesta a její definice je následující:

Je-li $G = (U, H)$ obyčejný graf, potom cesta mezi uzly u, v o délce n je sled $(u = w_0, h_1, w_1, h_2, \dots, w_{n-1}, h_n, w_n = v)$ mezi uzly u, v takový, že platí $i \neq j \Rightarrow w_i \neq w_j \wedge h_i \neq h_j, 0 \leq i, j \leq n$. Jedná se tedy o posloupnost, ve které se neopakují uzly, ani hrany.

Cesta může být pro další kroky generování města také důležitá. V následující části se tedy budeme zabývat jak extrakcí cyklů, tak i cest.

Během extrakce primitiv je potřeba najít uzel s minimální x -ovou hodnotou. Abychom nemusely pokaždé prohledávat graf, budou uzly předem seřazeny a uloženy do struktury typu heap, kde minimální prvek bude umístěn v kořenu struktury (více viz kapitola 4.6). Pro seřazení budou použita následující pravidla:

$$(x_0, y_0) < (x_1, y_1) \text{ pokud } \{(x_0 < x_1) \vee [(x_0 = x_1) \wedge (y_0 < y_1)]\}$$

4.4 Extrakce cest

Pokud má V_0 jeden sousední uzel, jedná se o počáteční bod cesty. V takovém případě jsou navštíveny všechny uzly dané cesty, informace o nich jsou zaznamenány pro další použití a uzly s hranami jsou odstraněny z grafu. Hraný a uzly jsou odstraňovány postupně, dokud má aktuální uzel pouze jednoho souseda. V případě, že aktuální uzel neobsahuje žádného souseda, jedná se o koncový bod cesty. Pokud obsahuje více než jednoho souseda, jedná se o větvení, kde se může nacházet cyklus, nebo jiná cesta. V obou případech se jedná o ukončení cesty.

Cesta nemusí začínat jenom v místě uzlu, který má pouze jednoho souseda. Může vycházet i z uzlu, který představuje větvení grafu. Jedná se tedy o místo, ve kterém má uzel nejméně 3 sousední uzly.

Jak již bylo zmíněno, pokud nalezneme cyklus grafu, odstraníme pouze některé hraný. Ostatní hraný ovšem musí být označeny pro další zpracování. Pokud budou takovéto hraný nalezeny při extrakci cesty, budou z grafu odstraněny bez dalšího uložení. V opačném případě by mohly být některé z hran cyklu označeny za cestu.

4.5 Extrakce cyklů

Extrakce cyklu začíná nejlevějším uzlem na ose x . Jak již bylo řečeno, nejdříve se nalezne sousední uzel jehož hrana je natočena nejvíce po směru hodinových ručiček. V případě obrázku 4.1 se jedná o hranu (V_0, V_1) . Ostatní ze sousedních uzlů jsou procházeny proti směru hodinových ručiček. Pokud se dostaneme k uzlu, který nemá žádného souseda, budeme posloupnost uzlů považovat za cestu. Pokud se ovšem dostaneme zpět k uzlu V_0 , našli jsme cyklus. Po nalezení cyklu můžeme odstranit hranu (V_0, V_1) . Ostatní hraný budou označeny jako hraný cyklu. Pokud nyní uzel V_0 nebo V_1 bude obsahovat pouze jednoho souseda, budou hraný těchto uzlů extrahovány, jako by se jednalo o cesty. Vzhledem k tomu, že jsou již označeny jako hraný cyklu, budou odstraněny bez dalšího uložení. Ostatní hraný cyklu budou odstraněny až při dalším průchodu.

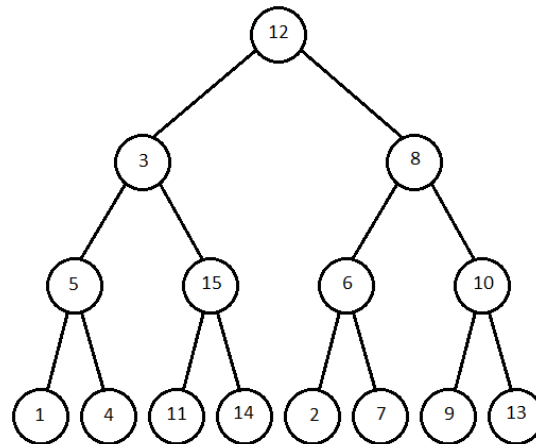
4.6 Heap (hromada)

Hromada je stromová struktura, pro kterou platí, že existuje stejný vztah mezi rodičovským uzlem a všemi jeho potomky, tedy například rodič je vždy menší než všichni jeho potomci. Nejčastěji se můžeme setkat s hromadou tvořenou binárním stromem. Pro nás je podstatné, že se v kořeni stromu vždy nachází extrém daného stromu, tedy maximální nebo minimální hodnota.

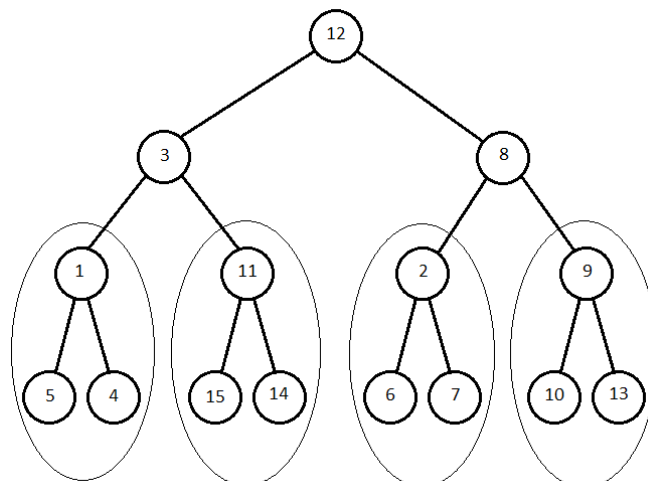
Pro hromadu je důležitá operace znovuoustavení hromady poté, co bylo porušeno pravidlo hromady v jednom z uzlů. Nejvýznamnějším případem porušení hromady je v jejím kořeni. Proces, který umožňuje znovuoustavit hromadu, a tedy odstranit její porušení, se nazývá „sift“ (prosetí). Jeho princip spočívá v tom, že prvek porušující pravidlo hromady je postupně vyměňován se svými

potomky, až propadne na místo, kde již splňuje pravidlo hromady. Tímto se i do kořene dostane prvek dodržující pravidlo hromady. Složitost této operace je v nejhorším případě $\log_2 n$.

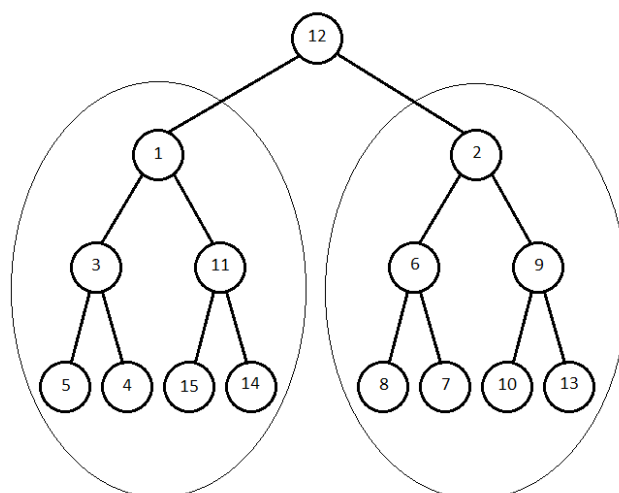
Pokud chceme z obecného binárního stromu vytvořit hromadu, je nutné opakovaně aplikovat proceduru „sift“ na uzly stromu, počínaje rodičovskými uzly na nejnižší úrovni a konče kořenem stromu. Postupné vytvoření hromady můžeme vidět na obrázcích 4.6.1, 4.6.2, 4.6.3 a 4.6.4. Všimněme si, že začínáme vytvářením sady menších hromad, které postupně spojujeme do větších, až dostaneme hromadu celého stromu.



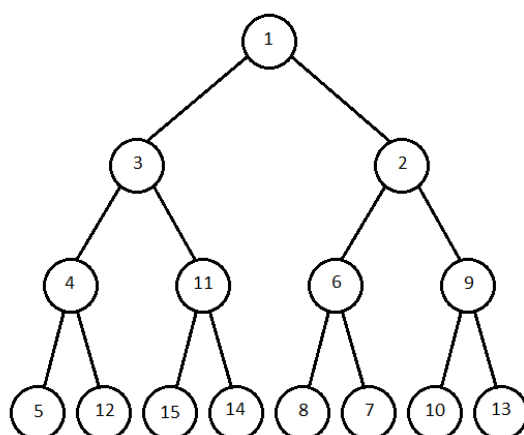
Obrázek 4.6.1: Výchozí strom



Obrázek 4.6.2: Vytvoření čtyř menších hromad



Obrázek 4.6.3: Vytvoření dvou větších hromad



Obrázek 4.6.4: Výsledná hromada

Pro hromadu je velmi podstatné, že ji lze snadno implementovat pomocí pole. Aby ale mohla být takto navržena, je potřeba, aby pro binární strom o n úrovních platilo, že na všech $n-1$ úrovních existují všechny uzly. Pokud je tato podmínka splněna, má každý rodičovský uzel s indexem i levého potomka na indexu $2i$ a pravého potomka na indexu $2i+1$.

Pokud je hromada implementována pomocí pole, můžeme její prvky řadit. Při řazení se využívá vlastnosti hromady, která může být definována tak, že v jejím kořeni je vždy nejmenší hodnota. Tato hodnota se umístí do výstupního pole a na místo kořene je vložen nejpravější uzel na nejnižší úrovni. Po zatřesení stromem se v kořeni opět nachází nejmenší hodnota. Hodnotu opět vložíme do výstupního pole. Takto pokračujeme, dokud jsou ve stromu nějaké prvky. Poté co jsou

tyto kroky provedeny nad všemi prvky hromady, dostáváme seřazenou posloupnost. Složitost algoritmu je lineární $(n \cdot \log_2 n)$.

V naší aplikaci je velmi podstatná schopnost efektivně znovuořadit hromadu po odebrání jednoho jejího prvku. V takovém případě je na pozici odebraného prvku vložen nejpravější uzel na nejnižší úrovni. Uzel porušující hromadu je poté prosetím posunut na správné místo. Jak již bylo řečeno, složitost operace prosetí je $\log_2 n$. Díky tomu jsme tedy schopni hledat minimum v dané struktuře s logaritmickou rychlostí.

5 Generování budov

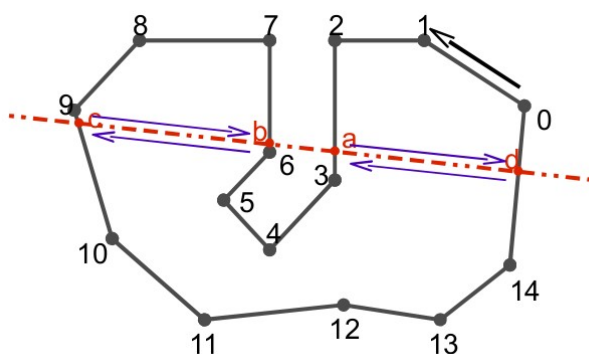
V této kapitole se budeme zabývat určením přesné pozice jednotlivých budov a jejich následným generováním.

5.1 Dělení stavebních pozemků

V předchozí kapitole jsme se zabývali hledáním minimálních cyklů v grafu, které by představovali stavební pozemky, na nichž bychom mohly umístit budovy. Na jednom stavebním pozemku ale bude umístěno více budov, musíme tedy rozdělit takovéto prostory na menší bloky.

Přístup navržený pro dělení prostor je založen na [3]. Jedná se o algoritmus, který rekurzivně dělí prostor na dvě části, dokud je dodržena minimální velikost bloku. Minimální velikost bloku může být ovlivněna hustotou zalidnění. V místech s velkým počtem obyvatel jsou obvykle vyšší budovy a tomu odpovídá i velikost jejich podstav. Po dokončení algoritmu jsou odstraněny všechny bloky, které nemají přímý přístup k silnici.

Algoritmus je přizpůsoben k dělení jak konkávních, tak i konvexních prostor. Dělicí čára je vždy kolmicí k nejdelší hraně představující hranici prostoru.



Obrázek 5.1: Dělení stavebních pozemků

(zdroj obrázku [3])

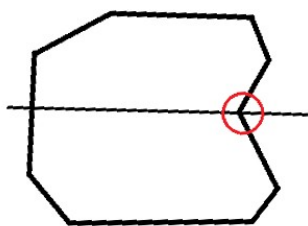
Jednotlivé kroky algoritmu jsou následující (průběh algoritmu je znázorněn na obrázku 5.1):

1. Nejdříve je identifikována nejdelší hrana okolo daného pozemku. Následně je vypočítána kolmice ze středu této hrany.
2. Vypočtené průsečíky mezi dělicí čarou a hraničními čarami jsou uloženy společně s hodnotou indexu vrcholu patřícího protnuté hraniční čáře. Průsečíky jsou poté seřazeny podle tohoto indexu. Seřazené průsečíky, které spolu sousedí, tvoří buď novou hranu bloku pro stavbu budovy, nebo hranu mimo stavební plochu. {a:2, b:7, c:9, d:14}

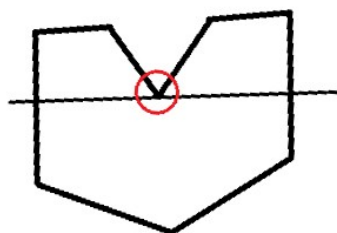
3. Následně jsou odstraněny části dělicí čáry, které se nacházejí mimo stavební plochu a jsou uloženy ty, které se nacházejí uvnitř. $\{c \rightarrow b, a \rightarrow d\}$
4. Původní polygon je spojen s grafem tvořeným novými hranami nacházejícími se uvnitř polygonu. Poté jsou nalezeny cykly v tomto nově vytvořeném grafu.

Identifikace čar uvnitř polygonu

Pro rozlišení části dělicí čáry nacházející se uvnitř a vně polygonu je použita metoda velmi podobná řádkovému vyplňování polygonu. Využívá se tedy seřazení jednotlivých průsečíků, přičemž oblast mezi lichým a sudým průsečíkem se nachází uvnitř polygonu. Problémové je ovšem protnutí dělicí čáry ve vrcholu polygonu. V takovém případě vznikají průsečíky dva. Při protínání vrcholů polygonů ale rozlišujeme dva případy. V jednom případě požadujeme dva průsečíky, ve druhém jenom jeden. Situace, kdy chceme pouze jeden průsečík nastává v případě, že daný vrchol polygonu je lokálním extrémem ve směru dělicí čáry (viz obrázek 5.1.1 a). V případě, že se naopak jedná o lokální extrém kolmý ke směru dělicí čáry, vyžadujeme vrcholy dva (viz obrázek 5.1.1 b). Naším cílem tedy je, aby počet průsečíků byl vždy sudý.

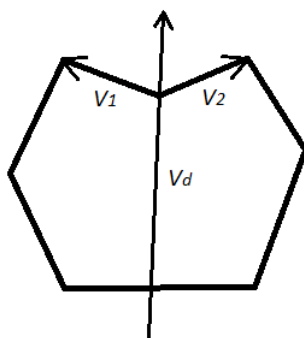


Obrázek 5.1.1 a: Jeden průsečík ve vrcholu polygonu



Obrázek 5.1.1 b: Dva průsečíky ve vrcholu polygonu

Všimněme si, že v případě obrázku 5.1.1 a jsou sousední vrcholy protnutého vrcholu na opačných stranách dělicí čáry, zatímco na obrázku 5.1.1 b jsou sousední vrcholy protnutého vrcholu na stejné straně dělicí čáry. Tato skutečnost nám pomůže odlišit, kdy použít dva průsečíky a kdy pouze jeden. Jednoduchým řešením tohoto problému je použití vektorového součinu dvou vektorů. Nejdříve vypočítáme vektorový součin pro vektor daný hranou vedoucí z protnutého vrcholu (v_1) a vektor daný dělicí čarou (v_d). Dále vypočítáme součin mezi vektorem daným druhou hranou vedoucí z protnutého vrcholu (v_2) a opět vektorem daným dělicí čarou (v_d). Pokud mají výsledné vektory daných součinů různou orientaci, nacházejí se sousední vrcholy protnutého vrcholu na opačné straně od dělicí čáry a v místě průniku vzniká pouze jeden průsečík. Podrobnější informace o vektorovém součinu nalezneme v kapitole 5.3.

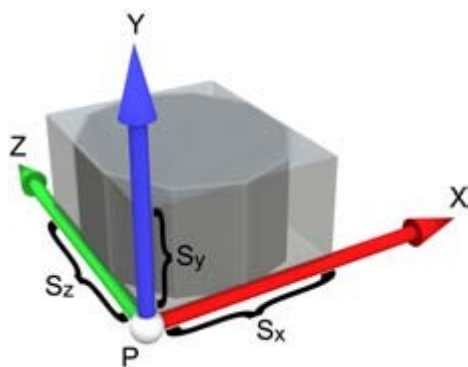


*Obrázek 5.1.2: Vektory
použité při vektorovém
součtinu*

5.2 L-systém pro generování budov

Pro generování geometrie budov bude sloužit rozšířený L-systém vycházející z práce [2]. Zatímco běžný L-systém používá pro generování paralelní přístup, pro generování budov je vhodnější sekvenční přístup.

Navržená gramatika pracuje se sadou geometrických objektů. Každý objekt se skládá ze symbolu, geometrických vlastností a číselných atributů. Symbol označuje typ objektu. Může se jednat buď o terminální symbol $\in \Sigma$ nebo non-terminální symbol $\in V$. Objekty jsou poté nazývány terminální nebo non-terminální. Mezi nejdůležitější geometrické vlastnosti patří pozice P , pravoúhlé vektory X, Y, Z popisující souřadnicový systém a vektor S popisující velikost. Tyto atributy definují obalové těleso.



*Obrázek 5.2.1: Objekt s obalovým
tělesem (zdroj obrázku [2])*

Proces použití přepisovacích pravidel

Máme konečnou sadu základních objektů. Aplikace přepisovacích pravidel začíná použitím sady objektů (zpravidla obsahující jediný objekt) nazývané axiom a pokračuje následovně:

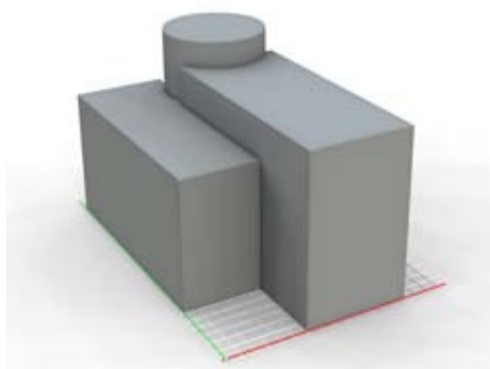
1. Z nabízené sady objektů vybereme aktivní objekt se symbolem B .
2. Vybereme pravidlo se symbolem B na levé straně, použijeme pravidlo a získáme nové objekty z pravé strany, označené jako $BNEW$.
3. Nastavíme objekt B jako neaktivní, vložíme objekty $BNEW$ do sady objektů a pokračujeme bodem 1.

Jakmile sada objektů neobsahuje žádné non-terminály, používání přepisovacích pravidel je ukončeno. V závislosti na způsobu výběru v prvním kroku může být derivační strom vytvářen do hloubky nebo do šířky.

Pravidla pro obalové těleso

Podobně jako u běžného L-systému i zde používáme obecná pravidla, abychom modifikovali aktuální objekt. Pro posun je použit vektor $T(t_x, t_y, t_z)$, který upravuje pozici bodu P . R_x , R_y , R_z provádí rotace kolem příslušných os souřadného systému. $S(s_x, s_y, s_z)$ mění velikost obalového tělesa. Dále používáme symbol „[“ k uložení současného obalového tělesa na zásobník a „]“ k odebrání současného obalového tělesa ze zásobníku. Příkaz $I(objId)$ přidává instanci objektu. Příklad níže popisuje vytvoření objektu na obrázku 5.2.2:

$A \rightarrow [T(0,0,6) S(8,10,18) I(„cube“)] T(6,0,0) S(7,13,18) I(„cube“) T(0,0,16) S(8,15,8) I(„cylinder“)$



Obrázek 5.2.2: Vygenerovaný objekt
(zdroj obrázku [2])

Základní pravidla pro dělení

Tato pravidla dělí současné obalové těleso podél jedné osy. Uvažujme například rozdělení stěny na čtyři patra a jednu římsu:

$$\text{fac} \rightarrow \text{Subdiv}(,Y'', 3.5, 0.3, 3, 3, 3) \{ \text{floor} \mid \text{ledge} \mid \text{floor} \mid \text{floor} \mid \text{floor} \}$$

První parametr popisuje osu, podle které budeme dělit („X“, „Y“, „Z“) a zbývající parametry popisují velikosti jednotlivých částí rozděleného objektu. Mezi složenými závorkami je seznam nových objektů.

Pravidla pro změnu velikosti

Z předchozího příkladu je zřejmé, že dané pravidlo je navrženo pro změnu objektu o velikosti $y = 12,8$. Pro jiné objekty je ovšem třeba změnit velikosti, na které budeme objekt dělit. Ne vždy ovšem chceme změnit všechny rozměry stejně. Rozlišujeme tedy mezi absolutními hodnotami (jejich rozměry se nemění) a relativními hodnotami (jejich rozměry se mění). Implicitně jsou předpokládány absolutní hodnoty. Pro odlišení relativních hodnot používáme písmeno r .

$$\text{floor} \rightarrow \text{Subdiv}(,X'', 2, 1r, 1r, 2) \{ B \mid A \mid A \mid B \}$$

Zde jsou relativní hodnoty r_i nahrazeny hodnotou následující: $r_i \cdot (Scope.sx - \sum abs_i) / \sum r_i$. $Scope.sx$ reprezentuje velikost obalového tělesa ve směru osy x , $\sum abs_i$ odpovídá součtu absolutních hodnot a $\sum r_i$ je součtem relativních hodnot. Při dělení tedy budou mít objekty B hodnotu 2, zatímco objekty A si mezi sebou rozdělí zbývající velikost původního objektu.

Pravidla pro opakování

Často můžeme chtít opakovat některý model vícekrát. Příklad pravidla řešícího tento problém může být následující:

$$\text{floor} \rightarrow \text{Repeat}(,X'', 2) \{ B \}$$

V tomto případě bude podlaha po ose x vyskládána tolika dlaždicemi, jak jen to dovoluje daný prostor. Hodnota uvedená jako druhý parametr představuje rozměr dlaždice. Počet opakování je potom $\lfloor Scope.sx / 2 \rfloor$.

Pravidla pro rozdělení na složky

Až doted' byly všechny objekty trojrozměrné. Následující pravidlo umožňuje rozdělit objekt na jednotlivé méně rozměrné složky:

$$a \rightarrow \text{Comp}(\text{type}, \text{param}) \{ A \mid B \mid \dots \mid Z \}$$

Hodnota *type* představuje typ komponenty, kterou budeme dělit. Například $Comp(„faces“)\{A\}$ vytvoří model A pro každou stranu původního trojrozměrného objektu. Stejně tak můžeme použít $Comp(„edges“)\{B\}$ k rozdělení hran na vrcholy. Parametr *param* je volitelný a slouží k výběru konkrétní komponenty daného objektu. Například $Comp(„edge“, 3)\{A\}$ vytvoří objekt A odpovídající třetí hraně původního modelu.

Použitá gramatika budov

Pro samotný program je použita následující sada pravidel:

- p1: $lot \rightarrow S(1r, building_height, 1r) bounding$
- p2: $bounding : Scope.sy > min_height$
 $\rightarrow Subdiv(„Y“, Scope.sy*0.1, 1r) \{ base \mid S(Scope.sx*0.8, 1r, Scope.sz*0.8) main \}$
- p3: $bounding : Scope.sy \leq min_height \rightarrow main$
- p4: $main : size_of_base > min_size \wedge Scope.sx > Scope.sz$
 $\rightarrow Subdiv(„X“, 1r, 1r, 1r) \{ S(1r, Scope.sy*rand(0.1,0.6), 1r) sidewing \mid$
 $S(Scope.sx*2.0, 1r, Scope.sz*0.8) middle \mid S(1r, Scope.sy*rand(0.1,0.6), 1r) sidewing \} : 0.7$
 $\rightarrow middle : 0.3$
- p5: $main : size_of_base > min_size \wedge Scope.sx \leq Scope.sz$
 $\rightarrow Subdiv(„Z“, 1r, 1r, 1r) \{ S(1r, Scope.sy*rand(0.1,0.6), 1r) sidewing \mid$
 $S(Scope.sx*0.8, 1r, Scope.sz*2.0) middle \mid S(1r, Scope.sy*rand(0.1,0.6), 1r) sidewing \} : 0.7$
 $\rightarrow middle : 0.3$
- p6: $main : size_of_base \leq min_size \rightarrow middle$
- p7: $middle : size_of_base > min_size \wedge Scope.sx > Scope.sz$
 $\rightarrow Subdiv(„X“, 1r, 1r, 1r) \{ floors \mid S(Scope.sx*1.5f, 1r, Scope.sz*0.8f) floors \mid floors \} : 0.8$
 $\rightarrow floors : 0.2$
- p8: $middle : size_of_base > min_size \wedge Scope.sx \leq Scope.sz$
 $\rightarrow Subdiv(„Y“, 1r, 1r, 1r) \{ floors \mid S(Scope.sx*0.8f, 1r, Scope.sz*1.5f) floors \mid floors \} : 0.8$
 $\rightarrow floors : 0.2$
- p9: $middle : size_of_base \leq min_size \rightarrow floors$
- p10: $floors : Scope.sy > min_height$
 $\rightarrow Subdiv(„Y“, Scope.sy*0.618, 1r) \{ final \mid S(Scope.sx*0.9, 1r, Scope.sz*0.9) floors \} : 0.5$
 $\rightarrow final : 0.5$
- p11: $floors : Scope.sy \leq min_height \rightarrow final$

Pravidlo *p1* vytváří na místě stavebního pozemku základní tvar budovy. Tato operace je popsána jako zvětšení v ose *y*. Nový objekt se nazývá *bounding*. Výsledek aplikace tohoto pravidla můžeme vidět na obrázku 5.2.3.

Druhé a třetí pravidlo je použito na objekt typu *bounding*. Cílem třetího pravidla je rozdělit budovu v ose *y* na podstavu (*base*) a hlavní část budovy (*main*). Výška podstavu je 10 % výšky původního objektu, výška hlavní části budovy představuje zbytek, tedy 90 % výšky původní budovy. Hlavní část je navíc zmenšena v ose *x* a *z*, a to na hodnotu 80 % původní velikosti. Aby byla budova dělena výše popsaným způsobem, je potřeba, aby její výška splňovala určité minimum. V opačném případě je na objekt *bounding* aplikováno třetí pravidlo, a je tedy pouze změněn na objekt *main*. Na obrázku 5.2.4 můžeme vidět použití pravidla *p2*.

Pravidla *p4*, *p5* a *p6* jsou aplikována na objekty typu *main*. Pravidlo *p4* se stará o dělení budovy v ose *x*. Pravidlo *p5* dělí budovu v ose *z*. Pokud je velikost obalového tělesa v ose *x* větší, než velikost obalového tělesa v ose *z*, pak se použije pravidlo *p4*, v opačném případě se použije pravidlo *p5*. Aby došlo k dělení, je opět potřeba, aby objekt splňoval jistou minimální velikost. Pokud této velikosti nedosahuje, je použito pravidlo *p6*. V pravidlech *p4* a *p5* je objekt dělen na tři stejně velké části, tedy na objekty o rozměru 1/3 původní velikosti. Postranní části se nazývají *sidewing*, prostřední se nazývá *middle*. Výška postranních částí je zmenšena na náhodnou hodnotu v rozmezí 10 až 60 % původní výšky objektu. V případě prostřední části se mění velikost v ose *x* a *z*. Na ose, podle které jsme dělili, zvětšíme objekt na dvojnásobek původní velikosti, na druhé ose objekt naopak zmenšíme, a to na hodnotu 80 % původní velikosti. K dělení ovšem navíc dochází pouze v 70 % případů. Pokud k dělení nedojde, je objekt typu *main* přepsán na objekt typu *middle*. Na obrázku 5.2.5 můžeme vidět aplikaci pravidla *p4* v případě, že k dělení dojde.

Na objekty typu *middle* jsou použita pravidla *p7*, *p8*, *p9*. Tato pravidla jsou velmi podobná pravidlům z předchozího odstavce (*p4*, *p5*, *p6*). Dochází tedy opět k dělení v ose *x* v případě pravidla *p7* a dělení v ose *z* v případě pravidla *p8*. Opět vybíráme pravidlo v závislosti na tom v které ose je objekt větší a je potřeba pro dělení dodržet jistou minimální velikost podstavu. Objekt je také dělen na tři stejně velké části. Všechny nové objekty jsou tentokrát stejného typu *floors*. U tohoto pravidla ovšem dochází k dělení v 80 % případů. Další odlišností je změna velikosti nových objektů. Postranní objekty zůstávají beze změny. Prostřední objekt mění velikost na ose *x* a *z*. Na ose podle které jsme dělili, zvětšíme objekt na 150 % původní velikosti. Na druhé ose zmenšíme objekt na 80 % původní velikosti. Pokud k dělení nedojde, přepíše se objekt typu *middle* na objekt typu *floors*. Na obrázku 5.2.6 můžeme vidět aplikaci pravidla *p7* v případě, že k dělení dojde.

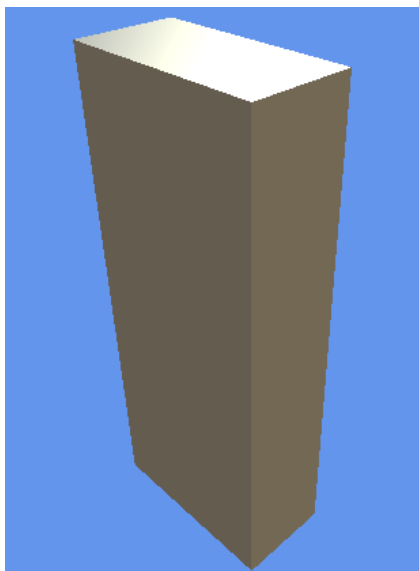
Pravidla *p10* a *p11* jsou použita na objekty typu *floors*. Pravidlo *p10* umožňuje dělení objektu v ose *y*. Aby ale k dělení došlo, musí být opět dodržena jistá minimální výška. Pokud dodržena není, použije se pravidlo *p11*, které přepíše objekt typu *floors* na objekt typu *final*. K dělení u pravidla *p10* navíc dochází pouze v 50 % případů. Objekt je dělen v poměru zlatého řezu, což je:

$$\frac{1+\sqrt{5}}{2} \approx 1.618\,033\,988\,749\,894\,848 \quad (1)$$

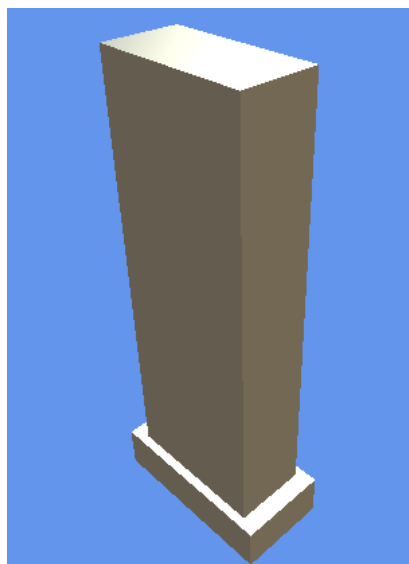
Nás ovšem zajímá, kolik je to procent vzhledem k celkové výšce objektu. To spočítáme jako:

$$\frac{1+\sqrt{5}}{1+\sqrt{5}+2} \approx 0.618\,033\,988\,749\,894\,848 \quad (2)$$

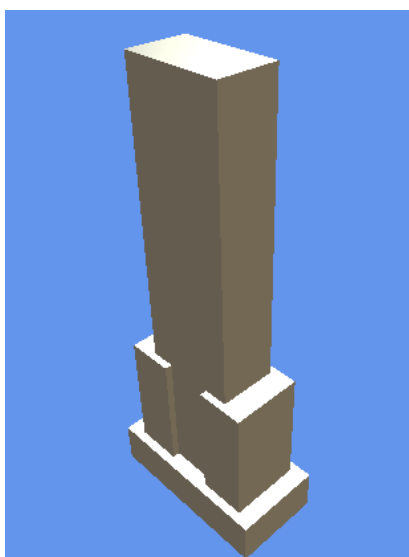
Spodní část rozděleného objektu se poté nazývá *final*. Horní část je stejného typu jako levá strana pravidla a je tedy možné ji dále dělit. Horní část je navíc zúžena na 90 % původní hodnoty. Na obrázku 5.2.7 vidíme případ, kdy s použitím těchto pravidel došlo ke třem dělením.



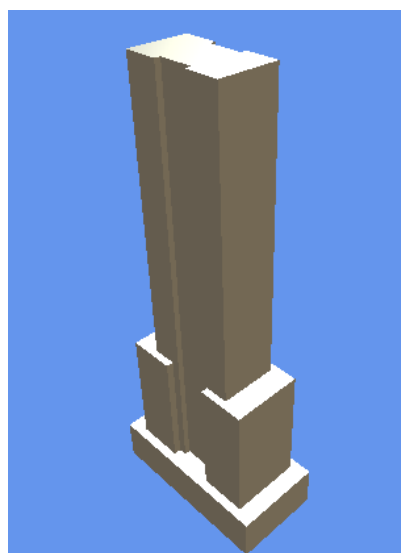
Obrázek 5.2.3: Aplikace pravidla p1



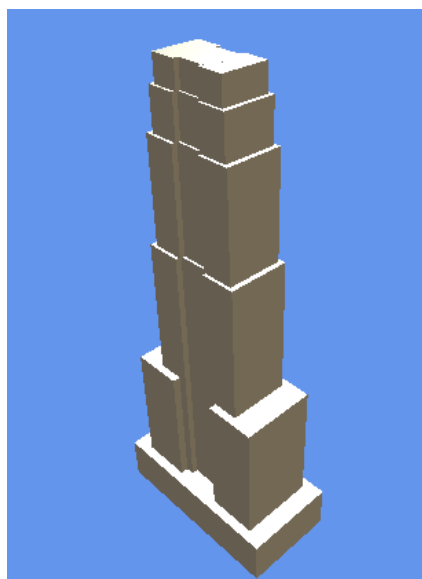
Obrázek 5.2.4: Aplikace pravidla p2



Obrázek 5.2.5: Aplikace pravidla p4



Obrázek 5.2.6: Aplikace pravidla p7

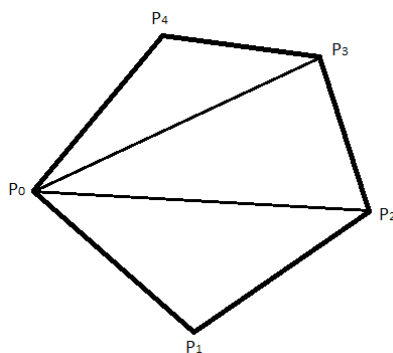


*Obrázek 5.2.7: Aplikace
pravidla p10*

5.3 Triangulace konkávního polygonu

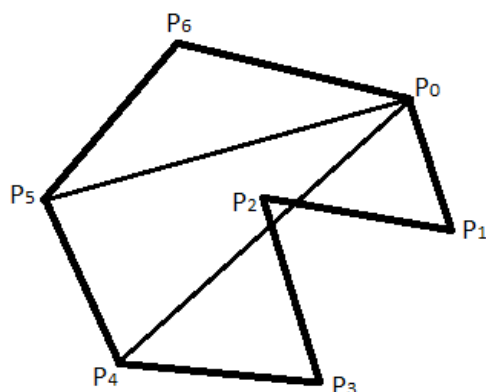
Při vytváření budov se často setkáváme se situací, že jedna ze stěn případně střecha odpovídá svým tvarem nepravidelnému polygonu, a to navíc polygonu konkávnímu. Abychom mohli takovýto polygon dále zpracovávat, je potřeba ho převést na trojúhelníky. Tomuto procesu se říká triangulace a popíšeme si ho právě v této kapitole.

Pro rozložení konvexního polygonu na trojúhelníky je situace jednoduchá. Vybereme jakýkoliv vrchol polygonu a označíme ho P_0 . Každý trojúhelník poté bude tvořen dvěma sousedními vrcholy polygonu a právě vrcholem P_0 . Zjednodušeně řečeno propojíme vrchol P_0 se všemi ostatními vrcholy.



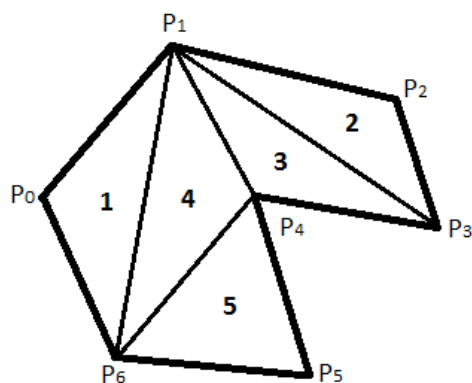
Obrázek 5.3.1: Triangulace konvexního polygonu

V případě konkávního polygonu je ovšem situace o něco složitější. Jak můžeme vidět na obrázku 5.3.2, při spojování vrcholů vznikají překrývající se trojúhelníky.



Obrázek 5.3.2: Nesprávná triangulace konkávního polygonu

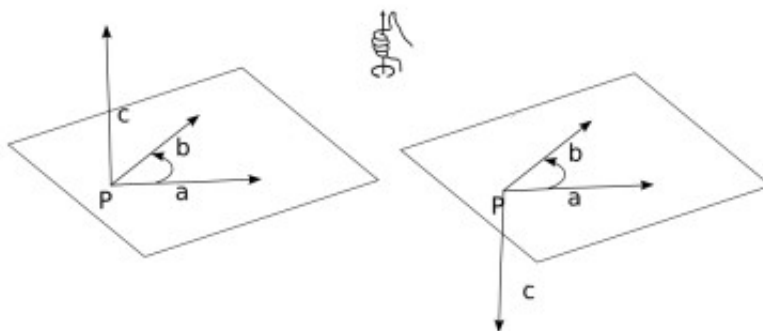
Algoritmus, který by řešil konkávní polygony je tedy o něco složitější. Procházíme všechny vrcholy a pro každý z nich se pokoušíme vytvořit trojúhelník z aktuálního vrcholu a dvou vrcholů sousedních. Podstatné ovšem je, aby v místě vrcholu byl polygon konvexní, v opačném případě by nový trojúhelník vznikl mimo daný polygon. Druhou významnou podmínkou vzniku trojúhelníku je, aby se uvnitř něj nenacházel jiný vrchol polygonu. Pokud jsou dodržena tato kritéria, je trojúhelník vytvořen a je odstraněn z daného polygonu. Přesněji řečeno je odstraněn daný vrchol, ze kterého jsme při vytváření trojúhelníku vycházeli. Procházení vrcholů poté probíhá nad novým polygonem. Těmito kroky postupně zanikají vrcholy, které jsou pro daný polygon konkávní a s nimi zaniká i možnost, že by se uvnitř trojúhelníku nacházel jiný vrchol polygonu. Na obrázku 5.3.3 vidíme vytváření trojúhelníků pomocí tohoto algoritmu, včetně pořadí v jakém byly trojúhelníky vytvořeny.



Obrázek 5.3.3: Správná triangulace konkávního polygonu

V předchozích odstavcích jsme došli k závěru, že je potřeba zjišťovat, zda je polygon v daném místě vrcholu konvexní či konkávní. K řešení takového problému je velmi vhodné využít vektorového součinu vektorů. Problematikou součinu vektorů jsem se částečně zabývali již v kapitole 4.

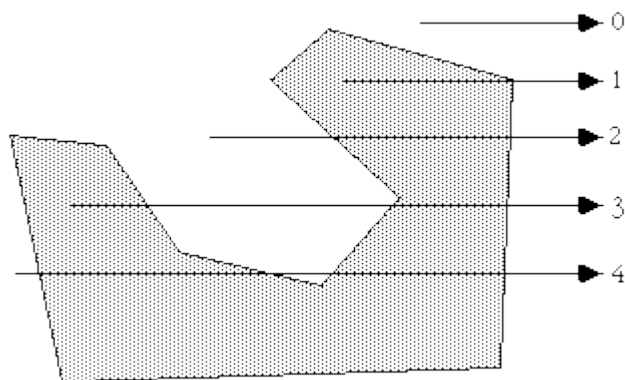
Víme, že vektorový součin dvou vektorů je definován jako vektor kolmý k těmto dvěma vektorům. Nás ovšem zajímá i orientace daného vektoru, která je dána pravidlem pravé ruky, a to následujícím způsobem. Mějme vektory a a b a jejich součin $a \times b$. Pokud položíme pravou ruku na pomyslnou rovinu určenou vektory a a b tak, aby pokrčené prsty směřovaly od vektoru a k vektoru b nejkratším směrem, pak vztyčený palec ukazuje směr výsledného vektoru vzniklého daným součinem. Co se tedy stane, když dané pravidlo aplikujeme na sousední vektory polygonu z obrázku 5.3.3? Pokud budeme při součinu vybírat vektory například ve směru hodinových ručiček, zjistíme, že pro vrcholy, které jsou konvexní, je výsledný vektor kladný, zatímco pro konkávní vrcholy je vektor záporný. Orientace vektoru vzniklého vektorovým součinem nám tedy umožňuje rozlišit, zda je polygon v daném vrcholu konvexní či konkávní.



*Obrázek 5.3.4: Orientace vektoru daná pravidlem pravé ruky
(zdroj obrázku [15])*

Jak již bylo řečeno, při vytváření trojúhelníků je vždy potřeba ověřit, zda trojúhelník neobsahuje jiný vrchol polygonu. Popíšeme si tedy algoritmus vycházející z [14], který určuje, zda se bod nenachází uvnitř polygonu.

Hlavní myšlenka tohoto algoritmu je následující. Abychom určily stav v jakém se zkoumaný bod nachází, je třeba uvažovat horizontální čáru směřující z onoho bodu směrem doprava. Pokud je počet průsečíků dané čáry s hraničními čarami polygonu sudý, bod je mimo polygon. Pokud je počet průsečíků lichý, bod je uvnitř polygonu. Že tato myšlenka opravdu platí, můžeme vidět na obrázku 5.3.5.



*Obrázek 5.3.5: Testování stavu bodu
(zdroj obrázku (14))*

V implementaci je potom každá hrana testována vůči horizontální čáře následujícím způsobem. Je potřeba ověřit, že je y -ová hodnota zkoumaného bodu uvnitř rozmezí testované hrany. Dále je potřeba ověřit, že x -ová hodnota zkoumaného bodu je nalevo od testované hrany. Pokud jsou obě podmínky splněny došlo k průniku.

6 Analytická geometrie

Při implementaci programu pro generování města je často zapotřebí řešit průniky různých geometrických útvarů. Jedná se především o hledání průsečíku dvou úseček, který je potřebný při napojování protínajících se silnic, dělení stavebních pozemků apod. Řešení tohoto problému je popsáno v kapitole 6.1. Druhým problémem, který si popíšeme, je hledání průniku kružnice a úsečky. To je opět zapotřebí při napojování silnic, tentokrát ovšem při hledání silnic v blízkosti koncového bodu jiné silnice. Řešení tohoto problému nalezneme v kapitole 6.2.

6.1 Průsečík dvou úseček

Metoda vybraná pro hledání průsečíku dvou úseček vychází z [12]. Nejdříve definujeme dvě úsečky pomocí jejich koncových bodů. Definice první úsečky v parametrickém tvaru je následující:

$$P_a = P_1 + U_a(P_2 - P_1) \quad (1)$$

P_1 představuje počáteční bod úsečky, P_2 je koncovým bodem úsečky. $P_2 - P_1$ tedy odpovídá vektoru úsečky. U_a nabývá hodnoty od 0 do 1. Povšimněme si, že pokud je hodnota U_a rovna 0, nacházíme se v počátečním bodě úsečky (P_1), zatímco v případě hodnoty 1 se nacházíme v koncovém bodě úsečky (P_2).

Rovnice druhé úsečky je zapsána v totožném tvaru, tedy:

$$P_b = P_3 + U_b(P_4 - P_3) \quad (2)$$

P_3 je první bod úsečky, P_4 je druhý bod úsečky a U_b může nabývat hodnoty od 0 do 1. Pozici průsečíku poté můžeme najít v bodě, kde se dvě výše uvedené rovnice rovnají, tedy:

$$P_a = P_b \quad (3)$$

Po dosazení:

$$P_1 + U_a(P_2 - P_1) = P_3 + U_b(P_4 - P_3) \quad (4)$$

Pokud rovnici (4) rozepíšeme pro osu x a y , dostaneme soustavu rovnic:

$$x_1 + U_a(x_2 - x_1) = x_3 + U_b(x_4 - x_3) \quad (5)$$

$$y_1 + U_a(y_2 - y_1) = y_3 + U_b(y_4 - y_3) \quad (6)$$

Polohu bodu nám určuje hodnota proměnných U_a a U_b . Pokud tedy chceme nalézt průsečík, hledáme jejich hodnoty v situaci, kdy se rovnice úseček rovnají. Proměnné U_a a U_b proto vyjádříme z předchozí soustavy rovnic:

$$U_a = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (7)$$

$$U_b = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \quad (8)$$

Z rovnic (7) a (8) vidíme, že jmenovatel je v obou případech stejný. Pro úsečky platí, že pokud je hodnota daného jmenovatele nulová, jsou úsečky rovnoběžné. Pokud jsou nulové i oba čitatele, jsou úsečky shodné.

Všimněme si ale, že pokud úsečky nejsou rovnoběžné nebo shodné, je nalezen průsečík vždy. Je to z toho důvodu, že výše uvedené řešení je určené pro průsečík přímek. Abychom vypočítaly průsečík úseček, musíme navíc omezit hodnotu U_a a U_b tak, že bude platit:

$$0 \leq U_a \leq 1 \quad (9)$$

$$0 \leq U_b \leq 1 \quad (10)$$

Pokud nyní chceme dostat průsečík úseček, stačí hodnotu U dosadit do parametrické rovnice jedné z úseček. Do rovnice úsečky první dosazujeme U_a , do rovnice úsečky druhé dosazujeme U_b . Dosazení do rovnice první úsečky je tedy následující:

$$x = x_1 + U_a(x_2 - x_1) \quad (11)$$

$$y = y_1 + U_a(y_2 - y_1) \quad (12)$$

Poznámka k implementaci

Při samotné implementaci samozřejmě není vhodné hledat průsečík dané úsečky (silnice) se všemi ostatními úsečkami. Pokusíme se proto odfiltrovat nepotřebné úsečky. Představme si, že jsou jednotlivé úsečky obaleny obdélníky. Pokud se dané obdélníky překrývají, je šance, že dojde k průniku. Testování překrytí obdélníků je pak velmi jednoduché. Otestujeme možné překrytí na ose x a y . Pokud je levý vrchol jednoho obdélníku více vpravo, než pravý vrchol druhého obdélníku, k překrytí nedošlo. Pokud je horní vrchol jednoho obdélníku níže, než dolní vrchol druhého obdélníku, k překrytí nedošlo. V opačném případě k překrytí došlo. Pouhým porovnáváním se tedy zbavíme většiny nepotřebných úseček.

6.2 Průsečík kružnice a úsečky

Řešení navržené pro tento projekt vychází z [13]. Ze všeho nejdříve definujeme rovnici úsečky v parametrickém tvaru, stejně jako v kapitole 6.1, tedy:

$$P = P_1 + U(P_2 - P_1) \quad (1)$$

P_1 je počáteční bod úsečky, P_2 je koncový bod úsečky. U nabývá hodnoty od 0 do 1. Pro jednotlivé osy jsou potom rovnice následující:

$$x = x_1 + U(x_2 - x_1) \quad (2)$$

$$y = y_1 + U(y_2 - y_1) \quad (3)$$

Rovnice kružnice se středem v bodě $P_3(x_3, y_3)$ a poloměrem r má rovnici:

$$(x - x_3)^2 + (y - y_3)^2 = r^2 \quad (4)$$

Pokud substituujeme rovnici úsečky do rovnice kružnice dostaneme následující kvadratickou rovnici:

$$au^2 + bu + c = 0 \quad (5)$$

kde

$$a = (x_2 - x_1)^2 + (y_2 - y_1)^2 \quad (6)$$

$$b = 2[(x_2 - x_1)(x_1 - x_3) + (y_2 - y_1)(y_1 - y_3)] \quad (7)$$

$$c = x_3^2 + y_3^2 + x_1^2 + y_1^2 - 2(x_3x_1 + y_3y_1) - r^2 \quad (8)$$

Řešením kvadratické rovnice je potom:

$$U_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Diskriminant $b^2 - 4ac$ je při vyhodnocení průniku velmi podstatný. Pokud je jeho hodnota menší než nula, nedošlo k průniku. Pokud je diskriminant roven nule, je možné, že došlo k nalezení tečny. Pokud by se jednalo o průnik kružnice a přímky došlo by k nalezení tečny zcela jistě, v případě průniku kružnice a úsečky je ovšem navíc nutné ověřit, že se výsledek nachází v rozsahu 0 až 1. Pokud je hodnota diskriminantu větší než nula, je možné, že došlo k nalezení průsečíku. Opět je potřeba, aby hodnota U byla v rozsahu od 0 do 1. Pokud budeme hledat nejen průsečík kružnice s úsečkou, ale jakoukoliv kolizi těchto dvou objektů, bude nás zajímat i případ, kdy je úsečka uvnitř kružnice. K takovéto situaci dochází v případě, že je jedna hodnota U kladná a druhá záporná.

7 Implementace

Aplikace je implementována v jazyce C# s použitím XNA Frameworku. Vývojovým prostředím je XNA Game Studio 3.1 spolu s Visual Studiem 2008. Cílovou platformou je operační systém Windows, minimálně ve verzi XP.

7.1 XNA Framework

XNA Framework vychází z .NET Frameworku v případě implementace pro Windows. Pro Xbox 360 a Windows Phone 7 je použit .NET Compact Framework. XNA Framework obsahuje rozšířenou sadu knihoven pro vývoj her se snahou o maximální znovupoužitelnost kódu při převodu na jednotlivé cílové platformy. Grafika je založena na DirectX. Jako vývojové prostředí se používá rozšíření Visual Studia zvané XNA Game Studio. Hry mohou být napsány v jakémkoliv .NET kompatibilním jazyce. Pouze C# je ovšem podporovaný v XNA Game Studiu.



Obrázek 7.1.1: Princip XNA Frameworku (zdroj obrázku [17])

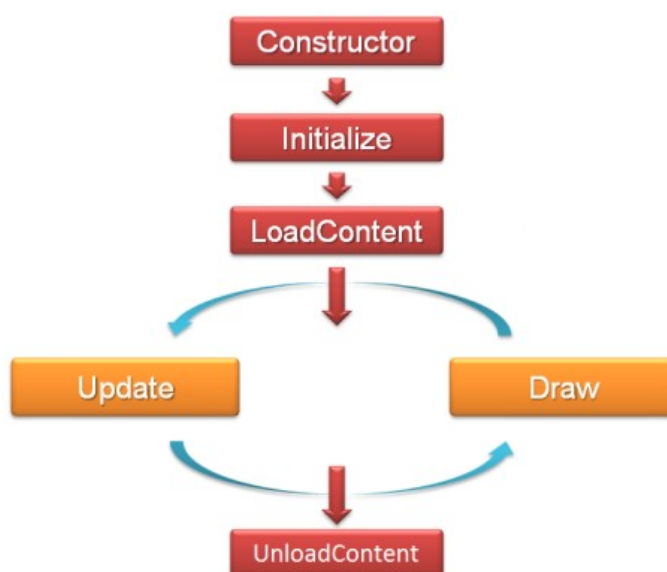
XNA Framework se skládá ze dvou základních částí. Těmi jsou Core Framework a Extended Framework. Core Framework obsahuje všechny nezbytné součásti, jako matematickou knihovnu pro práci s vektory a maticemi, knihovnu pro ošetření vstupů, knihovnu pro práci se zvukem apod. Extended Framework navíc řeší další problémy, se kterými se vývojáři mohou často setkat. Extended Framework se skládá z části Application Model a Content Pipeline.



*Obrázek 7.1.2: Rozdělení XNA Frameworku
(zdroj obrázku [17])*

Application Model

Application Model představuje šablonu aplikace. Každý nový XNA Game projekt má třídu `Game1`, která obsahuje sadu metod. Těmito metodami jsou `Initialize`, `LoadContent`, `Update`, `Draw` a `UnloadContent`. `Initialize` slouží k počátečnímu hernímu a grafickému nastavení, nastavení kamery apod. `LoadContent` může být použit k načítání herního obsahu, jako jsou například obrázky, modely a zvuky. `Update` by měl obstarávat herní logiku a zpracování vstupů. `Draw` slouží k vykreslování scény. `UnloadContent` by měl uvolňovat herní obsah načtený v `LoadContent`. Extended Framework tedy vývojářům obstarává herní smyčku, určuje, kde zpracovat vstupy apod. Běh aplikace můžeme vidět na obrázku 7.1.3.

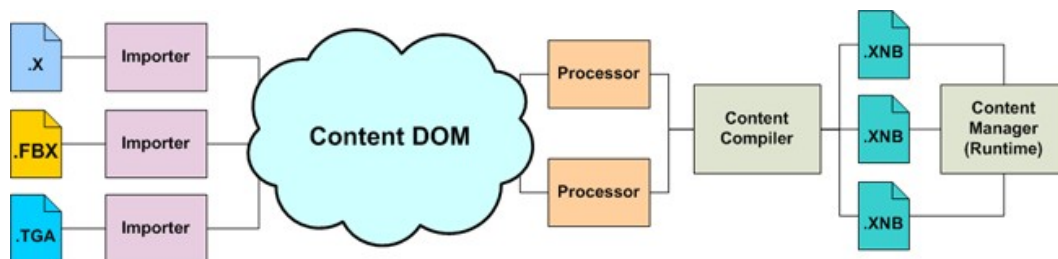


Obrázek 7.1.3: Průběh XNA aplikace (zdroj obrázku [17])

Content Pipeline

Content Pipeline sjednocuje zpracování herního obsahu, jako jsou obrázky, modely apod. Jednotlivé části Content Pipeline můžeme vidět na obrázku 7.1.4. První částí je `Importer`, který vezme data a

normalizuje je. To znamená, že se například nemusíme starat o natočení modelu dané modelovacím nástrojem, ve kterém byl model vytvořen. Poté, co Importer dokončí svou práci, se data nacházejí v tzv. content DOM. Zde jsou data silně typovaná, to znamená, že jsou reprezentována jako dobře známý formát. Například série vrcholů, texturová data, apod. Processor bere data z content DOM a vytváří objekt použitelný za běhu programu. Content Compiler potom ukládá výsledná data do souboru (typicky s koncovkou .xnb). Content Manager se nakonec stará o načtení dat ze souboru a převod na objekt použitelný v XNA.



Obrázek 7.1.4: Content Pipeline (zdroj obrázku [18])

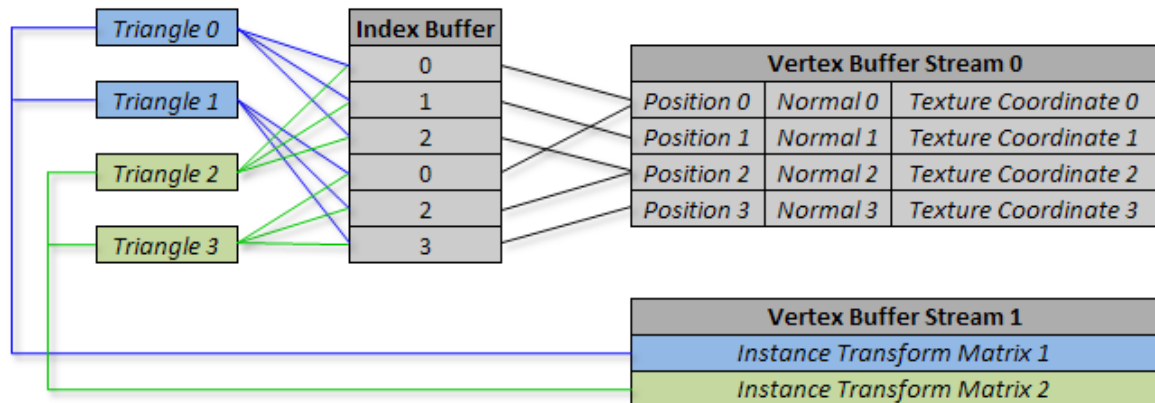
Instancování

XNA Framework umožňuje vykreslovat objekty pomocí tzv. instancování (anglicky instancing). Přestože tato technika nakonec nebyla ve výsledném programu implementována, zmíníme se o ní, alespoň jako o možném rozšíření.

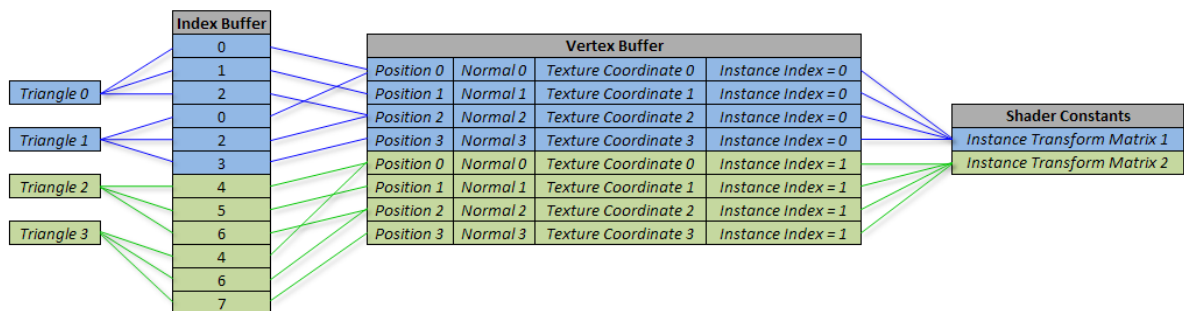
Instancování je technika umožňující vykreslování několika instancí jednoho objektu za použití GPU, zatímco CPU je zatíženo co nejméně. Instancování funguje obecně tak, že mnoho instancí objektu je posláno na GPU, takže mohou být vykresleny, aniž by bylo nutné čekat na CPU. Popíšeme si nyní několik technik instancování. Jsou jimi *Hardware Instancing*, *Shader Instancing* a *VFatch Instancing*.

- *Hardware Instancing*: Tato technika využívá dva buffer streamy. První je v podstatě obyčejný vertex buffer a slouží pro uložení vrcholů. Druhý stream představuje data jednotlivých instancí. V našem případě se bude jednat o transformační matice určující polohu a natočení jednotlivých instancí. Při vykreslování jsou poté vytvořeny grafické modely z vertex bufferu s různou transformací určenou druhým buffer streamem. Pro použití této techniky je zapotřebí podpora vertex shaderu 3.0.
- *Shader Instancing*: Shader Instancing uchovává ve vertex bufferu několik kopií původního objektu. Nastavení instancí probíhá přímo v shaderu. Na rozdíl od předchozí techniky, není vždy možné vykreslit všechny instance najednou. Počet instancí vykreslených v jednom kroku závisí na velikosti dat daného modelu. Pro tuto techniku je zapotřebí vertex shader 2.0.

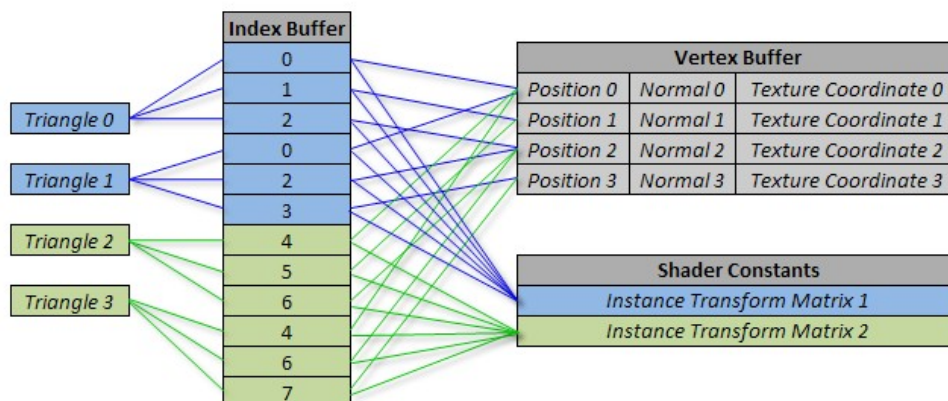
- *VFetch Instancing*: Tato technika je určena výhradně pro platformu Xbox 360. Je velmi podobná technice Shader Instancing, s tím rozdílem, že jsou vytvářeny pouze kopie dat v index bufferu. Kopie vrcholů jsou získány technikou VFetch.



Obrázek 7.1.5: Hardware Instancing (zdroj obrázku [5])



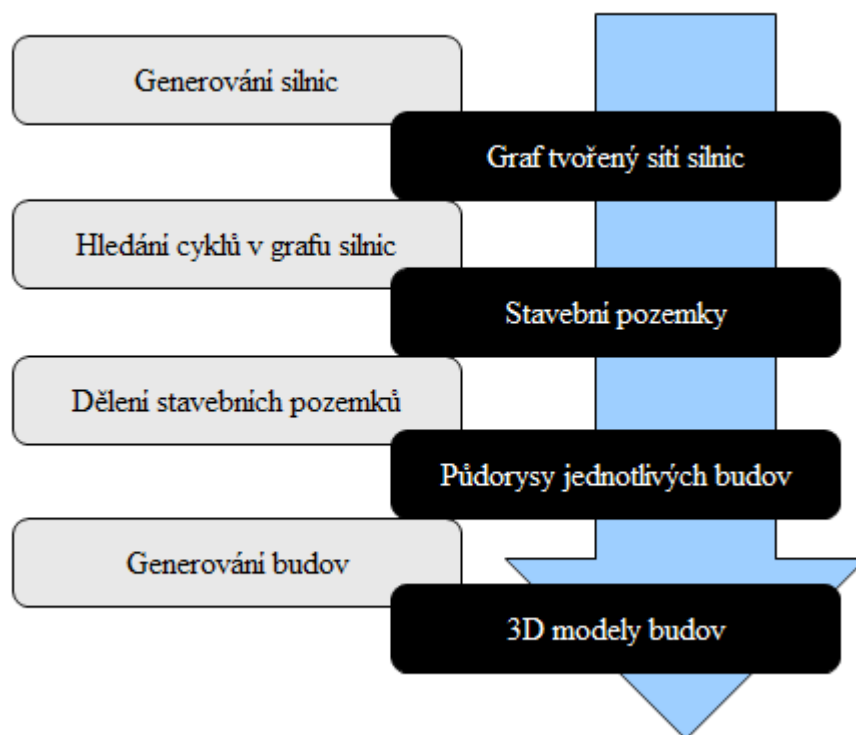
Obrázek 7.1.6: Shader Instancing (zdroj obrázku [5])



Obrázek 7.1.7: VFetch Instancing (zdroj obrázku [5])

7.2 Struktura programu

V této kapitole si popíšeme nejdůležitější třídy související s generováním města. Na obrázku 7.2 vidíme jednotlivé kroky generování města. Třídy jsou zde tedy členěny do skupin odpovídajících daným krokům.



Obrázek 7.2: Proces generování města

- *Generator*: Jedná se o hlavní třídu, která zřetězuje jednotlivé operace tak, jak je znázorněno na obrázku 7.2. Kromě toho se stará o ošetření vstupů a vykreslování scény.

Generování silnic

- *RoadLsystem*: Tato třída implementuje L-systém pro generování silnic. Vytváří objekt typu *GlobalGoals*, který je použit na jednotlivé části silnic pro nastavení počátečních hodnot. Dále vytváří objekt typu *LocalConstraints* k ošetření omezujících podmínek pro jednotlivé části silnic. Třída *RoadLsystem* také komunikuje s objektem třídy *Canvas*, do kterého jsou vykreslovány silnice. Výsledná data silnic jsou zároveň ukládána do objektu třídy *RoadData* pro další zpracování. Více o L-systému pro generování silnic najdeme v kapitole 3.1.

- *GlobalGoals*: Tato třída aplikuje na jednotlivé moduly L-systému tzv. globální cíle. Pomocí této třídy jsou tedy nastavovány počáteční atributy modulů podle vzorů silnic a podle hustoty zalidnění (více viz kapitola 3.2).
- *LocalConstraints*: Tato třída aplikuje na moduly L-systému lokální omezení. Jedná se o ošetření přístupu do zakázaných oblastí a napojování silnic. V této třídě je také prováděno vyhlazování silnic. Po použití této třídy je daný úsek silnice dokončen a k žádným dalším úpravám již nedochází. Proto zde navíc probíhá ukládání informací o silnici. Více o lokálních omezeních najdeme v kapitole 3.3.
- *Canvas*: Pomocí této třídy jsou silnice zakreslovány do textury terénu. Nové hodnoty se ukládají do pole barev. To je následně převáděno na texturu, která je použita na povrch terénu.
- *RoadData*: V této třídě se ukládají data grafu silnic. Konkrétně se zde ukládají uzly grafu a hrany grafu. Speciálním případem hrany grafu je most, který se při hledání cyklů v grafu neuvažuje.
- *RModule*: Tato třída odpovídá modulu R z L-systému definovaném v kapitole 3.1. Obsahuje tedy odpovídající parametry *del* a *ruleAttr*. Parametry *ruleAttr* jsou navíc definovány ve zvláštní třídě *RoadPattern*.
- *IModule*: Tato třída odpovídá modulu I z L-systému definovaném v kapitole 3.1. Jejím obsahem jsou tedy parametry *state* a *roadAttr*. Parametry *roadAttr* jsou opět definovány ve zvláštní třídě, tentokrát *RoadAttr*.
- *BModule*: Tato třída odpovídá modulu B z L-systému definovaném v kapitole 3.1. Obsahuje tedy odpovídající parametry *del*, *ruleAttr* a *roadAttr*. Parametry *ruleAttr* a *roadAttr* jsou opět definovány ve třídách *RoadPattern* a *RoadAttr*.
- *RoadAttr*: Tato třída obsahuje parametry silnice, jako jsou pozice, úhel natočení, délka apod.
- *RoadPattern*: Tato třída obsahuje parametry, kterými se silnice při vytváření řídí. Jedná se o takzvané vzory silnic z kapitoly 3.2. Jednotlivými parametry jsou například počáteční délka, počáteční úhel apod. *RoadPattern* je ovšem pouze abstraktní třídou. Konkrétní implementace vzorů silnic můžeme nalézt ve třídách *BasicPattern* a *RasterPattern*. Tyto třídy odpovídají základnímu vzoru a rastrovému vzoru z kapitoly 3.2.

Hledání cyklů v grafu silnic

- *Graph*: Tato třída pracuje nad grafem silnic. V grafu hledá nejenom cykly, ale i cesty a izolované uzly. Výsledná data jsou ukládána do objektů třídy Primitive. Prohledávání grafu probíhá nad uzly, které jsou umístěny ve třídě Heap. Odtud jsou vybírány uzly s nejmenší hodnotou x . Hledání cyklů grafu je více popsáno v kapitole 4.
- *Heap*: Tato třída implementuje stromovou strukturu heap (hromada). Jsou v ní umístěny jednotlivé uzly grafu. V kořeni stromu se vždy nachází uzel s nejmenší hodnotou pozice na ose x . Vytváření základních primitiv grafu probíhá právě nad touto třídou (více viz kapitola 4.6).

Dělení stavebních pozemků

- *Subdivision*: Tato třída je použita k dělení stavebních pozemků na menší části, které poté slouží jako podstava pro jednotlivé budovy. Metody této třídy jsou použity také k dělení budov při jejich generování. Teorie k dělení stavebních pozemků je popsána v kapitole 5.1.

Generování budov

- *Building*: Třída Building se stará o generování jedné budovy pomocí L-systému. Vstupem pro generování budovy je její půdorys, který je dán svými vrcholy. Kromě generování se třída stará také o vykreslování budovy. Jednotlivé symboly v generovaném řetězci odpovídají částem budovy. Parametry těchto symbolů jsou ukládány do objektů třídy Shape. Více k L-systému pro generování budov můžeme najít v kapitole 5.2.
- *Shape*: Tato třída ukládá parametry části budovy. Objekty této třídy jsou součástí řetězce L-systému pro generování budov.
- *Triangulation*: V této třídě jsou polygony převáděny na trojúhelníky. Přesněji řečeno je vytvářeno pole indexů pro jednotlivé trojúhelníky. Toto pole je později použito jako index buffer. Více informací k triangulaci najdeme v kapitole 5.3.

8 Závěr

V této práci bylo představeno řešení, které by nám umožňovalo procedurálně generovat město. Byly nastíněny jednotlivé kroky návrhu od generování silnic až po vytváření budov. Generování budov i silnic bylo navrženo s pomocí L-systému.

Podle daného návrhu se podařilo vytvořit program generující město na základě vstupních map, jako jsou výšková mapa, mapa hustoty zalidnění apod. Použití daného programu je zamýšleno jako součást většího celku (počítačové hry nebo simulátoru), ve kterém by se město generovalo a následně použilo. Výsledný model města tedy není možné uložit. Vytvořená aplikace ovládaná přes roletové menu slouží pouze k prezentaci schopností daného programu.

Pro tuto práci se nabízí celá řada rozšíření, která by mohla být do aplikace přidána. Jistě by se mohlo zapracovat na optimalizaci, a to především v rámci zobrazení budov. Budovy by mohli být vytvářeny v několika úrovních detailu, případně by mohla být použita metoda instancování. Pro zlepšení vzhledu města by bylo vhodné použít více vzorů silnic, které by dávali silnicím větší rozmanitost. Gramatika L-systému pro generování měst by také mohla být obohacena o další pravidla, aby bylo dosaženo realističtějšího vzhledu města. Samotné budovy navíc nemají žádnou texturu. Měli bychom se tedy zaměřit na vytváření povrchu budov, a to nejlépe generovaného pomocí L-systému. Dalším vhodným rozšířením by bylo generování jednotlivých vstupních map. V současném stavu je potřeba, aby uživatel vkládal jednotlivé vstupní mapy sám, a to minimálně výškovou mapu a mapu hustoty zalidnění. Pokud by byla aplikace použita např. jako součást počítačové hry, byla by nutnost vkládání vstupních map velmi omezující.

Literatura

- [1] PARISH, Yoav; MÜLLER, Pascal. Procedural Modeling of Cities [online]. 2001, [cit. 2010-12-21]. Dostupný z WWW: <<http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperCities>>.
- [2] MÜLLER, Pascal; WONKA, Peter; HAEGLER, Simon; ULMER, Andreas; VAN GOOL, Luc. Procedural Modeling of Buildings [online]. 2006, [cit. 2010-12-21]. Dostupný z WWW: <<http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperBuildings>>.
- [3] KELLY, George; MCCABE, Hugh. Citygen: An Interactive System for Procedural City Generation [online]. 2007, [cit. 2010-12-21]. Dostupný z WWW: <<http://citygen.net/>>.
- [4] EBERLY, David. The Minimal Cycle Basis for a Planar Graph [online]. 2005-11-02, [cit. 2010-12-21]. Dostupný z WWW: <<http://www.geometrictools.com/>>.
- [5] APP HUB: Develop for Windows Phone & Xbox 360 [online]. 2007-10-27 [cit. 2010-12-21]. Instanced Model. Dostupné z WWW: <http://create.msdn.com/en-US/education/catalog/sample/mesh_instancing>.
- [6] MDX info [online]. 2006-01-05 [cit. 2010-12-21]. True hardware and shader based instancing. Dostupné z WWW: <<http://www.mdxinfo.com/tutorials/instancing.php>>.
- [7] L-system. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2010-12-21]. Dostupné z WWW: <<http://en.wikipedia.org/wiki/L-system>>.
- [8] MÜLLER, Pascal. Design und Implementation einer Preprocessing Pipeline zur Visualisierung prozedural erzeugter Stadtmodelle [online]. 2001, [cit. 2010-12-21]. Dostupný z WWW: <<http://www.vision.ee.ethz.ch/~pmueller/wiki/CityEngine/PaperCities>>.
- [9] L-systém. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-05-01]. Dostupné z WWW: <<http://cs.wikipedia.org/wiki/L-systém>>.
- [10] TIŠNOVSKÝ, Pavel. Root.cz [online]. 2006-10-24 [cit. 2011-05-01]. L-systémy: přírodní objekty i umělé artefakty. Dostupné z WWW: <<http://www.root.cz/clanky/l-systemy-prirodni-objekty-i-umele-artefakty/>>.
- [11] TIŠNOVSKÝ, Pavel. Root.cz [online]. 2006-11-14 [cit. 2011-05-01]. Závorkové a stochastické L-systémy. Dostupné z WWW: <<http://www.root.cz/clanky/zavorkove-a-stochasticke-l-systemy/>>.
- [12] Third Party Ninjas [online]. 2008-10-07 [cit. 2011-05-02]. Line Segment Intersection. Dostupné z WWW: <<http://thirdpartyninjas.com/blog/2008/10/07/line-segment-intersection/>>.
- [13] BOURKE, Paul. Paul Bourke - Personal Pages [online]. 1992 [cit. 2011-05-02]. Intersection of a Line and a Sphere. Dostupné z WWW: <<http://paulbourke.net/geometry/sphereline/>>.
- [14] BOURKE, Paul. Paul Bourke - Personal Pages [online]. 1987 [cit. 2011-05-02]. Determining if a point lies on the interior of a polygon. Dostupné z WWW: <<http://paulbourke.net/geometry/insidepoly/>>.

- [15] KONČEL, Jan. Analytická geometrie [online]. 2009 [cit. 2011-05-02]. Vektorový součin. Dostupné z WWW: <http://www.karlin.mff.cuni.cz/katedry/kdm/diplomky/jan_koncel/vektory.php?kapitola=vektorovySoucin>.
- [16] Microsoft XNA. In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, [cit. 2011-05-06]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Microsoft_XNA>.
- [17] The Code Project [online]. 2011-04-02 [cit. 2011-05-06]. 3D Graphics for Windows Phone 7 Using the XNA Framework. Dostupné z WWW: <<http://www.codeproject.com/KB/windows-phone-7/3DGraphicsWP7.aspx>>.
- [18] KLUCHER, Michael . XNA Game Studio Team Blog [online]. 2006-08-29 [cit. 2011-05-06]. The XNA Framework Content Pipeline. Dostupné z WWW: <<http://blogs.msdn.com/b/xna/archive/2006/08/29/730168.aspx>>.

Příloha A

Obsah DVD

- /bin spustitelná aplikace pro Windows XP, Windows Vista a Windows 7
- /bin/maps vstupní mapy pro generování města
- /doc text práce ve formátu PDF, jeho zdrojový tvar a plakát
- /src projekt Visual Studio 2008

Příloha B

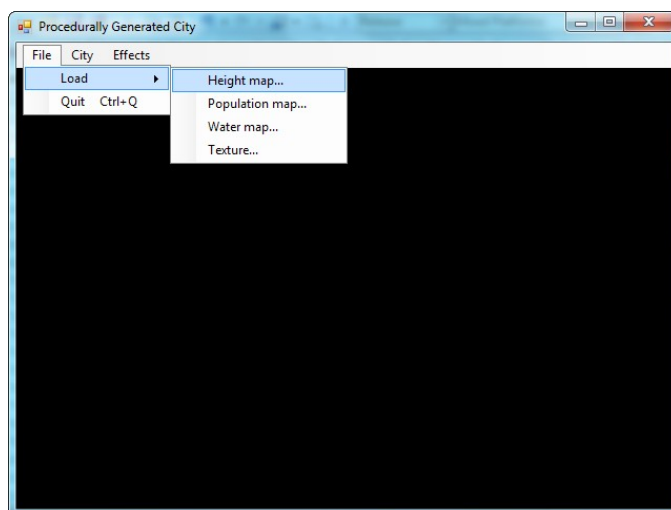
Uživatelská příručka

Příložená aplikace je určena pro operační systém Windows, minimálně ve verzi XP. Program je možné spustit na počítačích v CVT. Pro jeho funkčnost v jiném prostředí je zapotřebí nainstalovat XNA Game Studio.

B.1 Ovládání aplikace

Načítání vstupních map

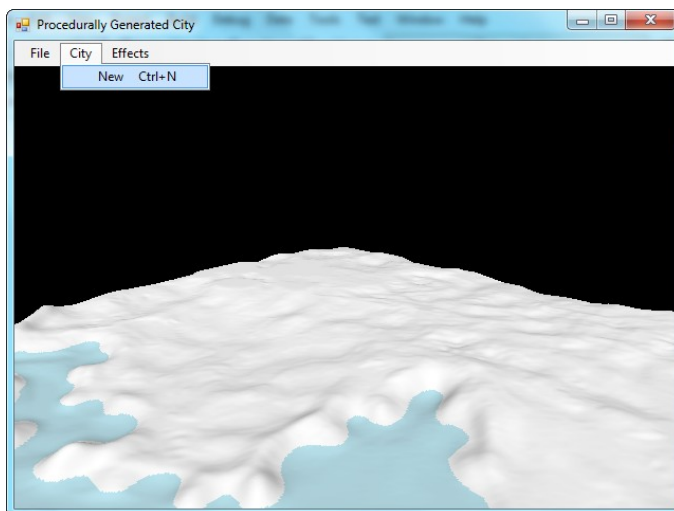
Pro generování města je nejdříve zapotřebí načíst sadu vstupních obrazů. Načítání obrazů probíhá přes roletové menu (viz obrázek B.1.1). Povinná je pouze výšková mapa a mapa hustoty zalidnění. V případě těchto map by se mělo jednat o obrazy s různou úrovní šedi, kde bílá znamená maximální hodnotu a černá znamená minimální hodnotu. Dále je možné načíst obraz popisující vodní plochy. Část obrazu, která není vodní plochou, by měla být bílá. Část obrazu, která je vodní plochou, může mít libovolnou jinou barvu (detekce vodní plochy je založena na změně barvy). Nakonec je možné načíst také obraz textury terénu. Pokud není textura načtena, použije se bílá barva. V případě prvních třech map (výšková mapa, mapa populace, mapa vodních ploch) je zapotřebí, aby jejich velikosti byly stejné. Pro všechny obrazy navíc platí, že jejich rozměry (šířka i výška) by měli být násobkem dvou.



Obrázek B.1.1: Načítání vstupních map

Generování města

Samotné generování města probíhá opět přes roletové menu (viz obrázek B.1.2) nebo pomocí klávesové zkratky *Ctrl+N*. Ke generování pochopitelně dojde pouze v případě, že je načtena výšková mapa a mapa hustoty zalidnění.



Obrázek B.1.2: Generování města

Procházení scény

Pokud chceme danou scénou procházet, musíme kliknout levým tlačítkem myši na plochu, kde je scéna vykreslována. Jakmile tak učiníme, zmizí kurzor myši, a my se můžeme pomocí myši rozhlížet a procházet scénou pomocí kláves (viz tabulka B.1). Pokud chceme průchod scénou ukončit, stiskneme tlačítko *Esc*.

Tabulka B.1: Ovládání průchodu scénou

Klávesa	Akce
W / Šipka nahoru	Pohyb dopředu
S / Šipka dolů	Pohyb dozadu
A / Šipka doleva	Pohyb doleva
D / Šipka doprava	Pohyb doprava
Q	Pohyb nahoru
Z	Pohyb dolů