České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Petr Frantál**

Studijní program: Softwarové technologie a management
Obor: Web a multimedia

Název tématu: **Procedurální modelování měst vyvíjejících se v čase**

Pokyny pro vypracování:

Seznamte se s metodami pro procedurální modelování měst. Na základě analýzy navrhněte jak do gramatik pro modelování měst zahrnout faktor času. Dále analyzujte a navrhněte jak může uživatel ovlivňovat vývoj města v čase. Váš návrh demonstrujte na prototypu implementovaném v programu CityEngine. Pomocí prototypu vytvořte alespoň tři rozdílná města.

Seznam odborné literatury:

G. Kelly, and H. McCabe. A survey of procedural techniques for city generation. ITB Journal, 14:87-130, 2006

Y.I. Parish, and P. Müller. Procedural modeling of cities. Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 301-308, ACM, 2001.

P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool. Procedural modeling of buildings. ACM Transactions On Graphics, 25(3):614-623, 2006.

M. Honda, K. Mizuno, Y. Fukui, and S. Nishihara. Generating autonomous time-varying virtual cities. International Conference on Cyberworlds, pp. 45-52, IEEE, 2004.

Vedoucí: Ing. Ladislav Čmolík, Ph.D.

Platnost zadání: do konce zimního semestru 2017/2018

L.S.

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

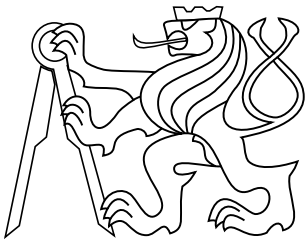
prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 29. 2. 2016

bakalářská práce

Procedurální modelování měst vyvíjejících se v čase

Petr Frantál



Květen 2016

Vedoucí práce: Ing. Ladislav Čmolík, Ph.D.

České vysoké učení technické v Praze
Fakulta elektrotechnická, Katedra počítačové grafiky a interakce

Poděkování

Chtěl bych poděkovat Ing. Ladislavu Čmolíkovi, Ph.D. za rady a trpělivost se mnou při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 27.5.2016 .....

Abstrakt

Tato bakalářská práce představuje možnosti tvorby 3D modelů měst, které simulují jejich vývoj v průběhu času. Pro modelování měst v této práci uvažujeme využití procedurálního modelování. Práce nejdříve uvádí motivaci pro tvorbu časově proměnných modelů měst. Poté obsahuje úvod do řešené problematiky a procedurálního modelování. Dále je uvedena analýza vývoje města v čase a změn, které přitom nastávají. Další částí práce je návrh, kde jsou představeny principy popisující tvorbu města vyvíjejícího se v čase a také parametry, kterými lze ovládat vývoj města v průběhu času. Další součástí práce je implementace těchto principů v programu CityEngine, který slouží k tvorbě modelů měst a využívá procedurálního modelování. Tato implementace je popsána v následující stejnojmenné kapitole. Implementace je otestována vytvořením tří rozdílných vzorových měst s uvedením postupu a využitím parametrů pro ovládání vývoje. Tyto výsledky jsou následně zhodnoceny.

Klíčová slova

3D modely měst; Procedurální modelování; Časově proměnný 3D model; Gramatiky

Abstrakt

This bachelor thesis introduces the possibilities of creation of 3D models of cities that evolve in time. We intend to use procedural modeling for this task. First we present a motivation for the creation of the time varying 3D models of cities. Then we introduce the issues addressed in this thesis and procedural modeling techniques. After that we present an analysis of the city evolution in time and we notice the visible changes that occur during that process. The next part is a design of the time varying 3D models and the parameters to control the city evolution. After that the implementation of the models is introduced, using procedural city modeling software CityEngine. We test the implementation by creating three different cities that evolve in time and present the details of the modeling process and the control parameter usage. The results are evaluated in the end.

Keywords

3D models of cities; Procedural modeling; Time varying 3D model; Grammars

Obsah

1 Úvod

Modelujeme-li v počítačové grafice města, narážíme na problém složitosti tohoto procesu. Vzhled města obsahuje velké množství detailů, které musíme zachytit. Jednou z částí města je struktura ulic, která může být velmi rozsáhlá. Také každá z budov je unikátní svým tvarem, výškou, či použitými stavebními prvky. Kdybychom chtěli ručně modelovat všechny tyto detaily, bude nám to trvat velmi dlouho.

K modelování měst můžeme využít technik procedurálního modelování. Při využití tohoto přístupu místo ručního modelování podobu modelu popíšeme a vytvoříme proceduru, která ho za nás na základě tohoto popisu vymodeluje. Procedurám můžeme definovat vstupní parametry, kterými lze řídit jejich činnost. Tak můžeme ovlivňovat části vytvářeného modelu (uvažujeme-li budovy, může to být například jejich výška). V procedurách také užíváme pseudonáhodných čísel, které zakomponováváme do procesu tvorby a které také mohou ovlivnit výslednou podobu modelu.

Máme-li k dispozici například proceduru pro tvorbu budov, jejíž činnost lze řídit předáním vstupních parametrů jako je výška budovy či architektonický sloh, můžeme jednodušeji vytvářet různé typy budov. Díky parametrům a pseudonáhodným číslům tyto budovy budou odlišné. Vytvoříme-li procedury pro generování jednotlivých částí města (ulic, parcel, či dalších), přičemž podobu jimi vytvářených modelů budeme schopni efektivně ovlivňovat vstupními parametry, tvorbu modelu města si tím oproti ručnímu modelování urychlíme.

Dalším způsobem, jak mít proces tvorby města pod kontrolou, je simulovat jeho vývoj v čase. Místo nutnosti vytvořit celý model najednou můžeme jít postupně a ovládat procedury pro vytváření jednotlivých částí města v určitých časových bodech. Proces tvorby města v čase bude simulovat reálný vývoj města a definujeme-li ho správně (co nejvíce věrně realitě), můžeme tak dosáhnout také výsledků více odpovídajících realitě.

Kdybychom vytvořili procedurální model města, jehož podoba bude závislá na vstupním parametru času, získáme tak navíc možnost prohlížet si v jednotlivých krocích celý vývoj modelovaného města. Vytvořený model města bychom mohli prohlásit za kompletní (model je schopen zachytit celý vývoj města se všemi jeho podobami). Mohli bychom ho využít ke sledování vývoje vybraného existujícího města, simulování jeho vývoje v budoucích letech, ale i k vytváření vlastních měst. Jejich vývoj bychom chtěli definovat podle našich přání. Potřebujeme proto být schopni ovládat ho. Toho dosáhneme návrhem vhodných vstupních parametrů procedur generujících části modelu.

Naší myšlenkou je tedy usnadnit lidem vytváření 3D modelů měst, dát jim možnost modelovat vývoj měst, který budou moci efektivně a jednoduše ovládat. Tak budou moci vymodelovat města podle svých představ, rychleji než využitím ručního modelování a navíc bude tvorba simulovat reálný vývoj města a přinese tak výsledky více odpovídající realitě.

1.1 Cíle práce

V práci se zaměříme jednak na analýzu procedurálních technik pro modelování měst a představíme způsoby, jak města generovat. Uvedeme si také přehled vývoje měst a

budeme pozorovat jednotlivé změny, které při něm nastávají. Prozkoumáme možnosti modelování měst simulováním tohoto vývoje a představíme způsoby, jak tuto simulaci ovládat. Definujeme proto vstupní parametry, po kterých budeme požadovat smysl, efektivitu a jednoduchost. Z těchto poznatků vyjdeme a navrhneme, jak tuto simulaci implementovat. Implementujeme ji v programu CityEngine pro procedurální modelování měst a otestujeme vytvořením tří rozdílných vzorových měst.

1.2 Struktura práce

Text je rozdělen do pěti kapitol. Po úvodu následuje kapitola analýzy a návrhu, kde rozebereme techniky pro procedurální modelování měst, průběh vývoje měst a možnosti jeho ovládání. Na základě těchto znalostí si navrhneme, jak vytvořit simulaci vývoje města a jak ji ovládat. V třetí kapitole si popíšeme implementaci simulace v programu CityEngine. Ve čtvrté kapitole demonstrujeme výsledky vytvořením tří odlišných měst, jejichž vývoj v čase budeme řídit námi navrhnutými parametry. Výsledky zhodnotíme. Práci shrneme v páté kapitole, kterou je závěr.

2 Analýza a návrh

Tato kapitola přináší analýzu problematiky práce a návrh algoritmů pro simulaci vývoje města v čase. Nejprve si uvedeme techniky procedurálního modelování využitelné pro modelování měst. Poté si uvedeme analýzu vlastností programu CityEngine a podíváme se, jak v něm můžeme vytvářet modely měst a jak lze tvorbu částí těchto modelů ovládat. Následuje analýza vývoje měst v čase s popisem sledovatelných změn a návrh technik pro ovládání tohoto vývoje.

2.1 Procedurální modelování

Počítačová grafika má řadu odvětví, mezi než patří například virtuální realita, počítačové hry, či simulace (například simulace chodu nějakého zařízení). V těchto odvětvích vytváříme více či méně úplné virtuální světy, fiktivní či založené na tom našem. Ať jde již o jakoukoliv z těchto či podobných aplikací, často v nich pracujeme s modely. Model představuje objekt reálného či fiktivního světa. Může jít o jakoukoliv hmotnou věc. Námi vytvářené (modelované) světy se pak skládají z počítačových modelů.

Model je kontejner informací o podobě předmětu, který chceme zobrazit. Jde o jeden celek, ucelenou jednotku. Přesná podoba informací je závislá na reprezentaci modelovaného předmětu. V rámci této práce budeme uvažovat jen hraniční reprezentaci, kdy modelujeme pouze povrch předmětu. Model předmětu je pak kolekcí vrcholů, hran a ploch, které ho tvoří. Dále může obsahovat definice barvy či simulace materiálu, ze kterého je předmět vyroben a další informace.

Proces tvorby modelu nazýváme modelováním. Modelovat předměty můžeme například ručně v 3D modelovacích programech. To je jeden z častých způsobů vytváření modelů například pro počítačové hry či animované filmy. 3D modelovací programy umožňují modelování předmětů v grafickém uživatelském rozhraní. Uživatel zde ovlivňuje podobu modelu do té doby, než je s ní spokojen. To dělá tak, že definuje parametry jednotlivých vrcholů, ze kterých se model skládá. Definovat je může například přímo (třeba přesunout vrchol) nebo může použít modifikátor, čili proceduru, která ovlivní model podle svojí definice. Příkladem modifikátoru je aplikace algoritmu pro dělené plochy (z angl. Subdivision Surfaces) [1], který zjemní povrch modelu vytvořením dodatečných vrcholů.

3D modelování poskytuje tvůrci absolutní kontrolu nad podobou vytvářeného modelu. To ale přináší také řadu nevýhod. Čím detailnější model vytváříme, tím více času nám jeho tvorba zabere. Například modely pro animované filmy bývají velmi detailní, a proto na nich pracují celé týmy tvůrců. Další nevýhodou je, že aby byl tvůrce schopen vytvořit hezký model, musí být umělecky nadán. Také musí hlouběji rozumět modelovacímu programu, který často bývá velmi složitý a trvá dlouho se s ním naučit pracovat.

Těmto nevýhodám se můžeme vyhnout použitím odlišného přístupu k vytváření modelů, kterým je procedurální modelování [3]. Při něm tvůrce nedefinuje přesnou podobu všech vrcholů jako při 3D modelování. Ta je místo toho vytvořena procedurami, které definují, jaký objekt nebo jaká jeho část budou vytvořeny. Tvůrce může procedurám předat vstupní prametry, které mohou ovlivnit vytvoření konkrétní podoby.

Takto může ovlivnit proceduru, aby generovala model s jím požadovanými vlastnostmi. Procedury navíc pracují s pseudonáhodnými čísly. Ta jsou také využita k ovlivnění výsledné podoby modelu. Díky nim a parametrům tak může vzniknout velké množství podob výsledného modelu. Díky pseudonáhodnosti zadáním stejného semínka pro generátor pseudonáhodných čísel získáme stejný model (musí být stejné i všechny vstupní parametry). Pokud se nám některý vytvořený model líbil, stačí tedy příště proceduře pro jeho vytvoření předat stejné vstupní parametry a toto semínko a přesně tento stejný model bude vygenerován.

Použití procedurálního modelování nám přináší řadu výhod. Tvorba některých předmětů pomocí procedur může být jednodušší než tvorba 3D modelováním. Příkladem takového předmětu je rostlina. Zde je využito faktu, že jednotlivé části rostliny se generují podobně (větve a stonek lze vytvořit kreslením čar). Další výhodou je snížení paměťové náročnosti při práci s modely. Místo toho, abychom například v hraniční reprezentaci ukládali velké množství modelů s velkým množstvím vrcholů, stačí uložit pouze proceduru generující model, vstupní parametry a semínko pro generátor pseudonáhodných čísel, na základě kterých byly tyto modely vytvořeny. To je ve většině případů nesporně výhodnější než ukládat celé modely. Další výhodou je, že model může být vykreslen až tehdy, je-li k tomu v uvažované grafické aplikaci dán pokyn. Do té doby je reprezentován pouze výše uvedeným způsobem a jsou tak šetřeny prostředky počítače do té doby, než je skutečně nutné model vykreslit a těchto prostředků využít.

S definovanou procedurou, jejíž činnost je ovladatelná vstupními parametry, navíc může 3D model vytvářet i méně zkušený nebo talentovaný tvůrce. Odpadá nutnost mít umělecké nadání pro tvorbu modelů v počítačové grafice. Vytvoření kvalitního hezkého modelu ale vyžaduje od uživatele zadání správných vstupních parametrů. Tedy takových, se kterými bude mít generování modelu požadované hezké výsledky (například tvoření konkrétního typu rostliny, která bude odpovídat představám tvůrce). K tomu je v první řadě potřeba, aby vstupní parametry správně ovlivňovaly podobu výsledného modelu. Tedy aby měly smysl (výška rostliny), byly silné - změna parametru se skutečně projeví (chtěný počet větví rostliny vede skutečně na model s takovým počtem větví), a také aby byly pro uživatele procedury jednoduché na pochopení (uživatel ví, že parametr definuje počet větví rostliny) a nebylo jich mnoho. Při dobrém návrhu vstupních parametrů je pak pro tvůrce velmi jednoduché vytvářet odlišné modely téhož předmětu.

V následujících sekcích si uvedeme definici gramatik, jelikož jsou na nich založené procedurální techniky, které můžeme využít pro modelování měst. Poté si tyto techniky popíšeme.

2.1.1 Gramatiky

Nyní si uvedeme definici gramatik podle Melichara [4]. Jak si postupně ukážeme, procedurální techniky, které budeme využívat pro modelování měst, jsou na gramatikách založeny.

Gramatika je čtveřice

$$G = (N, T, P, S) \tag{1}$$

Přitom platí:

- N je konečná množina neterminálních symbolů. Tyto symboly mohou být nahrazeny.
- T je konečná množina terminálních symbolů. Tyto symboly jsou konstantní.

- P je množina pravidel. Pravidlo, tj. element $(\alpha, \beta) \in P$, zapsaný $\alpha \rightarrow \beta$, definuje způsob nahrazení neterminálního symbolu řetězcem neterminálních či terminálních symbolů.
- $S \in N$ je počáteční (startovací) symbol gramatiky.

Gramatika tedy obsahuje množinu nahrazovacích pravidel, které definují způsoby nahrazení neterminálních symbolů jinými neterminálními nebo i terminálními symboly. Gramatika je dále definována i počátečním symbolem, ze kterého vycházíme při nahrazování podle pravidel. Nahrazením toho symbolu získáme jiné (jinou posloupnost), ty můžeme opět nahradit a tento proces můžeme dále opakovat, buď ve stanoveném počtu kroků nebo dokud nejsou všechny symboly vznikajícího řetězce terminální.

Nyní si ukážeme příklad gramatiky a nahrazování symbolů podle pravidel. Mějme gramatiku $G = (N, T, P, S)$, kde $N = \{a, b\}$, $T = \{\}$, $P = \{a \rightarrow ab, b \rightarrow ba\}$, $S = \{a\}$. Pak pro počet nahrazení symbolů podle pravidel (počet iterací) n je výsledný řetězec následující:

$$\begin{aligned} n &= 0 : a \\ n &= 1 : ab \\ n &= 2 : abba \\ n &= 3 : abbabaab \end{aligned}$$

Na tomto příkladě vidíme postup nahrazování symbolů. Tyto symboly mohou mít různý význam a jejich nahrazováním určujeme výžnam výsledného řetězce těchto symbolů. V následujících sekcích si ukážeme, jak jsou na gramatikách založeny techniky procedurálního modelování, které bychom mohli využít pro modelování měst. Těmito technikami jsou l-systémy a gramatiky tvarů (z angl. Shape Grammars).

2.1.2 L-Systémy

Lindenmayerův systém (l-systém) je jednou z aplikací gramatik v procedurálním modelování objektů. L-systém definuje množiny neterminálních a terminálních symbolů, pravidla a počáteční symbol. Stejně jako v sekci o gramatikách vyjdeme z počátečního symbolu, který podle příslušného pravidla nahradíme jinými a ty zase dalšími symboly. Po námi stanovém počtu kroků vznikne řetězec symbolů, který definuje výsledou podobu objektu. Výsledná podoba objektu je vykreslena perem želví grafiky.

Symboly v l-systémech reprezentují operace tohoto pera. Těmi jsou například nakreslení čáry o stanovené délce nebo natočení pera o nějaký úhel. Dále také operace uložení a načtení pozice pera. Díky tomu se můžeme vracet na předešlá místa, ze kterých jsme již jednou či víckrát kreslili směrem definovaným symboly, a tudíž můžeme z jednoho místa vykreslit více čar a větvit tak vykreslení modelu.

L-systémy se používají například pro vykreslování rostlin či stromů. Využívá se přitom podobného způsobu vykreslení stonku či kmenu a větví, obojí se kreslí jako čáry. Čím vícekrát nahradíme symboly podle pravidel příslušného l-systému (iterujeme), tím více detailů a částí modelu přidáváme. Jejich využití ale na rostliny není nutně omezeno, což si ukážeme v následující sekci.

Vykreslení rostliny definované l-systémem želví grafikou si ukážeme na konkrétním příkladě. Mějme L systém $L = (N, T, P, S)$, kde:

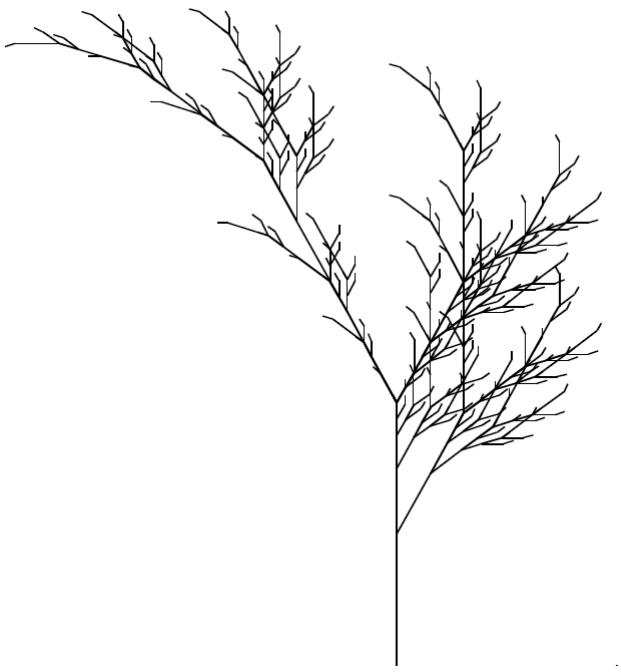
$$
\begin{aligned}
N &= \{X, F\}, \\
T &= \{+, -, [,]\}, \\
P &= \{X \to F - [[X] + X] + F[+FX] - X, F \to FF\}, \\
S &= \{X\}.
\end{aligned}
$$

Přitom symboly mají tento grafický význam:

- X je počátečním symbolem a nemá grafický význam.
- F znamená posun pera želví grafiky dopředu o jednu jednotku.
- [je uložení pozice pera na zásobník.
-] je načtení pozice pera, která se nachází na vrchu zásobníku.
- $+$ znamená otočení pera želví grafiky doleva o úhel δ.
- $-$ znamená otočení pera želví grafiky doprava o úhel δ.
- $\delta = 22.5°$

Počáteční symbol X nahradíme dle definovaných pravidel, a stejně tak nahrazujeme nově vzniklé symboly. Po zvoleném počtu nahrazení vykreslíme dle grafického významu symbolů popisovanou rostlinu želví grafikou.

Čáry, tedy stonek a větve kreslíme posunutím pera vpřed. Díky operacím natáčení můžeme z prvotní rovné čáry vykreslit větve rostliny. Předtím si ale zapamatováváme pozice, na kterých jsme se nacházeli. Když jsme s vykreslením větve hotoví, můžeme se tak vrátit na původní místo a pokračovat v kreslení stonku a po čase se třeba opět natočit k vykreslení další větve. Výsledek vykreslení s pěti iteracemi nahrazení symbolů můžeme vidět na následujícím obrázku.



Obrázek 1 Rostlina definovaná l-systémem [3].

2.1.3 Využití l-systémů v modelování měst

V předchozí sekci jsme si uvedli použití l-systémů na přírodních objektech jako jsou rostliny a stromy. Podívejme se nyní, zda-li bychom jimi mohli definovat i podobu objektů ve městě, tedy ulic a budov.

Uvedli jsme si, že symboly l-systémů reprezentují operace pera želví grafiky, které vykresluje podobu modelů podle výsledných řetězců symbolů. Kdybychom kreslili perem želví grafiky v trojrozměrném prostoru a definovali takové významy symbolů, že jejich posloupnost nám umožní vykreslit ulice a budovy, mohli bychom l-systémy využít v procedurálním modelování měst. Uvažovaný l-systém bude opět definován množinami neterminálních a terminálních symbolů, pravidly a počátečním prvkem. Nahrazováním symbolů získáme výsledný řetězec, který bude určovat způsob vykreslení objektů v trojrozměrném prostoru.

Podívejme se nejprve na modelování ulic a parcel definovaných l-systémy, které si ukážeme na konkrétním příkladě. Mějme l-systém $L = (N, T, P, S)$, kde:

$$\begin{aligned} N &= \{U, P\}, \\ T &= \{\}, \\ P &= \{U \to UUP\}, \\ S &= \{U\}. \end{aligned}$$

Přitom symboly mají tento grafický význam:

- U - Pero želví grafiky vykreslí ulici.
- P - Pero želví grafiky vykreslí parcelu.

Vyjdeme z počátečního symbolu U, který definuje, jak vykreslit ulici. Podle pravidla $U \to UUP$ je počáteční symbol nahrazen řetězcem symbolů UUP. Podíváme-li se, jak bude pero želví grafiky vykreslovat model podle tohoto řetězce, vidíme, že:

1. Pero vykreslí ulici podle definice prvního symbolu U.
2. Pero vykreslí druhou navazující ulici podle druhého symbolu U.
3. Pero vykreslí přilehlou parcelu podle definice symbolu P.

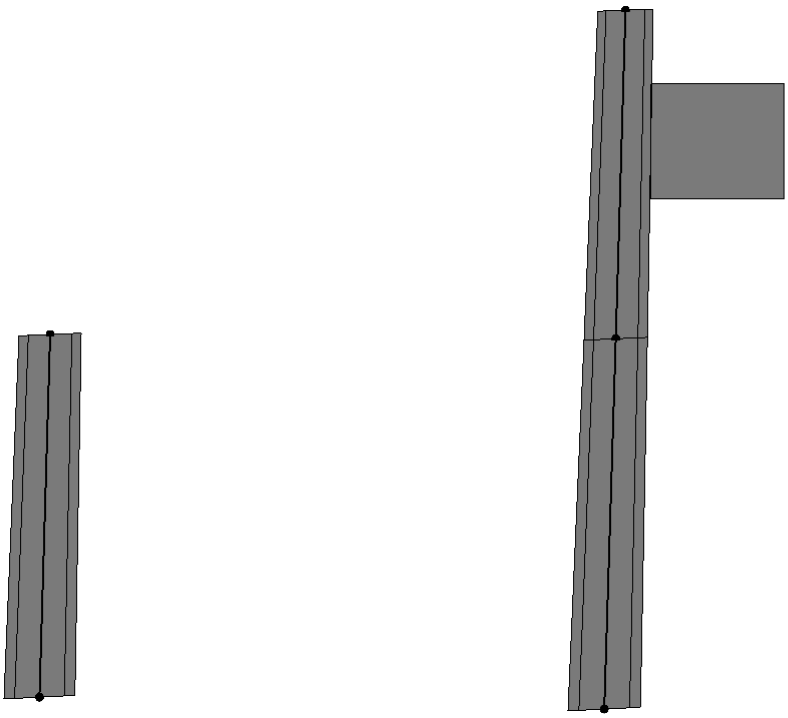
Podíváme-li se na ilustrační obrázek 2, vidíme, jak by model vypadal, kdybychom ho vykreslili před nahrazením počátečního symbolu výše uvedeným pravidlem a po něm. Po nahrazení je nejprve vykreslena spodní ulice, poté vrchní a vedle té je posléze vykreslena parcela.

Na tomto příkladu vidíme, že správnými definicemi symbolů můžeme využít l-systémy k modelování ulic a parcel. Symboly musí být definovány tak, aby pero želví grafiky vykreslilo posloupnost čar (hran), která tvoří tyto objekty. Také musíme definovat nahrazovací pravidla, pak bude pero místo jedné ulice vykreslovat dvě a podobně.

Obdobným způsobem bychom mohli použít l-systémy i pro modelování podoby jiných částí městské zástavby, tentokrát podoby budov. Tuto metodu zmiňují Parish a Müller ve své práci Procedural Modeling of Cities [5].

Symboly l-systémů budou nyní reprezentovat způsob vykreslení částí budov. K nim definujeme vhodná pravidla pro jejich nahrazení. Pokud nahradíme symboly, tak pero želví grafiky vykreslí místo původního objektu jiné podle definice příslušnými symboly. Takto může pero místo parcely kreslit plášť budovy. Místo pláště může kreslit jednotlivá patra a místo nich stěny s okny a s fásadou.

Takto můžeme s využitím l-systémů modelovat také budovy. Proces nahrazování symbolů a vykreslení výsledných řetězců perem želví grafiky v několika iteracích ilustruje obrázek 3.



a) Před nahrazením.

b) Po něm.

Obrázek 2 Aplikace pravidla a vykreslení objektů před ní a po ní.

Obrázek 3 Tvorba budovy definované l-systémem [5].

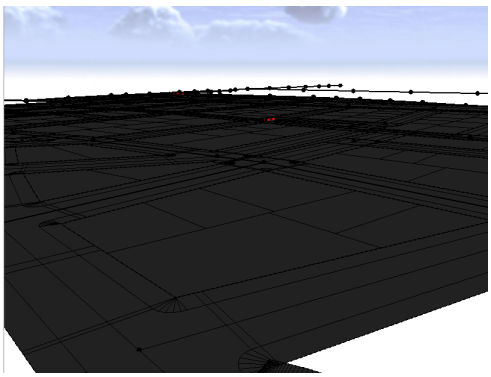
Zde vidíme postup, kdy místo původních částí budovy pero kreslí nahrazující části. V první iteraci vykreslí plášť budovy. V další iteraci vykreslí místo pláště přízemí budovy a část, která reprezentuje všechna ostatní patra. V další iteraci místo této části vykreslí část představující širší spodní patra a část představující užší vrchní patra. Nahrazováním částí pero vykresluje stále detailnější části, s čímž roste i detail budovy. Tímto způsobem také můžeme řídit úroveň detailu budovy.

Vidíme tedy, že definujeme-li význam symbolů jako způsob vykreslení trojrozměrných objektů nacházejících se ve města, můžeme využít l-systémy k modelování jednotlivých částí města jako jsou ulice, parcely a budovy.

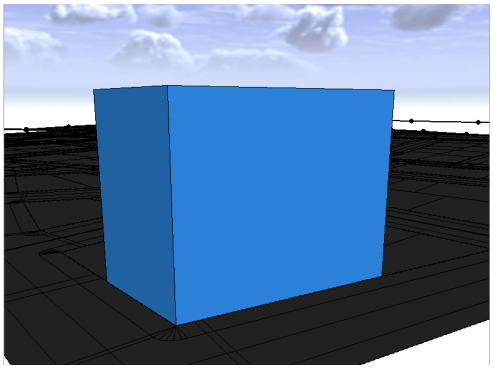
2.1.4 Gramatiky tvarů (Shape Grammars)

Nyní víme, že s využitím l-systému můžeme definovat podobu objektů jako jsou například ulice či budovy. Výsledná podoba modelu je vykreslena perem želví grafiky. Podobný způsob definice poskytuje i technika, kterou si nyní popíšeme, a kterou jsou gramatiky tvarů.

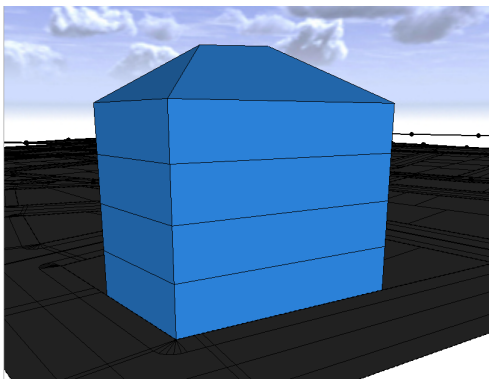
Stejně jako l-systém, i tyto gramatiky vycházejí z principů, které jsme si uvedli v



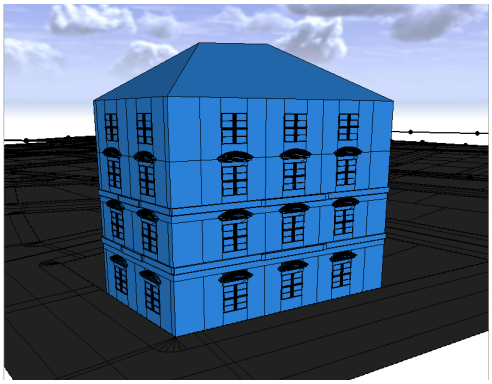
a) Počátečním symbolem je parcela.



b) Parcela nahrazena pláštěm budovy.



c) Plášť je nahrazen patry a střechou.



d) Po několika dalších iteracích dostáváme výslednou podobu budovy.

Obrázek 4 Modelování budovy pomocí gramatik tvarů.

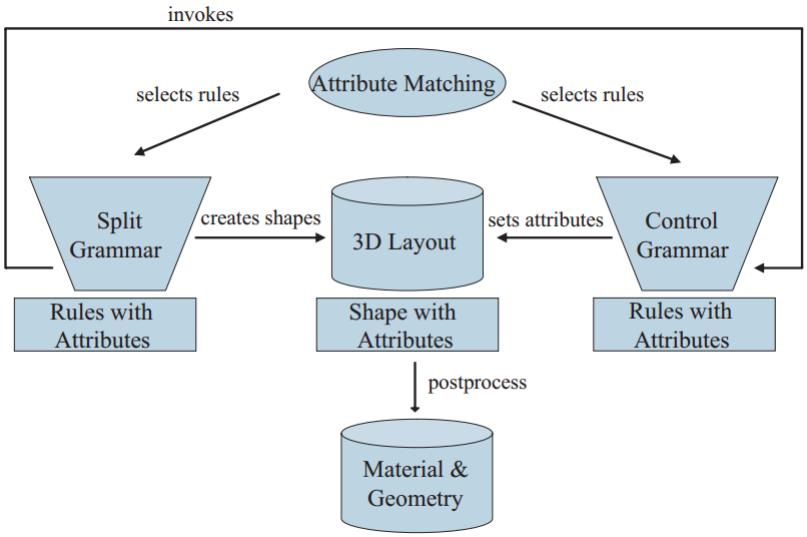
sekci 2.1.1. Taková gramatika je také definovaná množinami neterminálních symbolů, terminálních symbolů, nahrazovacích pravidel a počátečním symbolem. Rozdílem oproti l-systémům je, že zatímco u něj symboly definovaly operace pera želví grafiky, symboly gramatik tvarů přímo reprezentují geometrické objekty. Nahrazování symbolů potom znamená, že jednotlivé objekty přímo jsou nahrazovány jinými. Nahrazující objekty jsou umístěny v místě původního objektu a také jejich velikost mu odpovídá.

Práce s gramatikami tvarů je proto velmi podobná práci s l-systémy. I zde definujeme všechny množiny gramatiky a poté stanovíme počet iterací nahrazování. Výsledný řetězec symbolů představuje objekt složený z jednotlivých částí, kterými jsou jiné objekty. Při vykreslení jsou vykresleny jednotlivé objekty tvořící celkový model.

Nyní si uvedeme příklad gramatiky tvarů na modelování budovy. Předpokládejme, že počátečním symbolem je parcela budovy. V první iteraci nahradíme parcelu pláštěm budovy. V druhé iteraci nahradíme plášť jednotlivými patry a střechou. V další iteraci nahradíme patra zdmi fasády a okny. Takto postupujeme stále dál, až získáme výslednou podobu budovy. Tu vygenerujeme tak, že vygenerujeme jednotlivé objekty, ze kterých se budova skládá. Postup tvoření této budovy vidíme na obrázku 4. Můžeme zde vidět vygenerování budovy v prvních dvou iteracích a poté finální podobu.

2.1.5 Gramatiky dělení (Split Grammars)

Podobný způsob modelování objektů jako jsou gramatiky tvarů popsané v předchozí sekci, definovali Peter Wonka a kol. [6]. Ve svém článku představili gramatiku dělení, Split Grammar. Tento způsob modelování budov se podobá výše uvedenému způsobu modelování budov s využitím l-systémů. I zde definujeme budovy s využitím gramatiky a i zde dochází k nahrazování částí budov jinými. Komponenty systému využívajícího gramatiky dělení jsou vidět na obrázku 5.



Obrázek 5 Komponenty systému využívajícího gramatiky dělení [6].

Gramatika dělení je jednou z komponent systému. Jejím úkolem je vytvářet geometrická primitiva ve výsledném modelu budovy. Množiny terminálních a neterminálních symbolů gramatiky dělení tato geometrická primitiva obsahují a gramatika dále obsahuje pravidla, která umožňují nahrazovat části budov jinými. Jedná se o atributovou gramatiku. Geometrická primitiva této gramatiky jsou ovlivňována atributy, kterými lze měnit jejich podobu. Také pravidla, která dělí části budovy, mají své atributy. Díky nim lze pro část budovy s určitými atributy vybrat vhodné dělící pravidlo na základě nich a atributů tohoto pravidla.

Další komponentou systému je kontrolní gramatika (Control Grammar). Jak gramatika dělení dělí části budovy na další, atributy, které dělená část budovy obsahovala, se propagují do těch nově vznikajících. Předtím ale mohou být korigovány dle nahrazovacích pravidel právě z kontrolní gramatiky. Tímto způsobem můžeme do výsledného modelu budovy vkládat svoje další představy, například to, že první patro budovy má mít určitou výšku.

Poslední komponentou systému, kterou si popíšeme, je komponenta porovnání atributů (Attribute Matching). Ta se stará o výběr pravidel pro dělení částí budovy, když k tomuto dělení dochází. Vybrána jsou ta pravidla, která odpovídají atributům, které komponentě porovnání atributů předáme. Také z kontrolní gramatiky jsou vybrána pravidla pro korekci atributů propagovaných do nově vznikajících částí budovy a také ta jsou vybírána podle námi předaných atributů.

Podívejme se nyní na jeden krok procesu dělení části budovy na nové části:

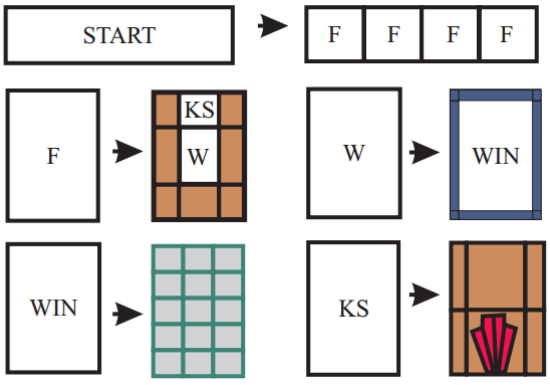
1. Gramatika dělení vyhledá pro dělenou část budovy pravidla, kterými tato část může být rozdělena.

2. Komponenta porovnání atributů porovná atributy přiřazené dělené části budovy s atributy přiřazenými všem vybraným pravidlům z předchozího bodu a vybere to nejvíce odpovídající.
3. Dojde k rozdělení původní části budovy a k vytvoření nových částí podle vybraného pravidla. Do nových částí jsou následně propagovány atributy z původní části.
4. Dojde ke korekci atributů nových částí budovy podle pravidla kontrolní gramatiky. Toto pravidlo je vybráno podobně jako pravidlo pro dělení budovy, tedy na základě toho, jestli může být aplikováno na vzniklou část budovy a na základě porovnání jeho atributů a námi předaných.
5. Tím končí krok rozdělení části budovy. Tento algoritmus je nyní rekurzivně zavolán na nově vzniklé části budov.

Dle popisu výše vidíme, že budovy můžeme modelovat i tímto postupem. Jako tvůrci navíc můžeme ovlivňovat tvorbu budovy těmito způsoby:

- Můžeme ovlivnit podobu geometrických primitiv nastavením ovlivňujících atributů.
- Nastavením atributů také můžeme ovlivnit výběr pravidel pro nahrazení určité části budovy.
- Pomocí kontrolní gramatiky a jejích pravidel můžeme dále ovlivňovat podobu budovy. Tímto způsobem jsou upraveny atributy definující podobu nově vznikajících částí.
- Můžeme definovat nové prvky gramatiky jak gramatiky dělení, tak kontrolní gramatiky.

Konkrétní příklad modelování budovy systémem s gramatikou dělení můžeme vidět na obrázku 6. Zde vidíme počáteční primitivum, kterým je stěna budovy či jejího patra. Tato stěna je nahrazena čtyřmi částmi (označeny F). Každá část je dále dělena na část okolo okna a prostředek (W) dále nahrazen oknem (WIN). To je dále nahrazeno tabulkou skla a část nad oknem (KS) je nahrazena částí se zdobným prvkem fasády.



Obrázek 6 Modelování částí budov s využitím gramatiky dělení [6].

Vidíme, že využití tohoto systému poskytuje dostatečně účinný postup pro modelování budov. Podobný princip využívá také program CityEngine, kde můžeme modelovat budovy využitím gramatiky CGA Shape Grammar.

2.2 CityEngine

V této kapitole se budeme zabývat programem CityEngine, který slouží k procedurálnímu modelování měst. V tomto programu jsou v rámci této práce implementovány časově proměnné modely měst. Nejprve si představíme CityEngine. Poté navážeme na předchozí kapitolu o procedurálním modelování a uvedeme si, které procedurální techniky CityEngine využívá. V další části si ukážeme základní techniky, jak v programu vytvořit procedurálně generované město.

2.2.1 Úvod do CityEngine

CityEngine je program pro procedurální modelování měst. Umožňuje rychlé vytváření ulic pomocí připravených nástrojů a modelování ulic sestavením vlastních nahrazovacích pravidel pro řetězce symbolů reprezentující objekty s využitím operací, které poskytuje CGA gramatika (Computer Generated Architecture). Program poskytuje grafické uživatelské rozhraní pro jednodušší ovládání modelovacích nástrojů a také aplikační rozhraní pro jejich ovládání v jazyce Python.

CityEngine využívá modelování ve vrstvách scény, kdy ulice a parcely vznikají ve vrstvách ulic a objekty ve svých vlastních. Objekty v CityEngine mají atributy, které ovlivňují jejich vzhled. U ulic jde například o jejich šířku, u budov může jít o výšku patra či šířku okna. Atributy mohou být ovládany jak s voláním metod API, tak v inspektoru scény.

Výhodou CityEngine je jeho jednoduchost a rychlost při tvoření ulic a budov. Jeho nevýhodou je skrytá implementace jeho elementárních principů a nemožnost přistupovat přímo k datovým strukturám objektů a k ukazatelům objektů na ostatní objekty. Díky tomu například rychle nezjistíme, jaká ulice je následníkem nějaké naší zvolené nebo jestli spolu dva domy sousedí. Všechny sousednosti musíme určovat z porovávání trojrozměrných vrcholů.

Parish a Müller v článku představujícím CityEngine [4] zmiňují využití l-systémů pro generování ulic i budov. CityEngine verze 2011.2, který v této práci využívám já, ale využívá ke generování budov již zmíněnou CGA gramatiku. Budovy tedy můžeme vytvářet nahrazováním symbolů reprezentujících objekty (části budovy) podle pravidel. Tato pravidla přitom sestavujeme sami. CGA gramatika definuje různé operace (vytváření geometrie, texturování, transformace a jiné), které v nich můžeme využít. Do pravidel můžeme vkládat výše zmíněné atributy, které pak ovlivňují funkcionalitu operací CGA gramatiky. Tato gramatika také umožňuje dělení objektů podle délek. Při nahrazování objektů jinými si tak můžeme vybrat, jaká část s určenou délkou bude výsledným objektem nahrazena.

2.2.2 Techniky pro vytvoření procedurálně generovaného města

V této části nalezneme analýzu funkcionality CityEngine a podíváme se na to, jak vytvářet jednotlivé části procedurálně generovaných měst, kterými jsou ulice a domy. Tyto znalosti poté využijeme implementaci modelů měst vyvjíjejících se v čase.

Tvorba ulic

CityEngine poskytuje dva přístupy tvorby ulic. Jsou jimi manuální vytvoření ulice nástrojem Create Street Tool a vytváření většího množství ulic nástrojem Grow Streets. Tyto nástroje poskytují rozdílné možnosti a oba jsou vyvolatelné pythonovým skriptem, který může být v programu spuštěn.

Create Street Tool slouží k vytváření samostatných ulic - tvoříme je vždy po jedné. Po vybrání nástroje v grafickém uživatelském rozhraní můžeme kliknutím do scény volit přesná místa, kde bude ulice začínat a končit. Jakmile vytvoříme jednu ulici, nástroj automaticky pokračuje možností vytvářet ulice s ní spojené právě z místa, kde jsme skončili. Takto tedy můžeme tvořit více navazujících ulic. Není to ale nutné, můžeme tvorbu ulic z aktuálního koncového místa ukončit a pokračovat tvorbou ulic jinde. Pakliže se chceme vrátit, nástroj nás graficky navede tak, že víme, že budeme ulici vytvářet opět z jednoho z konců jiné ulice (zvýraznění koncového vrcholu ulice). Nemusíme ale začínat či končit jen na koncích ulice. V případě, že chceme začít jinde a vytvořit křižovatku, je to možné (nástroj nás opět graficky nevede zvýrazněním místa).

Výhodou tohoto nástroje je fakt, že můžeme přesně definovat pozici ulice. Tak můžeme tvořit ulice zcela podle svých představ. Nevýhodou je to, že ulice jsou tvořeny po jedné a tak je tento přístup pomalý na to, abychom tvořili rozsáhlé sítě ulic.

Tento nástroj také můžeme vyvolat přes pythonové rozhraní CityEngine. Odpovídající metoda se jmenuje CE.createGraphSegments() a předáním pole s pozicemi vrcholů můžeme přesně definovat, na jakých pozicích se budou generované ulice nacházet. Dále předáváme vrstvu ulic, ve které chceme ulice vytvořit (je také možné nechat vytvořit novou vrstvu). Předáme-li metodě pole vertexů, které bude obsahovat více než dva (takové pole tedy bude mít více než šest prvků, ale jejich počet bude stále dělitelný třemi), metoda vytvoří napojené ulice. Pokud bychom stejné ulice vytvořili postupným voláním této metody, ulice nebudou propojené, i když budou začínat či končit na stejných pozicích. Ulice lze ještě dodatečně propojit nástrojem Cleanup Graph, který si popíšeme níže v této části. Vytvoří-li propojené ulice kružnici (spojí-li se linka z více ulic okolo nějaké plochy), vznikne mezi nimi blok s parcelami.

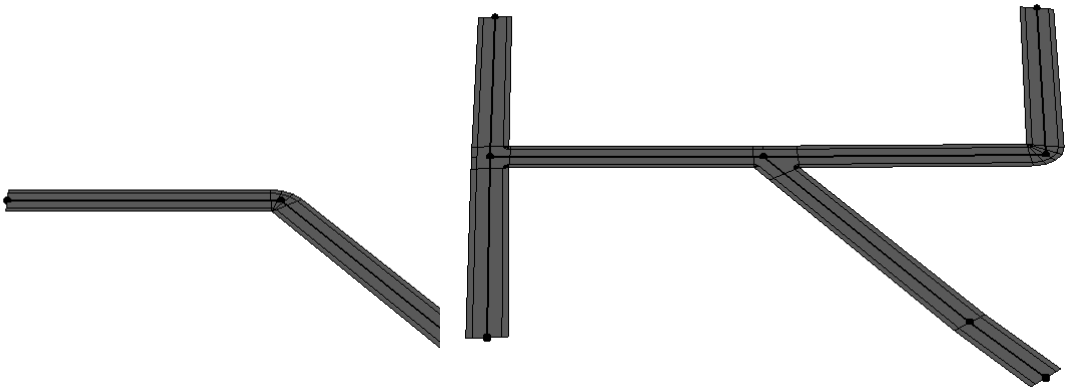
Zatímco ruční modelování ulic je pomalé na vytvoření rozsáhlých sítí ulic, tvorba voláním metody CE.createGraphSegments() je samozřejmě rychlejší. S tímto způsobem již můžeme pomýšlet na generování větších sítí ulic.

Příklad modelování ulic s využitím Create Street Tool vidíme na obrázku 7c. Zde můžeme pozorovat malou simulaci rozvoje sítě ulic. Na obrázku vpravo vidíme, že při vzniku kružnic napojených ulic vznikají bloky s parcelami.

Jak jsme již zmínili, tímto nástrojem můžeme vytvořit napojené či nenapojené ulice. Nenapojené bychom ale chtěli propojit, protože pouze v kružnici těch může vzniknout blok s parcelami. Také jsme zmínili, že nenapojené ulice můžeme dodatečně propojit aplikací nástroje Cleanup Graph. Tento nástroj si nyní popíšeme.

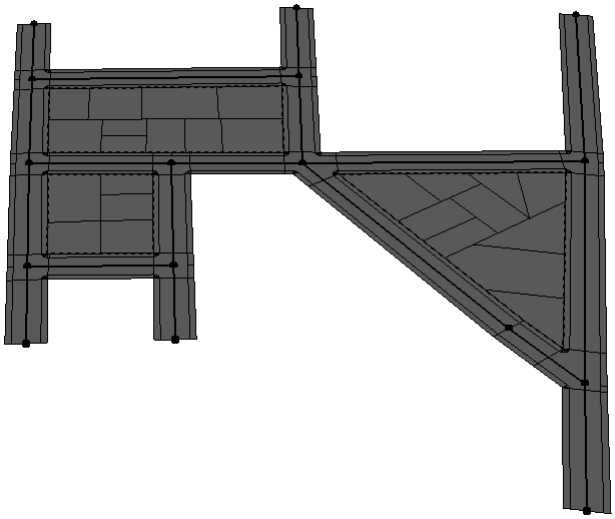
Cleanup Graph je vyvolatelný jak ručně z grafického uživatelského rozhraní, tak voláním metody CE.cleanupGraph(). Nástroj má více možností nastavení jeho funkcionality. Toto nastavení se definuje objektem CleanupGraphSettings a můžeme jím řídit například, zdali vůbec chceme ulice napojovat, v jakých vzdálenostech od sebe se musí nacházet vrcholy či vrchol a hrana ulic, které mohou být propojeny. Další možností je také nastavit úpravu předané sítě ulic, pokud došlo ke konfliktům. Ty nastávají, když dojde k vygenerování příliš krátkých ulic s křižovatkami, které se překrývají. Kromě aktuálního nastavení, které vyjadřuje objekt CleanupGraphSettings předáváme metodě ještě seznam ulic, na kterých chceme nástroj použít. Příklad použití Cleanup Graph nalezneme na obrázku 8. Zde vidíme spojení dříve oddělených ulic a vytvoření bloku mezi nimi.

Další možností, jak generovat ulice je nástroj Grow Streets. Tento nástroj poskytuje trochu jiné možnosti než Create Street Tool. Umožňuje vytvářet ulice ve větším počtu, který můžeme definovat. Po vytvoření jsou všechny ulice napojené a automaticky tedy vzniknou bloky. Ulice můžeme vytvářet jednak v nové vrstvě. To se děje ve chvíli, kdy nemáme ve výběru žádnou z již existujících ulic. Pokud máme vybranou alespoň jednu



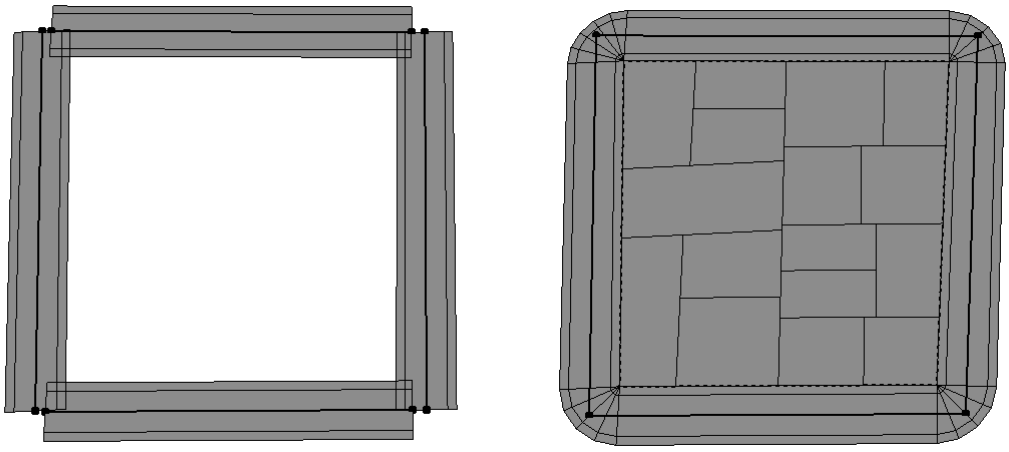
a) Fáze 1.

b) Fáze 2.



c) Fáze 3.

Obrázek 7 Vytváření ulic nástrojem Create Street Tool.



a) Ulice před propojením.

b) Ulice po propojení, vznikl nový blok.

Obrázek 8 Propojování ulic nástrojem Cleanup Graph.

ulici, její hranu či vrchol, ulice se vytvoří do vrstvy, ve které je i tento prvek.

Grow Streets má vlastní nastavení, kterým můžeme generování ulic ovlivnit. Tím můžeme jednak zadat počet ulic, které chceme vytvořit. Dále si můžeme zvolit schéma, ve kterém chceme ulice generovat, a toto schéma může být odlišné pro hlavní větší ulice i menší vedlejší ulice, které tyto hlavní spojují. Schéma může být buď křivolaké (Organic), pravoúhlé či rovnoběžné (Raster) a kruhové (Radial). Více o významu schémat si povíme v části 2.3.3. Dále můžeme definovat délky vznikajících ulic a odchylku od této délky. Můžeme také definovat šířky ulic a chodníků odlišně pro hlavní a vedlejší ulice a odchylky ke všem těmto hodnotám. Dále můžeme generování ulic řídit také předáním výškové mapy (Heightmap) nebo mapy překážek (Obstaclemap). Výšková mapa reprezentuje definovaný terén, kopce na něm či údolí a mapa překážek zase lesy či vodní toky a jezera. Grow Streets z těchto definic vyjde. Pak budou ulice stavěny ve výšce podle výškové mapy a ne v rovině. Při použití mapy překážek nebudou stavěny v místech definovaných jako vodní plochy.

Výhodou Grow Streets je, že urychluje generování ulic, kdy jsme ale stále schopni toto generování řídit pomocí výše uvedených způsobů. Další výhodou je, že vzniknou napojené ulice a automaticky tak i bloky s parcelami. Tento nástroj odstiňuje uživatele, který model vytváří, od definování přesných pozic ulic. To urychluje vytváření procedurálního modelování města. Grow Streets nám tak umožňuje rychlé generování ulic za cenu nemožnosti definovat jejich přesné pozice. To ale nepotřebujeme vždy (nevytváříme město s přesně definovanými ulicemi). Kdybychom pozice ulic přeci jen chtěli přesně definovat, můžeme použít metodu CE.createGraphSegments() (nástroj Create Street Tool), které tyto pozice předáme. Další drobnou nevýhodou nástroje Grow Streets může být například fakt, že vytváříme-li ulice z křižovatky, ze které již vedou čtyři ulice, pátá se nevytvoří.

Nástroj Grow Streets můžeme vyvolat jak z GUI, tak voláním metody CE.growStreets(). Nastavení pak definujeme přímo v GUI, případně předáním objektu GrowStreetsSettings. Na obrázku 9 můžeme pozorovat vytváření sítě ulic nástrojem Grow Streets. Na prvním obrázku vidíme vytvoření první sady ulic. Na druhém vidíme situaci po vytvoření další sady a můžeme si všimnout vzniku bloku.

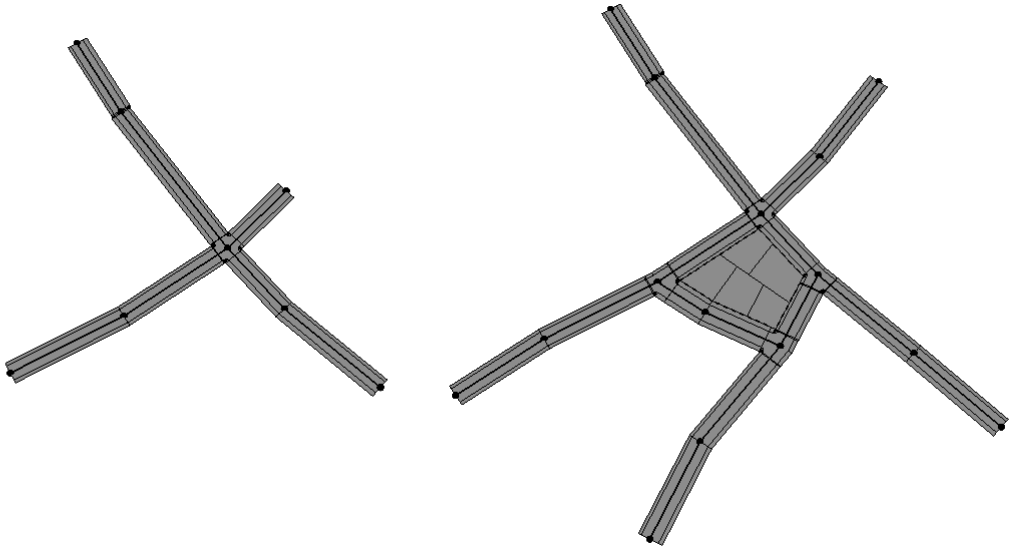
Vidíme tedy, že CityEngine poskytuje dva různé nástroje pro generování ulic. Create Street Tool umožňuje maximální kontrolu nad vytvářenými ulicemi včetně jejich přesných délek a pozic, Grow Streets zase odstiňuje tvůrce od definování těchto hodnot a umožňuje rychlejší vytváření ulic, které ale tvůrce stále může do jisté míry řídit podle své vůle, například nastavením přibližných délek ulic, počtu vznikajících ulic či vybráním generovacího schématu.

Tvorba parcel

Po nastínění možností, jak v CityEngine modelovat ulice, se přesuneme k parcelám. Ty potřebujeme modelovat proto, abychom na nich mohli později vygenerovat budovy.

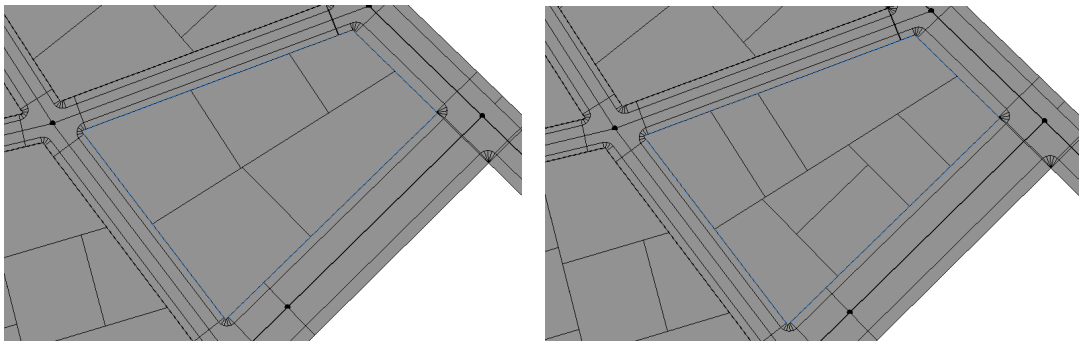
V předchozí části jsme si již uvedli, že parcely se vytvoří, když v modelu vznikne kružnice (uzavřená smyčka) propojených ulic. Poté je na ploše uvnitř této smyčky vytvořen blok. Ten je rozdělen na jednotlivé parcely. Způsob tohoto rozdělení určuje hodnota semínka a změníme-li tuto hodnotu, vzniknou rozdělením jiné parcely. Semínko můžeme změnit v panelu GUI po vybrání bloku. Tuto změnu ilustruje obrázek 10. Stejné semínko vede na stejné rozdělení parcel.

Tento přístup je výhodný, protože nám šetří hodně času. Parcely jsou vytvořeny automaticky programem s vytvořením sítě ulic. To nám umožňuje se o tvorbu parcel příliš nestarat a nechat CityEngine vytvářet parcely za nás. Používáme-li navíc nástroj



a) Fáze 1. **b)** Fáze 2.

Obrázek 9 Generování ulic nástrojem Grow Streets.



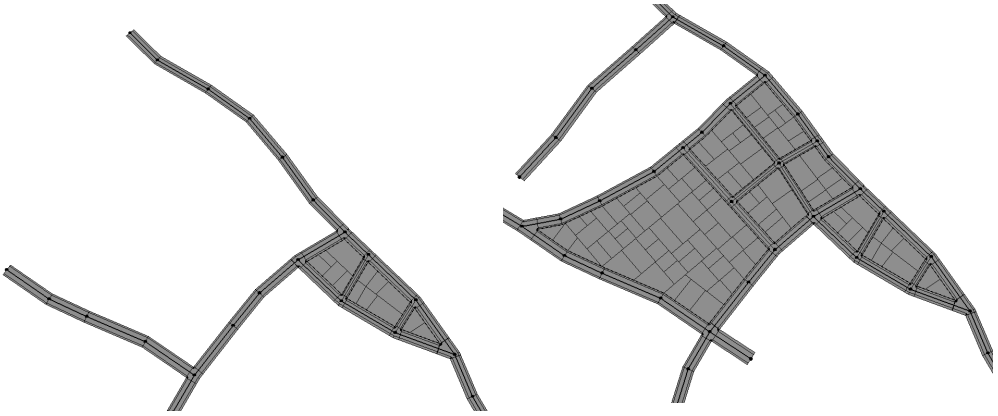
a) Rozdělení bloku před změnou semínka. **b)** Rozdělení bloku po změně semínka.

Obrázek 10 Vliv semínka na rozdělení bloku.

Grow Streets, síť ulic s parcelami okolo nich vzniká bez našeho velkého úsilí. Vznik ulic a bloků s využitím tohoto nástroje vidíme na obrázku 11.

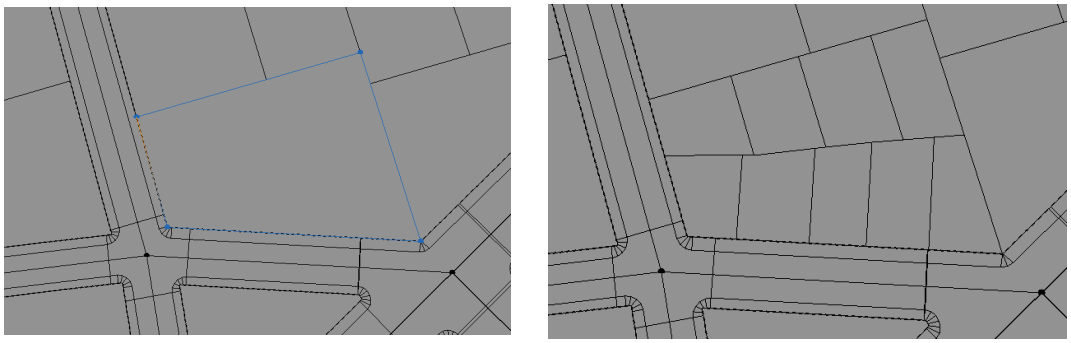
Kromě tohoto způsobu můžeme parcely vytvářet také nástrojem Create Shape Tool. Ten je svým využitím velice podobný nástroji Create Street Tool popsanému v předchozí sekci, pouze slouží pro vytváření parcel. Opět je ale vyvolatelný jak z GUI, tak metodou rozhraní programu, která se jmenuje CE.createShape(). S tímto nástrojem máme opět možnost definovat přesnou pozici a velikost vznikajících parcel.

V GUI s tímto nástrojem tvoříme parcely po jedné. Klikáním do scény definujeme pozici jednotlivých vrcholů vznikající parcely. Počet těchto vrcholů není omezený a můžeme tak dosáhnout velkého množství tvarů parcely. Chceme-li vytvářet další parcelu, musíme tvorbu aktuální parcely ukončit, například klávesou Escape. Chceme-li vyvolat nástroj v pythonovém skriptu, musíme metodě CE.createShape() předat pole souřadnic vrcholů, na kterých nová parcela vznikne. Předat můžeme také vrstvu, ve které chceme parcelu vytvořit, či nechat vytvořit vrstvu novou. Jedním zavoláním CE.createShape() vytvoříme jednu parcelu.



a) Fáze 1. **b)** Fáze 2.

Obrázek 11 Grow Streets - fáze generování ulic a bloků parcel.



a) Parcela před rozdělením. **b)** Parcela po rozdělení.

Obrázek 12 Dělení parcely na menší nástrojem Subdivide.

Výhodou této metody je možnost přesně definovat pozici, velikost a tvar parcely. Nevýhodou je pracnost a délka trvání tvorby.

Doplňme si, že vznikající parcely mohou být dvojího typu, dynamické a statické. Ty, které vznikají automaticky uzavíráním smyček propojených ulic, jsou dynamické. Ty, které vytváříme nástrojem Create Shape Tool, jsou statické. Dynamické parcely můžeme konvertovat na statické volbou Convert to Static Shapes v GUI a metodou CE.convertToStaticShapes() ve skriptu. Metodě musíme předat seznam dynamických parcel, které chceme konvertovat. Po konvertování budou na stejné pozici existovat dvě parcely - původní dynamická a nově vzniklá statická.

Nad statickými parcelami můžeme provádět operace rozdělení na další parcely (nástroj Subdivide), seskupení parcel (Combine Faces) a opětovného oddělení (Separate Faces). Po seskupení parcel se tyto parcely chovají jako jedna. Postavíme-li domy na dvou oddělených parcelách, vygenerujeme dva modely. Necháme-li ale postavit dům na dvou seskupených parcelách, bude vytvořen pouze jeden společný model stojící na obou parcelách. Operaci rozdělení můžeme vidět na obrázku 12. Výše uvedené operace nemůžeme provádět s dynamickými parcelami. Navíc pouze ze statických parcel jsme schopni přečíst jejich atributy, kterými je ovlivněna podoba domů na nich postavených. Tímto se dále zabýváme v kapitole 3.4.

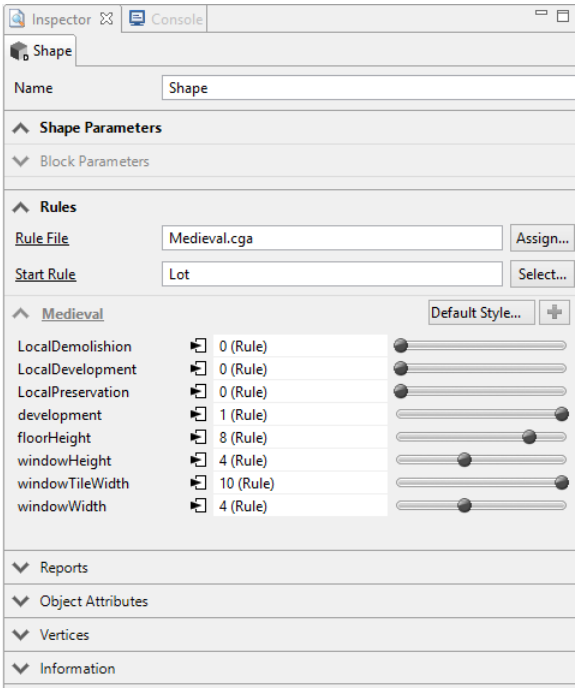
Tvorba budov

Nyní, když jsme si nastínili možnosti modelování ulic a parcel, podívejme se na to, jak modelovat budovy města.

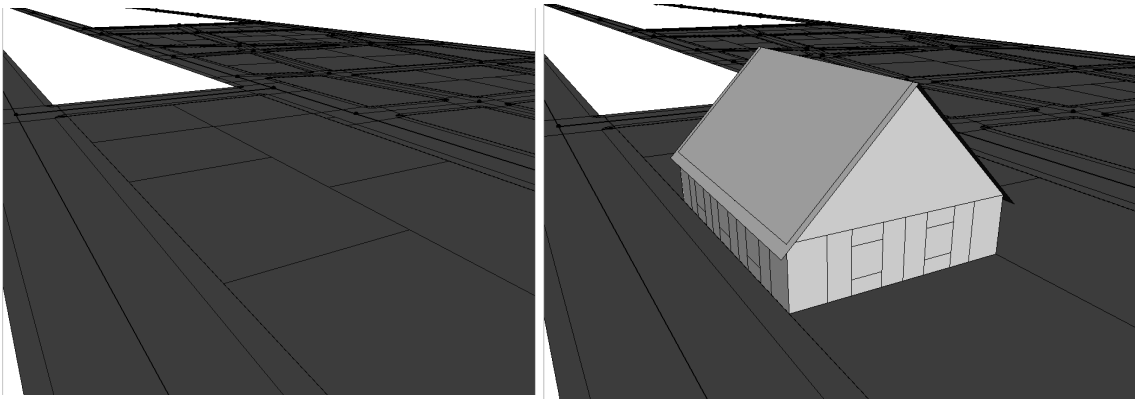
V CityEngine modelujeme podobu objektů definicí posloupností pravidel CGA gramatiky. Každá taková posloupnost spolu se vstupními parametry objektu přesně definuje jeho podobu. Tyto posloupnosti zapisujeme do souborů s příponou .cga, souborů pravidel (rulefiles).

Chceme-li vygenerovat budovu, musíme soubor pravidel přiřadit parcele, na které chceme, aby se budova nacházela. Budova je vygenerována, dáme-li k tomu pokyn (vybereme budovu a buď klikneme na volbu Generate v GUI programu nebo použijeme klávesovou zkratku CTRL + G). Podívejme se na ukázku souboru pravidel:

$$
\begin{aligned}
&attrdevelopment = 1.0\\
&attrfloorHeight = 8.0\\
&attrwindowTileWidth = 10.0\\
&attrwindowWidth = 4.0\\
&attrwindowHeight = 4.0\\
&\\
&@StartRule\\
&Lot --> \\
&casedevelopment == 1.0 :\\
&extrude(world.y, floorHeight)Building\\
&else :\\
&extrude(world.y, 2 * floorHeight)Building\\
&\\
&Building --> \\
&comp(f)\{side : Facade | top : Roof\}\\
&\\
&Roof --> \\
&roofGable(45, 1.5, 1.0)RoofTexture\\
&\\
&Facade --> \\
&split(y)\{floorHeight : Floor\} *\\
&\\
&Floor --> \\
&split(x)\{\tilde{}1 : Wall | \{windowTileWidth : WindowTile\} * | \tilde{}1 : Wall\}\\
&\\
&WindowTile --> \\
&split(x)\{\tilde{}1 : Wall | \{windowWidth : WindowPart\} | \tilde{}1 : Wall\}\\
&\\
&WindowPart --> \\
&split(y)\{\tilde{}1 : Wall | \{windowHeight : Window(split.index)\} | \tilde{}1 : Wall\}
\end{aligned}
\tag{2}
$$



Obrázek 13 Přiřazení souboru pravidel (pole RuleFile).



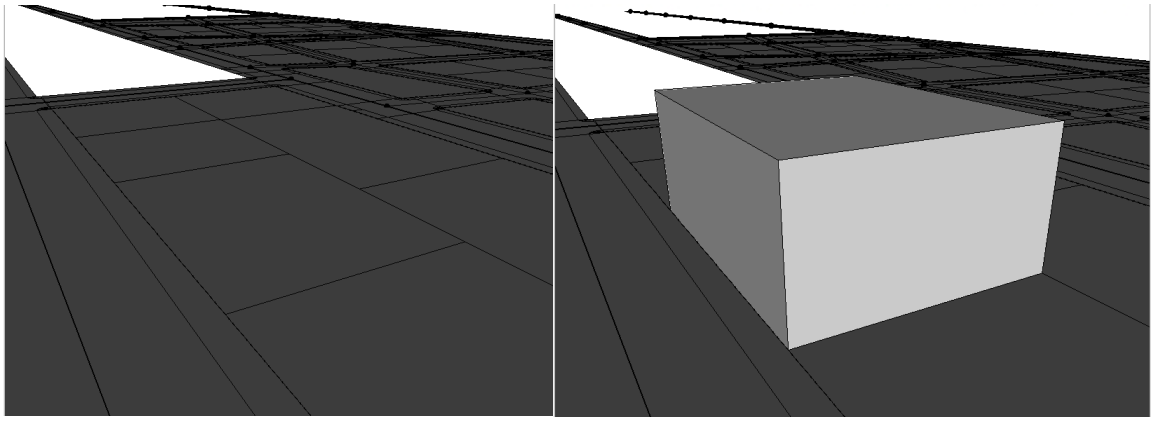
a) Před vygenerováním budovy. **b)** Po něm.

Obrázek 14 Vytvoření budovy podle souboru pravidel.

Přiřazení souboru pravidel můžeme udělat v inspektoru scény, viz. obrázek 13 (věnujme pozornost poli Rule File v záložce Rules) nebo ve skriptu. Může jít jak o dynamickou, tak statickou parcelu.

Každý soubor musí obsahovat počáteční symbol, od kterého začíná generování geometrie. Tento symbol je opatřen anotací *@StartRule*, ze kterou následuje definice nahrazovacího pravidla. V případě generování budov na parcelách vytvořených postupem uvedeným v předchozích odstavcích se může jednat o jeden z dvojice symbolů *Lot* (parcela se nachází na okraji bloku, tj. sousedí s ulicí) a *InnerLot* (jedná se o parcelu uvnitř bloku, je ze všech stran obklopena dalšími parcelami). Po pokynu k vygenerování je geometrie budovy vytvořena. Na obrázku 14 můžeme vidět vytvoření budovy, jejíž podobu definuje výše uvedený soubor pravidel 2.

Kromě budov můžeme ale tímto způsobem definovat také podobu dalších objektů ve scéně jako jsou třeba chodníky či ulice. Pokud objektu žádný soubor nepřiřadíme, Ci-



a) Před vytažením geometrie. **b)** Vytažením získáme plášť budovy.

Obrázek 15 Operace extude.

tyEngine jeho podobu vytvoří podle své implementace. Ta se pro vyjmenované objekty liší. Podoba parcel není bez přiřazení souboru pravidel nijak dále ovlivnitelná a vidíme skutečně pouze prázdnou parcelu. Podoba ulic je ale definována několika atributy jako jsou například šířka ulice a chodníků po obou jejích stranách. I když pro oba tyto objekty můžeme vytvořit soubor pravidel, implementace definující původní podobu těchto objektů, je nám skrytá.

V kapitole 2.2.1 jsme si uvedli, že do pravidel můžeme vkládat operace, které nám CGA gramatika poskytuje. Vložením těchto operací pak definujeme přesnou podobu nahrazovacího pravidla. Pojďme se nyní detailněji podívat na některé z těchto operací, abychom si udělali představu, jak můžeme budovy modelovat. Tyto operace se dělí do skupin pro vytváření geometrie, kterými jsou například dělení geometrie, manipulace s ní, texturování a další.

Ze skupiny vytváření geometrie si představíme operaci extrude (vytažení geometrie), i (insert, vložení geometrie) a operace pro vytváření střech domů, kterých je několik.

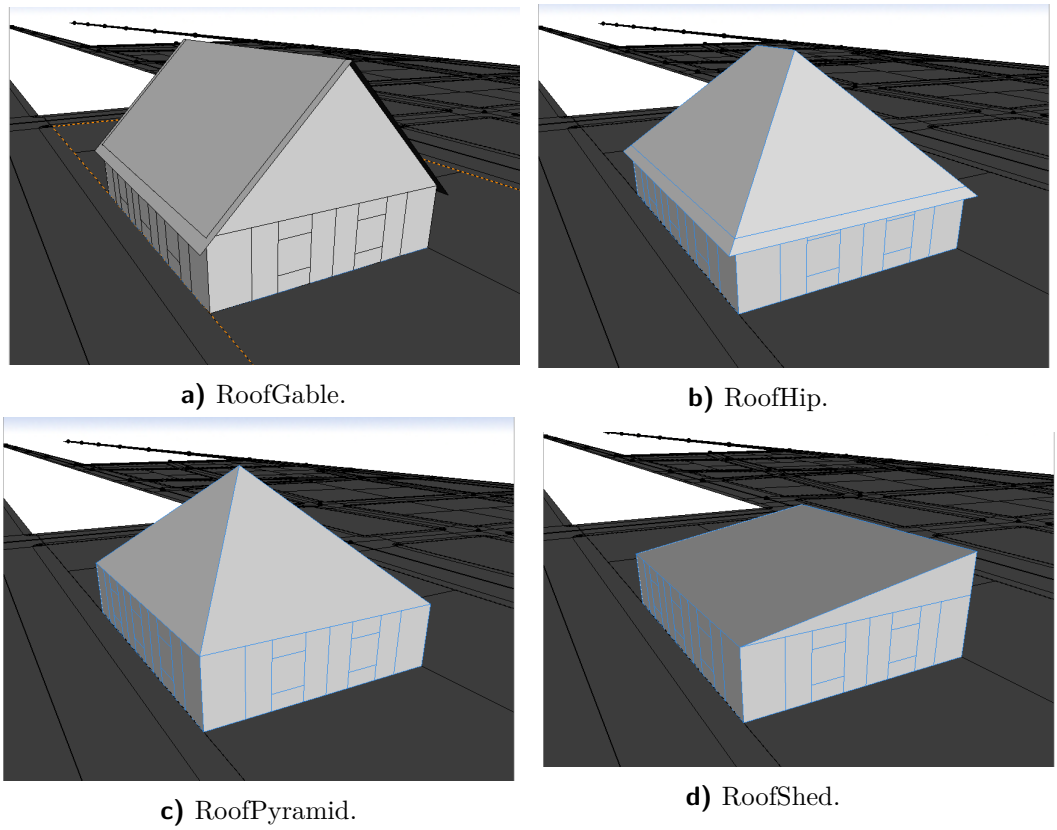
Podívejme se nejprve na extrude, jelikož tuto operaci používáme většinou již v počátečním pravidle. Jedná se o vytažení geometrie, kdy část objektu, na kterou je operace zavolaná, posouváme po úsečce zvoleného směru a délky. Tím nám vznikne nová oblast geometrie. Syntaxe operace extrude je $extrude(axisWorld, height)$, kde $axisWorld$ je zvolená světová souřadnicové osa a $height$ je délka vytažení.

Takto můžeme například z dvojrozměrného objektu vytažením geometrie vygenerovat trojrozměrný. Přesně podle tohoto postupu bývá v prvním pravidle pro generování budovy vytažena geometrie parcely v kladném směru souřadnicové osy y ve světových souřadnicích a tak získáme plášť budovy. Příkladem pravidla pro toto vytažení může být

$$\begin{aligned} &@StartRule \\ &Lot \rightarrow extrude(world.y, 5.5)Building \end{aligned} \tag{3}$$

Aplikaci takového pravidla na budově můžeme vidět na obrázku 15.

Další operací, kterou můžeme vytvářet geometrii, je operace vložení geometrie, $i(geometryPath)$, kde $geometryPath$ je cesta k souboru. Tímto způsobem můžeme do naší budovy vložit nějaký již vytvořený 3D model. Používáme-li tento způsob, můžeme takto pro naši budovu vytvořit její části (okna, římsy, a další) a ty potom vkládat do příslušných nahrazovacích pravidel touto operací. To je rychlejší než modelovat kompletně



a) RoofGable. **b)** RoofHip. **c)** RoofPyramid. **d)** RoofShed.

Obrázek 16 Různé druhy střech podle uvedených operací.

celou budovu.

Posledními operacemi, které si uvedeme ze skupiny vytváření geometrie, jsou operace pro vytváření střech domů. Tyto operace jsou:

- $roofGable(angle, overhangX, overhangY)$, kde $angle$ představuje úhel sklonu dvou protilehlých stěn střechy, $overhangX$ a $overhangY$ pak přesahy střechy v osách kolmých na hrany základny střechy.
- $roofHip(angle, overhang)$, kde $angle$ představuje úhel sklonu všech stran střechy a $overhang$ hodnotu přesahu střechy na každé straně.
- $roofPyramid(angle)$, kde $angle$ je úhel sklonu první zvolené stěny střechy. Tato stěna je vybrána dle skryté implementace CityEngine.
- $roofShed(angle)$, kde $angle$ je úhel sklonu celé střechy.

Příklady pravidel s jednotlivými typy střech jsou

$$\begin{aligned} Roof &\rightarrow roofGable(45, 1.5, 1.0)GeneratedRoof \\ Roof &\rightarrow roofHip(45, 1.5)GeneratedRoof \\ Roof &\rightarrow roofPyramid(45)GeneratedRoof \\ Roof &\rightarrow roofShed(10)GeneratedRoof \end{aligned} \tag{4}$$

Výsledky můžeme vidět na obrázku 16. Pořadí pravidel a pořadí obrázků odpovídá.

Nyní víme, jakých operací můžeme využít, abychom vytvořili geometrii a uvedli jsme si příklady, jak tyto operace můžeme použít pro tvorbu budov. Podívejme se nyní na operací dělení geometrie. Tyto operace využijeme k tomu, abychom rozdělili části objektů na další části. Na ty pak můžeme dále aplikovat nahrazovací pravidla.

Nejprve si představme operaci dělení na komponenty (Component Split). Přidáním této operace do nahrazovacího pravidla můžeme původní část objektu rozdělit na stěny (plochy), hrany či vrcholy. Syntaxe pravidla je
$comp(compSelector)\,\{selector\ operator\ operations|selector\ operator\ operations...\}$. Zde *compSelector* je selektor způsobu dělení. Může nabývat tří hodnot, podle kterých dělíme na různá geometrická primitiva:

- f - plochy.
- e - hrany.
- v - vrcholy.

Pomocí parametru *selector* specifikujeme část objektu, kterou chceme vydělit z původního objektu. Jedná se například o jednotlivé stěny budovy či střechu. Pomocí parametru *operator* zvolíme, jestli nově vzniklé části budou stát každá samostatně nebo jestli budou opět sjednoceny do jedné. Pro jednotlivé nové části můžeme ještě přidat další operace parametrem *operations*, které se po rozdělení provedou. Pak můžeme objekt rozdělit, aplikovat rozdílné operace na každou část a poté části opět sloučit do jednoho objektu nebo nechat rozdělené.

Jako příklad využití této operace si uveďme rozdělení domu na stěny a střechu

$$Building \rightarrow comp(f)\,\{side : Facade|top : Roof\} \tag{5}$$

Kromě dělení na komponenty můžeme dělit objekt ještě na části podle vzdálenosti. To provedeme operací dělení (Split), jejíž syntaxe je
$split(splitAxis)\,\{size_1 : operations_1|...|size_n : operations_n\}$. Parametr $splitAxis$ určuje osu v lokálním souřadném systému dělené části, podle které budeme část objektu dělit. Takto nejsme vázání předdefinovanými částmi objektů jako v předchozím případě, ale dělíme pouze podle námi zadaných vzdáleností.

Jako příklad si můžeme uvést dělení podlaží na části, do kterých chceme později vsadit okna

$$Floor \rightarrow split(x)\,\{\tilde{}1 : Wall|\,\{3 : WindowTile\} * |\tilde{}1 : Wall\} \tag{6}$$

Vidíme, že dělíme v ose x, tedy podélně. Dále vidíme, že pro vznik části $Wall$ definujeme délku jako ˜1, zatímco pro část WindowTile, do které bychom později chtěli vsadit okno, definujeme délku 3. Dalším prvkem je zde operátor opakování $*$ ve $\{3 : WindowTile\}*$. Nahrazení s touto definicí je vyhodnoceno následovně:

1. Nejprve rozděl objekt na maximální možný počet částí $WindowTile$ s délkou 3.
2. Zbývající nerozdělenou délku objektu rozděl na dvě části $Wall$ a vlož je po jedné na začátek a na konec dělené délky tak, že mezi těmito dvěma částmi budou všechny části $WindowTile$.

Definice délky ˜1 znamená zbývající délku po aplikování ostatních délek. Jelikož jsou zde definovány dvě části $Wall$ s touto délkou, je tato zbývající délka rozdělena ještě na dvě stejné části.

Obě uvedené operace rozdělení využijeme pro modelování budov. Pomocí rozdělení na komponenty můžeme rozdělit objekt na plášť budovy nebo i její jednotlivé stěny a její střechu. Vzniklé části pak můžeme dále rozvíjet.

Další skupinou operací jsou transformace částí objektů, posunutí, rotace a změna měřítka. Do pravidel můžeme dále vkládat matematické funkce (například goniometrické, logaritmus, umocňování) či můžeme generovat náhodná čísla.

V souborech pravidel můžeme dále definovat atributy a konstanty a jako hodnoty je využít v jednotlivých nahrazovacích pravidlech. Atribut oproti konstantě můžeme měnit v inspektoru scény nebo vykonáním změny ve skriptu. Atributy jsou tedy způsobem, jak dát tvůrci objektu využívající proceduru pro jeho vytváření možnost předat této proceduře vstupní parametry. Příkladem atributu a pravidla, které ho využívá, je

$$\begin{aligned} &attr height = 20 \\ &Lot \rightarrow extrude(world.y, height) Building \end{aligned} \tag{7}$$

Zde je definován atribut výšky budovy *height*, který je v pravidle nastaven na hodnotu 20. Uživatel může tuto hodnotu v inspektoru scény změnit. Jelikož na základě této hodnoty je vytažena geometrie (plášť budovy) ve světové souřadnicové ose y, je výška budovy přímo závislá na hodnotě atributu *height*.

Posledními důležitými částmi, které potřebujeme pro efektivní modelování budov, jsou parametrizovaná a podmíněná pravidla. Parametrizované pravidlo přijímá hodnotu, kterou může využít při nahrazování části objektu. Opět si to ukážeme na příkladě s vytažením pláště budovy. Podle vstupní hodnoty height je nyní určena výška budovy:

$$Lot(height) \rightarrow extrude(world.y, height) Building \tag{8}$$

Podmíněné pravidlo je takové, které ve své definici vyhodnocuje podmínku. Na základě toho, jak je tato podmínka vyhodnocena, je pak ovlivněno nahrazování. Vrátíme-li se opět k našemu příkladu s výškou budovy, můžeme tak definovat rozdílnou výšku budovy na základě toho, jaký typ budovy chceme vytvořit:

$$\begin{aligned} &Lot(buildingType) \rightarrow \\ &case buildingType == "skyscraper" : extrude(200) Building \\ &else : extrude(10) Building \end{aligned} \tag{9}$$

Parametrizované a podmíněné pravidlo můžeme také zkombinovat. Pak se můžeme v pravidle rozhodovat podle vstupní hodnoty. Tou může být například atribut předaný tvůrcem objektu. Taková pravidla nám dávají další možnost ovlivňování geometrie budov.

Nyní si shrňme, co nám tato část přinesla a uvažme, jak bychom tyto poznatky mohli aplikovat na modelování budov. Můžeme vytvořit části budovy nebo místo některých částí vložit vytvořený 3D model. Víme také, jak dělit budovy na tyto části, abychom mohli dále definovat jejich podobu. Do nahrazovacích pravidel můžeme vkládat operace CGA gramatiky, kterými definujeme funkčnost vykonanou při nahrazení těmito pravidly. Tvorbu budovy také můžeme ovládat, pokud dobře navrhneme atributy, které na ni budou mít vliv. Můžeme tak ovládat pouze některé aspekty budovy (například její výšku) nebo tyto atributy předat do pravidla a definovat jinou funkčnost.

2.3 Analýza vývoje města v čase a návrh jeho ovládání

Po uvedení technik pro procedurální modelování měst a představení metod, jak modelovat části městské zástavby v programu CityEngine, nyní analyzujeme vývoj měst v čase. Dostáváme se tak hlavní části této práce. Během vývoje měst dochází k mnoha změnám. My si uvedeme všechny relevantní pro tuto práci. Tato analýza nám dá přehled

o tom, co musíme navrhnout a implementovat v časově proměnných modelech měst, kterými se táto práce zabývá. Nejprve si tak vytvoříme celkový obrázek o vývoji města. Poté projdeme jednotlivými změnami, které si rozebereme detailně, a navrhneme, jak tyto změny implementovat. Samotná implementace podle tohoto návrhu v CityEngine pak bude popsána v následující kapitole, 3.

2.3.1 Přehled vývoje měst v čase

Město během svého životního cyklu prochází velkým množstvím změn. Tyto změny můžeme rozlišovat v různých rovinách. Jsou to třeba změny fyzické, například když někdo zboural dům a na jeho místě poté postaví jiný. Můžeme sledovat změny ekonomické, například když se obyvatelé města stávají bohatšími. To pak přirozeně vede k fyzické změně stavění nových domů či vylepšování (rozšiřování) existujících. Dále bychom mohli uvažovat změny kulturní či vkusové. Takovou změnou může být například změna obliby architektonického slohu, což vede také k fyzickým změnám. Budovy se pak budou stavět ve slohu, který začne být oblíbený. V naší práci si analyzujeme pouze změny fyzické (fyzický vývoj). Ekonomické změny, které jsou častou příčinou některých fyzických změn, nás zajímat nebudou, stejně jako žádné další druhy změn.

Podíváme-li se na fyzický vývoj města nejdříve z celkového hlediska, pozorujeme, že města s plynutím času rostou dvěma způsoby. Jednak se rozšiřují (plocha města se zvětšuje). Pak řekneme, že město roste do šířky. To se stává, když jsou na okrajích takového města stavěny nové domy či celá satelitní města. Volitelně bychom do tohoto druhu růstu mohli řadit i aglomerace. Druhý druh růstu města je růst do výšky. S postupem času se budovy ve městě stávají vyššími. Existujícím domům mohou být přistavěna nová patra, protože jejich majitel chce využívat větší obytnou plochu domu. Nebo se ve městě začnou stavět vyšší budovy, ať již například obytné panelové domy či administrativní budovy s kancelářemi. Některé z nových vysokých budov mohou nahradit původní nižší budovy, které jsou předtím zbourány.

Celkový vývoj měst i jejich růst provází celá řada fyzických změn. Tyto změny bychom mohli rozdělit do skupin podle jednotlivých prvků města, za které budeme nyní považovat ulice, parcely a budovy. Představme si stav vývoje města, ze kterého budeme vycházet a postupovat dál. Tento stav je určen právě ulicemi, parcelami a budovami, přesněji řečeno jejich detaily jako jsou počet, poloha jednotlivých prvků, jejich podoba či jejich stav (fyzická kondice). S postupem času je každá z těchto skupin doplňována o nové prvky. Jsou tedy stavěny nové ulice, u kterých mohou vzniknout nové parcely, na těch se zase staví nové domy. Současně se změny projevují i na stávajících prvcích: ulice i domy mohou být rekonstruovány, domy zbořeny či přestavěny nebo může být upravena jejich fasáda. S těmito změnami město roste a obměňuje se. Na jednotlivé skupiny prvků a jejich změny se nyní podíváme ještě detailněji.

Zaměříme-li se na změny ulic, napadne nás ihned stavění nových ulic. Považujeme-li nyní ulici za nadřazený pojem, můžeme stavbu dále rozdělit na stavbu silnic, které vedou z daného města k okolním městům, a na stavbu ulic přímo ve městě. Pro druhý případ platí, že ulice bývají stavěny spíše na okrajích města, či v částech, kde zatím není moc městské zástavby, která by stavbě nových ulic překážela. Na místech, která již bývají hustěji zastavěna domy, ale například i parky či jinými ulicemi, většinou již obyvatelé nové ulice nestaví. Zástavba zde již existuje a tak k tomu není důvod. Opačná je situace na okrajích měst či v méně zastavěných částech. Se stavbou nových ulic v těchto místech vznikají také nové parcely, na kterých obyvatelé posléze postaví nové domy. Tak vznikají nové bloky a čtvrti a město roste do šířky.

V místech, kde se již nové ulice nestaví, ale může dojít k jiné fyzické změně těchto

ulic. Tou je jejich rekonstrukce, která typicky nastává, když se fyzický stav těchto ulic zhorší.

Všimnout si můžeme také toho, že v různých obdobích se ulice staví v různých natočeních vůči sobě, či schématech. Zatímco v dřívějších obdobích (například gotika, renesance) nemusela mít vzájemná orientace ulic žádná pravidla a cesta z na sebe navazujících ulic byla křivolaká, dnes bývají ulice stavěny rovnoběžně vůči sobě. Tím také vznikají pravidelné bloky budov.

Podíváme-li se na vývoj budov města, opět nás může napadnout stavba nových domů. Jak se město postupně rozšiřuje, jsou stavěny na místech, kde předtím žádné nestály nebo na místech původních budov, které byly předtím zbourány. Bourání budov můžeme považovat za další změnu.

Podoba nových domů je většinou určena aktuálním stavebním stylem. Ten udává velké množství detailů na budově, jejichž přesný rozbor není součástí této práce. Můžeme si ale představit detaily jako jsou prvky fasád, ale také výška pater či jejich počet. Zde platí, že čím je období pozdější, tím je počet pater vyšší. S postupem času přestává být využíváno stávajícího stylu a nastupuje nový.

Změny nastávají i na již existujících budovách. Zde můžeme uvažovat rekonstrukci či rozvoj takové budovy. Ten se může projevit přistavěním nových pater na původní vrchní patro. Jako další projev rozvoje budovy můžeme chápat změnu fasády, kdy je původní fasáda stržena a vybudována nová, která vychází z aktuálního architektonického stylu.

2.3.2 Časově proměnný model města

Nyní máme celkový přehled o změnách, které se v městě dějí v průběhu času. Na jejich základě navrhneme časově proměnný model města, který posléze implementujeme v programu CityEngine. Změny, které byly popsány v předchozí části, analyzujeme detailněji z hlediska návrhu časově proměnného modelu města. Než se k tomu ale dostaneme, musíme si uvést ještě základní myšlenky takového modelu a požadavky na něj.

Chceme, aby časově proměnný model města simuloval vývoj podoby města. Proto takový model musí implementovat veličinu času. Na jejím základě bude město růst a měnit se. Model bude pro různé vstupní hodnoty času zobrazovat různé podoby města. Je otázkou, jak zvolit časovou jednotku. Nejvhodnějším způsobem se nám zdá nepoužívat jednotky reálného světa, ale zvolit jednotku vlastní, imaginární.

Jednotky reálného světa, například roky, se vztahují k pevně daným časovým obdobím. Jsme zvyklí, že během těchto období může být vykonáno určité množství práce, a tak i města se během těchto období mohou změnit jen nějakým způsobem. My se ale v naší aplikaci soustředíme spíše na základní okolnosti postupu času a toho, jak se projeví na podobě města. Například uvažujeme, že za určitý čas mohou ve městě nastat určité změny (jako třeba postavení určitého počtu nových ulic), ale nezabýváme se již tím, jestli je doba pro provedení této změny dostatečná. Navíc můžeme chtít při vytváření nových modelů délku časové jednotky definovat rozdílně. Někdy bychom chtěli, aby se změny děly rychleji, někdy zase pomaleji. Pak by délka období určeného touto jednotkou nemohla být ve všech modelech stejná, zatímco délka reálných časových jednotek vždy je.

Časové jednotky dále reprezentují jisté zákonitosti, například daný počet let určuje daný počet architektonických stylů používaných během těchto let a také určuje posloupnost těchto stylů. Námi vytvářený časově proměnný model ale tyto zákonitosti nemusí respektovat. To by navíc bylo i velmi těžké korektně zachytit a není to předmětem této práce. Navíc teoreticky můžeme chtít i definovat vlastní posloupnosti architektotnických stylů nebo vlastní délky období jejich používání. Toto opět nemusí odpovídat realitě.

Vidíme tedy, že použití reálných jednotek by neodpovídalo zákonitostem, na které jsme u nich v realitě zvyklí a je proto nevhodné. Zvolíme tedy imaginární časovou jednotku, kterou budeme v následujícím textu označovat jednoduše "časová jednotka".

Již jsme si uvedli, že časově proměnný model bude pro různé vstupní hodnoty času zobrazovat různé podoby měst. To můžeme využít jednak k sledování vývoje nějakého města krok po kroku nebo k zobrazování podob města v námi vybraných časech (například se chceme podívat na nějakou konkrétní hodnotu v nějakém minulém období, které pro nás mohlo být něčím význačné). Chtěli bychom tak jednak mít možnost sledovat vývoj města v postupu času. Mohli bychom nechat čas plynout a sledovat, jak se nám město mění před očima. Tyto změny budou vytvářeny procedurálně, takže my skutečně budeme moci nechat čas plynout a nechat město měnit se samo, ale na základě námi předaných parametrů.

Chtěli bychom totiž vytvářet města, která se nám líbí. Proto bychom chtěli být schopni vývoj města ovládat, abychom toho docílili. Proto potřebujeme parametry, na základě kterých budeme moci vývoj města řídit. Tyto parametry musí být dobře definované. Parametry uznáme dobře definovanými, když budou mít následující vlastnosti:

- Síla (efektivita) - Parametry musí být dostatečné silné na to, aby se v procedurách jejich změna projevila a abychom změnou tohoto parametru skutečně ovlivnili výslednou podobu města.
- Smysl - Parametry musí procedury ovlivňovat smysluplně. Jelikož ovlivňují aspekt výsledného modelu, musí jít o smysluplný aspekt. Těmi mohou být například počet nově vytvářených ulic nebo pokyn ke zbourání vybraných domů.
- Jednoduchost na pochopení - Aby uživatelé mohli ovlivňovat vývoj svých měst těmito parametry, musí být pro uživatele snadno pochopitelné, co který parametr ovlivní. A to i pro uživatele, jejichž znalosti o modelování města nejsou na úrovni expertů.
- Ideálně nízký počet parametrů - Velké množství by mohlo vést sice k dobré ovladatelnosti vývoje města, ale nutnost mnoha definic by práci s modelem zpomalila.

Podíváme-li se na poznatky z předchozí části o vývoji měst v čase, vidíme, že jsme si změny provázející vývoj města rozdělili do změn ulic, parcel a budov. Jednotlivé změny z těchto kategorií budeme chtít těmito parametry ovlivnit. Tak bychom mohli ovládat prvky vývoje města jako například zmíněný počet ulic či výběr domů ke zbourání.

Řekli jsme si, že časově proměnné modely měst chceme využít ke sledování vývoje měst či k nahlížení do historie modelovaného města. Model proto musí mít tyto dvě funkcionality. Dále nám musí umožňit procedurálně vytvořit zcela nové město o počátečním stavu, se kterým poté budeme pracovat. Shrňme si tedy základní myšlenky o těchto druzích funkcionality:

- Vytvoření nového města - Uživatel modelu chce tento krok realizovat jako první, aby procedurálně vytvořil prvotní stav města.
- Přirozené plynutí času - Umožní sledovat vývoj města v čase. Uživatel přitom bude chtít ovládat vývoj města tak, aby vytvořil město, které se mu líbí.
- Skok v čase - Umožní vrátit se do vybrané časové hodnoty a prohlédnout si podobu města v této době. Pokud se uživateli nelíbí, jak se město vyvíjelo od této chvíle nebo by chtěl něco změnit, může tak učinit předáním parametrů a opět sledovat vývoj města s přirozeným plynutím času. Tak může vytvořit jiné město než bylo vytvořeno původně.

Pro vytvoření nového města a přirozené plynutí času platí, že procedurálně generujeme nové prvky ve městě. Oproti tomu při skoku v čase pouze načítáme již vytvořené podoby a nic nového nevytváříme. Abychom ale mohli načítat podoby města při skoku

v čase, musíme tedy při vytvoření města či při plynutí času nově vytvářené prvky a detaily o nich ukládat.

Na jednotlivé funkcionality uvedené výše se nyní podíváme podrobně. Potřebné věci si dovysvětlíme a navrhneme, jak je realizovat. Každá z funkcionalit se stane jednou z hlavních procedur pro tvorbu modelu.

2.3.3 Vytvoření nového města

Časově proměnný model města vzniká vytvořením iniciálního stavu. V tomto stavu je město definováno prvními ulicemi, parcelami a budovami. Z tohoto stavu poté uživatel modelu vychází a nechá plynout čas, takže jsou vytvořeny další tyto prvky.

Proces tvorby nového města začíná tedy ve chvíli, kdy model města neobsahuje žádná geometrická primitiva. Neexistují žádné ulice ani budovy. Před vlastní tvorbou města uživatel definuje parametry, podle kterých chce, aby se město vytvářelo. Těmito parametry jsou:

- Počet stavěných ulic.
- Schéma stavěných ulic.
- Použité architektonické slohy a časy jejich nástupů v časových jednotkách.
- Perioda změn - Doba v časových jednotkách, po které je pravděpodobnost, že budova ve městě bude zbourána či rozšířena, maximální.

Tvorba prvotního stavu města pak probíhá v následujících krocích:

1. Hodnotu času stanovíme na počáteční, tedy 1 časovou jednotku.
2. Určíme skutečný počet stavěných ulic. Uživatel předtím předal jím preferovanou hodnotu, výsledný počet ale není určen jen uživatelem, ale také na základě náhody. My ho definujeme jako náhodnou hodnotu z určitého rozsahu okolo hodnoty předané uživatelem, tedy:

$$\begin{aligned} &StreetCount \in [UserCount - \delta, UserCount + \delta], \\ &StreetCount, UserCount, \delta \in \mathbb{Z} \end{aligned} \tag{10}$$

   Zde $StreetCount$ je výsledný počet vytvořených ulic, $UserCount$ je počet ulic předaný uživatelem a δ je hodnota definující okolo $UserCount$. Schémata ulic, které v této práci uvažujeme, jsou křivolaké, rovnoběžné a kruhové. Více o nich si můžeme přečíst v kapitole 2.3.3.
3. Vytvoříme výše určený počet ulic, které jsou vystavěny dle schématu předaného uživatelem.
4. Vytvoříme parcely města. Ty vytvoříme v blocích (smyčky ulic) a okolo ulic, kde po této operaci nebyly vytvořeny.
5. Na každé parcele vytvoříme dům v prvním architektonickém slohu.
6. Každé parcele přiřadíme násobek k periodě změn. Ta je předána uživatelem a touto hodnotou je násobena. Tuto hodnotu určíme opět náhodně:

$$PeakMultiple \in [1 - \epsilon, 1 + \epsilon] \tag{11}$$

   Zde ϵ udává rozsah okolo hodnoty 1. Na základě ϵ tedy může být doba do změny více či méně prodloužena či zkrácena. Násobek je pojmenován $PeakMultiple$, jelikož v čase $Časposlednízměnynaparcele + dobadozměny$ je pravděpodobnost, že bude budova zbourána či rozšířena, maximální.

7. Uložíme všechny detaily města, které potřebujeme k opětovnému načtení modelu v tomto časovém okamžiku. Těmito hodnotami jsou:
   - Postavené ulice.
   - Způsob, jakým jsou bloky rozděleny na parcely. To se může dít náhodně, v takovém případě pak stačí uložit semínko pro generátor náhodných čísel.
   - Násobky periody změn pro jednotlivé parcely.
   - Detaily jednotlivých parcel - jaký typ domu je na parcele postaven a jak je tento dům rozvinutý.
   - Parametr výběru schématu stavění ulic předaný uživatelem. V tomto schématu budou ulice stavěny i nadále.
   - Časy posledních změn na domech (Zde je to tedy čas 1 u všech domů, jelikož všechny domy byly postaveny v tomto kroce).
   - Aktuální hodnotu času. Zde je to hodnota 1.

Tímto postupem vytvoříme nové město a uložíme všechny hodnoty, které jsou potřebné opětovnému načtení této podoby. Po tomto postupu uživatel nechá alespoň jednou plynout čas, aby se město dále rozšířilo.

Schémata stavby ulic

Již jsme si uvedli, že ulice mohou být stavěny v různých natočeních, orientacích vůči sobě. Ty nazýváme schémata a v této práci jich rozeznáváme tři typy:

- Křivolaké. Toto schéma bylo používáno spíše v dřívějších obdobích. Jde o schéma, kdy jsou ulice stavěny náhodně natočené k sobě. Vytvoříme-li tak z ulic cestu, je tato cesta křivolaká.
- Kruhové okolo nějakého bodu. Okolo něj se ulice vyskytují v kružnicích. Prvky tohoto schématu můžeme najít například v Paříži.
- Rovnoběžné (pravoúhlé). Ulice jsou stavěny do pravého úhlu. Tak vznikají rovnoběžné bloky. Toto schéma se vyskytuje spíše v pozdějších obdobích. Vidět ho můžeme například v New Yorku.

Znázornění schémat můžeme vidět na obrázku 17.

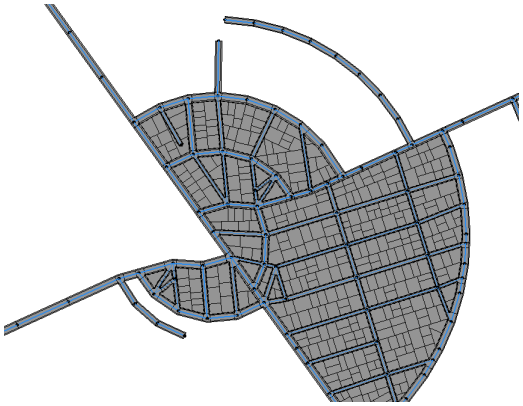
2.3.4 Přirozené plynutí času

Nyní definujeme postup pro rozvoj města za přirozeného plynutí času. Díky němu budeme moci sledovat vývoj města krok po kroku. Stavem, ze kterého vycházíme, je město, které je již určeno ulicemi, parcelami a domy. Navíc uživatel, který ovlivňuje podobu modelu, předal nejméně jednou (buď před vytvořením nového města nebo před krokem tohoto typu) tyto parametry:

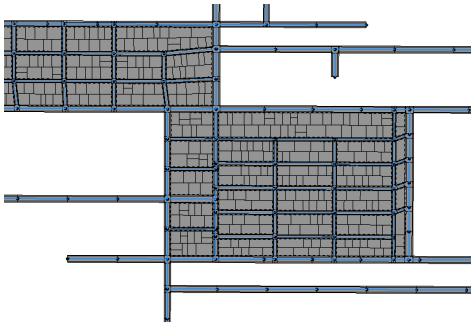
- Počet stavěných ulic.
- Schéma stavěných ulic.
- Perioda změn - Doba v časových jednotkách, po které je pravděpodobnost, že budova ve městě bude zbourána či rozšířena, maximální.

Tyto parametry může kdykoliv (i nyní) změnit. Také definoval použité architektonické slohy a časy jejich nástupů v časových jednotkách. Uživatel může chtít dále zasáhnout do vývoje města tím, že bude měnit pravděpodobnost některé ze změn existujících budov. Toho může dosáhnout předáním parametrů z tabulky 1. Tyto parametry se dělí na globální a lokální. Globální parametry ovlivní všechny existující budovy ve městě. Lokální ovlivní pouze budovy vybrané uživatelem. V tabulce 1 také můžeme

a) Křivolaké.



b) Kruhové.



c) Rovnoběžné (pravoúhlé).

Obrázek 17 Schémata stavby ulic.

Typ parametru	Název	Efekt
Global	City Reconstruction	Budovy města budou zbourány a na jejich místě postaveny nové v aktuálním stavebním stylu.
	Preservation	Budovy města budou zachovány. Nebudou bourány ani přestavovány.
	Development	Budovy města budou přestavovány (rozšiřovány).
Local	Preservation	Budova bude zachována. Nebude zbořena či přestavěna.
	Development	Budova bude přestavěna (rozšířena).
	Demolishion	Budova bude zbourána.

Tabulka 1 Tabulka parametrů ovlivňujících existující budovy.

vidět, jak se jednotlivé parametry projeví. Mějme na paměti, že nastavení parametru pouze zvýší pravděpodobnost patřičné změny. Může ale nemusí tuto změnu vyvolat.

Poté, co uživatel dodefinuje požadované parametry, provedeme vývoj aktuální podoby města v těchto krocích:

1. Načteme hodnotu času, kterou zvýšíme o 1 časovou jednotku.
2. Načteme detaily města z poslední časové jednotky. Těmito detaily jsou:
   - Časy posledních změn na domech.
   - Pakliže uživatel nedefinoval nové hodnoty pro některý z následujících parametrů, jak je uvedeno výše, jsou tyto hodnoty načteny z poslední časové jednotky a použity. Jedná se o tyto parametry:
     - Perioda změn.
     - Počet stavěných ulic.
     - Schéma stavených ulic.
     - Globální parametry ovlivňující změny existujících budov.
     - Lokální parametry ovlivňující změny existujících budov.
   - Násobky periody změn pro každou parcelu.
3. Vykonáme vývoj města a jeho jednotlivých objektů v tomto pořadí:
   a) Postavíme určený počet ulic ve schématu určeném uživatelem. Počet ulic opět určíme na základě uživatelem preferovaného počtu ulic a náhody podle vzorce 10.
   b) Pro každou existující budovu vypočítáme pravděpodobnosti jejího zbourání (a postavení nové budovy na jejím místě) či rozšíření. Na základě těchto pravděpodobností a náhody rozhodneme, zda budou budovy zbourány či rozšířeny. Nejdřív zjišťujeme, jestli bude budova zbourána. Pokud ano, zbouráme ji a již nezkoumáme rozšíření. Pokud budovu nezbouráme, zjišťujeme ještě, jestli bude rozvinuta. I toto se může stát a nemusí. Pokud zboříme budovu, postavíme na jejím místě novou. Rozvinutost nové budovy se přitom bude odvíjet od rozvinutosti ostatních budov v bloku. Když zboříme nějakou budovu a postavíme na jejím místě novou, bude například podobně vysoká jako okolní budovy. Rozvinutost nově vzniklé budovy definujeme jako průměr rozvinutostí okolních budov. Pokud původní budovu zboříme nebo rozvineme, pak:
      - Zapíšeme aktuální čas jako čas poslední změny na budově.
      - Pokud uživatel předal před změnou nějaký z lokálních parametrů ovlivňující změněnou budovu, tento parametr zahodíme.

      Více o výpočtu pravděpodobnosti změn uvádíme v části 2.3.4.
   c) Vytvoříme nové parcely.
   d) Na těch postavíme nové budovy v aktuálním stavebním stylu.
4. Uložíme všechny potřebné detaily, abychom mohli aktuální stav města opět načíst v případě požadavku uživatele. Kromě hodnot uvedených výše (které jsme načítali či je definoval uživatel) jsou to následující:
   - Způsob rozdělení bloku na parcely.
   - Detaily jednotlivých parcel - typ postaveného domu a jeho rozvinutost.
5. Uložíme aktuální hodnotu času.

Takto rozvineme vstupní podobu města. Shrneme-li si tento proces, zjistíme, že stavíme nové ulice, parcely a domy a zkoumáme možnosti zbourání či rozšíření existujících domů. Tyto změny budou provedeny s určitou pravděpodobností. Ta je závislá na čase uplynulém od poslední změny a na dalších věcech. Těmito tématy se zabýváme v následující části.

Výpočty pravděpodobností změn

Během přirozeného postupu v čase kromě stavby nových objektů ve městě zjišťujeme, jestli provedeme změny na existujících budovách města. Ty mohou být buď zbourány, aby jejich místo zaujala budova nová, nebo mohou být rozšířeny (přestavěny). Obě tyto změny nastanou s určitou pravděpodobností. Na základě jí a náhody rozhodneme, zda-li je vykonáme. Nyní si vysvětlíme výpočet těchto pravděpodobností.

Ty jsou závislé jednak na základní pravděpodobnosti změny, která se zvyšuje s plynutím času. K této hodnotě dále přičítáme pravděpodobnostní příspěvek od uživatele. Ten ho definuje globálními či lokálními parametry změn existujících budov. Pravděpodobnost rozvoje budovy navíc závisí ještě na tom, kolik budov z bloku, kde se tato budova nachází, je rozvinutějších.

Ať jde již o pravděpodobnost zbourání nebo rozšíření, základem výpočtu je určení základní pravděpodobnosti. Jedná se o pravděpodobnost závislou čistě na stáří poslední změny budovy (poslední rozšíření či postavení této budovy). Tato pravděpodobnost s přibývajícím časem od změny nejdříve stoupá a poté opět klesá. Simuluje tak stárnutí budovy a změn na ní provedených. Platí, že v krátké době po provedení změny není pravděpodobné, že bude provedena nová. Poté, co postavíme budovu, ji nebudeme brzy rozšiřovat. Poté, co postavíme nová patra budovy, nebudeme brzy stavět další ani upravenou budovu nezbouráme.

Uživatel nicméně může ovlivnit, kdy základní pravděpodobnost dosáhne svého maxima. To udělá definicí periody změn. To je uživatelova preference doby, za kterou by chtěl, aby základní pravděpodobnost dosáhla své maximální hodnoty. Čas dosažení maxima je pro každou budovu jiný a je závislý na násobku periody změn, který je pro každou budovu unikátní. Jeho výpočet jsme si uvedli v zápise 11. Čas dosažení maxima pro každou budovu se pak vypočítá jako:

$$PeakTime = PeriodOfChanges * PeakMultiple \tag{12}$$

Zde $PeriodOfChanges$ je perioda změn definovaná uživatelem a $PeakMultiple$ je násobek této doby odlišný pro každou budovu.

V čase změny je pravděpodobnost další změny nulová. Od času změny do času dosažení maxima roste. Poté začne opět klesat. První část průběhu (růst) simuluje fakt, že čím je budova déle bez změny, tím pravděpodobnější se tato změna stává. To platí ovšem jen do určité chvíle, poté se pravděpodobnost řídí druhou částí průběhu (pokles). Zde začne rychle klesat a opět klesá k nule. Tímto simuluje fakt, že poté, co stáří budovy překročí určitou hodnotu (čas dosažení maxima), začne být budova považována za historickou, chráněnou, či se na ní začne vztahovat památková péče. Takové budovy již nepřestavujeme ani nebouráme, pouze je můžeme rekonstruovat. Pravděpodobnost jakékoliv další změny na budově proto rychle klesá.

Základní pravděpodobnost změny budovy tedy vypočítáme rozdílně pro čas před dosažením maxima a pro čas po dosažení maxima. Vztah pro výpočet je následující:

$$DefaultProbability = \begin{cases} \alpha \frac{Age}{PeakTime} & \text{Age <= PeakTime} \\ \frac{\alpha}{1+\beta*(Age-PeakTime)} & \text{Age > PeakTime} \end{cases} \tag{13}$$

V tomto vztahu Age je stáří poslední změny (doba uplynulá od této změny) a $PeakTime$ je čas dosažení maximální hodnoty. Pro $DefaultProbability$ platí: $DefaultProbability \in [0, 1]$. Vzorec dále obsahuje dvě konstanty $\alpha \in [0, 1]$, $\beta \in [1, \infty]$, jejich význam je následující:

Typ změny	Typ parametru	Parametr	Vliv na pravděpodobnost změny
Zboření	Globální	City Reconstruction	Zvyšuje
	Globální	Preservation	Snižuje
	Lokální	Demolishion	Zvyšuje
	Lokální	Preservation	Snižuje
Rozvoj	Globální	Dvelopment	Zvyšuje
	Globální	Preservation	Snižuje
	Lokální	Development	Zvyšuje
	Lokální	Preservation	Snižuje

Tabulka 2 Vliv parametrů na pravděpodobnosti změn.

- α je omezující konstanta výsledné hodnoty. Hodnota pravděpodobnosti by totiž v čase PeakTime byla rovna 1. Pravděpodobnost uvažované změny by pak byla 100%. Touto konstantou tedy můžeme pravděpodobnost omezit. To učiníme, jelikož základní pravděpodobnost je pouze jedním z příspěvků do výsledné pravděpodobnosti, jak si ještě ukážeme. Proto nechceme, aby měla takovou váhu. Navíc takto můžeme definovat i rozdílnou základní pravděpodobnost pro zbourání budovy a její rozšíření.
- β určuje strmost poklesu hodnot pravděpodobnosti v závislosti na čase. Jejím zvýšením dosáhneme rychlejšího poklesu pravděpodobnosti změny k nule.

Základní pravděpodobnost tedy určuje podobně pro zbourání budovy i rozšíření, lišit se mohou v definici konstant α a β. Dalším příspěvkem k celkové hodnotě pravděpodobnosti je hodnota definovaná parametry předanými uživatelem. Těmi tedy uživatel může zvýšit či snížit celkovou pravděpodobnost. Nyní se nabízí otázka, jak definovat rozsah parametrů a jak je promítnout do výpočtu pravděpodobnosti.

Podívejme se nejprve na celkovou funkčnost parametrů, tak jak jsme si je uvedli v tabulce 1. Zde vidíme, že máme parametry lokální a globální. Globálními parametry uživatel může uživatel ovlivňovat změny existujících budov v celém města, lokálními může ovlivnit existující budovy, které si sám vybere. My definujeme priority parametrů tak, že lokální budou mít přednost před globálními. V každé skupině jsou dále tři parametry. Ty vedou na zcela odlišné výsledky. Development zvyšuje pravděpodobnost rozvoje budov, Demolishion (lokální) potažmo City Reconstruction (globální) způsobují bourání starých budov a stavění nových a Preservation snižuje pravděpodobnost jakýchkoliv změn na budovách. Jedná se tedy o odlišné trendy. Ve výpočtech pravděpodobnosti také tyto parametry působí protichůdně, podívejme se na tabulku 2.

Rozsah parametrů definujeme od nuly do jedné. To odpovídá práci s pravděpodobností. Pak tedy uživatel předáním těchto parametrů předává přímo příspěvky do celkové hodnoty pravděpodobnosti. Nyní definujeme výpočty pravděpodobnosti pro obě změny pro uvažovanou budovu z parametrů na základě tabulky 2. Záleží přitom na tom, jestli uživatel definoval lokální parametry pro tuto budovy nebo ne.

$$ParameterProbability_{Demolishion} =$$
$$= \begin{cases} CityReconstruction - GlobalPreservation & \text{Lokální parametry nedefinovány.} \\ LocalDemolishion - LocalPreservation & \text{Lokální parametry definovány.} \end{cases} \tag{14}$$

$$ParameterProbability_{Development} = \\ = \begin{cases} GlobalDevelopment - GlobalPreservation & \text{Lokální parametry nedefinovány.} \\ LocalDevelopment - LocalPreservation & \text{Lokální parametry definovány.} \end{cases} \tag{15}$$

Jelikož hodnoty parametrů jsou v rozsahu od nuly do jedné, pro obě pravděpodobnosti z parametrů platí:

$$\begin{aligned} ParameterProbability_{Demolishion} &\in [-1, 1] \\ ParameterProbability_{Development} &\in [-1, 1] \end{aligned} \tag{16}$$

Je nutné mít zde možnost jít do záporných hodnot, protože tuto hodnotu přičítáme k hodnotě základní pravděpodobnosti. Tu tak můžeme zvýšit i snížit. Na samém konci budeme pracovat s pravděpodobností v rozsahu $[0, 1]$, takže hodnoty ořízneme do tohoto rozsahu. Jelikož je ale nynější výsledek v rozsahu $[-1, 1]$, dává nám to možnost zvýšit i snížit dle naší preference pravděpodobnost o celý výsledný rozsah hodnot. Můžeme tedy sami plně rozhodnout, co se s domem stane, bez ohledu na další pravděpodobnostní příspěvky.

Předtím, než si uvedeme celkové vzorce pro výpočet pravděpodobností pro oba typy změn na budovách, definujeme ještě poslední pravděpodobnostní příspěvek. Ten přispívá pouze k pravděpodobnosti rozvoje budovy. Je jím příspěvek bloku, ve kterém budova stojí.

Tento příspěvek simuluje fakt rozvoje budovy s ohledem na rozvinutost ostatních budov v bloku. Majitelé budov v bloku často chtějí mít budovy podobně nebo stejně vysoké jako jejich sousedi, také renovují fasádu domu spíše ve chvíli, kdy tak učiní jejich soused v bloku. Příspěvek bloku k pravděpodobnosti rozvoje budovy se tedy zvyšuje se zvyšujícím se počtem rozvinutějších (vyšších, s novou fasádou) budov v bloku.

Vzorec pro výpočet takového příspěvku definovali ve své práci Honda a kol. [2]. Zde kromě dalších věcí závisel na hodnotách rozvinutosti ostatních domů v bloku. My se tímto výpočtem inspirujeme, ale definujeme ho pro zkoumanou budovu jinak. Dodejme si, že také tento pravděpodobnostní příspěvěk je závislý na stáří poslední změny budovy. Proto do výpočtu přidáváme násobení základní pravděpodobností, která je sama závislá na stáří poslední změny, přičemž α ze vzorce 13 bude rovna 1. Pravděpodobnost bloku definujeme jako náhodné číslo:

$$BlockProbability \in [0, DefaultProbability * \frac{MoreDeveloped}{Buildings}] \tag{17}$$

Zde $MoreDeveloped$ je počet budov v bloku, které jsou rozvinutější než zkoumaná budova, a $Buildings$ je počet všech budov v bloku. Hodnota $BlockProbability$ je z rozsahu $[0, 1)$. Takovýto příspěvek odráží výše uvedené situace v bloku domů a je přičítán k celkové pravděpodobnosti rozvoje budovy.

Celkové pravděpodobnosti obou typů změn můžeme nyní definovat jako součet výše definovaných pravděpodobnostních příspěvků. Nezapomeňme, že ve vzorci pro výpočet základní pravděpodobnosti se mohou konstanty α a β lišit. Výsledné hodnoty obou pravděpodobností ořízneme tak, aby ležely v intervalu $[0, 1]$. Vzorce pro výpočet pravděpodobností zboření a rozšíření pro uvažovanou budovu a její poslední změnu o stáří

Age jsou:

$$DemolishionProbability = DefaultProbability + \\ + ParameterProbability_{Demolishion} \tag{18}$$

$$DevelopmentProbability = DefaultProbability + \\ + ParameterProbability_{Development} + BlockProbability \tag{19}$$

Podle těchto vzorců počítáme pravděpodobnost provedení obou typů změn existujících budov.

2.3.5 Skok v čase

Chtěli bychom, aby uživatel mohl využít námi definované časově proměnné modely měst také k zobrazování minulých podob měst. Proto nyní definujeme postup pro skok z jedné časové hodnoty (a od jedné podoby města) k druhé.

Na začátku tohoto procesu se nacházíme v daném stavu města, prvků v něm, s definovanými parametry uživatele a dalšími detaily, které byly popisovány v postupu pro přirozené plynutí času, 2.3.4. Tento stav nyní opustíme a vrátíme se k nějakému již existujícímu stavu. Často to pravděpodobně bude ke stavu minulému, ale zjistíme, že chceme-li se například z této minulosti vrátit do již existující budoucnosti (během celého vývoje časově proměnného modelu jsme si detaily všech stavů ukládali), postup bude stejný. Uživatel může před skokem v čase definovat libovolné parametry, které se zapíšou do aktuálního stavu (do toho, ze kterého vycházíme). Jelikož ale pouze načítáme podobu a detaily modelu, tyto parametry při časovém skoku nevyužijeme.

Postup pro časový skok do požadované časové jednotky definujeme takto:

1. Načteme všechny detaily a parametry uložené v cílové časové jednotce. Těmito detaily tedy jsou:
   - Časy posledních změn budov.
   - Parametry definované uživatelem. Těmi jsou:
     - Globální a lokální parametry změn existujících budov.
     - Perioda změn.
     - Počet stavěných ulic.
     - Schéma, ve kterém jsou ulice stavěny.
   - Násobky periody změn pro každou budovu.
   - Pozice původních ulic.
   - Rozdělení na parcely každého z bloků.
   - Detaily parcel (typ zde postaveného domu a jeho rozvinutost).
2. Načteme podobu modelu města z cílové časové jednotky v těchto krocích:
   a) Obnovíme ulice z cílové časové jednotky ve dvou krocích. Tímto nám vznikne původní struktura ulic města a obnoví se i bloky.
      - Smažeme ty aktuální ulice, které v cílové časové jednotce neexistují.
      - Vytvoříme chybějící ulice, které v modelu zatím nejsou, ale v této časové jednotce existovaly.
   b) Pro každý blok napravíme způsob jeho rozdělení na parcely tak, aby odpovídal původnímu způsobu.
   c) Vytvoříme chybějící parcely.
   d) Upravíme detaily všech parcel tak, aby odpovídaly cílové časové jednotce. Na těchto parcelách podle těchto detailů postavíme domy.

3. Aktuální čas nahradíme požadovanou časovou jednotkou.

Podle tohoto postupu jsme schopni načíst podobu modelu a všechny potřebné detaily ze zvolené časové jednotky. Tento postup je přitom univerzální a může být využit do jakéhokoliv uloženého stavu modelu, tedy z aktuálního času do minulosti, ale z této minulost i do již vytvořené pozdější časové jednotky. Uživatel si tak může krok po kroku prohlížet již vytvořené podoby modely. Také se ale nabízí návrat do minulosti a poté opětovná tvorba modelu přirozeným plynutím času. Předtím může také uživatel přenastavit načtené parametry. To udělá, pokud chce nechat vytvořit jiné podoby modelu, než se vytvořily původně.

3 Implementace

V této kapitole si představíme implementaci simulace vývoje měst na základě technik popsaných v předchozích kapitolách. Podíváme se na to, jak je simulace implementována v programu CityEngine. Podíváme se také na to, jaká je architektura implementace, popíšeme si jednotlivé třídy a uvedeme si důležité implementační detaily. Také si ukážeme, jak může uživatel importovat kód umožňující vytvoření simulace do CityEngine.

3.1 Použitá technologie

Časově proměnné modely měst jsem implementoval v programu CityEngine Advanced verze 2011.2. Jedná se již o starší verzi, ve které jsou ale všechny principy popsané v této práci implementovatelné. Také popis programu v kapitole 2.2.2 se týká této verze. Od novějších verzí se 2011.2 liší absencí některých modelovacích nástrojů a metod v aplikačním rozhraní. Ty jsme ale v této práci nevyužili.

Časově proměnné modely jsou napsány v jazyce Python, CityEngine 2011.2 ale využívá implementaci Pythonu v Javě, tedy Jython verze 2.5.2. Z hlediska provedené implementace je rozdíl verze 2.5.2 oproti novějším verzím v absenci některých datových struktur.

3.2 Architektura

Časově proměnné modely jsou definovány v jednom modulu s názvem TimeVaryingCity. Implementace je rozdělena do pěti tříd:

- TimeVaryingCity
- GraphManager
- BuildingManager
- ParameterStorage
- FileManager

Třída TimeVaryingCity je vstupním bodem pro práci s modely. GraphManager se stará o generování ulic a parcel v modelu. BuildingManager se stará o generování budov. ParameterStorage uchovává parametry ovlivňující vývoj města, které definuje uživatel. FileManager se stará o ukládání a načítání některých údajů ze souborů.

Další částí projektu jsou skripty spustitelné v CityEngine. Spouštěním těchto skriptů vykonáme různé druhy úloh:

- CityGeneration.py - Vytvoříme nové město v prázdné scéně v CityEngine. Město bude obsahovat počáteční sadu ulic a parcel s postavenými domy, které budou na nejnižším stupni rozšíření.
- TimeFlow.py - Postoupíme přirozeným způsobem v čase. Uplyne 1 časová jednotka. Proběhne rozvoj města.
- TimeSkip.py - Skočíme do definované časové hodnoty. Město pro tuto časovou hodnotu muselo být dříve vytvořeno.

- SetParameter.py - Před spouštěním skriptu vybereme budovy, pro které chceme nastavit lokální parametry ovlivňující jejich následné změny. Parametry nastavené budovám v inspektoru scény jsou uloženy.
- SetStreetCount0.py - Nastavíme počet ulic, který bude vytvářen při postupu vpřed v čase (při spouštění skriptu TimeFlow.py), na 0.
- SetStreetCount10.py - Nastavíme počet ulic, který bude vytvářen při postupu vpřed v čase (při spouštění skriptu TimeFlow.py), na 10.
- SetStreetCount20.py - Nastavíme počet ulic, který bude vytvářen při postupu vpřed v čase (při spouštění skriptu TimeFlow.py), na 20.
- SetCityReconstruction.py - Nastaví hodnotu globálního parametru stavby budov (City Reconstruction) na 0.4.
- UnsetCityReconstruction.py - Nastaví hodnotu globálního parametru stavby budov (City Reconstruction) na 0.0.
- SetGlobalDevelopment.py - Nastaví hodnotu globálního parametru rozvoje budov na 0.4.

Projekt dále obsahuje soubory CGA pravidel pro generování budov. Tyto soubory implementují tyto typy budov:

- Medieval.cga, MedievalInner.cga - Nízká středověká obytná budova.
- Older.cga, OlderInner.cga - Zachovalá starší budova se zdobnější fasádou.
- Modern.cga, ModernInner.cga - Moderní administrativní budova, která může být rozvinuta v mrakodrap.

Nyní si více popíšeme uvedeme účely jednotlivých tříd.

3.2.1 Popis tříd

Třídou, jejímiž metodami ovládáme časově proměnné modely, je třída TimeVaryingCity. Zde voláme metody pro vytvoření nového města, jeho rozvinutí v čase (postoupí o 1 časovou jednotku) a pro načtení modelu z jiné časové jednotky (skok v čase).

O práci s ulicemi a parcelami se stará třída GraphManager. Poskytuje všechny metody potřebné pro rozvoj města či jeho načítání při skoku v čase. Třída implementuje například následující úlohy:

- Ulice
  - Tvorba ulic dle předaných parametrů při přirozeném postupu v čase.
  - Rekonstrukce ulic při načítání modelu při skoku v čase. Také mazání nepotřebných ulic v aktuální scéně, které načítáný model neobsahuje.
  - Vyhledávání ulic a vrcholů ve scéně dle zadaných souřadnic.
  - Úprava rozdělení bloků parcel. Rozdělení bloku závisí na semínku pro generátor náhodných čísel.
- Parcely
  - Vytváření statických parcel.
  - Mazání zaniklých statických parcel (mohly zaniknout během postupu nebo skoku v čase).
  - Úprava detailů parcel při skoku v čase.

BuildingManager umožňuje práci s budovami v modelu města. Uchovává časy posledních změn budov. Dále implementuje tyto úlohy:

- Výběr a přiřazení souborů pravidel parcelám na základě časové jednotky.
- Změny existujících budov.
  - Zboření budovy a postavení nové budovy na jejím místě.
  - Rozvoj budovy.

- Výpočet pravděpodobností provedení těchto změn (základní pravděpodobnost, ovlivnění parametry předanými uživatelem, pravděpodobnost rozvoje podle ostatních budov v bloku).

ParameterStorage představuje úložiště parametrů předaných uživatelem:

- Globální a lokální parametry ovlivňující změnu existujících budov.
- Perioda změn budov.
- Počet generovaných ulic.
- Architektonické styly s časy jejich nástupů.
- Schémata pro generování ulic s časy jejich nástupů.

Dále uchovává:

- Násobky periody změn pro jednotlivé budovy.

Dále implementuje tyto úlohy:

- Ukládání a načítání parametrů.
- Zanáší prvek náhody do hodnoty času, kdy budou jednotlivé budovy změněny prostřednictvím násobků periody změn.

FileManager implementuje ukládání a načítání seznamů a map ze souborů.

3.3 Hlavní rysy implementace

V této části si zběžně nastíníme způsoby hlavních částí implementace.

3.3.1 Hlavní metody

V sekci 2.3 jsme si uvedli tři hlavní postupy pro práci s časově proměnným modelem. Tyto postupy a jim odpovídající metody třídy TimeVaryingCity jsou:

- Vytvoření nového města - generateNewCity(architecturalStyles, architecturalStylesInner, majorStreetPatterns, minorStreetPatterns)
- Přirozené plynutí času - proceedInTime()
- Skok v čase (načtení modelu města z jiné časové hodnoty) - skipToTime(desiredTime)

Parametry architecturalStyles a architecturalStylesInner jsou mapy architektonických stylů (dict). V mapách jsou klíči názvy souborů pravidel se styly a hodnotami časové jednotky nástupů příslušných stylů. Parametry majorStreetPatterns a minorStreetPatterns jsou mapy, kde jsou klíči schémata tvorby ulic. Hodnotami jsou opět časové jednotky nástupů. Parametr desiredTime je časová jednotka, ze které chceme načíst model.

3.3.2 Změny existujících budov a jejich rozvinutost

Při přirozeném plynutí v čase mohou být budovy města změněny (zbourány nebo rozšířeny). Metoda pro provedení změn je BuildingManager.manageBuildings(actualTime, parameterStorage), kde parametr actualTime je nastupující časová jednotka a parameterStorage je úložiště parametrů, instance třídy ParameterStorage.

Změny budov jsou vyhodnocovány po jednotlivých blocích. Postup je následující:

1. Pro každý blok jsou vyhledány hodnoty rozvinutosti jeho jednotlivých budov.
2. Na základě vypočtené pravděpodobnosti zbourání a náhody může být každá budova zbourána a na jejím místě ihned postavena nová.
3. Pokud budova nebyla zbourána, tak na základě vypočtené pravděpodobnosti rozvinutí může být budova rozvinuta.

Metoda pro zbourání a postavení nové budovy se jmenuje
BuildingManager.destroyAndBuild
(staticLot, actualTime, developmentList, parameterStorage). Parametr staticLot je statická parcela, na které budova stojí, actualTime je nastupující časová jednotka a developmentList je seznam hodnot rozvinutosti budov v bloku, kde se nachází zkoumaná budova.

Budova je zbourána a na jejím místě je postavena nová v aktuálním stavebním stylu. Hodnota její rozvinutosti odpovídá průměru hodnot rozvinutosti z developmentList, který je zaokrouhlen na celé číslo.

Metoda pro rozvoj budovy se jmenuje developBuilding(staticLot), kde parametr staticLot je statická parcela, na které budova stojí. Aktuální hodnota rozvinutosti budovy je načtena, zvýšena o 1 a nastavena budově.

Rozvinutost budovy je implementována jako atribut v souboru CGA pravidel, které spolu se vstupními atributy definují podobu budovy.

3.4 Omezení plynoucí z CityEngine

Při implementaci časově proměnných modelů v programu CityEngine se setkáváme s dvěma omezeními. Prvním je nutnost používání statických parcel pro tvorbu budovy. Důvod je ten, že pouze ze statických parcel jsme v Pythonu schopni přečíst hodnoty jejich atributů. Dynamické parcely toto neumožňují, a proto na nich vůbec nic nestavíme. Mají pouze tu funkci, že jsou vytvářeny automaticky s generováním ulic, a tak definují pozice pro statické parcely, které z nich generujeme a na kterých stavíme. Touto kombinací můžeme využít možnosti CityEngine vytvářet ulice a parcely rychle pomocí nástroje Grow Streets, ale jsme schopni stále ovládat domy na vznikajících parcelách.

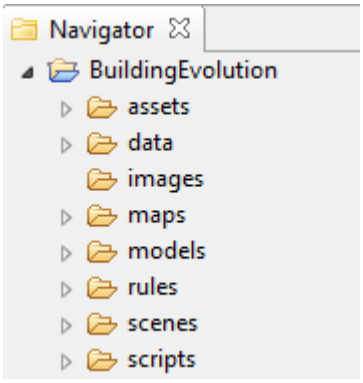
Dalším omezením je implementace souřadnic vrcholů v CityEngine. Tyto souřadnice můžeme přepisovat, jejich přesné hodnoty ale nejsou zcela pod naší kontrolou. Hodnoty souřadnic jsou vyčísleny na 15 desetinných míst, ne všechna tato místa ale jdou přepsat. Přepíšeme-li je, CityEngine někdy upraví hodnotu na tomto místě na jinou. Navíc pokud uložíme souřadnice nějakého objektu a vygenerujeme ho podle nich znovu, nemusí se nacházet na přesně stejných souřadnicích (CityEngine hodnotu jeho souřadnic opět upraví). Jelikož ukládáme spoustu hodnot jako jsou detaily parcel či souřadnice ulic do map, kde jsou klíčem právě souřadnice vrcholů, nelze pak tyto hodnoty jednoduše načíst. CityEngine nicméně provádí změny souřadnic jen na posledních desetinných místech. Proto hodnoty souřadnic zaokrouhlujeme na 4 desetinná místa a takto ukládáme. Když objekty vygenerujeme podle uložených souřadnic a hledáme jejich detaily v uložených hodnotách, zkoumáme pak odchylku souřadnic vygenerovaného objektu od zaokrouhlené uložené hodnoty a je-li tato odchylka dostatečně malá, pak rozhodneme o shodě. Odchylku považujeme za dostatečně malou, je-li menší než desetina rozdílu vygenerované a zaokrouhlené (uložené) souřadnice.

3.5 Import časově proměnných modelů do CityEngine

Nyní si popíšeme, jak importovat časově proměnné modely do CityEnigne.

Podíváme-li se na obrázek 18, vidíme strukturu CityEngine projektu s názvem BuildingEvolution.

Soubor s modulem implementujícím modely TimeVaryingCity.py umístíme do adresáře scripts. Sem můžeme umístit také skripty pro vykonání různých úloh, které jsme



Obrázek 18 Struktura projektu v CityEngine.

popsali v minulé sekci. Do složky rules umístíme soubory pravidel. Do složky assets/-facades dále musíme umístit objekty částí domů. Jsou jimi:

- OlderWindow.obj - Okno pro budovu dle Older.cga.
- WindowTop.obj - Prvek fasády nad oknem budovy dle Older.cga.
- Ledge.obj - Římsa dělící patra budovy dle Older.cga.
- ModernWindow.obj - Okno pro budovu dle Modern.cga.

Pokud chceme používat časově proměnné modely z modulu TimeVaryingCity, musíme dále tento modul importovat. To uděláme následujícím způsobem, nehledě na to, zda importujeme do vlastního skriptu či třídy či do konzole programu CityEngine:

$$\begin{aligned} &import\ sys \\ &sys.path.append(filePath) \\ &import\ TimeVaryingCity \end{aligned} \tag{20}$$

Přitom $filePath$ je cesta k souboru s modulem TimeVaryingCity. Po tomto importu je možné ve skriptu nebo v konzoli založit instance tříd v modulu a pracovat s nimi.

4 Výsledky

V této kapitole se podíváme, jaké výsledky tato práce přinesla či může přinést. Podle analýzy a návrhu jsem implementoval časová proměnná města, která může uživatel, který bude modely měst vytvářet, ovládat. Pro demonstrování výsledků jsem vytvořil tři rozdílná vzorová města. Jsou jimi:

- Město s historickým centrem (na okolí centra nebyly žádné požadavky).
- Město s budovami dle Older.cga (má simulovat město, kde převládá určitý architektonický styl budov).
- Město s moderními domy a mrakodrapy.

Tvorbu města, kde převažuje jeden architektonický styl, si popíšeme detailněji. Uvedeme si, jak se tato představa uživatele projevila v nastavování parametrů pro ovládání vývoje. Průběh tvorby města si popíšeme a ukážeme výsledek na obrázcích. Tvorbu ostatních měst již nebudeme popisovat tak detailně, ale opět si vše ukážeme na obrázcích. Na konci této kapitoly se ještě zamyslíme nad tím, jak tato práce pomáhá uživatelům při vytváření měst a jaké jsou rozdíly klasického 3D modelování proti použití této práce. Dodejme si, že jednotlivé architektonické styly budov jsou odlišeny barevně pro snadnější orientaci. Styl Mediaval je žlutý, Older modrý a Modern zelený.

4.1 Město s převládajícím architektonickým stylem

Představme si situaci, kdy chce uživatel vytvořit město do počítačové hry. Tato hra se přitom odehrává v konkrétním období v minulosti, například v renesanci. Uživatel proto chce vytvořit město, kde bude dominovat renesanční architektonický styl. Tato potřeba uživatele vede na vytvoření města s jedním dominantním architektonickým stylem, kterou si ukážeme. Uživatel požaduje město s jedním dominantním stylem, ale předpokládá, že ve městě zůstanou i některé starší domy v předchozích architektonických slozích, jak to bývá v realitě. Podívejme se nyní na to, jak uživatel toto město vytvoří s naší implementací v CityEngine. Předpokládejme, že uživatel zná parametry, kterými může tvorbu města ovládat.

Pro vytvoření prvotního stavu města je třeba definovat stavební slohy, které budou v modelu používány, s jejich časy nástupů. Dále musíme definovat schémata tvorby ulic s jejich časy nástupů. V našem příkladě jsme tuto definici provedli podle tabulky 3. Perioda změn budov byla 5 časových jednotek.

Typ parametru	Název	Čas nástupu
Architektonický styl	Medieval	1
	Older	10
Schémata stavby ulic	Hlavní ulice - křivolaký	1
	Vedlejší ulice - křivolaký	1
	Vedlejší ulice - pravoúhlý	10

Tabulka 3 Tabulka vstupních parametrů pro město s dominantním architektonickým stylem.

Město necháme generovat přirozeným postupem v čase. Během tohoto procesu do vývoje města zasahujeme těmito způsoby v těchto časech:

- 1 - Po vygenerování města nastavíme počet generovaných ulic na 20 (Původně bylo 50). Nechceme generovat ulice tak rychle.
- 7 - Počet generovaných ulic nastavíme na 10. Město dosud rostlo do šířky docela rychle. Chceme růst zpomalit.
- 8 - Počet generovaných ulic nastavíme na 0. Růst zpomalíme ještě víc.
- 10 - Nastavíme globální parametr stavby nových domů (City Reconstruction) na hodnotu 0,4. Tímto způsobem se budou staré budovy bourat a místo nich stavět nové, ne však příliš rychle. V tuto časovou jednotku již nastoupil architektonický styl Older, který bude naším dominantním. Parametr stavby nových domů zvýší pravděpodobnost zbourání existujících domů a postavení domů v tomto stylu.
- 14 - Počet generovaných ulic nastavíme na 20. Chceme vytvořit nové městské části ve stylu Older.
- 15 - Globální parametr stavby nových domů nastavíme na 0. Množství domů ve stylu Older nám už stačí a chceme zachovat také některé ve stylu Medieval.
- 16 - Nastavíme globální parametr rozvoje na 0,7. Chceme, aby se budovy ve městě trochu rychleji rozvíjely.
- 17 - Globální parametr rozvoje nastavíme na 0. Rozvoj nám stačí.
- 18 - Počet generovaných ulic nastavíme na 10. Chceme vygenerovat ještě nějaké nové budovy.
- 19 - V tomto čase končíme. S podobou města jsme spokojeni.

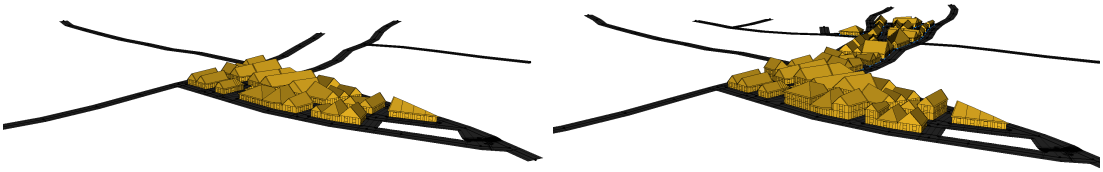
Takto námi definované parametry mohou posloužit našim cílům při tvorbě města. Rozvoj města s výše uvedeným způsobem ovládání je vidět na obrázku 19. Zde vidíme podoby města v uvedených časech a u každého podobrázku je napsáno, jaká změna se odehrála mezi touto a minulou podobou. Některé části výsledné podoby města si dále můžeme prohlédnout na obrázku 20.

Tímto způsobem jsme si vytvořili město, kde převládá architektonický styl Older, který je označený modrou barvou. Vidíme, že ve městě se vyskytují rozvinutější budovy (mají více pater) i méně rozvinuté (nižší) tohoto stylu. Dále vidíme, že ve městě zůstalo menší množství žlutých budov. Ty zůstaly ve městě proto, že během periody změn 5 časových jednotek nebyly nijak změněny. V tomto čase byla pravděpodobnost změny nejvyšší. Potom ale začala klesat, jelikož budovy začaly být považovány za zachovalé. Začala se na ně vztahovat památková péče a tak již nebyly změněny.

Na tomto příkladu vidíme, že můžeme definovat časy nástupů jednotlivých stylů a můžeme ovládat počet generovaných ulic. Použili jsme také globální parametr stavby nových budov, díky kterému se nám podařilo dosáhnout rychlejšího bourání starších (Medieval, žlutých) budov a jejich nahrazení novějšími (Older, modré). Když jsme potřebovali, aby se budovy ve městě trochu rozvinuly (byla jim přistavěna patra), použili jsme globální parametr rozvoje. Vidíme, že jsme tedy schopni do jisté míry ovládat určité aspekty modelování města v čase, zatímco program CityEngine se stará o ostatní věci jako je pozice stavěných ulic či vygenerování budov.

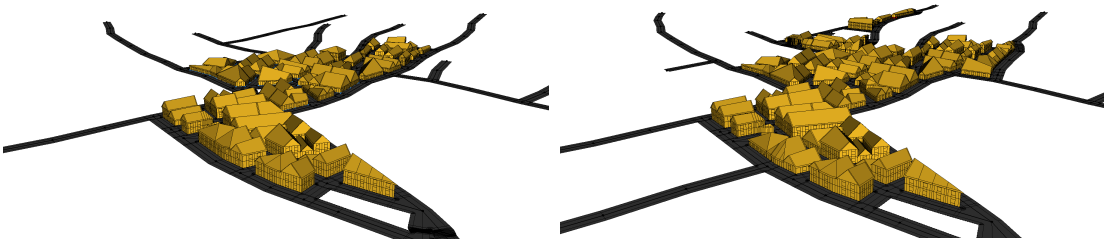
4.2 Město s historickým centrem

Představme si, že chceme vytvořit město, kde se nám nějaké budovy budou líbit natolik že je budeme chtít zachovat. Tyto budovy se nám mohou líbit například proto, že jsou postaveny v dříve užívaných architektonických slozích, které se dnes již nepoužívají. Jak zachovat oblíbené budovy si uvedeme v tomto příkladě a vyzkoušíme tak funkčnost



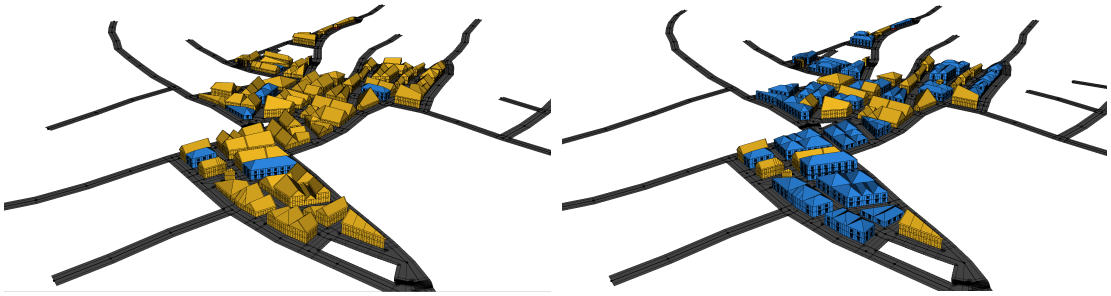
a) 1 - Vygenerování nového města.

b) 3 - Přibyla nová část města.



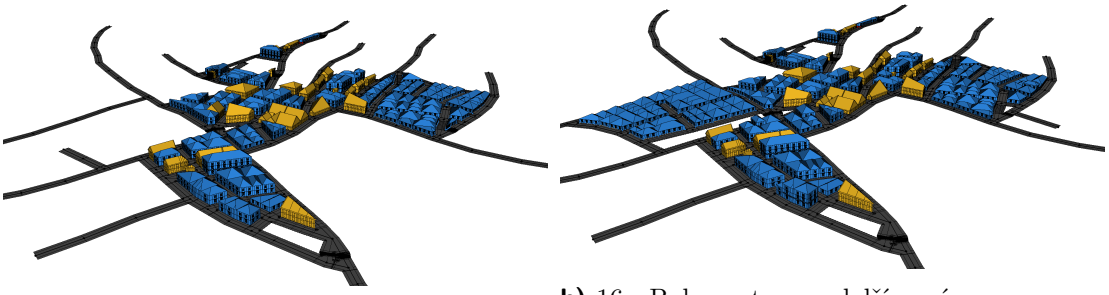
c) 6 - Byla postavena další nová část.

d) 9 - Vidíme rozvoj budov ve městě.



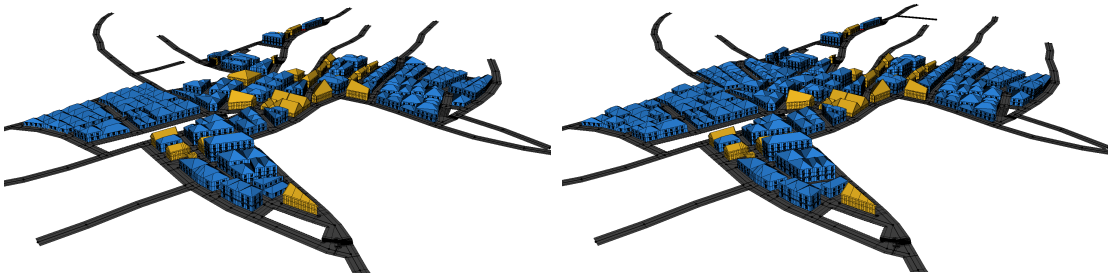
e) 10 - Začínají se objevovat první domy ve stylu Older.

f) 14 - Staré budovy se bourají a staví se nové.



g) 15 - Byla postavena čtvrť nových domů.

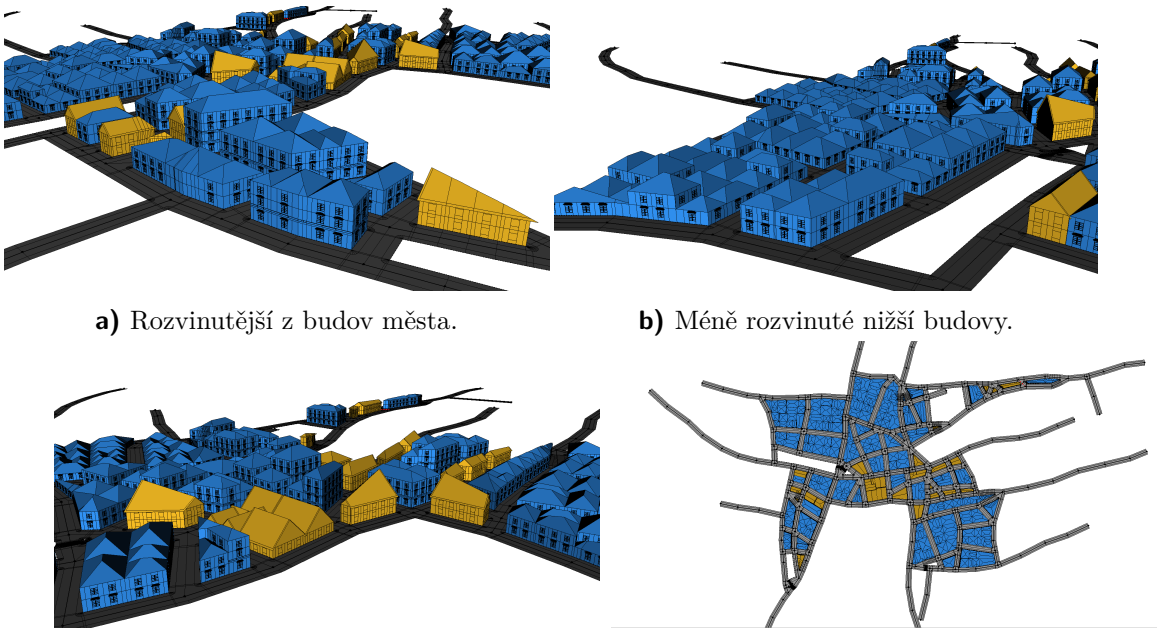
h) 16 - Byla postavena další nová čtvrť.



i) 17 - Urychlili jsme rozvoj budov.

j) 19 - Finální podoba města.

Obrázek 19 Průběh vývoje města.



a) Rozvinutější z budov města.

b) Méně rozvinuté nižší budovy.

c) Střed města se starými i novými budovami.

d) Půdorys města.

Obrázek 20 Výsledky vývoje města s dominantním architektonickm stylem.

Typ parametru	Název	Čas nástupu
Architektonický styl	Medieval	1
	Older	5
	Modern	10
Schémata stavby ulic	Hlavní ulice - křivolaký	1
	Hlavní ulice - pravoúhlý	10
	Vedlejší ulice - křivolaký	1
	Vedlejší ulice - pravoúhlý	10

Tabulka 4 Tabulka vstupních parametrů pro město s historickým centrem.

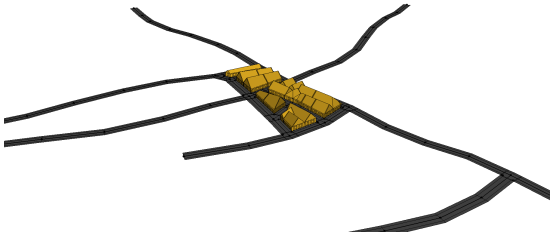
dalšího z parametrů. Předpokládejme, že tyto budovy se nachází v centru. Nastavení vstupních parametrů provedeme podle tabulky 4. Perioda změn bude opět 5 časových jednotek.

V tomto příkladě nemusíme do vývoje města příliš zasahovat. Soustředíme se pouze na budovy vybrané námi pro zachování, které se budou nacházet v centru města. Postup ovlivnění města během vývoje bude tento:

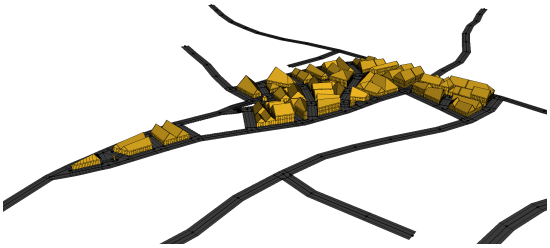
- 1 - Po vygenerování města nastavíme počet generovaných ulic na 20, abychom nové části města generovali pomaleji.
- 4 - Vybereme budovy, které chceme zachovat (v tuto chvíli všechny ve scéně) a nastavíme jim lokální parametr zachování (Local Preservation) na 1.0.
- 14 Končíme modelování, s městem jsme spokojeni.

Průběh vývoje města můžeme sledovat na obrázku 21. U každého z podobrázků vidíme časovou jednotku pořízení a popis změny, která je na podobrázku patrná a která se odehrála mezi tímto a minulým podobrázkem. Detaily výsledků vidíme na obrázku 22. Můžeme zde vidět, že ve městě najdeme jak moderní vyšší budovy (Modern, zelené), tak starší budovy (Older, modré), ale také historické jádro (Medieval, žluté).

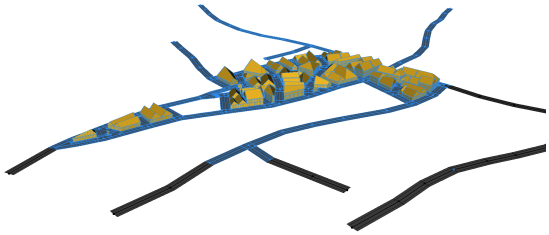
Vidíme tedy, že chceme-li, můžeme některé z budov města cíleně zachovat nasta-



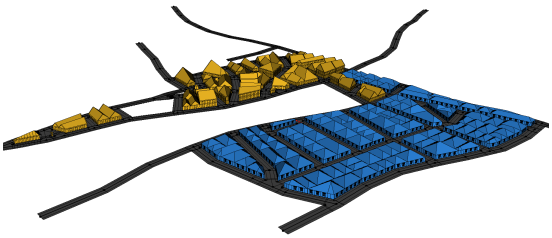
a) 1 - Vytvoření prvních budov ve městě.



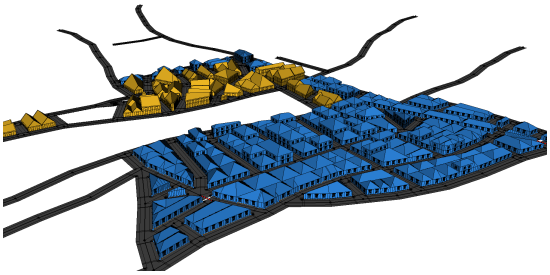
b) 4 - Byly postaveny nové domy.



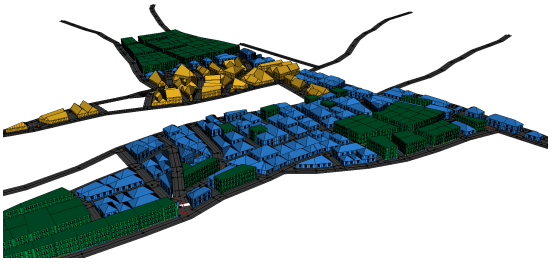
c) 4 - Vybereme budovy, které chceme zachovat.



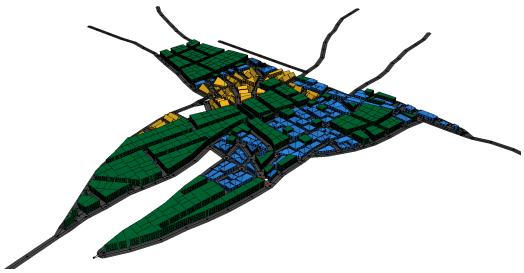
d) 5 - Byla postavena nová čtvrť.



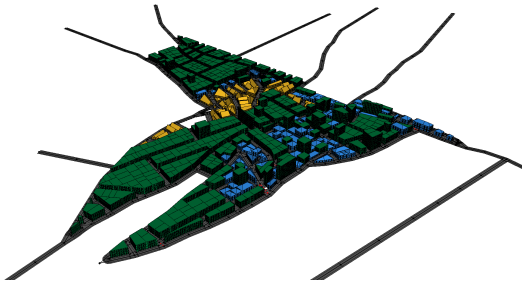
e) 8 - Některé budovy nové čtvrti byly rozvinuty.



f) 10 - Nastupuje moderní styl, centrum zůstává stejné.

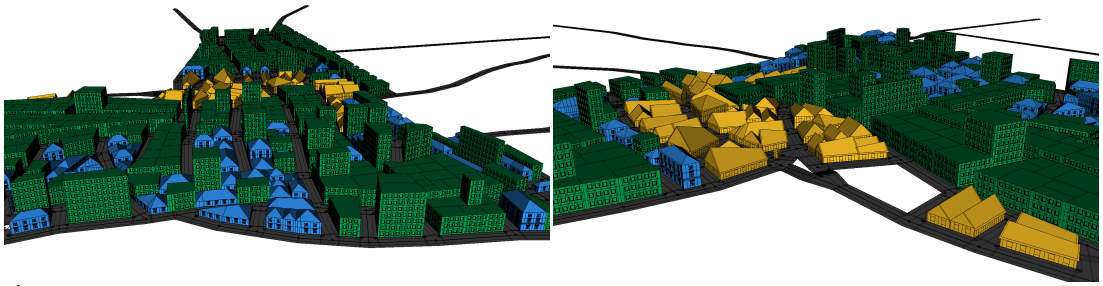


g) 11 - Nové části města jsou stavěny v moderním stylu.



h) 14 - Finální podoba města - historické centrum zůstalo zachováno.

Obrázek 21 Průběh vývoje města, kde zachováváme historické centrum.



a) Ve městě jsou přítomny jak moderní vyšší budovy, tak historické centrum.

b) Původní historické centrum.

Obrázek 22 Výsledky vývoje města s historickým centrem.

Typ parametru	Název	Čas nástupu
Architektonický styl	Modern	1
Schémata stavby ulic	Hlavní ulice - pravoúhlý	1
	Vedlejší ulice - křivolaký	1
	Vedlejší ulice - pravoúhlý	5

Tabulka 5 Tabulka vstupních parametrů pro moderní město s mrakodrapy.

vením lokálního parametru zachování. To je další možnost, jak ovládat vývoj námi modelovaných měst.

4.3 Moderní město

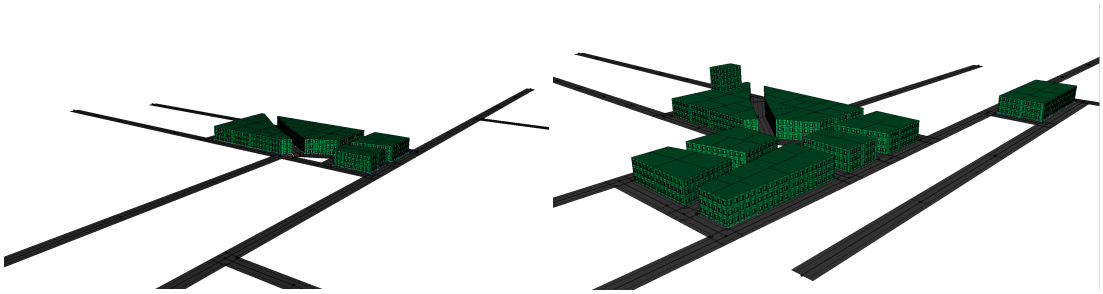
Posledním příkladem, který si ukážeme, je vytvoření moderního města s mrakodrapy. Tento scénář může uživatel využít tehdy, když si chce vygenerovat fiktivní město se současnými prvky architektury. Uživatel začne definicí architektonických stylů a schémat stavby ulic. Příklad těchto definic vidíme v tabulce 5. Perioda změny budov bude opět 5 časových jednotek.

Uživatel bude během přirozeného plynutí času do vývoje města zasahovat tímto způsobem:

- 1 - Po vygenerování města nastaví počet generovaných ulic na 20. Růst města do šířky bude pomalejší.
- 3 - Nastaví počet generovaných ulic na 10, aby ještě více zpomalil růst města do šířky. V této časové jednotce navíc nastaví globální parametr rozvoje na hodnotu 0,4, aby podpořil rozvoj budov. Ten se projeví přístavbou pater.
- 11 - Uživatel je s městem spokojen a ukončuje modelování.

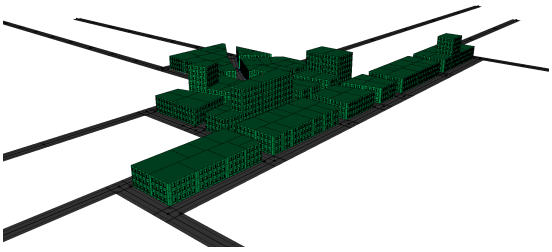
Kromě těchto zásahů nebude uživatel do vývoje města zasahovat a nechá CityEngine, aby pro něj definoval zbytek z podoby města. S těmito parametry jsem i já vygeneroval město, které si ukážeme na obrázku 23. Na podobrázcích vidíme uvedeny časy, kdy byly tyto podobrázky pořízeny a významné změny v podobě města, které ilustrují. Detaily vytvořeného modelu vidíme na obrázku 24. Zde můžeme vidět, že ve městě vznikly jak mrakodrapy, tak i nižší budovy.

Na tomto příkladě jsme demonstrovali možnost generovat specifické moderní město. Definovali schéma tvorby jeho ulic, které bylo hlavně pravoúhlé. Dále jsme zvýšili pravděpodobnost rozvoje budov, takže ve městě začaly vznikat mrakodrapy.

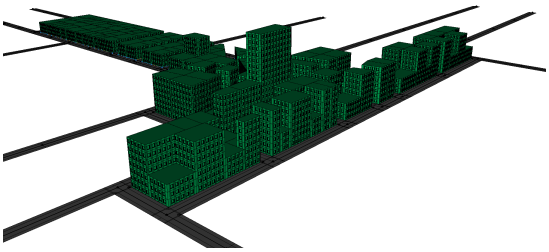


a) 1 - Vytvoření města.

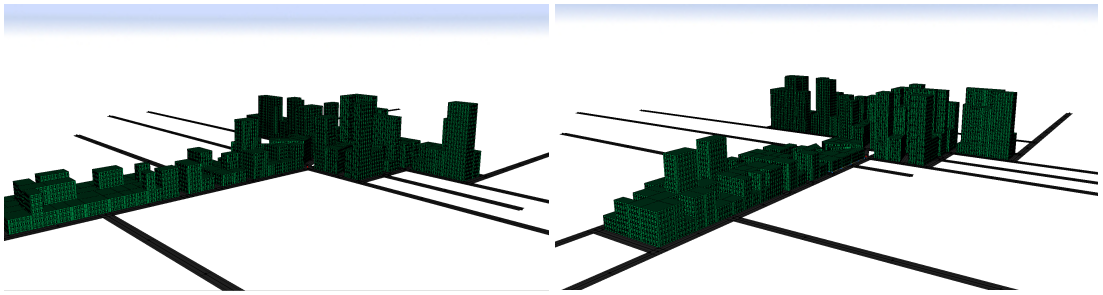
b) 3 - Byly postaveny další domy.



c) 5 - Byly postaveny další domy a předchozí byly rozvinuty.



d) 7 - Byla postavena nová čtvrť.



e) 9 - Další fáze rozvoje budov.

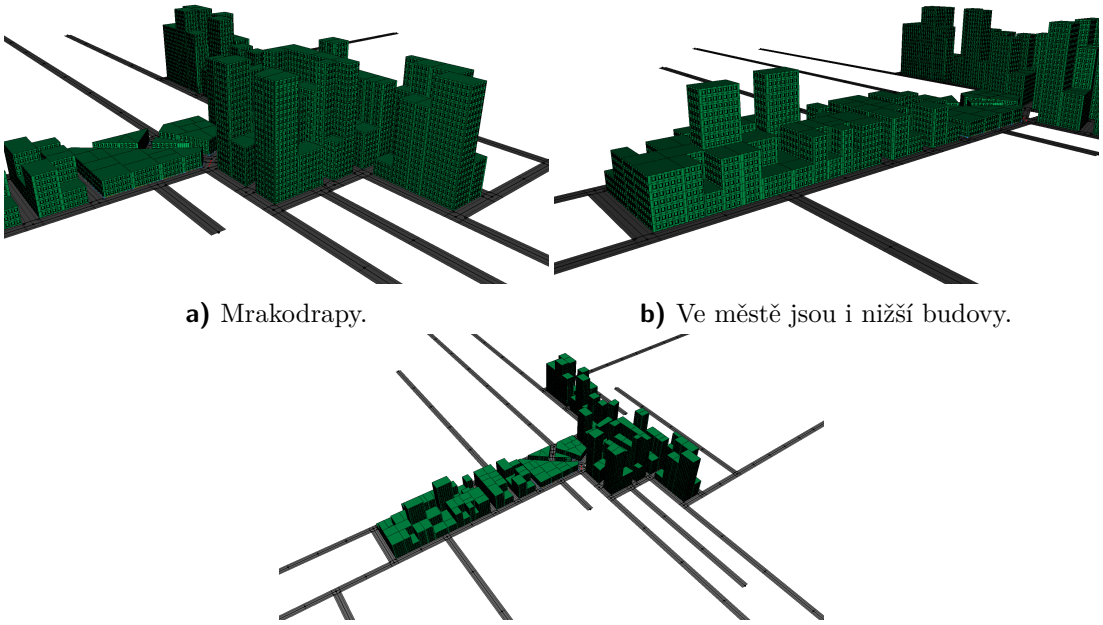
f) 11 - Konečná podoba města.

Obrázek 23 Průběh vývoje moderního města.

4.4 Celkové zhodnocení výsledků

Vidíme, že se nám podařilo implementovat simulaci vývoje měst. Tu můžeme využít pro vytváření měst a jejich vývoj můžeme ovládat pomocí vstupních parametrů. Funkčnost modelů jsme si demonstrovali na třech vzorových městech, které mohou být inspirací k vytváření dalších. Vidíme, že předáním parametrů můžeme ovlivnit generování ulic, podobu vznikajících budov a změny těch stávajících. Můžeme tak vytvořit města s rozličnými charakteristikami.

Porovnáme-li tvorbu měst vyvíjejících se v čase 3D modelováním a modelováním pomocí časově proměnných modelů implementovaných v této práci, můžeme prohlásit, že zde představovaná technika může být rychlejší. Značně rychlejší je hlavně generování ulic. Definici jejich pozic přitom necháváme výhradně na CityEngine, sami ale řídíme počet a schéma pro generování ulic. Manuální vytváření ulic by bylo pomalejší. Také modelování budov zpravidla bude rychlejší. Nemusíme totiž modelovat celou budovu, ale pouze její části jako jsou okna či prvky fasády. Poté využijeme vložení těchto prvků do modelů budov vytvořených podle námi definovaných pravidel CGA gramatiky a podle vstupních parametrů. Již existující budovy implementují rozhodování o jejich změně na základě jejich aktuální kondice (stáří poslední změny). Pravděpodobnost změny budovy



a) Mrakodrapy.

b) Ve městě jsou i nižší budovy.

c) Pohled na město shora.

Obrázek 24 Výsledky vývoje moderního města.

se určí na základě několika faktorů jako je stáří poslední změny, rozvinutost ostatních budov v bloku, kde se budova nachází, a ovlivnění uživatelem.

Výhodou oproti 3D modelování je fakt, že geometrické modelování za nás provádí CityEngine, my ho ale můžeme ovládat pomocí parametrů. To je rychlejší i jednodušší. Nevýhodou může být fakt, že nemáme absolutní kontrolu nad modelem. Z toho plyne například ta nevýhoda, že za stávající implementace nemůžeme zabránit CityEngine v tendenci vytvářet nové bloky parcel s protáhlými podlouhlými tvary.

5 Závěr

Cílem práce bylo analyzovat možnosti pro vytvoření modelů měst vyvíjejících se v čase a vytvořit aplikační logiku, která by umožňovala uživatelům takové modely vytvořit. Nejprve jsem analyzoval techniky procedurálního modelování využitelné pro modelování měst. Poté byl analyzován CityEngine, program pro procedurální modelování měst. Následovala analýza vývoje měst v čase a změn, které při tomto vývoji můžeme pozorovat. Získané znalosti byly využity k návrhu postupů, jak modelovat časově proměnné modely měst a také jak ovládat časový vývoj modelovaných měst. Na základě tohoto návrhu byla provedena implementace v CityEngine. Výsledky byly demonstrovány na modelech tří rozdílných měst vytvořených implementovanou logikou.

K ovládání vývoje měst byly navrženy parametry, u kterých byl kladen důraz na jejich smysluplnost, jednoduchost a nízký počet. Činnost parametrů byla popsána a demonstrována na vzorových příkladech. Byl navržen způsob práce s již postavenými budovami v modelu, který popisuje možnosti změn budov na základě jejich akutálního stavu (stáří posledních změn), zahrnuje interakci s okolními budovami v bloku a také ovládání uživatelem. Dále byly vytvořeny tři jednoduché vzorové budovy pro demonstraci různých architektonických stylů, bourání, stavby a rozvoje budov.

Příloha A

Obsah CD

Obsah přiloženého CD je rozdělen do následujících adresářů:

- Adresář **text** obsahuje text této práce a dále adresář se zdrojovými kódy textu v Latex.
- Adresář **src** obsahuje adresáře s Pythonovými soubory s implementací práce (adresář Python), soubory .cga s pravidly tří vzorových budov (RuleFiles) a 3D modely použité v implementaci těchto budov (Models).
- Adresář **sampleCities** obsahuje použité obrázky dokumentující vývoj měst rozdělené do tří adresářů podle těchto měst.
- Adresář **cityEngine** obsahuje instalační soubor programu CityEngine ve verzi, pro kterou byla provedena implementace (2011.2).

Literatura

[1] CATMULL, Edwin; CLARK, James. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 1978, 10.6: 350-355.

[2] HONDA, Masanobu, et al. Generating autonomous time-varying virtual cities. In: *Cyberworlds, 2004 International Conference on.* IEEE, 2004. p. 45-52.

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

[4] MELICHAR, Bořivoj. *Jazyky a překlady.* Vydavatelství ČVUT, 2003.

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

[6] Wonka, Peter; Wimmer, Michael; Sillion, Francois X; Ribarsky, William. *Instant Architecture.* ACM Transactions on Graphics, ACM, 2003, 22 (4), pp.669 – 677.