UNIVERZITA KARLOVA V PRAZE

Přírodovědecká fakulta

Katedra aplikované geoinformatiky a kartografie

Studijní program: Geografie
Studijní obor: Kartografie a geoinformatika

Bc. Ivo Brýdl

Automatická generalizace 3D modelů budov

Automatic generalization of 3D building models

Diplomová práce

Vedoucí diplomové práce: Mgr. Lukáš Brůha, Ph.D.

Praha 2017

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 26. 7. 2017 Podpis autora

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce Mgr. Lukáši Brůhovi, Ph.D. za věnovaný čas, cenné konzultace, rady a připomínky. Velký dík patří také mé rodině a přátelům za podporu v průběhu celého studia.

Abstrakt

Hlavním cílem této práce je navrhnout metodu generalizace 3D modelů budov s ohledem na topologii v 3D prostoru. První část představuje přehled prostorových datových reprezentací a metod generalizace 3D objektů. Druhá část popisuje navrženou metodu pro vytvoření více-měřítkového modelu, složenou z rekonstrukce objemových modelů, kontroly topologie a zjednodušení geometrie objektů založené na konceptu matematické morfologie. Metoda je následně implementována a jsou představeny experimentální výsledky. Na závěr jsou diskutovány výsledky práce a návrhy na možné vylepšení.

Klíčová slova: 3D GIS, 3D modely budov, generalizace

Abstract

The main aim of this thesis is to propose a novel method for generalization of 3D building models with respect to topology in 3D space. The first part gives an overview over spatial data structures and methods for generalization of 3D objects. The second part presents the methodology composed of volumetric model reconstruction, topology validation and simplification of geometry based on mathematical morphology approach to build multi scale model. Method is then implemented and experimental results are presented. In conclusion results are discussed and possible improvements are suggested.

Keywords: 3D GIS, 3D building models, generalization

Obsah

1. Úvod

Jednou z hlavních úloh výzkumu na poli 3D geoinformačních technologií je automatická generalizace trojrozměrných modelů měst, tvořených rozličnými geografickými objekty. Ačkoli je výzkum v této oblasti značně pokročilý, stále chybí vyhovující metoda, která by 3D generalizaci spolehlivě řešila i s ohledem na interakci s okolními objekty a dohromady vytvořila koncept více-měřítkového modelu. Mnoho úsilí bylo věnováno tvorbě postupů s cílem co nejlépe globálně zjednodušit modelované geografické objekty, a tím usnadnit jejich vykreslování při dynamické vizualizaci scény. Takové metody byly zpočátku vyvíjené pouze v oboru počítačové grafiky, ale dnes se již prolínají i s digitální kartografií. Na druhé straně existují typické GIS přístupy, které více než charakter zobrazení řeší generalizaci ve vztahu k okolí, tedy jak generalizace v určitém místě ovlivní místa jiná. Takto komplexní model však vyžaduje použití vhodné datové reprezentace, která vzájemné vztahy umožní zjišťovat a spravovat.

Datové modely měst a na nich prováděná generalizace se odlišují dimenzí dat a také dimenzí prostoru, ve kterém jsou tato data modelována. Dlouhou dobu přetrvávající koncept klasického 2.5D prostoru je postupně nahrazován rozšířenými modely, které umožňují reprezentaci komplexnějších objektů. S dimenzionalitou prostoru pak roste nejenom složitost metod, ale také datová náročnost a potřeba vyspělých datových reprezentací. Z těchto důvodů jsou metody plně podporující 3D modelování těles v 3D prostoru pro aplikaci v GIS stále do jisté míry nerozvinuté.

V následujících kapitolách této práce je provedena rešerše konceptů prostorových a topologických modelů doplněná o existující metody 3D generalizace. Následuje návrh vlastní metody pro dynamickou generalizaci 3D modelů budov. V další části jsou diskutovány dostupné datové zdroje, zjištěny jejich nedostatky a je představen návrh, jak z dat zrekonstruovat objemové modely. Navržená metoda bude implementována jako automatický proces, který bude následně experimentálně ověřen aplikací na reálná data. Poslední dvě kapitoly jsou pak věnovány diskuzi a závěru.

Tato práce si klade za cíl zkombinovat dostupné metody a znalosti pro modelování 3D GIS dat a navrhnout metodu 3D generalizace, která automaticky provede

zjednodušení 3D modelů budov. Metoda využije vhodných datových struktur, které umožní modelování 3D objektů v trojrozměrném prostoru a zjišťování topologických vztahů mezi těmito objekty. Aby mohlo být dosaženo vytyčeného cíle, bude potřeba splnit následující úkoly:

- Nalézt vhodnou datovou strukturu pro ukládání a práci s 3D modely budov, která umožní využití v GIS analýzách a zároveň bude vhodná pro účely vizualizace.
- Provést rešerši problematiky generalizace 3D modelů a vybrat vhodnou metodu pro generalizaci budov.
- Přizpůsobit tuto metodu zvolené datové struktuře a aplikovat metodu na 3D modely budov.
- Rozšířit metodu o kontrolu topologie ve 3D prostoru.
- Implementovat navrženou metodu, vytvořit funkční prototyp a otestovat řešení na vybraných datech.

2. Problematika modelování v GIS

Tato kapitola uvádí přehled základních definic a principů modelování těles v 3D prostoru a představuje datové reprezentace, které se při modelování používají. Uvedené koncepty pak slouží jako teoretický rámec pro navrhovanou metodu.

2.1 Dimenzionalita modelu

Proces modelování je základním konceptem, který je využívaný v metodách digitální kartografie a geoinformačních systémech. Modelování vždy znamená pokus o zachycení reality, která je ovšem mnohodimenzionální. Ačkoliv si v běžném životě lidé často tuto skutečnost neuvědomují, život lidstva se pohybuje v prostoru. Ve zjednodušeném pojetí si člověk vystačí se zobrazením v dvojrozměrném prostoru, tedy v rovině, kdy vidí před sebe, do stran a případně i za sebe. Pokud se však uvažuje nad modelem, který by popsal člověka a jeho prostředí, je nutné vzít v potaz i ostatní, na první pohled zanedbatelné dimenze. Pro úspěšné popsání geografického prostředí je však potřeba minimálně trojrozměrného prostoru, kde třetím rozměrem je výška. Obecnější modely mohou být rozšířené i o další dimenze, jako je například čas nebo měřítko. Vytvořit datový model, který by snadno pokryl alespoň výše zmíněné dimenze, je proto tak komplexní problém, že se v tradičních geoinformačních systémech volí postup zjednodušování modelů tak, že využívají pouze vybrané a předem určené dimenze.

Kartografie historicky začínala mapováním okolí a vytvářením map. Mapa musela obsáhnout co možná nejvíce prostorových informací na poměrně malé ploše. Je zřejmé, že tímto způsobem není možné zaznamenat kompletní reálný svět. V tu chvíli přichází na řadu zjednodušování kresby, vynechání nepodstatných informací, což se dnes nazývá termínem generalizace. Pomocí digitálních technologií je v současné době možné zaznamenat mnohonásobně více dat ve větším měřítku než v minulosti, ale i přesto jsou datové kapacity a výpočetní možnosti svým způsobem omezené, a tak má generalizace i dnes své opodstatnění (Dong et al.,

2001).

Přestože se o dimenzi obecně mluví jako o dimenzi prostoru ve třech směrech, mělo by se rozlišovat základní rozdělení mezi dimenzí vnitřní a vnější. Vnitřní dimenze se vztahuje k modelovanému objektu a znamená nejvyšší dimenzi, kterou objekt představuje. Například obrys hranice státu v mapě je nakreslen v rovině a proto má pouze dvě dimenze, zatímco model výškové budovy má dimenze již tři. Oproti tomu vnější dimenze charakterizuje dimenzi prostoru, ve kterém se dané objekty modelují. V n-rozměrném prostoru se tedy mohou modelovat objekty se stejnou nebo nižší dimenzí, nicméně opačně to možné není (Pilouk, 2006).

Na základě definic pro vnější a vnitřní dimenzi modelovaného objektu a prostoru je možné vytvořit konceptuální modely reálného světa, které Pilouk (Pilouk, 2006) rozděluje na:

2D model se pohybuje pouze v rovině. Je to intuitivní představa mapy, kde je topografický povrch projektován do roviny. V tomto konceptu je možné modelovat pouze bod, linii nebo polygon, protože maximální dimenzionalita prostoru je dvojrozměrná.

2,5D model se již posouvá částečně do trojrozměrného prostoru, ale přesto obsahuje prvky s nižší dimenzí, což celý model oproti 3D výrazně zjednodušuje. Typicky se takto modeluje terén, kdy se ke každé dvojici souřadnic (x,y) může přiřadit maximálně jedna výšková souřadnice. Známou technikou je modelování trojúhelníkových sítí (TIN). Kvůli svému omezení však tento model není schopný zachytit svislé prvky (Kolingerová, Žalik, 2006).

2,5D+ model rozšiřuje předešlý systém o možnost zachytit vertikální objekty, jako jsou obvodové zdi. Je tedy možné takto vytvořit pseudo-3D model města, který nebude objemový, nýbrž bude složený pouze z plošek. Pro vizualizační účely to však může být dostačující.

3D model je jediným konceptem, který podporuje primitivum neexistující v předchozích modelech s nižší dimenzí. Tímto primitivem je těleso, tedy modelování objemu. Na první pohled se přechod zdá snadným, ovšem z hlediska výpočetní geometrie se jedná o velkou změnu. V tomto modelu již mohou existovat takzvané vodotěsné objekty, tedy modely s plně uzavřeným povrchem nebo také objemové modely (Pilouk, 2006).

Cílem této práce je navrhnout metodu generalizace, která bude plně podporovat 3D prostor. Z toho důvodu je koncept 3D jediným možným použitelným modelem. Ostatní uvedené modely totiž nepodporují modelování objemových těles, které jsou pro tuto práci nezbytné.

2.2 Datové reprezentace ve 3D

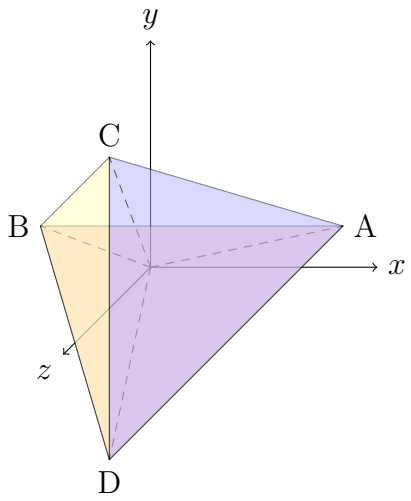
Metody, které slouží k ukládání strukturovaných informací, se obecně označují jako datové reprezentace. V počítačových systémech existují datové reprezentace pro ukládání různých datových typů, jako jsou čísla nebo text. V oblasti 3D GIS nebo obecně 3D modelování existují specifické datové reprezentace pro popis a uchování informací o modelovaných 3D objektech. Tyto datové reprezentace se liší mírou své složitosti, z čehož plynou výhody a nevýhody jejich použití. Na jedné straně existují datové struktury jednoduché, které ukládají tvar a polohu základních geometrických objektů, na straně druhé pak jsou struktury složitější, které v sobě zahrnují i informace o vztazích mezi objekty, které modelují.

Mezi jednodušší datové struktury patří vektorová reprezentace geometrických objektů metodou CSG (*Constructive Solid Geometry*). Tato metoda spočívá ve vzájemném spojení základních geometrických těles pro modelování tvarově složitějších objektů. Kombinací základních geometrických těles, jako je například krychle, kvádr, válec nebo koule, pomocí booleovských operací[1] je možné modelovat různé tvarově složitější objekty. Tato metoda se využívá hlavně ve strojírenství nebo v průmyslu. Ze své podstaty se CSG reprezentace nehodí pro modelování komplexních geografických 3D objektů, protože je úzce zaměřená na konstrukci součástek a nepodporuje potřeby GIS, tedy například modelování vztahů (Tuan et al., 2013).

Blíže GIS jsou metody objemového modelování, které tělesa popisují soustavou geometrických primitiv, tedy n-úhelníky nebo mnohostěny. První metodou je objemová dekompozice objektu na buňky. Druhou možností je mnohostěn (*polyhedron*), jehož povrch tvoří souvislé navazující plošky, které dohromady modelují uzavřené těleso (Lattuada, 2005).

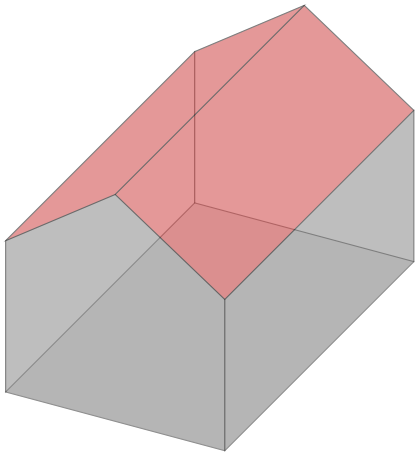
Dekompozice na čtyřstěny je technikou obdobnou z 2,5D modelu, kdy se povrch dělí na trojúhelníkové sítě, které dokáží dobře popsat terénní strukturu. Vzhledem k plné podpoře třetí dimenze zde však dělení neprobíhá na trojúhelníky, ale na čtyřstěny. Jedná se tak o analogii 2D triangulace ve 3D, pro kterou je

[1] Booleovská algebra definuje množinové operace průnik, sjednocení a doplněk označované jako booleovské operace.



Obrázek 2.1: Čtyřstěn. Zdroj: vlastní tvorba.

lepší použít anglický termín *tetrahedronisation* („tetrahedronizace“). Základní 3D buňkou této struktury je těleso zvané čtyřstěn (*tetrahedron*, obr. 2.1), označovaný také jako 3-simplex. Tento 3-simplex je tvořen simplexy s nižší dimenzí, které se také nazývají primitiva. Čtyřstěn je tedy složen ze čtyř vrcholů (0-simplexy), šesti hran (1-simplexy), čtyř stěn (2-simplexy) a jednoho objemu (3-simplex). Kvůli dekompozici na čtyřstěny je tato struktura velmi náročná na objem ukládaných dat, což zvyšuje případnou výpočetní náročnost nad takovým modelem. Zároveň oproti triangulaci jsou tetrahedronizační algoritmy výrazně složitější. Tato vyloženě objemová reprezentace nalezne své uplatnění např. v geologických modelech, kde je objem primární modelovanou doménou, zatímco pro potřeby 3D GIS je příliš komplexní (Tuan et al., 2013).



Obrázek 2.2: Mnohostěn, respektive polyhedron. Zdroj: vlastní tvorba.

Plošková reprezentace umožňuje modelovat těleso zvané mnohostěn. Stejně jako hranice polygonu určují plochu, tak povrch polyhedronu určuje těleso s určitým objemem. Polyhedron (obr. 2.1) je složen z ploch, které navzájem sdílí

své hrany s protějšími plochami. Podle Aguilera (1998) musí validní polyhedron splňovat tyto kritéria:

- **Planarita**

  Každá plocha ohraničující polyhedron musí být tvořena body, které leží v jedné rovině. Tři body určují rovnici roviny, platí tedy, že $Ax + By + Cz + D = 0$. Pro čtvrtý až n-tý bod se kontroluje vzdálenost bodu P od roviny určené prvními třemi body. Vzdálenost D musí být nulová (Hazewinkel, 2000).

  $$D = \frac{aP_x + bP_y + cP_z + d}{\sqrt{a^2 + b^2 + c^2}} = 0$$

- **Souvislost povrchu**
  Souvislý povrch je nepřerušená hranice tělesa. Vlastnost souvislosti tedy znamená, že objem polyhedronu je uzavřený. Hrana polyhedronu je sdílena právě dvěma přilehlými plochami. Pokud je hrana sdílena pouze jednou plochou, povrch není uzavřený. Naopak, pokud je hrana sdílena více než třemi plochami, znamená to, že jsou modelovány alespoň dva objemy nebo dokonce non-manifold geometrie.

- **Struktura primitiv**
  Plochy polyhedronu musí být tvořeny nižšími primitivy. Hrany jsou tvořeny právě dvěma vrcholy.

- **Orientace**
  Povrch polyhedronu musí být orientovaný. Orientaci povrchu určují směry normál ploch. Orientované normály musí směřovat vně polyhedronu.

Z porovnání datových struktur objemové dekompozice na čtyřstěny a plošková reprezentace polyhedronu je zřejmé, že při modelování budov v 3D GIS není nutné provádět dekompozici na čtyřstěny. To by se hodilo pouze v případě, kdy by zájmovou oblastí byl také vnitřek budov a jednotlivé buňky by reprezentovaly například interiér budov. Polyhedronová struktura je vhodnější, má menší paměťové nároky a pro modelování 3D GIS objektů je dostačující.

2.3 Prostorové prohledávání

Při zpracování velkého objemu prostorových dat dojde k situaci, kdy je prohledávání datových struktur časově velmi náročné a zpomaluje celý model. V tu chvíli je na místě používat techniky prostorového prohledávání dat, které jsou

	polyhedron	**dekompozice na čtyřstěny**
0-simplex	10 vrcholů	10 vrcholů
1-simplex	15 hran	25 hran
2-simplex	7 ploch	24 ploch (trojúhelníků)
3-simplex	1 objem	8 objemů (čtyřstěnů)

Tabulka 2.1: Porovnání požadavků na počet uložených primitiv mezi datovou strukturou polyhedronu a dekompozicí na čtyřstěny, při modelování budovy na obr. 2.2. Převzato z Penninga (2008).

schopné v krátkém čase odpovědět na prostorový dotaz k nalezení dat podle jejich polohy v prostoru. Tyto techniky prohledávání dat se souhrnně označují jako *prostorové indexy*. V klasických relačních databázích se již běžně používá metod indexace, které umožní databázovému systému rychlý přístup k datům. Při práci s prostorovými daty v GIS je situace o to složitější, že se nejedná o diskrétně kategorizovaná data, ale o polohové údaje pokrývající velké číselné rozsahy ve více dimenzích. Prostorový index umožní modelovaný prostor rozdělit na podprostory, které jsou pak zaznamenány prostorovými indexy do specifických stromových struktur (Kothuri et al., 2002).

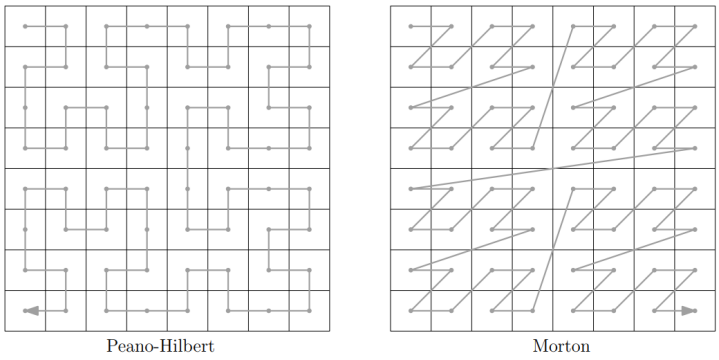
Stromy Pro dekompozici prostoru na podprostory se často používají stromy, které jsou speciálním případem grafu. Jakožto základní objekty matematické teorie grafů se obecně využívají pro reprezentaci vztahů. Grafy jsou vytvářeny tak, že objektům jsou přiřazeny *vrcholy* a jejich propojení je realizováno *hranami*. Hrana mezi vrcholy může být orientovaná nebo neorientovaná, u orientovaných hran poté rozlišujeme počáteční a koncový vrchol hrany. Hrana může vycházet a končit ve stejném vrcholu, potom se nazývá smyčkou. U neorientovaného grafu nerozlišujeme pořadí vrcholů v každé dvojici, u orientovaných grafů je pořadí udávající orientaci podstatné. Graf je v matematice definován jako trojice $G = (V,E,\epsilon)$, kde V je neprázdná množina vrcholů, E je množina hran a ϵ je zobrazení (neboli vztah incidence), které ke každé hraně $e \in E$ přiřazuje jeden nebo dva vrcholy (Demel, 2002).

Stromy patří k nejčastěji používaným grafům. Kořenový strom je graf, který obsahuje významný vrchol, označovaný také jako kořen, ze kterého se graf dále větví k níže postaveným vrcholům, takzvaným potomkům. Hloubka vrcholu je rovna počtu hran, které je třeba projít od vrcholu ke kořenu grafu. V informatice nalézá kořenový strom praktické využití právě jako efektivní datová struktura (Demel, 2002).

Známými metodami vícerozměrné indexace jsou stromové struktury *B-tree*,

R-tree. Je zřejmé, že použití indexu s sebou přinese potřebu ukládání dalších informací, proto je třeba zvážit, zda a v jaké dimenzionalitě tyto stromy používat. Pro praktické použití jsou stromové struktury výhodné, protože jsou často již implementovány v databázových systémech (Penninga, 2008).

Řadící indexy jsou odlišným typem indexování prostoru. Jedná se o spojité fraktální křivky, které prochází a vyplňují n-dimenzionální prostor a svým průchodem určují pořadí, ve jakém dochází k prohledávání prostoru. Příklad Hilbertovy a Mortonovy křivky je na obr. č. 2.3 (Penninga, 2008).



Obrázek 2.3: Hilbertova a Mortonova fraktální křivka. Převzato z Penninga (2008).

2.4 Topologické modely

Topologie je obor matematiky, který zkoumá vztahy mezi objekty v prostoru. V rámci GIS má topologie také svou nezastupitelnou úlohu, protože přenáší obecné matematické teorie do praktického využití. Obecně lze topologii popsat jako sadu určitých podmínek, při jejichž splnění označíme prostor za topologicky správný. Potřeba zavedení topologie pramení z toho, že geografická data, respektive mapy, jsou člověkem případně strojem generované datové soubory, které mapují reálné objekty. Tento sběr dat s sebou však přináší i chyby, ať už náhodné, systematické nebo zaviněné nedostatečnou přesností. Pokud není v datovém modelu zavedena topologie, snadno se může stát, že modelované objekty jsou v datech zaneseny s chybnou polohou, což ovlivní i vzájemné vazby mezi objekty. Vinou zmíněných chyb pak může dojít k chybné interpretaci dat. Mezi systémové výhody topologie patří možnost dotazování se na lokální vztahy nebo snížení redundance dat (Zeitouni et al., 1995).

2.4.1 Teorie bodových množin

Každá možná kombinace vztahů n-dimenzionálních objektů v n-rozměrném prostoru je topologií klasifikována a explicitně tak formalizuje topologické vztahy v prostoru. Důležitou vlastností topologických vazeb je invariance vůči prostorovým transformacím, jako je translace, rotace nebo změna měřítka.

Klasickým modelem bodových množin je možné popsat i zájmový objekt této práce, kterým je 3D těleso. Nechť X je *topologický prostor*. Mějme množinu bodů $A \subset X$. Vnitřní prostor, říkejme mu *vnitřek*, je tvořen množinou bodů značených jako A° a je definován jako sjednocení všech otevřených množin, které jsou podmnožinou A. *Uzávěr* A, značíme $\overline{A}$, je definován jako průnik všech uzavřených množin, které jsou podmnožinou A. *Hranice*, značená ∂A, je definována jako průnik uzavření A a uzávěr doplňku A, formálně tady jako $\partial A = \overline{A} \cap \overline{X - A}$. *Separabilita* množiny A je dvojice podmnožin $S \subset X, T \subset X$, pokud je splněno $S \neq \emptyset$, a zároveň $T \neq \emptyset$, a zároveň $S \cup T = A$, a zároveň $\overline{S} \cap T = \emptyset$, a zároveň $S \cap \overline{T} = \emptyset$. Pokud tedy existuje separabilita množiny A, potom je množina A nesouvislá, v opačném případě je souvislá. *Okolí* je poté neprázdná množina $X \in \mathbb{R}^n$ (Pultr, 2005).

Jak uvádí Egenhofer a Herring (1990), počet nezávislých vektorů, které tvoří základní prvky vektorového prostoru, definuje *dimenzi* prostoru. Základní vlastnost n-rozměrných prostorů je ta, že mohou obsahovat prvky nanejvýš o rozměru n. Zároveň pokud v $\mathbb{R}^n$ existuje prvek o rozměru n, neexistuje takové zobrazení tohoto prvku do prostoru o rozměru $(n-1)$. Základní prvky topologických prostorů jsou:

- 0-D prvek je uzel,
- 1-D prvek je hrana,
- 2-D prvek je plocha,
- 3-D prvek je těleso.

Dále je definována *kodimenze*, což je rozdíl mezi rozměrem prostoru a rozměrem prvku toho prostoru.

2.4.2 Průnikové topologické modely

Počet možných kombinací vzájemných interakcí mezi primitivy v $\mathbb{R}^2$ je 2^4, tedy šestnáct, a pouze devět z nich se nachází mezi prvky s kodimenzí rovnou nule. Z těchto devíti vztahů je osm takových, které mají souvislou hranici. Pro-

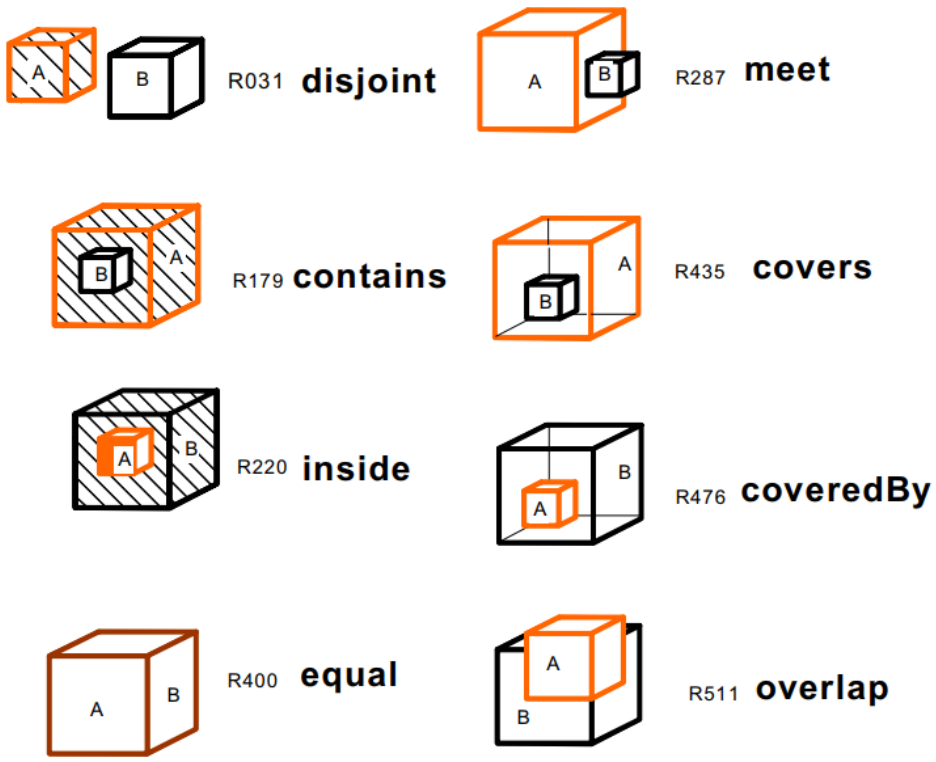
tože dochází k binárnímu porovnávání 4 primitiv, je tento model označován také jako *4-intersection-model* (4-IM). Mějme dva 2D polygony P_1 a P_2. Pokud zkombinujeme navzájem jejich hranici ∂ a vnitřek ^{o}, mluvíme o vztazích:

a) společné části hranic jako průnik hranic, značíme $\partial\partial$,

b) společný vnitřek, značíme ^{oo},

c) hranice jako část vnitřku, značíme ∂^{o},

d) vnitřek jako část hranice, značíme $^{o}\partial$.



Obrázek 2.4: Výsledné vztahy 4-IM modelu. Převzato z Zlatanova et al. (2004).

Topologické vztahy můžeme odvodit z toho, zda je průnikem polygonů P_1, P_2 prázdná nebo neprázdná množina, podle počtu nesouvislých průniků hranic a také podle dimenze průniku. Přehled těchto vztahů v $\mathbb{R}^2$ mezi dvěma polygony s kodimenzí 0 je v tabulce 2.2, a zároveň jsou vyobrazeny na obr. č. 2.4 (Egenhofer, Herring, 1990).

Model průniků 4-IM představený Egenhoferem a Herringem však nedokáže popsat všechny možné vztahy, které mohou mezi dvěma objekty nastat v $\mathbb{R}^n$ nastat. Lze si to představit v prostoru $\mathbb{R}^3$, kdy dimenze jednoho z objektů je menší než n, tedy například vztahy mezi tělesem a polygonem. Aby bylo možné formálně popsat i ostatní vztahy, byl zaveden *9-intersection-model* (9-IM), který oproti předešlému 4-IM zkoumá vztahy jak vnitřku a hranice, tak i vnějšku. Existuje

vztah	$\partial P_1 \cap \partial P_2$	${P_1}^{\circ} \cap {P_2}^{\circ}$	$\partial P_1 \cap {P_2}^{\circ}$	${P_1}^{\circ} \cap \partial P_2$	**popis vztahu**
r_0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	disjunktní
r_1	$\neg\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	dotýká se
r_2	$\emptyset$	$\neg\emptyset$	$\emptyset$	$\emptyset$	—
r_3	$\neg\emptyset$	$\neg\emptyset$	$\emptyset$	$\emptyset$	ekvivalentní
r_4	$\emptyset$	$\emptyset$	$\neg\emptyset$	$\emptyset$	—
r_5	$\neg\emptyset$	$\emptyset$	$\neg\emptyset$	$\emptyset$	—
r_6	$\emptyset$	$\neg\emptyset$	$\neg\emptyset$	$\emptyset$	uvnitř
r_7	$\neg\emptyset$	$\neg\emptyset$	$\neg\emptyset$	$\emptyset$	je pokrytý
r_8	$\emptyset$	$\emptyset$	$\emptyset$	$\neg\emptyset$	—
r_9	$\neg\emptyset$	$\emptyset$	$\emptyset$	$\neg\emptyset$	—
r_{10}	$\emptyset$	$\neg\emptyset$	$\emptyset$	$\neg\emptyset$	obsahuje
r_{11}	$\neg\emptyset$	$\neg\emptyset$	$\emptyset$	$\neg\emptyset$	pokrývá
r_{12}	$\emptyset$	$\emptyset$	$\neg\emptyset$	$\neg\emptyset$	—
r_{13}	$\neg\emptyset$	$\emptyset$	$\neg\emptyset$	$\neg\emptyset$	—
r_{14}	$\emptyset$	$\neg\emptyset$	$\neg\emptyset$	$\neg\emptyset$	—
r_{15}	$\neg\emptyset$	$\neg\emptyset$	$\neg\emptyset$	$\neg\emptyset$	překrývá

Tabulka 2.2: 16 kombinací topologických vztahů mezi vnitřky a hranicemi pro polygony v $\mathbb{R}^2$. Převzato z Egenhofer, Herring (1990).

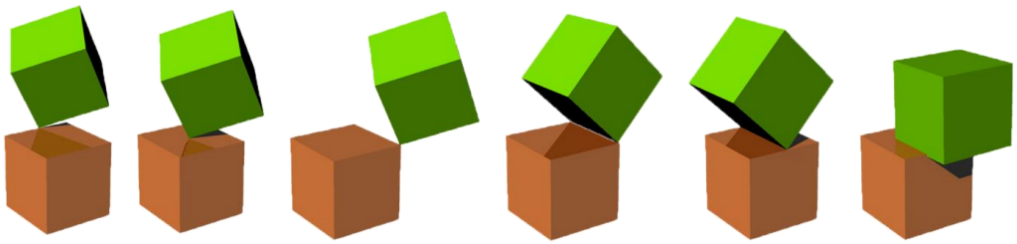
tedy devíti-rozměrná matice mezi těmito třemi primitivy, přičemž z původních 16 kombinací počet naroste na $2^9 = 512$. Model 9-IM tedy pro objekty A a B definuje hranice, vnitřek a vnějšek: ∂A, A°, A^{-} a ∂B, B°, B^{-}. Tento model můžeme formálně zapsat jako:

$$\Im_9(A,B) = \begin{pmatrix} A^{\circ} \cap B^{\circ} & A^{\circ} \cap \partial B & A^{\circ} \cap B^{-} \\ \partial A \cap B^{\circ} & \partial A \cap \partial B & \partial A \cap B^{-} \\ A^{-} \cap B^{\circ} & A^{-} \cap \partial B & A^{-} \cap B^{-} \end{pmatrix}$$

Zlatanova (2016) ukázala, že počet kombinací je možné dále redukovat zavedením tzv. negativních podmínek, které eliminují v reálném světě neexistující vztahy. Vzhledem k tomu, že ani 9-IM model nebere v úvahu tvar objektů, nastává mnoho situací, kdy vztahy v reálném světě lidským okem lehce odlišitelné se v 9-IM jeví ekvivalentně. Vizuální příklad takového jevu je na obr. č. 2.5. Možností, jak v tomto modelu rozlišit ekvivalentní situace, je rozšíření o dimenzionalitu průniku primitiv (Zlatanova, 2017).

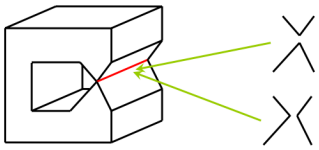
Shen, Zhou a Chen již zmíněné 4-IM, 9-IM, případně dimenzionální 9-IM modely rozšířili o počty průniků a popsali šest skupin topologických vztahů: uzel/uzel, uzel/hrana, uzel/plocha, hrana/hrana, hrana/plocha a plocha/plocha. Použi-

tím takového modelu je tedy možné popsat ještě větší počet topologických vztahů (Shen et al., 2017).



Obrázek 2.5: Ukázka vztahů, které se v 9-IM jeví ekvivalentní. Převzato z Zlatanova (2017).

Matematickými modely je možné popsat i objekty, které jsou takzvaně *non-manifold* a nejdou vyrobit ve skutečném světě. Takové modely mohou obsahovat abstraktní geometrické objekty jako například nekonečně tenkou linii. Ukázka non-manifold tělesa je na obr. 2.6, kde je zobrazena nemanifoldní situace spojení dvou ostrých částí. V případě manifold tělesa by zde spoj nebyl a existovala by úzká mezera nebo by byl spoj spojen tenkou vrstvou materiálu (Zlatanova, 2017). Popsané topologické modely pro GIS dokáží pracovat pouze se striktními *manifold* objekty. Zavádí se proto pojem *n-manifold*. Topologie v GIS pracuje s objekty s rozměrem maximálně 2-manifold.



Obrázek 2.6: Ukázka non-manifold tělesa. Převzato z Surynková (2017).

Lze konstatovat, že obecný GIS topologický model by měl používat minimálně 9-IM, protože v n-rozměrném prostoru může mezi objekty s rozdílnou kodimenzí existovat velký počet možných interakcí. Naopak, při testování topologických vztahů mezi objekty stejné kodimenze stačí aplikovat původní 4-IM model, ve kterém existuje pouze 8 možných vztahů, včetně disjunktního (Egenhofer et al., 1993).

3. Koncepty 3D generalizace

Využívání generalizace v tradičních mapových dílech je dnes již standardním způsobem, jak zajistit lepší čitelnost map (McMaster, Shea, 1992). Generalizace ve 3D vychází ze stejné potřeby, která stála u zrodu generalizace jako takové, přičemž zde dochází k zjednodušování (simplifikaci) geometrických 3D objektů. Zároveň je třeba vzít v potaz geografický aspekt, což v praxi znamená, že různé geografické typy objektů mohou vyžadovat odlišné způsoby generalizace (He et al., 2012).

Oproti klasickému dvourozměrnému pohledu je 3D řešení značně složitější a vyžaduje zobecnění algoritmů používaných v rovině do vyšší dimenze, případně použití zcela odlišných technik. Problém 3D generalizace je možné z pohledu geoinformatiky rozdělit do dvou proudů. První skupinou je zjednodušování obecných 3D modelů, které nacházejí využití hlavně v počítačové grafice a obecném matematickém modelování. Druhou kategorií jsou specifické generalizace 3D modelů budov, měst nebo jiných geografických objektů, které spadají právě do působnosti GIS. Specifické techniky zjednodušování objektů v 3D GIS začínají na poměrně jednoduchých principech vycházejících z rovinné generalizace, které se následně adaptují do 3D a končí na přístupech, které pracují přímo v 3D, případně dokonce v nD prostorech (Zhao et al., 2012).

3.1 Úroveň detailu

V 70. letech 20. století byl poprvé použit termín *úroveň detailu*, často zkracovaný jako LOD (*level of detail*). Koncept LOD se začal používat v oboru počítačové grafiky pro rozlišení scén z různou úrovní detailu a je úzce spojen s metodami, jak odlišných úrovní rozlišení dosáhnout. LOD je možné vnímat jako hierarchickou strukturu, dále jako lineární model přechodu mezi jednotlivými LOD nebo jako diskrétní oddělené modely v odlišném měřítku. Rozdílné je také vnímání toho, jak se má provádět zjednodušování geometrie v počítačové grafice od požadavků, které jsou kladeny v GIS. Pojítkem mezi oběma směry je parametr rozlišení, tedy určení toho, do jaké míry tvar geometrie zjednodušit a s tím

současně snaha o minimalizování prostorové chyby generalizovaného modelu vůči původní předloze (Biljecki, 2017).

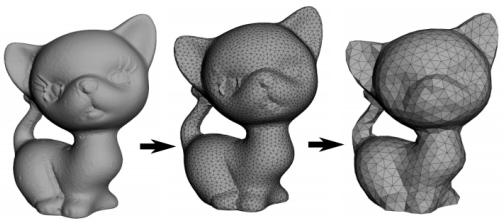
3.2 Generalizace za pomoci 2D projekce

Jedním ze způsobů, jak provádět generalizaci, je využití konceptu 2,5D modelu, který vychází z klasického 2D přístupu a umožňuje navíc modelovat svislé prvky. Využitím projekce modelu do roviny xy se vytvoří obtisk, takzvaný *footprint*. Před samotnou projekcí je vhodné klasifikovat jednotlivé částí budovy a použít rozhodovací pravidla pro další práci s těmito částmi. Prvky jako obvodové zdi nebo základy stavby poslouží pro generování zmiňovaného obtisku do 2D. Zbylé části, jako např. střecha, mohou být využity pro uložení informací o tvaru a pro extrahování informace o výšce objektu. Generalizace je provedena v rovině a výsledek je poté zrekonstruován zpět do 3D. Podobné postupy jsou ovšem nedostačující pro zpracování obecných objemových těles, protože prostor, ve kterém pracují, není úplným trojrozměrným prostorem (He et al., 2012).

3.3 Generalizace obecných povrchů

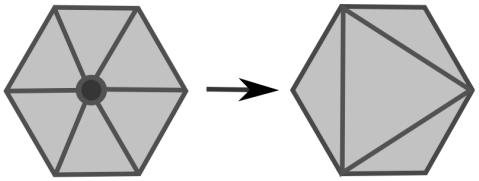
Simplifikace obecných objemových modelů řeší počítačová grafika již od poloviny 70. let 20. století. Hlavní motivací pro vývoj zjednodušujících algoritmů bylo zrychlení vykreslování v 3D grafických scénách (obr. 3.1). Metody zjednodušení geometrie obecných 3D povrchů jsou založené na iterativních změnách vstupní geometrie pomocí lokálních operátorů, které odstraňují primitiva nižší dimenzionality. V případě objemového polyhedronu je možné postupně odstraňovat plochy, hrany nebo vrcholy. Vzhledem k iterativnímu charakteru operací musí existovat metrika, kterou je možné měřit míru generalizace, tedy chybu polohy upravené geometrie oproti počátečnímu stavu. Pro měření chyby se používají metody měření odchylky vzdálenosti povrchů nebo Hausdorffova vzdálenost[1]. Při běhu algoritmu se v každé iteraci vyhledává prvek, jehož odstraněním vznikne nejmenší odchylka. Proces končí ve chvíli, kdy je překročena tolerovaná chyba nebo po určitém předem definovaném počtu kroků. Následující odstavce podrobněji popisují techniky lokální modifikace povrchu (Ovreiu, 2012).

[1]Hausdorffova vzdálenost je maximální odchylka mezi dvěma modely daná minimální vzdáleností mezi body obou modelů.



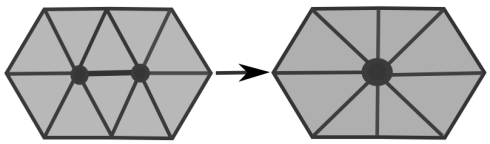
Obrázek 3.1: Postupné zjednodušení geometrie 3D modelu aplikací lokálních operátorů. Převzato z Ovreiu (2012).

Odstranění vrcholu Algoritmus prochází všemi dostupnými vrcholy a odstraní ten, který zajistí nejmenší chybu modelu. Pokud je povrch triangulován, tak se odstraněním jednoho vrcholu sníží počet ploch o dvě plochy (obr. 3.2).



Obrázek 3.2: Operátor odstranění vrcholu. Převzato z Ovreiu (2012).

Sjednocení hrany Tato operace prohledává model k odstranění vhodné hrany. Hrana je tvořena právě dvěma vrcholy a tato operace provede sjednocení těchto vrcholů. Z hrany se tedy stane singulární 0-simplex. Současně s tím dojde v triangulovaném modelu k odstranění jednoho vrcholu, třech hran a dvou ploch (obr. 3.3).



Obrázek 3.3: Operátor sjednocení hrany. Převzato z Ovreiu (2012).

Sloučení ploch Posledním typem je operace na 2-simplexech. Dochází k hledání koplanárních nebo téměř koplanárních ploch. Tolerance odchylky od koplanarity je zjišťována porovnáním normálových vektorů ploch. Identifikované plochy jsou za účelem zjednodušení geometrie sloučeny a okolní povrch je triangulován, aby bylo dosaženo validní orientace (Ovreiu, 2012).

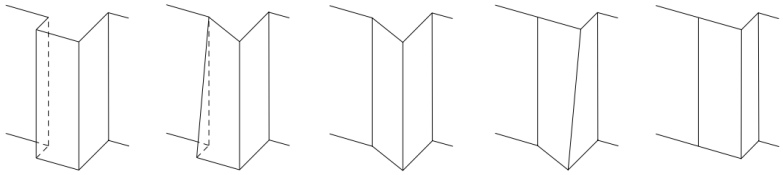
Aplikací uvedených operátorů tedy dojde k zjednodušení geometrie. Díky metrikám, které sledují odchylku od původního modelu, se zachovává i původní tvar. Obecně tyto modely nejsou příliš vhodné pro generalizaci pravidelných struktur,

jako jsou modely města. Nedokáží totiž zachovat to, co se od generalizace budov očekává, tedy rovné svislé stěny nebo pravoúhlost. Pro tento účel by musely být metody modifikovány přidáním podmínek, které budou vynucovat typické orientace ploch a hran. Přesto může být v určitých situacích použití zmíněných operátorů vhodné, a to zejména operace sloučení ploch, která zachovává rovinnost a zmenšuje počet nutných geometrických primitiv.

3.4 Generalizace specifických objektů

Speciální kategorií jsou metody generalizace ve 3D uzpůsobené přímo pro modely budov nebo jiné podobné geografické objekty. Tyto metody je možné rozčlenit do třech kategorií. První skupina těchto metod částečně vychází z obecných lokálních operátorů popsaných v předchozích odstavcích, další jsou založené na konceptu dělení prostoru na poloprostory a poslední využívají operátorů matematické morfologie.

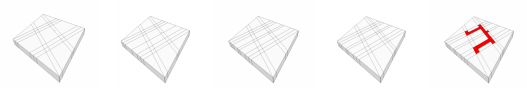
Lokálních operátorů pro decimaci geometrie využil Kada (2002) k vývoji algoritmu pro identifikaci a odstranění lokálních nerovností. Do modelu jsou přidány podmínky vynucující koplanaritu sousedních ploch nebo jejich pravoúhlost. V průběhu algoritmu dochází k prohledávání geometrie a hledání plošek s malým obsahem, které s největší pravděpodobností představují výklenky, mezery nebo jim podobné útvary. Opakovanou aplikací odstranění vrcholů nebo hran jsou tyto nerovnosti odstraněny. Na rozdíl od obecného použití, při kterém jsou hrany eliminovány tak, aby byla minimalizována chyba, zde se používá vynucení pravoúhlosti. Tento postup je popsán na obr. 3.4 V poslední fázi dojde k výpočtu nových aproximačních rovin pomocí metody nejmenších čtverců. Nově vypočtené roviny nahradí zbylou geometrii a tvoří finální zjednodušený model(Kada, 2002).



Obrázek 3.4: Schéma operace sjednocení hran při vynucení pravoúhlosti. Převzato z Kada (2002).

3.4.1 Metoda poloprostorů

Generalizaci 3D modelu budovy pomocí poloprostorů představili (Thiemann, Sester, 2004). Navržený postup ve zdrojovém modelu identifikuje v 3D prostoru množinu rovin, které obsahují významné části obvodových prvků budovy. Významnost určuje kategorizaci orientací všech ploch do skupin. Tímto jsou nalezeny hlavní aproximační roviny, jejichž množství ovlivňuje výslednou hladinu generalizace. Příklad prokládání rovin prostorem je na obr. 3.5.



Obrázek 3.5: Příklad dělení na poloprostory. Převzato z Kada (2006).

Množinou vybraných rovin je následně prostor rozdělen na disjunktní podprostory. Každý takto vytvořený podprostor generuje průnik s původní geometrií budovy. Zde dochází k porovnání, jaký podíl z objemu podprostoru je vyplněn původním tělesem. Nyní nastává rozhodovací fáze, definují se hraniční podmínky pro podíl objemu a dojde k binární klasifikaci podprostorů. Ty podprostory, které splní rozhodovací pravidlo, formují zjednodušenou konstrukci modelu budovy. Tato metoda může ve finálním modelu ponechat množství nežádoucích fragmentů, a proto musí být nakonec uplatněny ještě lokální techniky generalizace (Kada, 2006).

3.4.2 Metoda morfologických operátorů

Metoda generalizace pomocí struktur matematické morfologie má svůj počátek v technikách zpracování a vyhlazování signálu nebo obrazu. Metodu používanou na rastrový obraz je možné úspěšně použít také na vektorovou geometrii.

Matematická morfologie se zabývá studiem tvarů, které popisuje pomocí množin. Celý koncept je podpořen soustavou matematických teorií. Signál se zpracovává matematickým operátorem konvoluce signálu s konvolučním jádrem. Při zpracování obrazu je konvoluční jádro reprezentováno maskou, která prochází všemi body obrazu a mění výsledné hodnoty pixelů. Tento princip je možné převést do 3D prostoru tak, že zpracovávaným objektem bude polyhedron a konvolučním elementem nebude maska, ale 3D struktura, tedy libovolný trojrozměrný objekt. Stejně jako u konvolučních masek i zde je vhodné, aby byl tvar konvolučního operátoru jednoduchý, protože cílem je zjednodušení předlohy. Matematická

morfologie rozeznává čtyři základní morfologické operátory pro transformaci objektů v prostoru - dilataci, erozi, otevření a uzavření (Najman, Talbot, 2010).

Základní matematické operátory jsou invariantní vůči translaci. *Dilatace* δ je v geometrii ekvivalentní Minkowského sumě dvou polohových vektorů A, B. V Eukleidovském prostoru je tato Minkowského suma provedena přičtením polohového vektoru B k polohovému vektoru A. Vektory A, B mohou být reprezentovány množinou vektorů. Při aplikaci matematické morfologie představuje množina B strukturní element a množina A je zpracovávaná množina. Obecně platí pro dilataci, že:

$$\delta(A) = A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$$

Dilatace, která je Minkowského součtem (sumou), se označuje $A \oplus B$. Jak vyplývá z definice, původní velikost objektu se po aplikaci dilatace zvětší, protože dochází ke sčítání vektorových množin. Dilatace má vlastnosti komutativity $A \oplus B = B \oplus A$, asociativity $A \oplus (B \oplus C) = (A \oplus B) \oplus C)$ a je invariantní vůči posunu.

Doplněk X je definován všemi prvky, které nejsou součástí X. Doplněk doplňku X má za výsledek opět X, tedy $(X^{\mathsf{c}})^{\mathsf{c}} = X$.

Duálním operátorem k dilataci je *eroze*, nebo také Minkowského rozdíl:

$$\varepsilon(A) = A \ominus B = \{a \mid \forall b \in B, a + b \in A\}$$

.

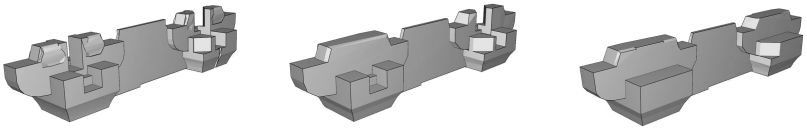
Erozi si lze představit jako průnik všech posunů obrazu A o vektor $-b \in B$. Eroze se používá pro odstranění úzkých ostrovů nebo pro rozdělení obrazu. Důležitou vlastností je dualita eroze a dilatace:

$$(A \ominus B)^{\mathsf{c}} = A^{\mathsf{c}} \oplus B$$

Dilatace následovaná erozí je nazývána uzavření $A \bullet B = (A \oplus B) \ominus B$. Eroze následovaná dilatací je nazývána otevření $A \circ B = (A \ominus B) \oplus B$ (Najman, Talbot, 2010).

Myšlenku založenou na matematické morfologii, tedy uzavření, použil Forberg (2007) pro generalizaci typických 3D modelů budov. Při aplikaci operátorů dilatace a eroze se kolem geometrie vytvořil pás o definované šířce. Podle typu operátoru se tento pás nacházel vně nebo uvnitř, a tak docházelo ke zvětšení nebo zmenšení objemu geometrie. Dále se vypočítalo, v jaké vzdálenosti od původní

geometrie měly vzniknout nové stěny. V druhé fázi došlo k posunu všech stěn do vypočtené vzdálenosti. Duální operací podobné erozi byl geometrii vrácen původní tvar, ovšem již bez drobných nerovností, které byly odstraněny (Forberg, 2007).



Obrázek 3.6: Příklad generalizace pomocí matematické morfologie. Převzato z Zhao et al. (2012).

Rozšířením předchozí práce pomocí výpočtu 3D Minkowského součtu pokračoval Zhao s kolektivem (Zhao et al., 2012). Aplikací 3D operátorů bylo možné provést generalizaci na libovolném manifoldním polyhedronu. Experiment při generalizaci čínského paláce charakteristického tvaru ukázal, že je metoda schopná zpracovat i takto komplexní objekty a snížit počty ukládaných primitiv v méně detailních LOD v závislosti na složitosti původní geometrie. Při generalizaci byly odděleně zjednodušovány objekty ze stejných výrobních materiálů, aby byla zachována jejich sémantika (obr. č. 3.6). Finální výsledek je ještě možné upravit použitím obecných technik zjednodušování povrchu, jako je slučování hran (Zhao et al., 2012).

Popsané metody generalizace 3D modelů budov mají svá specifika a je zřejmé, že budou existovat objekty, na kterých bude metoda fungovat bez problému, a naopak objekty, na kterých bude kolabovat. Přesto je metoda morfologických operátorů mírně výjimečná, a to díky své obecnosti a možnosti použití i mimo geoinformatiku. Metodu je možné i parametrizovat volbou tvaru strukturního elementu a jeho velikostí.

4. Návrh metody 3D generalizace

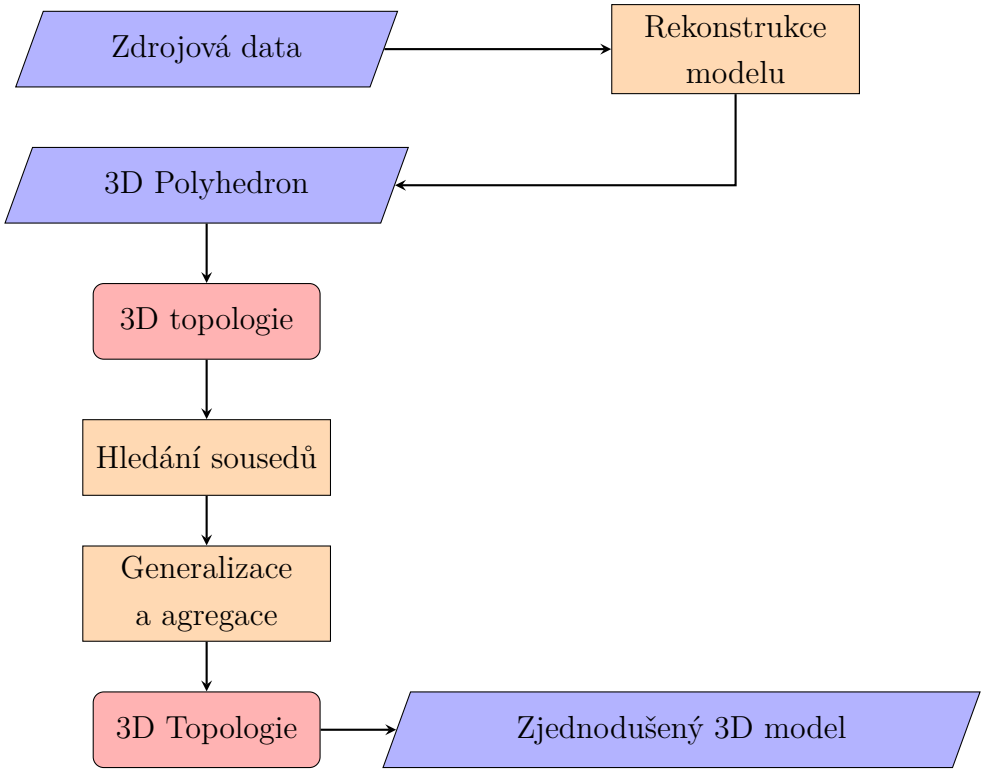
V předchozích kapitolách byl představen teoretický rámec, na jehož základech bude sestavena navrhovaná metoda generalizace. Tato část textu představí návrh metody generalizace 3D modelu městské zástavby, jejímž výsledkem bude topologicky korektní model s více úrovněmi detailu. Metoda navazuje na hlavní koncepty popsané v teoretické části, což jsou 3D datové struktury, teorie topologie a uplatnění jejích poznatků a generalizační postupy využívající technik matematické morfologie. Postup se skládá z několika kroků a jeho schématický přehled lze nalézt na obr. č. 4.1. Výstupem metody je 3D model nesoucí více úrovní detailu.

Navržený postup je možné rozdělit do těchto částí, které jsou podrobně rozebrány v následujících podkapitolách:

- Rekonstrukce polyhedronu
- Hledání sousedů
- Generalizace a agregace
- Víceúrovňové rozlišení

4.1 Rekonstrukce Polyhedronu

Na začátku celého procesu je nutné zhodnotit stav zdrojové geometrie. Fáze preprocessingu tedy bude zahrnovat identifikaci a opravu případných nedokonalostí a chyb ve zdrojových datech. Vstupní 3D data pro modely budov jsou zpravidla tvořena množinou polygonů, které reprezentují jednotlivé plošné části objektu, jako jsou například střechy nebo obvodové zdi. Problémem těchto shluků polygonů je, že samy o sobě nemodelují těleso, ale pouze ho ohraničují, což není z hlediska modelování objemových těles korektní přístup. Nad neobjemovými strukturami totiž není možné provádět analytické dotazy vztažené k objemu budov (Gröger, Plümer, 2011).



Obrázek 4.1: Schéma navrhované metody. Zdroj: vlastní tvorba.

4.1.1 Předzpracování modelu

Zdrojové polygony budou na začátku seskupeny podle příslušnosti k jednotlivým budovám a budou vstupem pro vytvoření souvislého uzavřeného povrchu pro každou budovu. Zpracování začne klasifikací polygonů do skupin podle směru jejich normálového vektoru, což dále umožní určit jejich pravděpodobnou sémantiku. Vodorovné nebo téměř vodorovné povrchy reprezentují základy budov nebo ploché střechy, svislé stěny obvodové nebo vnitřní zdi a šikmé polygony střešní plochy.

Před rekonstrukcí spodního základu budovy je zjištěna minimální hodnota výškové souřadnice $Min(z)$ napříč celou geometrií. V dalším kroku jsou vybrány všechny hrany polygonů obvodových zdí a jsou posunuty do stejné výškové hladiny na dříve zjištěnou hodnotu $Min(z)$. Z těchto hran je zpětně vytvořen základní polygon. Dále je zjištěna maximální hodnota výšky $Max(z)$, provede se duplikace spodního polygonu, jež se afinní transformací posune do výškové hladiny $Max(z)$. Pro zjednodušení procesu rekonstrukce jsou tedy původní šikmé střechy, pokud existovaly, nahrazeny plochými střechami. Mezi již vygenerovanými polygony jsou dopočítány zbylé obvodové polygony. Tímto procesem je garantována validita polygonů, tvořících model budovy.

Aby bylo možné vytvořit platný polyhedron, je nutné zajistit i správnou orien-

taci normál jednotlivých polygonů, které nakonec ovlivní orientaci celého povrchu polyhedronu. Sjednocení orientace se provede pomocí rekurzivního hledání sousedních incidenčních polygonů a kontroly jejich orientace v 3D prostoru. V prvé řadě je nutné zajistit, aby spodní polygon (základ budovy) byl orientován negativně, tedy po směru hodinových ručiček, protože jeho orientace musí směřovat směrem k zemi. Na tuto orientaci lze použít pravidlo pravé ruky. Podle Zwillingera (2003) se orientace polygonu O_P pro uspořádanou množinu souřadnic $(x_1, y_1), ...,(x_n, y_n)$ v rovině xy vypočte pomocí následujícího součtu determinantů (Zwillinger, 2003):

$$O_P = \text{sgn}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + ... + \begin{vmatrix} x_n & x_1 \\ y_n & y_1 \end{vmatrix}\right)$$

$$O_P = \text{sgn}\left(x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + ... + x_{n-1}y_n - x_ny_{n-1} + x_ny_1 - x_1y_n\right)$$

Pořadí vrcholů v základním polygonu se tedy upraví tak, aby byla jeho orientace záporná. Pro první polygon je dále hledán jeho soused, který s ním sdílí společnou hranu, což znamená, že mají dva společné vrcholy. Poté co je sousední polygon nalezen, se jeho orientace přizpůsobí prvnímu polygonu, přičemž orientace společné hrany musí být opačná. Takto je procházen celý seznam polygonů a jejich orientace je upravována podle jejich předchůdce. Tento postup sjednocení orientace povrchu formalizuje algoritmus č. 1 níže, na jehož vstupu jsou seznamy polygonů definované uspořádaným seznamem vrcholů. Výstupem z algoritmu jsou polygony s vyrovnanou orientací normál.

Po sjednocení orientace polygonů tvořících povrch polyhedronu je možné přistoupit ke konverzi sestavy těchto plošek do 3D datové struktury.

4.1.2 Konverze do půl-hranové datové struktury

Pro další operace bude zavedena půl-hranová datová struktura (*halfedge data structure*), do které se vloží předzpracované polygony a vytvoří se tak objemový model. Datová struktura se nejprve naplní unikátními vrcholy $V_i(x_i,y_i,z_i)$. Dále jsou vkládány uspořádané dvojice vrcholů (V_i, V_{i+1}). Cyklus těchto dvojic nakonec definuje plochu - část povrchu.

Konverzí do halfedge datové struktury se inicializuje i lokální topologický prostor, protože struktura si ukládá ukazatele na incidenční primitiva. Po dotazu na incidenční prvky struktura vrátí seznam všech vrcholů, hran nebo ploch, což umožní další geometricko-topologické operace nad modelem. Blíže je tato datová struktura popsána v podkapitole č. 5.4.1.

```
   Data: polygony
   Result: jednotně orientované polygony
 1 initialization;
 2 for každý polygon P do
 3 |   if je základní polygon then
 4 |   |   vypočti orientaci ulož P do seznamu již zpracovaných
 5 |   end
 6 end
 7 while nejsou všechny polygony zpracované do
 8 |   for každý polygon P do
 9 |   |   if P ještě není zpracovaný then
10 |   |   |   if P má právě 2 společné vrcholy s P již zpracovaným then
11 |   |   |   |   if mají na společné hraně stejnou orientaci then
12 |   |   |   |   |   zmeň pořadí vrcholů P
13 |   |   |   |   end
14 |   |   |   |   ulož P do seznamu již zpracovaných
15 |   |   |   end
16 |   |   end
17 |   end
18 end
```

Algoritmus 1: Sjednocení orientace polygonů v polyhedronu

4.1.3 Zavedení 3D topologie

Důležitým aspektem tohoto postupu je zavedení 3D topologie nad celým datovým modelem, což umožní sledovat vzájemné vztahy mezi jednotlivými objekty a identifikovat potenciální kolizní situace. Topologický systém bude těžit ze spolupráce s prostorovým indexem, který zmapuje celý model pomocí speciální datové struktury a umožní rychlé rozhodování při testování topologických incidentů.

Prostorový index

Funkce prostorového indexu v celém procesu není nezbytná, avšak umožní výrazné zrychlení 3D topologických operací, které jsou časově náročné. Není žádoucí, aby byly booleovské operace prováděny bezpodmínečně mezi všemi dvojicemi objektů, ale jen mezi těmi, u kterých to má smysl a jsou si prostorově blízké. Prostorový index si pro každý objekt udržuje rozsah geometrie ve všech dimenzích, což je ve skutečnosti ohraničující kvádr zarovnaný s osami kartézské soustavy souřadnic. Prostorový index používá R-tree strom, jehož předností je

schopnost indexování vícerozměrných struktur. Dotazem do prostorového indexu je tak možno v krátkém čase získat odpověď na otázku, které dvojice objektů mají případný blízký nebo jistý prostorový vztah. Je zřejmé, že porovnání ohraničujících kvádrů není exaktní metodou, protože mohou existovat objekty nezarovnané se souřadnicovými osami, nicméně přesný prostorový vztah je zjištěn až následně pomocí booleovských operací.

Identifikace topologických vztahů

Prvním krokem při testování topologických vztahů je prvotní odhad incidenčních dvojic pomocí prostorového indexu. V krátkém čase je vrácena množina dvojic objektů, u kterých se prostorový průnik ohraničujících kvádrů nerovná prázdné množině. Z původního počtu $\frac{1}{2}n^2$ dvojic, kde n je počet objektů, vznikne výrazně menší podmnožina dvojic, zlomek původního počtu. Na těchto vybraných dvojicích bude prováděn výpočet booleovské operace průniku 3D polyhedronů a její výsledek bude současně sloužit k určení topologického vztahu. V kapitole 2, v pojednání o topologii, je uveden přehled průnikových topologických modelů, na nichž bylo dokázáno následující - zkoumá-li se topologický vztah dvou objektů o stejné dimenzi a pokud je zároveň stupeň jejich kodimenze roven nule, aplikujeme jednodušší *4-intersection model*, ve kterém se rozeznává pouze 8 typů prostorových vztahů. Objekty zpracovávané při tomto postupu tyto požadavky splňují. Na začátku byl z ploškového modelu vytvořen uzavřený souvislý polyhedron P s nenulovým objemem, jedná se tak o geometrii s dimenzí $dim(P) = 3$. Vzhledem k faktu, že topologickým prostorem je $\mathbb{R}_3$, pak kodimenze objektů je

$$codim(P) = codim(R) - codim(P) = 3 - 3 = 0,$$

a tak je použití 4-IM opodstatněné.

Průnikový model 4-IM rozlišuje, jak již bylo zmíněno, osm typů vztahů, které jsou však pro potřeby tohoto postupu sloučené do třech kategorií, jak ukazuje tabulka 4.1. Jak je vidět, po průniku dvou objektů muže dimenzionalita výsledného průniku nabývat právě jedné z hodnot 0, 1, 2 nebo 3. Pokud spolu modely po průniku sdílí alespoň část vnitřku, dimenze $dim = dim(P_1 \cap P_2)$ tohoto průniku je vždy $dim = 3$, což bude v této metodě označeno jako topologická chyba, protože nelze připustit, aby se ve validní scéně objekty jakkoli protínaly.

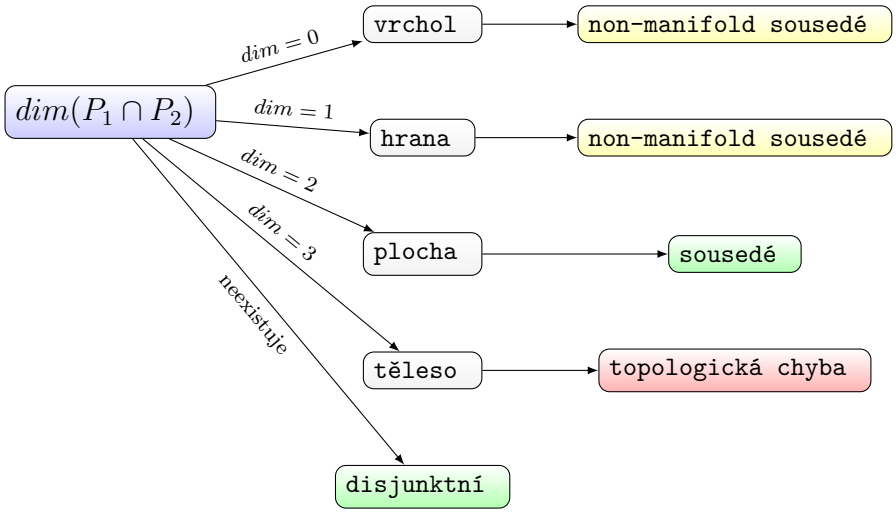
Rozhodovací strom na obr. č. 4.2 ukazuje, jaké mohou nastat topologické situace. Jsou následující:

disjunktní stav

Při disjunktním vztahu je situace jasná, objekty spolu prostorově nesouvisí.

	topologický vztah	výsledek průniku	dimenze průniku			
			0-D	1-D	2-D	3-D
r_0	disjunktní	disjunktní	—	—	—	—
r_1	dotýká se	sdílí hranice	•	•	•	—
r_3	ekvivalentní	sdílí vnitřek	—	—	—	•
r_6	uvnitř	sdílí vnitřek	—	—	—	•
r_7	je pokrytý	sdílí vnitřek	—	—	—	•
r_{10}	obsahuje	sdílí vnitřek	—	—	—	•
r_{11}	pokrývá	sdílí vnitřek	—	—	—	•
r_{15}	překrývá	sdílí vnitřek	—	—	—	•

Tabulka 4.1: Souhrn operace průniku dvou 3D polyhedronů ve 3D prostoru. Upraveno dle Egenhofer, Herring (1990).



Obrázek 4.2: Rozhodovací strom topologických pravidel. Zdroj: vlastní tvorba.

těleso

Pokud dimenze průniku dosáhne hodnoty 3, jedná se o průnik vnitřků a tedy konfliktní situaci, která v této metodě představuje *topologickou chybu*. Pokud budou takové topologické chyby nalezeny, dojde k sjednocení obou objektů, aby byla chyba odstraněna a model byl validní. Zároveň se zaznamenají ukazatele na původní budovy, aby bylo zřejmé, které patří do nového agregátu.

plocha

Tento vztah značí, že budovy spolu sdílí část svého povrchu, jsou to tzv. *sousedé*. Topologicky je to validní situace a zároveň tato informace poslouží při hledání skupin sousedících objektů v další fázi této metody

vrchol nebo hrana

Tato situace je sama o sobě validní, přesto bude v topologické vazbě tato informace uložena, aby se zabránilo pokusu o sjednocení těchto dvou modelů, které by vyústilo v non-manifold geometrii. Dá se předpokládat, že taková situace v praxi nenastane, nicméně je na místě ji ošetřit. Pokud by zdrojová data obsahovala tento vztah, dalo by se rozhodnout i o vyjmutí jednoho z modelů z analýzy, případně by bylo možné manuálně upravit geometrii tak, aby byla validní.

Vstupem do procesu kontroly topologie jsou předzpracovaná zdrojová data, kde jsou jednotlivé 3D modely složené z uspořádaných seznamů vrcholů, hran a ploch a seznamu budov. Výsledkem jsou modely v halfedge datové struktuře a současně s tím je provedená kontrola topologie. Ta poskytne topologické informace o dvojicích s nedisjunktním topologickým vztahem a zároveň dimenzi průniku těchto dvojic. Tuto část procesu je možné formalizovat algoritmem č. 2.

```
   Data: vertexList, edgeList, faceList, buildingList
   Result: polyhedronList, relationList
 1 initialization halfedge data structure as heDS ;
 2 for každou budovu B z buildingList do
 3 |   nový polyhedron P
 4 |   for každý vertex v z vertexList do
 5 |   |   vlož v do heDS
 6 |   end
 7 |   for každou plochu f z faceList do
 8 |   |   for každou hranu e z f do
 9 |   |   |   vlož e do heDS
10 |   |   end
11 |   end
12 |   vlož P do polyhedronList
13 end
14 for každou kombinaci dvojic polyhedronů P_1, P_2 z polyhedronList do
15 |   if bbox(P_1) ∩ bbox(P_2) = ¬∅ then
16 |   |   I = P_1 ∩ P_2
17 |   |   if dim(I) >= 0 then
18 |   |   |   ulož dim(I) vztahu mezi P_1, P_2 do relationList
19 |   |   end
20 |   end
21 end
```

Algoritmus 2: Algoritmus konstrukce polyhedronů a kontroly topologie

Validace zdrojových dat Výše navržená pravidla poslouží zároveň pro kontrolu a pročištění vrstvy vstupních zdrojových dat, aby celý model od začátku pracoval pouze s topologicky validním modelem. Objekty se postupně načtou do připravené datové struktury, provede se kontrola validity geometrie a kontrola souvislosti jejich povrchu. Poté se provede zjištění topologických vztahů, a pokud se nalezne některá kombinace dvojic, která se objemově protíná, provede se sjednocení, a tím se odstraní topologická chyba. Sousedé sdílející plochu nebo více ploch jsou topologicky korektní.

4.2 Hledání sousedů

V této části postupu bude provedeno hledání shluků sousedů, tedy geometrií, které spolu vzájemně sousedí a tvoří tak například celistvý blok budov. Nalezení těchto shluků bude sloužit pro odlehčení výpočetních nároků - nebude tak nutné následnou generalizaci provádět na celém modelu, ale pouze iterativně po jednotlivých shlucích. Pro hledání těchto shluků budou využita data zjištěná při topologické operaci, konkrétně ta, která ukazují na dvojice sousedů. Tento seznam dvojic (*neighboursTuples*) se bude rekurzivně procházet, dokud nebudou všechny shluky jednoznačně odděleny. Jak probíhá proces hledání shluků je vidět v návrhu algoritmu č. 3, na jehož vstupu je seznam sousedících dvojic a výstupem je seznam shluků. Aby bylo možné provést shlukování sousedů na celém modelu, bude seznam dvojic sousedů neighboursTuples spojen se seznamem všech budov, což si lze představit jako stav, kdy je jedna a tatáž budova sama sobě sousedem. Příklad pro množinu budov ABCD, kdy A sousedí s B, B sousedí s C a D stojí samostatně, je popsán v tabulce č. 4.2.

dvojice	**vztah**
(A, B)	sousedé
(B, C)	sousedé
(A, A)	sousedem sám sobě
(B, B)	sousedem sám sobě
(C, C)	sousedem sám sobě
(D, D)	sousedem sám sobě

Tabulka 4.2: Příklad seznamu dvojic

Funkce hledání shluků je hladovým algoritmem, který prochází všechny vstupní dvojice sousedů do té doby, než jsou všechny jedinečné budovy rozřazeny do jednotlivých shluků. Podmínka konvergence je tedy splněna v tu chvíli, kdy je

```
   Data: neighboursTuples
   Result: clustersList
1  while nejsou všechny budovy shlukované do
2  |  for každou dvojici tuple z neighboursTuples a zároveň každý cluser
   |   z clustersList do
3  |  |  if tuple ∩ cluster = ∅ then
4  |  |  |  vlož tuple do clustersList jako nový cluster
5  |  |  else if tuple ∩ cluster = jedenprvek then
6  |  |  |  vlož nový prvek do již existujícího clusteru v clustersList
7  |  end
8  end
```

Algoritmus 3: Algoritmus hledání shluků z dvojic sousedů

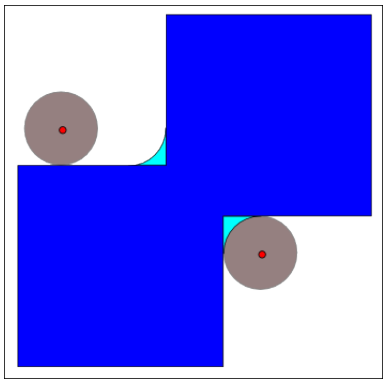
počet budov v seznamu shluků roven unikátnímu počtu budov na vstupu. Algoritmus začíná cyklem, kdy pro každou dvojici hledá, zda je již zařazena v některém ze shluků. Pokud jsou všechny budovy z dvojice v některém ze shluků, nic se neprovede. Pokud je první z dvojice v některém ze shluků a druhá z dvojice nikoliv, je tato druhá do tohoto shluku zařazena. Pokud není ani jedna z dvojic v žádném ze shluků, je celá dvojice zapsána jako nový shluk. Po celé první iteraci se může stát, že jsou některé budovy ve více shlucích a není tak splněna podmínka konvergence. Algoritmus tedy pokračuje další iterací, kdy jsou původní dvojice nahrazeny nově nalezenými shluky a vyhledávání analogicky pokračuje dále. Po splnění podmínky konvergence se algoritmus ukončí a výsledkem jsou disjunktní shluky vzájemně sousedících budov. Výstupem příkladu z předešlého odstavce by tedy byly dva shluky: první (A,B,C) a druhý (D).

4.3 Generalizace a agregace

V předchozích částech byly provedeny nezbytné kroky před zahájením procesu generalizace celé scény. Celý model je součástí halfedge datové struktury, byly zjištěny shluky objektů a po jednotlivých shlucích bude prováděn proces simplifikace. Budou se aplikovat metody matematické morfologie, jejichž výsledkem bude generalizace původních modelů a u shluků objektů dojde současně k agregaci.

Proces generalizace bude sestávat z operace uzavření a otevření. Opakovanou aplikací Minkowského součtu polyhedronu a strukturního elementu, se dosáhne odstranění elementů, jako jsou díry, zákoutí a výběžky, které jsou svou velikostí menší než používaný strukturní element. Podoba strukturního elementu má vý-

razný podíl na konečné podobě generalizace. Vzhledem k tomu, že je tato metoda navržena pro 3D generalizaci, musí i strukturní element dosahovat stejné dimenze. Tento element by měl mít ideálně podobu jednoduchého 3D objektu. Jednoduché symetrické 3D těleso je například koule. Vhodnější však bude pravděpodobně použití tělesa s tvarem, který je charakteristický pro modely budov. Takové pravoúhlé těleso představuje například krychle. Ukázka kruhového strukturního elementu a jeho efektu na výslednou dilataci je na obr. č. 4.3.



Obrázek 4.3: Demonstrace nevhodného použití kruhového, případně kulového strukturního elementu na pravoúhlý objekt. Převzato z Wikipedia (2008).

4.3.1 Strukturní element

Samotné použití jednotkové krychle jako strukturního elementu by však bylo nedostatečné a bude vhodné použít afinní transformace pro škálování a rotaci tohoto elementu. Cílem rotace krychle bude co největší přizpůsobení ke směru natočení 3D modelu budovy v prostoru. Pro tento účel se vypočte nejvýznačnější směr budovy pomocí orientace jejích hran rovnoběžných s rovinou xy. Vzhledem k tomu, že rotací krychle podle osy z má smysl provádět pouze na intervalu $\langle 0, \frac{\pi}{2})$, převedou se všechny vypočtené směry do prvního kvadrantu. Po tomto sloučení se sečtou délky hran, které náleží jednotlivým směrům. Finální význačný směr budovy je úhel s největším součtem délek hran. O tento úhel se krychle rotací kolem osy z otočí. Krychle bude zformována tak, že její těžiště leží v počátku soustavy souřadnic a její strana má délku $a = s$. Parametr s je takzvaný škálovací parametr, který umožní krychli zvětšovat či zmenšovat, dle požadavku na míru generalizace. Čím větší je krychle, tím více pohltí rušivé elementy na generalizovaném objektu. Reálná hodnota velikosti strukturního elementu bude diskutována v praktické části této práce.

4.3.2 Morfologické operace

V předchozích krocích byly zkonstruovány validní objemové modely budov a strukturního elementu uložené v halfedge datové struktuře, které nyní vstupují do fáze generalizace. Kombinací metod dilatace a eroze se budou postupně odstraňovat části budovy, které jsou svým rozměrem menší než strukturní element. Aplikaci morfologické generalizace je možné provádět kombinováním operací dilatace a eroze v různém pořadí a počtu provedených kroků. Nejvyužívanější jsou tyto kombinace (Damen et al., 2008):

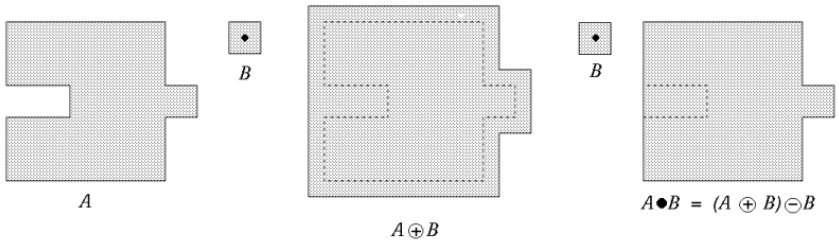
- pouze operace uzavření,
- pouze operace otevření,
- operace uzavření následovaná otevřením,
- operace otevření následovaná uzavřením.

Použití té které varianty se odvíjí od požadavku na typ generalizace. Operace uzavření dokáže odstranit nepravidelnosti typu děr, proluk, mezer atp. Naproti tomu operace otevření je schopna eliminovat např. úzké spoje a nahradit je mezerou. Schopnost eliminace je nicméně relativní k velikosti použitého strukturního elementu. Jak ukazuje Damen et al. (2008), je vhodné nejdříve provést uzavření a následně otevření. Opačná souslednost může mít za následek vyšší eliminaci modelů budov (Damen et al., 2008).

Operace uzavření je posloupností morfologických operátorů dilatace a eroze. Dilatace D je Minkowského sumou množin objektu A a strukturního elementu B, zapsané jako $D = A \oplus B$. Aplikací Minkowského sumy vznikne pás offsetu ve vzdálenosti poloviny délky hrany strukturního elementu. Dilatace tak v případě mezery, ve které dojde z obou stran k průniku offsetů, zajistí zacelení tohoto prostoru objemem objektu. Pro přiblížení zpět k původnímu tvaru budovy se použije komplementární operace eroze. Eroze E je opět Minkowského suma, a to množinového doplňku dilatace D a strukturního elementu B, formálně $E = (A \oplus B)^{\mathsf{c}} \oplus B$. Tímto je dokončena operace uzavření, jež dostala svůj název právě proto, že uzavírá objekt. Na obr. č. 4.4 je grafické znázornění uzavření ve 2D. Matematická formulace otevření je následující:

$$A \bullet B = (A \oplus B) \ominus B = \Big((A \oplus B)^{\mathsf{c}} \oplus B \Big)^{\mathsf{c}}$$

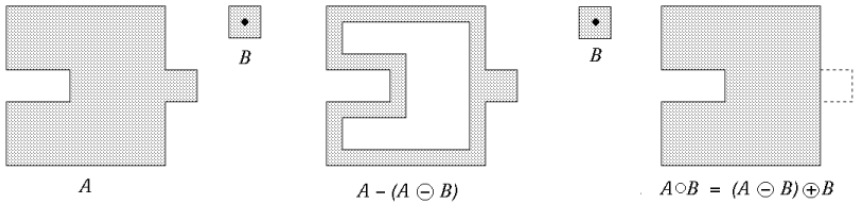
Generalizace dále pokračuje aplikací operace otevření, která je oproti uzavření v obráceném pořadí, nejdříve se použije eroze a poté dilatace. Eroze E, nebo také Minkowského rozdíl množin, se zapíše jako $E = A \ominus B$. Eroze opět vytváří offset,



Obrázek 4.4: Grafické znázornění morfologické operace uzavření ve 2D. Převzato z Peterlin (1996).

ovšem tentokrát na množinovém doplňku vstupní množiny A, eroze tedy zeroduje objekt, který tak ztrácí svůj objem, jak ve vidět na obr. č. 4.5 uprostřed. Zároveň však dochází k odstranění drobného výrůstku na pravé části objektu. Otevření se dokončí opětovnou aplikací komplementární dilatace. Původní tvar objektu se výrazně nemění a došlo k odstranění přebytečných artefaktů. Formální zápis popsané množinové operace je:

$$A \circ B = (A \ominus B) \oplus B = (A^{\mathsf{c}} \oplus B)^{\mathsf{c}} \oplus B$$



Obrázek 4.5: Grafické znázornění morfologické operace otevření ve 2D. Převzato z Peterlin (1996).

Postupnou aplikací operace uzavření a otevřeni, v praxi dilatace, eroze, eroze a dilatace v trojrozměrném prostoru dojde v relevantních částech původního objektu k zjednodušení jeho geometrie. Zároveň dojde nejen k snížení počtu vrcholů, ale i počtu hran a ploch, které tento objekt definují.

4.3.3 Agregace a postprocessing

Generalizace celého modelu bude dekomponována na samostatné procesy, které mohou běžet paralelně, či sériově, a to kvůli očekávaným vysokým výpočetním nárokům. Každý proces bude zpracovávat vždy jeden shluk budov identifikovaný v předchozím kroku této metody. Každý shluk budov je zároveň v každém

procesu podroben agregaci, která je důsledkem aplikace morfologického operátoru dilatace, jak je popsáno v podkapitole 4.3.2. Dilatace způsobí zvětšení objemu vstupních těles v daném shluku, což vede k agregaci těchto těles a výsledkem je tak jedno kompaktní těleso reprezentující celý shluk, na které jsou dále aplikovány zbylé morfologické operace. Detailně jsou pak tyto kroky vysvětleny v podkapitole č. 7.5.4 pojednávající o implementaci této metody. Postupným během procesů dojde ke generalizaci původního modelu a výsledné objekty jsou opět uloženy v halfedge datové struktuře.

```
   Data: vstupní 3D model (inputPolyhedronClusters)
   Result: výstupní 3D model (ouputPolyhedrons)
 1 for každý shluk Cluster z inputPolyhedronClusters do
 2     for každou halfedge z Cluster do
 3         vypočti hlavní směr rotace Angle pro Cluster
 4     end
 5     sestav strukturní element E
 6     proveď rotaci E o úhel Angle
 7     nový AggregatedPolyhedron
 8     for každý Polyhedron P z Cluster do
 9         dilatace
10         sjednocení
11         eroze
12         eroze
13         dilatace
14         ulož do AggregatedPolyhedron
15     end
16     for každou halfedge z AggregatedPolyhedron do
17         najdi incidenční plochy f_1a f_2
18         vypočti normálové vektory n_1, n_2 ploch f_1,f_2 obou ploch
19         if normály n_1, n_2 jsou kolineární then
20             odstraň he z heDS
21             proveď sloučení ploch f_1,f_2
22         end
23     end
24 end
25 return AggregatedPolyhedron
```

Algoritmus 4: Generalizace a postprocessing.

Při agregaci budov, ale také při generalizaci solitérních objektů může docházet ke generování vrcholů a hran, které od sebe dělí zdánlivě koplanární plochy.

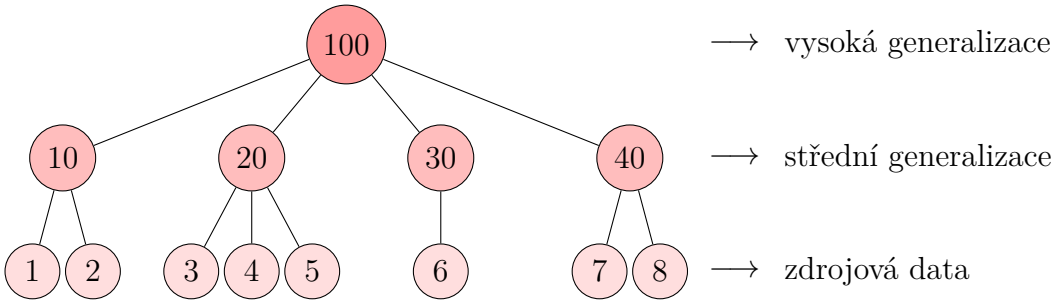
Jakmile je však orientace normálového vektoru těchto ploch byť jen mírně odlišná, nové hrany mezi těmito plochami se musí zákonitě objevit. Z toho důvodu bude provedeno vyrovnání a sloučení ploch s velmi blízkou orientací normál do jedné plochy, což vyústí v úsporu dat, definujících geometrii. Platí, že každá hrana je v datové struktuře halfedge složena ze dvou opačně orientovaných půlhran. Dále pro souvislé povrchy platí, že každá halfedge má incidenci právě se dvěma plochami. Budou se tedy procházet všechny dvojice vzájemně incidenčních ploch a bude se zjišťovat, zda jsou tyto plochy koplanární. Pokud ano, dojde k vyjmutí dotyčných halfedge hran a rekonstrukci okolí tak, aby nově vzniklá plocha měla správnou orientaci s incidenčními halfedge hranami (Donkers et al., 2015).

Algoritmus č. 4 souhrnně popisuje výše popsaný proces generalizace. Na vstupu jsou jednotlivé shluky budov, tedy polyhedronů, vygeneruje se strukturní element a provede se jeho orientace se shlukem budov. Aplikací morfologických operátorů dojde ke generalizaci a agregaci shluků, u kterých jsou na konci odstraněny přebytečné koplanární plochy. Výstupem algoritmu jsou již agregované a generalizované modely budov.

Postupem popsaným v této podkapitole tedy byla provedena generalizace vstupních objektů, které byly v závislosti na velikosti strukturního elementu případně dále agregovány do souvislých bloků. Závěrem se provedlo sloučení koplanárních ploch, které mohou vzniknout důsledkem provádění booleovských operací nad polyhedrony. Výsledkem je zjednodušená geometrie na nižší úrovni rozlišení.

4.4 Víceúrovňové rozlišení

Provedením alespoň jedné iterace generalizace 3D modelu města dojde k vytvoření celého nového modelu na nižší úrovní detailu a vznikne potřeba připravit strukturu, která umožní s víceúrovňovým modelem pracovat a propojit odkazy na jednotlivé budovy v různých úrovních.



Obrázek 4.6: Příklad stromu pro ukládání informace o agregaci. Zdroj: vlastní tvorba.

Každá budova si udržuje svůj jednoznačný identifikátor, který si s sebou nese

v celém procesu. Pokud dojde k agregaci, vytvoří se nový náhodný identifikátor shluku, na který se navážou identifikátory původních modelů. Takto vznikne jednoduchá stromová struktura, kde jsou uzly tvořené jednotlivými budovami, případně agregovanými bloky budov, a hrany reprezentují odkazy. Hloubku stromu určuje počet úrovní detailu. Na obr. č. 4.6 je příklad stromu pro 3 úrovně rozlišení, který původní solitérní objekty agreguje do bloků.

5. Aplikační rámec a datové struktury

V této kapitole bude popsán aplikační rámec, ve kterém bude probíhat následná implementace. Na začátku budou představeny použité technologie, poté použitý software a popis důležitých datových struktur, se kterými se bude při implementaci pracovat.

Při hledání vhodných softwarových nástrojů pro implementaci navržené metody byl kladen důraz na to, vyhnout se komerčnímu a proprietárnímu programovému vybavení, a to jednak z důvodu, že komerční propracované software se schopností práce ve 3D vyžadují nemalé investiční náklady a jednak proto, aby byly výsledky práce široce dostupné pro akademickou komunitu. Především bylo nutné nalézt takový software, který bude schopen pracovat s reálně trojrozměrnými objekty a bude poskytovat dostatečnou infrastrukturu z pohledu dostupných funkcí a datových struktur, protože by nebylo v silách a možnostech autora této práce naprogramovat celé řešení od základů.

5.1 Systém a programové vybavení

Jako hlavní vývojové prostředí byl zvolen linuxový operační systém a jeho distribuce Ubuntu ve verzi 16.04 s kódovým označením *xenial* postavená na 64bitové architektuře. Hardwarový výbava zahrnuje dvou-jádrový procesor s frekvencí 2,2 GHz a paměť RAM o velikosti 7 GB. Celý systém není provozován lokálně, ale ve virtualizovaném prostředí serverových farem, což je v dnešní době trend získávání výpočetního výkonu. Výhodou těchto platforem je možnost sestavit si parametry systému dle vlastních potřeb a možnost platby pouze za spotřebovaný výpočetní čas.

Na poli softwarových nástrojů existuje již poměrně velký výběr programů, které umí pracovat s 3D daty. V oblasti GIS je velmi oblíbenou aplikací ArcGIS, která obsahuje i nadstavbu pro 3D aplikace, avšak jedná se spíše uživatelsky přívětivý front-endový nástroj, jehož aplikační rozhraní není pro 3D modelování

dostačující. Dále existuje velké množství programů na bázi CAD, jako je AutoCAD, OpenSCAD, nicméně ty jsou využívány spíše v konstruktérství. Velká část akademických implementací využívá knihovnu CGAL pro svoji robustnost výpočtu a schopnost zpracovávat velké datasety.

Dále bylo nutné vybrat vhodný typ úložiště, ve kterém budou data skladována. Vzhledem k tomu, že souborový systém ukládání není dostatečně flexibilní, byl vybrán databázový systém, kterých je k dispozici celá řada. Opět díky požadavku na open-source se jako nejvhodnějším systémem jevil databázový systém PostgreSQL s nadstavbou PostGIS, podporující prostorová data. Vzhledem k vybraných softwarovým nástrojům bude v této implementaci hlavním programovacím jazykem C++ v kombinaci s jazykem Python. Standardem pro práci s databází je dotazovací jazyk SQL.

5.2 CGAL

Knihovna CGAL (*Computational Geometry Algorithms Library*) je softwarovou knihovnou výpočetní geometrie poskytující efektivní a robustní algoritmy pro práci s geometrickými objekty. Počátky této knihovny se datují již od roku 1996, kdy vznikla jako společný projekt univerzit z osmi evropských měst a z Izraele podporovaný fondy Evropské Unie. Tato knihovna bude využita na všechny podstatné geometrické operace v následující implementaci.

Knihovna CGAL je napsána v jazyce C++ a z tohoto důvodu bude C++ hlavní jazyk této práce. CGAL je rozdělena na jednotlivé balíky, kdy každý je určen ke specifickému použití a má vlastní dokumentaci. Dokumentace CGAL je poměrně rozsáhlá a většinou obsahuje i praktické příklady. K dispozici jsou dokonce programová rozhraní do jazyků Java a Python, avšak nepodporují všechny dostupné balíky, a tak jsou pro tuto práci irelevantní. Silnou stránkou je možnost použití jádra (*Kernel*), které provádí přesné geometrické výpočty a konstrukce, a to bez zaokrouhlovacích chyb. Jádro je sice pomalejší, ale zato robustní. V současné době je CGAL k dispozici ve verzi 4.10 a je distribuován pod hlavičkou licence *Open Source*, ačkoli licenční podmínky jednotlivých balíků algoritmů mohou být odlišné. V rámci implementace navržené metody bude využito následujících balíků:

- *3D Polyhedral Surface*, licence GPL
- *Halfedge Data Structures*, licence LGPL
- *3D Boolean Operations on Nef Polyhedra*, licence GPL

- *3D Minkowski Sum of Polyhedra*, licence GPL
- *Boost Graph Library*, licence LGPL

Použité balíky jsou dostupné za licenčních podmínek *GPL* nebo *LGPL*. Této práce se týká hlavně licence GPL, která požaduje při distribuci softwaru nad CGAL datovými strukturami zveřejnění zdrojového kódu. Této podmínce bude vyhověno zveřejněním zdrojových kódů ve veřejném repozitáři GitHub. Licence LGPL pouze ošetřuje povinnost zveřejňování modifikace zdrojového kódu.

5.3 PostgreSQL a PostGIS

Databázový systém PostgreSQL bude sloužit jako hlavní prostředek pro uchovávání geometrických objektů před a po zpracování v knihovně CGAL. Cílem je zasadit výsledný model do jednoduché struktury databáze, která umožní snadné a rychlé dotazování a propojení s procesem vizualizace. PostgreSQL je objektově-relační databázový systém pod licencí MIT, která zaručuje svobodné použití. Konkurenční výhodou tohoto databázového systému je dostupnost nadstavby PostGIS, která z databáze dělá téměř plnohodnotný GIS software. PostGIS obsahuje snad většinu klasických 2D funkcí známých např. z proprietárního ArcGIS. Mimo to však v omezené míře umožňuje operace nad plnohodnotnými 3D geometriemi, a to za pomoci SFCGAL, což je C++ wrapper okolo knihovny CGAL pro podporu 3D operací a 3D datových typů v databází PostgreSQL. Software PostGIS a SFCGAL používají licenci GNU, tedy svobodné použití.

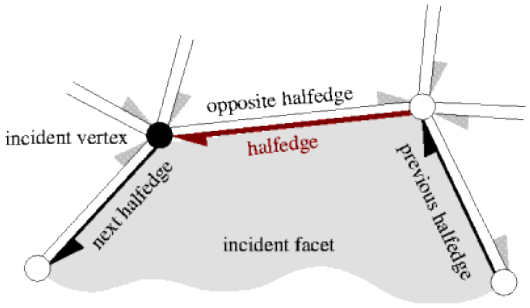
5.4 Datové struktury

Během implementace jsou použity datové struktury v rámci CGAL i PostGIS, které jsou výrazně odlišné. Jak již však bylo uvedeno, důležité geometrické operace s využitím sofistikovaných datových struktur jsou prováděny v rámci CGAL a zbytek je prováděn v databázi.

5.4.1 Halfedge datová struktura

Tuto datovou strukturu implementuje CGAL pro práci s 3D povrchy a polyhedrony. Je to hranově orientovaná datová struktura schopná uchovávat data o incidenčních primitivech, jako je bod, hrana a plocha. Každá hrana (*edge*) je dekomponována na dvě opačně orientované půlhrany (*halfedge*). Pro každou

halfedge se ukládá informace o incidenčních vrcholech a incidenčních plochách. Struktura podporuje pouze orientované 2-manifold objekty. K dispozici jsou pomocné třídy, díky kterým je možné z aktuální halfedge odkázat nejenom na přímo incidenční primitiva, ale také na opačnou, předešlou a následující halfedge. Dále jsou k dispozici iterátory, které vrací uspořádaný seznam incidenčních prvků (například okolo plochy) a také třídy pro lokální změnu povrchu, jako je přidávání a odstranění primitiv.



Obrázek 5.1: Schéma lokálního okolí halfedge datové struktury. Převzato z Kettner (2017).

5.4.2 Polyhedral Surface

Datové struktury používané v databázi PostreSQL jsou úplně jiného charakteru, nežli v CGAL, nativně neumožňují dotazy na incidenční primitiva nebo použití iterátorů. V podstatě se jedná o změť polygonů (*polygon soup*) na jejichž pořadí nezáleží a důležitá je pouze jejich orientace, tedy pořadí vrcholů, které definují plochu. Konstrukci modelu v tomto formátu je možné provádět např. pomocí WKT, což je lidsky čitelný formát geometrií. Například jednotková krychle je tvořena šesti polygony (čtverci) a definice požaduje, aby první a poslední vrchol polygonu byl identický a pomyslně tak uzavřel okruh. Zápis krychle poté vypadá následovně:

```
POLYHEDRALSURFACE Z~(
  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),  ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)),
  ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),  ((1 0 0, 1 0 1, 0 0 1, 0 0 0, 1 0 0)))
```

Tímto způsobem se v rámci databáze uloží modely budov a je možné nad nimi provádět různé analytické dotazy.

5.4.3 Prostorový index GIST

Nadstavba PostGIS také poskytuje možnost využití prostorového indexování GIST, což výrazně urychluje prostorové dotazy. Prostorový index je potřeba

nejdříve vybudovat, aby bylo možné využít jeho předností. Index GIST je upraveným R-Tree indexovacím mechanismem, který není úplně exaktní, protože neindexuje celou geometrii, ale pouze její ohraničující kvádr zarovnaný s osami souřadnic, což je v podstatě minimální a maximální hodnota souřadnic x,y,z. Při prohledávání stromu se tak pouze kontrolují přesahy ohraničujících kvádrů a vyhledávání je tak velmi rychlé i na velkých objemech dat. V rámci prohledávání pomocí tohoto indexu je možné používat prostorové operátory, které kontrolují vztahy mezi dvěma objekty. Příkladem zmíněného operátoru je například dotaz na všechny dvojice objektů, u kterých nastane jakýkoliv průnik jejich ohraničujících kvádrů.

Je zřejmé, že porovnávání pouhých ohraničujících kvádrů není dostatečné pro zjištění přesných topologických vztahů. Přesný vztah se bude zjišťovat podle dimenze průniků porovnávaných geometrií.

5.4.4 Komunikace mezi CGAL a databází

Knihovna CGAL podporuje export 3D geometrií ve formátu OFF (*Object File Format*), který je typický pro úzce specializovaný software pro 3D modelování. Systémy orientované na GIS tento formát nativně nepodporují. Z tohoto důvodu bylo nutné napsat potřebné čtecí a zapisovací metody, aby si mohly jednotlivé části programu předávat data.

Formát OFF se sestává z hlavičky, kde na prvním řádku je heslo OFF a na druhém řádku je počet vrcholů, počet ploch a počet hran oddělené mezerou. Následuje příklad zápisu geometrie ve formátu OFF pro čtyřstěn. Na dalších řádcích pokračuje definice geometrie. Na každém dalším řádku je umístěn jeden vrchol a jsou zapsány jeho souřadnice xyz oddělené mezerou. Po definici vrcholů následuje definice ploch a to opět způsobem, že každá plocha je na novém řádku. Řádek plochy začíná počtem vrcholů, které tvoří plochu, a následuje uspořádaný seznam indexů vrcholů v pořadí, v jakém jsou definované výše. Vše je opět odděleno mezerou. Na rozdíl od formátu WKT se ve standardu OFF při konstrukci ploch objevuje první vrchol pouze jednou.

```
OFF             //hlavicka
4 4 0           //pocet vrcholu, ploch
1 1 1           //zacatek definice vrcholu
-1 1 -1
-1 -1 1
1 -1 -1
3 0 1 2         //zacatek definice ploch
3 0 2 3
3 0 3 1
3 3 2 1
```

Čtení a zápis formátu OFF je nativní vlastností knihovny CGAL, nikoliv však databáze. Z tohoto důvodu byla v jazyce Python vytvořena metoda *readOFF()*, která čte OFF soubor ze zdrojového umístění a zapisuje do databáze ve formátu *PolyhedralSurface*. Metoda *writeOFF()* provádí export z databáze do souboru OFF a ukládá jej do cílového adresáře. Způsob fungování těchto konverzních metod je popsán v algoritmech č. 5 a 6. Implementace těchto metod je součástí zdrojového kódu v digitální příloze této práce.

```
   input  : zdrojový adresář
   output: databázová tabulka
 1 readOFF
 2 for pro každý soubor v adresáři do
 3 |   přečti hlavičku a nastav indexy řádků
 4 |   parsuj řádek → ulož pole vrcholů, ulož pole ploch
 5 |   for každou plochu do
 6 |   |   konstruuj WKT pro Polygon
 7 |   end
 8 |   konstruuj WKT pro PolyhedralSurface
 9 |   sql: INSERT geometrie do databázové tabulky
10 end
```

Algoritmus 5: Schéma procedury readOFF

```
   input  : databázová tabulka
   output: cílový adresář
 1 writeOFF
 2 sql: SELECT * FROM zdrojovaTabulka
 3 for pro každý záznam (Polyhedron) v tabulce do
 4 |   parsuj WKT geometrii na jednotlivé plochy → ulož pole ploch
 5 |   ulož pole unikátních vrcholů → sestav OFF seznam vrcholů
 6 |   for každou plochu do
 7 |   |   hledej index vrcholu → sestav OFF seznam ploch
 8 |   end
 9 |   vytiskni hlavičku
10 |   vytiskni seznam vrcholů
11 |   vytiskni seznam ploch
12 end
```

Algoritmus 6: Schéma procedury writeOFF

6. Datové zdroje

Obsah této kapitoly je věnován popisu dostupnosti datových zdrojů obsahujících 3D modely městské zástavby, jejich porovnání, zhodnocení a finální výběr.

6.1 Dostupnost datových zdrojů

Pořizování 3D modelů se v dnešní době stalo standardem a téměř každá větší metropole má takovýto model k dispozici, ať už volně přístupný veřejnosti, či za vyžádání za poplatek. Na internetu lze nicméně najít řadu volně dostupných zdrojů. Převážně se jedná o městské organizace, které mají na starosti plánování a urbanismus, případně weby, více či méně podobné geoportálům, známým z prostředí Česka. Dalším neméně důležitým zdrojem jsou akademická pracoviště. Zde se však může objevit problém, že ve vědeckých pracích publikovaných na těchto pracovištích jsou použita data, která běžně dostupná nejsou, případně se jedná o uměle vytvořené modely. Během hledání vhodného datového zdroje byly postupně otestovány datové modely měst Berlína, Toronta, Floridy a Prahy.

6.2 Datové formáty

V klasickém pojetí GIS, kde je drtivá většina datových sad ve 2D, jsou již zažité standardy pro přenos a ukládání těchto dat. Po přidání třetího rozměru je však situace diametrálně odlišná. Díky tomu, že se doména 3D modelování prolíná mezi více obory, existuje poměrně široká škála formátů dat. Existují typicky CAD zaměřené datové sady ve specifických CAD formátech, které nejsou GIS nástroji obecně podporované. Některá data lze najít ve formátu Shapefile, který je však možné dělit na dva typy, Polygon a Multipatch. Příslib sjednocení je vkládán do formátu CityGML, který byl navrhnut přesně pro potřeby 3D modelů měst.

Mimo to se data samozřejmě liší také v dalších parametrech. Podle použité metody akvizice dat se modely liší svou kvalitou zaměření. Některé modely jsou pouze blokové, vytvořené z katastrálních map. V lepším, případě se jedná o foto-

grammetricky zaměřené budovy, které vznikají náročným pozemním skenováním. Některé modely jsou generované i z dat laserového skenování, ať už pozemního či leteckého. To vše dohromady určuje výslednou míru detailu. V neposlední řadě je nutné zmínit, že pro potřeby této práce je velmi důležitým aspektem geometrická validita. Při testování vybraných vzorků se ukázalo, že modely jsou generovány spíše pro potřeby vizualizací a nesplňují parametry pro analytické použití.

6.2.1 CityGML standard

Potřeba vytvoření pevného rámce pro 3D modelování městského prostředí vedla organizaci *Open Geospatial Consortium* (OGC) k definování mezinárodního standardu s názvem CityGML. Za projektem stojí skupiny z akademické sféry, konkrétně *University in Bonn* a *Technical University in Delft*. Samotný standard je vytvářen již od roku 2008, avšak až s poslední verzí s rozšířenými vlastnosti se dostává více do obecného povědomí (Open Geospatial Consortium, 2017).

Formát CityGML byl vytvořen pro ukládání nebo sdílení a jak sám název napovídá, klade důraz na město a jeho prvky. Jedná se o modifikaci XML formátu, který umožňuje hierarchické členění prvků. Už méně vhodný je na správu a pro použití v analytických nástrojích, protože takové funkce tento formát nenabízí. Datový model je postavený na takových základech, aby vyhověl širšímu spektru použití, tedy nejen v geografii, nýbrž také pro plánování nebo navigaci a ukládá nejenom geometrické prvky, ale také sémantická data a textury. Mezi dostupné prvky patří budovy, městský mobiliář, model terénu, vodní plochy a dopravní sítě. Zároveň umožňuje ukládat objekty ve více úrovních rozlišení a obsahuje pět těchto úrovní, od blokového modelu po detailní texturovaný model. CityGML také definuje topologii, tedy že jednotlivé objekty musí splňovat zadaná kritéria.(Kolbe, 2009).

Čtení CityGML databáze

Vzhledem ke komplexnosti datového formátu CityGML není úplně triviální tento formát číst standardními GIS nástroji. Existuje však nadstavba 3DcityDB pro PostreSQL databázi, kterým je možné data načíst. Balík je možné stáhnout z otevřeného repozitáře GitHub. Níže uvádím postup instalace a importu dat, jelikož oficiální dokumentace je v tomto ohledu nedostatečná.

Následující sadou příkazů v shellu dojde ke zkopírování GitHub repozitářů a vytvoření struktury v databázi:

```
# naklonuj repozitare
git clone https://github.com/3dcitydb/3dcitydb.git 3dcitydb
```

```
git clone https://github.com/3dcitydb/importer-exporter.git importer-exporter
# nainstaluj ant, pokud neni k~dispozici
sudo apt install ant
ant run
# vytvor strukturu v~databazi
sudo -u postgres psql postgres -f CREATE_DB.sql
```

Soubor **project.xml**, který definuje parametry připojení k databázi, musí mít následující strukturu:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<project xmlns="http://www.3dcitydb.org/importer-exporter/config">
  <database>
    <connections>
      <connection id="foo" initialSize="0">
        <description>New connection</description>
        <type>PostGIS</type>
        <server>127.0.0.1</server>
        <port>5432</port>
        <sid>postgres</sid>
        <user>postgres</user>
        <password>abc</password>
        <savePassword>false</savePassword>
      </connection>
    </connections>
  </database>
</project>
```

Posledním shellovým příkazem provedeme import:

```
# naklonuj repozitare
java -jar -Xms128m -Xmx4096m lib/3dcitydb-impexp.jar \\
        -shell -import /home/ivos/data/berlin
```

Po importu dat 3D modelu Berlína byla provedena analýza kvality geometrie a 3D modelů. Jednotlivé polygony, které jsou hlavním stavebním prvkem celé databáze, se navzájem protínaly nebo vytvářely „non-manifoldní“ struktury, které nejsou při modelování objemových modelů akceptovatelné. Stejný problém byl nalezen i u testovacího vzorku ulice v německém Frankfurtu. Z těchto důvodů bylo od využití testovaných CityGML modelů upuštěno.

Výše zmíněná zjištění potvrzuje i projekt, který si dal za cíl otestovat několik desítek dostupných datových sad CityGML. Rozsáhlé šetření vedlo k výsledku, že geometricky validní modely bez chyb jsou velmi vzácné a pokud existují, jsou to velmi zjednodušené objekty typu kvádrových bloků (Biljecki et al., 2016).

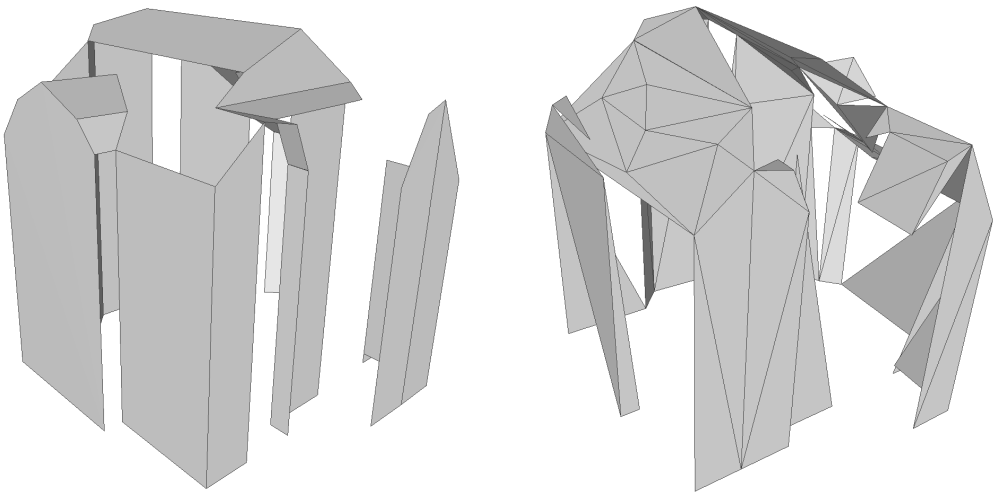
6.2.2 Shapefile

Shapefile je v oboru GIS velmi oblíbeným formátem. Přestože se primárně jedná o formát proprietárního softwaru, jeho standard je otevřený a existuje

mnoho pluginů nebo databázových systémů, které umí tento formát číst nebo do něj zapisovat, což je nesporná výhoda. Tento formát byl vytvořen hlavně pro snadné sdílení, protože definice geometrie, atributová složka a metadata o souřadnicovém systému jsou ve 4 povinných souborech. Přestože je shapefile určen převážně pro použití ve 2D a nativně nepodporuje specifikaci pro 3D objemové modely, je schopný ukládat i výškovou Z souřadnici a modely v něm uložené jsou tvořeny, podobně jako u CityGML, množinou polygonů.

6.2.3 3D model Prahy

Data ve formátu shapefile byla nalezena pro Prahu a Toronto. Import dat ve formátu shapefile proběhl bez problému, ovšem po exploratorní analýze dat je nutné konstatovat, že trpí stejným problémem, jako data z Berlína. Obsahovaly velké množství konfliktních polygonů, tedy topologických chyb, takže z dat nakonec nebylo možné zrekonstruovat geometricky korektní objemový model bez děr a bez vzájemně se protínajících ploch. Problém nastal hlavně u tvarově komplexnějších budov. Bylo provedeno více variant iterací rekonstrukce modelu, např. se snahou o eliminaci některých prvků, ovšem ani to nevedlo k úspěšné rekonstrukci. Níže na obr. 6.1 je vyobrazen pokus o rekonstrukci budovy s eliminováním konfliktních geometrií.



Obrázek 6.1: Ukázka topologicky nevalidní geometrie v datasetu 3D modelu Prahy. Zdroj: vlastní tvorba

Přestože všechny otestované veřejně dostupné datasety obsahovaly geometrické chyby, případně byly velmi zgeneralizované, jako nejvhodnější ze všech se pro další použití jevila data z Prahy. I přes počáteční generalizaci mají pražská data v porovnání s ostatními nejvyšší míru detailu. Proces předzpracování

Atribut	Datový typ	Význam
Geometrie	PolygonZ	binárně uložená geometrie
Aktualizace	integer	rok a měsíc poslední aktualizace
ML	string	mapový list
Typ	string	typ plochy
ID_BUD	integer	jedinečné ID budovy
Skupina	integer	kód typu plochy

Pozn: [a] Jednotlivé typy ploch jsou popsány v textu níže.

Tabulka 6.1: Atributy 3D modelu Praha.

dat a vytváření validních objemových modelů je podrobně rozebrán v podkapitole č. 7.2.

Dataset 3D modelu Prahy je veřejně dostupný na Geoportálu Praha (www.geoportalpraha.cz), jeho pořizování probíhá již od roku 2001 a poslední aktualizace byla dle zmíněného webu provedena v roce 2011. Základním prvkem při tvorbě modelu byly zlomové body střech, jež se získávaly fotogrammetrickým vyhodnocením leteckých měřičských snímků. Fakt, že model není geometricky zcela korektní, je způsoben metodou prost-processingu dat, která byla poloautomatická. Data jsou tedy bez problému využitelná pro základní vizualizaci, pro analytické potřeby už je však nutné přistoupit k přemodelovaní. Udávaná chyba modelu je maximálně 50 cm (Geoportal Praha, 2015).

Atribut *typ plochy* je v datech rozdělen do devíti kategorií:

- dílčí plocha střešní kruhové plochy,
- komín,
- šikmá střešní plocha,
- svislá obvodová stěna,
- vikýř / střešní nástavba,
- vodorovná střešní plocha,
- výtah / větrání / klimatizace,
- význačná věž na střeše,
- základová deska.

7. Implementace navrženého postupu

Tato kapitola se věnuje samotné implementaci navržené metody generalizace. Implementace bude používat navržený aplikační rámec, který byl představený v kapitole č. 5. Implementace začíná importem a předzpracováním zdrojových dat a dále vytvořením validních objemových modelů. Pokračuje zavedením topologické struktury, po které následuje samotný proces generalizace. Proces končí post-processingem výsledků a vytvořením víceúrovňové databáze. Zdrojové kódy použitých metod a pomocných funkcí jsou k dispozici v digitální příloze této práce, případně ve veřejném repozitáři GitHub na adrese `https://github.com/pivosak/3D-building-generalization`.

7.1 Instalace software

Vzhledem k použití linuxového operačního systému je instalace používaného software celkem přímočarou záležitostí. Na rozdíl od prostředí Windows je v Linux možné využít takzvaných závislostí (*dependency management*), tedy programu, který zajistí nejen instalací zvolených balíků, ale také ostatních závislostí. Při instalaci software je důležité používat uživatelský účet s dostatečnými právy - v tomto případě je to administrátorský účet *superuser*, který se vyvolává příkazem *sudo*.

Následujícími příkazy dojde k nainstalování databázového systému PostgreSQL, prostorové nadstavby PostGIS a balíku pro výpočetní geometrii CGAL z veřejného repozitáře systému Ubuntu. V rámci instalace balíku CGAL by mělo automaticky dojít i k pořízení balíků Boost pro podporu multi-threadingu a knihoven GMP a MPFR, které jsou nutné pro precizní výpočty nad reálnými čísly s pohyblivou desetinnou čárkou a vyhnutí se zaokrouhlovacím chybám.

```
sudo apt-get install postgresql-9.5-postgis-2.2
sudo apt-get install libcgal-dev
```

7.2 Import dat a preprocessing

Po instalaci databáze je k dispozici základní databázové schéma *public*, které je možné bez problému využít, protože v tuto chvíli je řízení uživatelských práv irelevantní. Import dat a všechny následující databázové dotazy budou probíhat v tomto schématu.

Zdrojová data 3D modelu Prahy jsou k dispozici ve formátu *Shapefile Polygon Z*. Vzhledem k povaze formátu shapefile, který není tvořen jedním souborem, ale sadou minimálně čtyř souborů, je vhodné všechny soubory umístit do jedné složky. Aplikace pro import shapefile přečte vstupní data a zapíše je do stejně strukturované databázové tabulky. Atributy se uloží jako textové řetězce nebo číselné typy. Geometrická reprezentace je uložená binárně a lze ji číst pomocí dostupných databázových funkcí.

```
shp2pgsql /data/Praha73 public.praha73 |
sudo -u postgres psql postgres
```

V tuto chvíli jsou data spolu s geometrií uložená v databázi a provede se první krok předzpracování dat. Zdrojová data netvoří objemový model a jejich datová struktura jej ani nepodporuje. Jedná se pouze o jednotlivé polygony, které se mohou vzájemně protínat nebo tvoří nesouvislý povrch modelu, což není žádoucí. V rámci experimentování s geometriemi bylo provedeno několik variant pokusů o rekonstrukci budov ze všech dostupných ploch. Zdrojová data však negarantují topologickou správnost a obsahují velké množství chyb, které v konečném důsledku znemožňují vytvoření validního objemového modelu. Vzhledem k faktu, že z podstaty metody je objemový model nutným předpokladem, bylo učiněno rozhodnutí vytvořit objemové modely pomocí projekce modelu do roviny xy a současného uložení informací o výškových poměrech. Z těchto dat se zrekonstruuje blokový model, který konstrukci objemové struktury umožní.

7.2.1 Rekonstrukce 3D modelu

Pro tento účel byla napsána vlastní procedura *repair()* v jazyce Python, na jejímž vstupu je databázová tabulka s nevalidními vstupními daty a výsledkem jsou zrekonstruované validní orientované plochy. Pomocí PostGIS funkcí *ST_Force2D*, *ST_MakeValid* a *ST_Union* se ze vstupních polygonů vytvoří obtisk geometrií do roviny xy a sestaví se výsledný půdorysný polygon pro každou budovu. Tyto půdorysné polygony se uloží do paměti do vícerozměrného pole, spolu s minimální (*minz*) a maximální (*maxz*) hodnotou výškové souřadnice (ř. 4-8). Dále se vypočte a nastaví negativní orientace půdorysného polygonu, aby jeho normála

směřovala směrem k zemi, a posune se do výškové hladiny minima *minz* (ř. 20-27). Poté se inkrementálně prochází dvojice vrcholů půdorysného polygonu a generují se nové obvodové plochy se správnou orientací tak, aby na společné hraně byla orientace hran opačná (ř. 30-39). Posledním krokem je sestavení ploché střechy posunutím půdorysného polygonu do výškové hladiny *maxz* a změnou orientace (ř. 30-39). Zrekonstruované modely se uloží zpět do databáze (ř. 44-47). Využité pomocné Python funkce jsou definované v souboru *helpers.py*.

```
1  ### repair.py
2  def repair(inputTable,targetTable):
3    from helpers import *
4    sql=""" with db as (select * from """+inputTable+""" where typ='svislastena')
5    select A.id_bud, zmin, zmax, ST_AsText(ST_Union(geom)) as geom from (
6    select id_bud, ST_MakeValid(ST_Force2D(geom)) as geom from db ) A
7    JOIN (select id_bud, MAX(ST_ZMax(geom)) as zmax, MIN(ST_ZMin(geom)) as zmin
8    from db group by 1) B ON A.id_bud=B.id_bud GROUP BY 1,2,3 ; """
9    conn=doConn() #pripojeni k~dtb
10   cur = conn.cursor()
11   cur.execute(sql) #dotaz na zdrojova data
12   rows = cur.fetchall()
13   dropTable(cur, targetTable)
14   cur.execute("CREATE TABLE "+targetTable+" (id_bud int, geom geometry);")
15   for row in rows:
16     id = int(row[0])
17     polygons = []
18     base = parse2Dpoly(row[3]) #parsuj zdrojova data
19     #pokud orientace nesmeruje negativne, udelej inverzi
20     if ( orientation(base) == 1):
21       base = base[::-1]
22     zmin = float(row[1]) #zapamatuj si vysku zakladu
23     zmax = float(row[2]) #zapamatuj si vysku strechy
24     basePoly = copy.deepcopy(base) #zaklad budovy
25     for v~in basePoly:
26       v.append(zmin)
27     polygons.append(listToPolyZ(basePoly))
28     i=0
29     #prochazej dvojice
30     for coor in base:
31       v1 = base[i]
32       v2 = base[i+1]
33       #generuj zdi budovy
34       wallPoly = liftUp(base[i],base[i+1],zmin,zmax)
35       wallPoly = listToPolyZ(wallPoly)
36       polygons.append(wallPoly)
37       i+=1
38       if (i+1)==len(base):
39         break
40     roofPoly = copy.deepcopy(base[::-1]) #strecha budovy
41     for v~in roofPoly:
42       v.append(zmax)
43     polygons.append(listToPolyZ(roofPoly))
44     for p in polygons:
45       sql="""INSERT INTO "+targetTable+" (id_bud, geom)
46         VALUES ("+str(id)+", ST_GeomFromText('"+p+"'));"""
47       cur.execute(sql) #uloz orientovave polygony do dtb
```

7.3 Struktura programu

Tato část implementace již pracuje s datovými strukturami knihovny CGAL, je psaná v jazyce C++ a obsahuje třídy a konstruktory, které obsluhují jednotlivé části celého procesu. Proces je rozdělen do jednotlivých bloků:

- Inicializace a naplnění half-edge datové struktury.
- Načtení původních modelů do databáze a kontrola topologie.
- Generování shluků budov.
- Generalizace a agregace.
- Načtení zjednodušených modelů do databáze a kontrola topologie.
- Vizualizace.

7.3.1 Datové třídy

Vzhledem ke komplexnosti knihovny CGAL je dále uveden seznam používaných datových typů a jejich krátké vysvětlení. Oproti standardním datovým typům jsou typy velmi specifické pro úlohy výpočetní geometrie.

Kernel *Jádro.* Třídy knihovny CGAL podporují několik typů výpočetního jádra. Na nejvyšší úrovni je možné tyto typy rozdělit na exaktní jádra a neexaktní jádra. Rozdíl je v zaokrouhlování desetinných čísel. Neexaktní jádra provádějí zaokrouhlení, což vede k rychlejším výpočtům, ale na druhé straně je jádro náchylné k chybě, kdy například při kontrole toho, zda jsou body v rovině, dojde díky přílišnému zaokrouhlení k chybným výsledkům. Exaktní jádro číslo co nejlépe aproximuje, jádro je tedy výpočetně pomalejší, ale tím více stabilní. Tato implementace využívá exaktního jádra *Exact_predicates_exact_constructions_kernel*, které nativně podporuje číselný typ *double* - standardní typ pro čísla s pohyblivou desetinnou čárkou.

Polyhedron_3 Reprezentuje nesouvislý povrch mnohostěnu nebo jeho souvislý uzavřený model a skládá se z vrcholů, hran, ploch a incidenčních vztahů mezi těmito primitivy. Pomocí nativních metod jde zjišťovat např. validita, uzavřenost nebo počet primitiv. Tento datový typ je nutné použít při konstrukci modelu. Součástí této třídy je datová struktura halfedge, a tak je každá hrana

složena ze dvou půlhran (halfedge). Metody datové reprezentace halfedge podporují procházení této struktury a dotazy na incidenční nebo okolní primitiva a lokální úpravy povrchu.

Nef_polyhedron_3 Speciální typ polyhedronu. Je vždy uzavřený a definovaný sadou průniků a doplňků z konečné množiny poloprostorů. Jako jediný typ podporuje booleovské operace (průnik, sjednocení, rozdíl a doplněk) a afinní transformaci pro škálování, rotaci a posunutí.

7.4 Vlastní metody

Pro zjednodušení kódu a struktury programu byly navrženy hlavní metody, které jsou používány vícenásobně a jsou parametrizované, a tak je možno těmto třídám na vstup předávat odlišná data nebo parametry. Metody následně vykonají požadovanou funkci a vrátí zpět výsledek. Návratový typ tříd se liší podle použití. Níže jsou popsány pro běh programu nejdůležitější metody. Zbylé pomocné funkce jsou k dispozici v digitální příloze této práce.

- *Build_surface* -

  ze vstupních vrcholů a ploch vytváří objemový model (C++)

- *Build_cube* - generuje strukturní element pro generalizaci (C++)
- *Merge_faces* - slučuje koplanární plochy objemového modelu (C++)
- *Get_direction* - vypočte hlavní směr natočení modelu (C++)
- *Build_topology* - vytváří topologickou strukturu a kontrolní mechanismy (Python, SQL)
- *Cluster_neighbours* - hledá shluky z dvojic sousedů (Python, SQL)
- *ReadOFF, WriteOFF* - import/export 3D modelu ve formátu OFF do/z databáze (Python, SQL)

7.4.1 Build_surface

Jak je zřejmé z názvu, tato třída obsluhuje vytváření objemových modelů do nativního CGAL datového typu. Na vstupu do této metody je trojrozměrný vektor souřadnic *coors* (x,y,z) a vícerozměrný vektor ploch *facets*, který nese definice ploch. Plochy jsou definovány indexy bodů v tom pořadí, v jakém

jsou předávány ve vektoru *coors*. Obě tyto datové struktury jsou zkonstruovány pomocí dat z databáze. Na začátku běhu se inicializuje halfedge datové struktura (HDS), nad kterou je možné volat metody nativní CGAL třídy *Polyhedron_incremental_builder_3*, která umožňuje naplnit strukturu HDS vstupní geometrií. V prvním kroku se metodou *begin_surface()* vytvoří nový prázdný povrch a přes konstruktor trojrozměrného bodu *Point_3* se struktura HDS naplní všemi vstupními vrcholy, a to pomocí metody *add_vertex()*.

Nyní jsou v HDS definovány vrcholy a následuje vkládání jednotlivých ploch pomocí konstruktoru *begin_facet* - tomu se posílají indexy požadovaných vrcholů v pořadí, v jakém tvoří hranici plochy. Před přidáním plochy do povrchu se v HDS zkontroluje, zda má vkládaná plocha správnou orientaci a splňuje podmínky opačně orientovaných halfedge hran. V případě nesprávné orientace program provede inverzi orientace plochy a teprve poté ji vloží do povrchu. Po skončení běhu třída navrací orientovaný objemový model datového typu *Polyhedron_3*.

```
// Main.cpp
Build_surface(vector<double> coors, vector<double> facets) {}
    void operator()( HDS& hds) {
        CGAL::Polyhedron_incremental_builder_3<HDS> B( hds, true);
        //Zacni konstrukci povrchu
                B.begin_surface( coors.size()/3, 0, 0);
                for (auto row : coors) {
                        //Pridej 3D bod do HDS
                        B.add_vertex( Point_3( row[0], row[1], row[2] ) );
                }
                //Pridej plochy
        for (auto row : facets) {
            std::vector<int> rowReverse;
            rowReverse = row;
            std::reverse(rowReverse.begin(), rowReverse.end());
                        /Test orientace plochy
            if (B.test_facet(row.begin(),row.end())) {
                B.begin_facet();
                for (auto col : row) {
                    B.add_vertex_to_facet( col );
                }
                B.end_facet();
            }
            //V pripade chyby zmen orientaci plochy
            else if (B.test_facet(rowReverse.begin(),rowReverse.end())) {
                B.begin_facet();
                for (auto col : rowReverse) {
                    B.add_vertex_to_facet( col );
                }
                B.end_facet();
            }
        }
        //Uzavri povrch pro zmeny a~uloz do pameti
        B.end_surface();
        B.remove_unconnected_vertices();
    }
```

7.4.2 Build_cube

Na podobném principu je založena i třída Build_cube modelující krychli, která bude sloužit jako strukturní element pro morfologické operace. Krychle je generována jako jednotková a její těžiště je umístěno do počátku soustavy souřadnic. Souřadnice jednotkové krychle jsou uvnitř třídy explicitně zadané a nejde je měnit. Velikost krychle určuje parametr *elementSize*, tedy velikost strany krychle, což dovoluje dynamicky měnit míru zvolené generalizace. Případná rotace krychle je řešena až za běhu programu pomocí afinní transformace. Návratovým typem je opět *Polyhedron_3*.

```
// Main.cpp
Build_cube(double elementSize) {}
    void operator()( HDS& hds ) {
        CGAL::Polyhedron_incremental_builder_3<HDS> B( hds, true);
        static const double cubeVertexArr[][3] = {
            {-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,-1.0},{-1.0,1.0,-1.0},
        };
        std::vector<std::vector<int> > cubeFacetArr = {
            {0,1,5,4},{1,2,6,5},{2,3,7,6},{3,0,4,7},{3,2,1,0},{4,5,6,7} };
        B.begin_surface( 8, 6, 24);
        for ( int i~= 0; i~< sizeof(cubeVertexArr)/3.0; i++ ) {
             B.add_vertex( Point_3(
                cubeVertexArr[i][0]*elementSize/2.0,
                cubeVertexArr[i][1]*elementSize/2.0,
                cubeVertexArr[i][2]*elementSize/2.0 ) );
        }
        for (auto row : cubeFacetArr) {
            if(B.test_facet(row.begin(),row.end())) {
                B.begin_facet();
                for (auto col : row) {
                    B.add_vertex_to_facet( col );
                } B.end_facet();
                }
        }
        B.end_surface();
    }
```

7.4.3 Merge_faces

Tato třída hledá na vstupním modelu koplanární plochy a provádí jejich sloučení, čímž zajišťuje maximální možnou úsporu dat, která definují model. Na vstupu této třídy je objemový model datového typu *Polyhedron_3*. Program zpočátku projde všechny plochy ve vstupním modelu a vypočítá jejich normálové vektory. Dále se iteruje přes všechny halfedge hrany v modelu a probíhá sada kontrol, zda může dojít k odstranění hrany. Předpokládají se následující podmínky:

1. plochy incidenční k halfedge jsou koplanární,

2. ke každému vrcholu halfedge musí existovat alespoň tři incidenční halfedge,

3. plochy incidenční s halfedge nesmí být trojúhelníky.

Při splnění všech podmínek je možné halfedge vymazat z HDS struktury, a tím automaticky dojde ke sloučení incidenčních ploch do jedné. Takto se iterativně projdou všechny prvky v HDS a dojde ke sloučení všech možných ploch. Třída vrací upravenou geometrii stejného datového typu jako na vstupu (Ledoux, 2014).

```
// Main.cpp
void Merge_faces(Polyhedron P) {
 P.normalize_border();
 if(!P.size_of_border_halfedges()) {
  //Vypocti normalovy vektor vsech ploch
  std::transform(P.facets_begin(),P.facets_end(),P.planes_begin(),compNormal());
  bool stillCoplanar = true;
  std::vector<Polyhedron::Halfedge_handle> Hh;
  while (stillCoplanar) {
   stillCoplanar = false;
   for (Polyhedron::Halfedge_iterator Iter = P.halfedges_begin();
    Iter != P.halfedges_end(); ++Iter) {
    //Zkontroluj koplanaritu ploch incidencnich k~halfedge
    if (isCoplanar(Iter)){
     Polyhedron::Halfedge_handle deleteHE = Iter;
     //Hledej vrchol, ktery bude mozne odstranit
     while (CGAL::circulator_size(deleteHE->vertex_begin()) < 3)
      deleteHE = deleteHE->next();
     if ( checkForHoles(Iter) ) {
      while (CGAL::circulator_size(deleteHE->opposite()->vertex_begin()) < 3)
       //Hranu je mozne odstranit pouze pokud neni v~incidenci s~trojuhelniky
       if (deleteHE->facet_degree()>3 && deleteHE->opposite()->facet_degree()>3)
        //Zhrouceni hrany do vrcholu
        deleteHE = (P.join_vertex(deleteHE))->next()->opposite();
       else
        break;
      if (CGAL::circulator_size(deleteHE->opposite()->vertex_begin()) < 3)
       P.erase_center_vertex(deleteHE->opposite());
      else
       P.join_facet(deleteHE);
      stillCoplanar = true;
      break;
}}}}}}
```

7.4.4 Get_direction

Metoda vypočítá hlavní směr natočení budovy (nebo shluků budov) kolem osy z, proto jsou na jejím vstupu všechny halfedge orientované ve vodorovném směru. Směr budovy je důležitý pro nastavení rotace strukturního elementu generalizace. Metoda je součástí C++ implementace. Algoritmus prochází přes všechny hrany a do proměnných Ax,Ay,Az a Bx,By,Bz hodnoty souřadnic počátečního A a koncového bodu B. Směr se počítá pouze pro hrany ležící v rovině xy, používá se proto podmínka stejné z souřadnice. Vzhledem k faktu, že strukturním elementem je

krychle, má smysl počítat úhel pouze na intervalu $\langle 0, \frac{\pi}{2})$, proto pokud je hrana kolineární s osou x nebo y, je výsledný směr *theta* roven nule. V ostatních případech je úhel mezi vektorem $\overrightarrow{AB}$ a osou x vypočten pomocí funkce *atan*2 a převeden do prvního kvadrantu. Pro každou halfedge se navíc vypočte její délka. Dvojice úhel - délka je uložena do pomocného vektoru *degreeDist*. Na konci algoritmu se hledá směr s největší přiřazenou délkou a tento výsledný úhel je také návratovou hodnotou této metody.

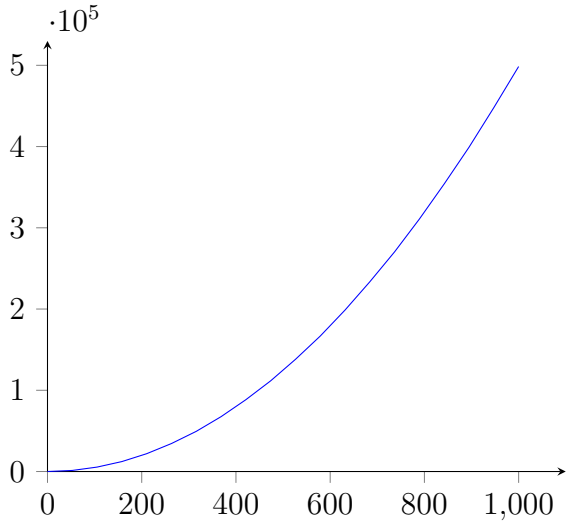
```
// Main.cpp
double Get_direction(std::vector<double> halfedges) {
  std::vector<std::vector<double> > degreeDist; //sem ukladej mezivysledky
  for (auto he : halfedges) {  //iteruj pres vsechny halfedge
    double Ax = he[0][0], Ay = he[0][1], Az = he[0][2];
    double Bx = he[1][0], By = he[1][1], Bz = he[1][2];
    double degree;
    if (Az==Bz) { //pocitej pouze pro vodorovne halfedge
      if (Ax==Bx or Ay==By) { theta=0; //kolinearni v~osou x nebo y
      } else {
        double theta = atan2(Bx - Ax, Ay - By); //vypocet uhlu
        if (theta < 0.0)
          theta += M_PI;
        if (theta >= M_PI/2.0)
          theta = theta - M_PI/2.0;
        double distance = sqrt(pow(Ax-Bx,2.0) + pow(Ay-By,2.0));
        int l=0;
        bool notFound = true;
        for (auto pair : degreeDist) { //pricti delku ke smeru
          if (degreeDist[l][0] == theta) {
            degreeDist[l][1] += distance;
            notFound = false;
          }
          l++;
        }
        if (notFound) { //pokud je to novy smer, pridej ho do vektoru
          std::vector<double>  newPair;
          newPair.push_back(theta);
          newPair.push_back(distance);
          degreeDist.push_back(newPair);
  }  }  }  }
  int m=0;
  int idx = 0;
  double max = 0.0;
  for (auto pair : degreeDist) { //Hledej uhel s~nejvetsi kumulovanou delkou
    if (pair[1] > max) {
      max = pair[1];
      idx = m;
  }  }
  return finalTheta = degreeDist[idx][0];
}
```

7.4.5 Build_topology

Metoda buduje topologickou strukturu nad 3D objemovými modely uložené v PostgreSQL databázi, zjišťuje topologické vazby a ukládá se pro další použití.

Tato metoda má pouze jeden vstupní parametr, a tím je název databázové tabulky se zdrojovými daty, nad kterými bude vytvořena topologická struktura. Výstupem metody je relační tabulka s definicí vzájemných topologických vztahů mezi jednotlivými objekty.

Základním principem topologie je zjištění vzájemných prostorových vztahů všech objektů v databázové tabulce. Pro získání informace o všech vztazích, včetně těch disjunktních, je však nutné porovnat vztah každé kombinace dvojic v modelu. Počet takových kombinací se vypočítá jako kombinace bez opakování $C_2(n) = \binom{n}{2} = \frac{n!}{2!(n-2)!}$. Jak je vidět na obr. č. 7.1, počet kombinací roste zhruba s polovinou druhé mocniny. Pro 1 000 budov by to znamenalo zjišťovat 499 500 kombinací.



Obrázek 7.1: Závislost počtu kombinací dvojic na počtu budov

Vzhledem k charakteru rozmístění budov v prostoru má smysl zjišťovat vztahy pouze mezi blízkými objekty. Nad tabulkou bude zaveden prostorový index GIST, který do speciální stromové struktury indexuje ohraničující kvádry budov. Tento prostorový index pak umožňuje dotaz na prostorový vztah ohraničujících kvádrů dvou objektů. Díky stromové struktuře není nutné provádět porovnání těch dvojic, které spolu prostorově nesouvisí, což zkrátí výpočetní dobu na minimum. Prostorový index se nad tabulkou zavede SQL příkazem:

```
CREATE INDEX models_index ON models USING GIST (geometry);
```

Pomocí speciálních prostorových operátorů je pak možné provádět dotazy a hledat takové dvojice, které mají prostorový průnik (&&&) ohraničujících kvádrů nebo jsou od sebe vzdálené do určené limitní vzdálenosti (*ST_DWithin*, např. 10 metrů).

```
SELECT A.ID_a, B.ID_b
FROM models as A, models as B
```

```
WHERE
        A.geometry && A.geometry = 't' OR
        ST_DWithin(A.geometry, B.geometry, 10.0);
```

Výše zmíněným dotazem se tedy zjistí dvojice, které spolu mohou prostorově souviset. U dvojic nezáleží na pořadí a budou se porovnávat jen takové dvojice, u kterých ID první budovy z dvojice bude menší než ID druhé budovy. Na této výrazně malé podmnožině všech možných kombinací bude provedena operace prostorového průniku (*ST_Intersection*) a výpočtem dimenze průniku (*ST_Dimension*) se zjistí výsledný topologický vztah dvojice. Uloženy budou pouze vztahy s existující prostorovou dimenzí nebo disjunktní vztahy v určené limitní vzdálenosti. Dimenze může nabývat hodnot: -1 (disjunktní), 0 (vrchol), 1(hrana), 2 (plocha) nebo 3 (objem). Relační tabulka vztahů má následující strukturu, první dva sloupce jsou vyhrazené identifikátorům budov, ve třetím sloupci je hodnota dimenze průniku.

```
CREATE TABLE budovy_vztahy (ID_a integer, ID_b integer, dimension integer);
```

Celá metoda je uloženou procedurou, která se spustí pokaždé, když je potvrzena transakce (commit) nad spravovanou tabulkou. Následuje přehled této procedury.

```
INSERT INTO models_relations
SELECT
        A.id as ID_a,
        B.id as ID_b,
        ST_Dimension((ST_3DIntersection(A.geom, B.geom))) as topo_dimension
FROM models A, models B
WHERE  A.geometry &&& B.geometry = 't' and A.id_bud>B.id_bud ;
```

7.4.6 Cluster_neighbours

Tato metoda implementuje hladový algoritmus hledání shluků budov, které budou potenciálními kandidáty k agregaci při samotném procesu generalizace. Na vstupu metody je seznam všech ID dvojic, mezi kterými existuje topologická vazba nebo jsou od sebe vzdálené do daného limitu, doplněný o seznam všech jednotlivých ID budov. Vstupní data se načítají z databázové tabulky topologických vztahů. Samotná metoda je napsaná v jazyce Python. Na začátku je definované prázdné pole shluků *Clusters[]*. Algoritmus v první iteraci prochází seznamem všech dvojic *Groups[]*. Pro každou dvojici provádí kontrolu, zda se alespoň jeden z prvků v *Clusters[]* již nenachází. Pokud ano, znamená to, že dvojice patří do testovaného shluku a je do tohoto shluku sloučena. Pokud se nenajde relevantní shluk, je dvojice zařazena jako nový shluk. V druhé a dalších iteracích je seznam dvojic nahrazen seznamem shluků a proces shlukování probíhá na stejném principu. Algoritmus končí ve chvíli, kdy jsou všechny budovy rozdělené do

disjunktních shluků, tedy dokud dochází k přerovnávání budov mezi shluky. Na počátku algoritmu se zjistí unikátní počet budov a algoritmus se zastaví ve chvíli, kdy počet prvků v shlucích je roven unikátnímu počtu budov. Jen tehdy je zaručeno, že shluky jsou disjunktní a každý prvek je zařazen právě v jednom shluku. Hlavní motivací této funkce je optimalizace výpočetního procesu generalizace, kterou je možné tímto paralelizovat.

```
## Cluster_neighbours.py
def Cluster_neighbours(Groups):
  flat = []
  for Group in Groups:
    for item in Group:
      flat.append(item)
  aim = len( list(set(flat)) ) #unikatni pocet budov, podminka zastaveni

  Clusters = [] #inicializace prazdneho pole shluku
  Clusters.append(list(set(Groups[0]))) #prvni dvojice je jiste novy shluk

  n=0
  while True:
    if (n>0):  #v druhe a~dalsi iteraci prochazej misto dvojic seznam shluku
      Groups = Clusters
      Clusters=[] #vynuluj seznam shluku
      Clusters.append(list(set(Groups[0])))

    for Group in Groups:
      Group = list(set(Group))  #odstraneni duplicit

      isnew = True
      pos = -1
      for Cluster in Clusters:
        pos += 1
        S1 = set(Group)
        S2 = set(Cluster)
        I~= S1.intersection(S2) #zjistit prunik
        if (len(I) > 0 ): #pokud existuje prunik, zarad skupinu do shluku
          isnew = False
          for value in Group:
            if (value not in Clusters[pos]):
              Clusters[pos].append(value)
          break
      if (isnew): #pokud je skupina nova, pridej ji jako novy shluk
        new = []
        for value in Group:
          new.append(value)
        Clusters.append(new)
    n += 1
    count=0
    for Cluster in Clusters:
      count = count + len(Cluster)
    if (count == aim):
      break  #pokud jsou shluky disjunktni, ukonci algoritmus
  return Clusters
```

7.5 Běh programu

Předchozí sekce této kapitoly nastavily nutné předpoklady pro samotný běh programu a ukázaly implementaci důležitých vlastních metod. V této části bude popsán samotný běh programu s použitím dříve definovaných datových struktur a metod.

7.5.1 Inicializace a naplnění halfedge datové struktury

V databázi jsou již připravené zrekonstruované plochy 3D modelů budov s validní orientací a jsou binárně uložené v geometrické reprezentaci *PolygonZ*, což je 3D polygon. Plochy jsou v databázi dekomponovány na orientovaný seznam vrcholů. To je provedeno funkcí *ST_DumpPoints*, která jeden databázový záznam obsahující polygon rozdělí na n záznamů, kde n je počet vrcholů v polygonu. Tyto záznamy nesou index, tedy pořadí vrcholu v polygonu. Následujícím dotazem se do dočasné tabulky *unique_vertex* uloží unikátní vrcholy pro model každé budovy s novým umělým identifikátorem *vertex_id*.

```
CREATE TEMPORARY TABLE unique_vertices AS
SELECT id, ST_X(geom) as x, ST_Y(geom) as y, ST_Z(geom) as z,
  (row_number() OVER(PARTITION BY id ORDER BY 1,2,3) - 1)::int as vertex_id
FROM ( SELECT id, (ST_DumpPoints(geom)).geom as geom from budovy_shapefile )foo
GROUP BY 1,2,3,4;
```

Následuje přiřazení umělých identifikátorů vrcholů zpět k dekomponovaným polygonům. Hodnotu souřadnic tak nahradí pouze odkaz na souřadnici pomocí indexu. Vzniká tabulka *facets*

```
CREATE TEMPORARY TABLE facets AS
SELECT facet_id, vertex_id, vertex_order, A.id FROM (
  SELECT id, (gd).path[1] as facet_id, (gd).path[3] as vertex_order,
    MAX((gd).path[3]) OVER (PARTITION BY id_bud, (gd).path[1] ) as max_vertex,
    ST_X((gd).geom) as x,  ST_Y((gd).geom) as y,  ST_Z((gd).geom) as z
  FROM (
    SELECT id_bud, ST_DumpPoints(geom) as gd from budovy_shapefile
    WHERE id_bud = %s
    )foo
  )A
JOIN unique_vertices B ON A.x=B.x and A.y=B.y and A.z=B.z AND A.id=B.id
WHERE vertex_order < max_vertex
ORDER BY A.id, facet_id, vertex_order
;
```

Z výše připravených tabulek se vytvoří dynamická pole vrcholů *vertices* a ploch *facets*, která se pošlou na vstup do další části programu, implementované v C++ nad knihovnou CGAL. Zde se pro každou budovu použije vlastní metoda *Build_surface* pro vytvoření objemového modelu. Takto jsou sestaveny všechny

vstupní modely do validní halfedge datové struktury a zároveň jsou ihned exportovány do souborového systému ve formátu OFF.

```
// Main.cpp
std::vector<int> buildings;
for (auto building : buildings ) {
  Polyhedron P; //deklarace objektu datoveho typu Polyhedron
  //naplneni halfedge struktury
  Build_surface<HalfedgeDS> surface( veriitces, facets );
  P.delegate( surface ); //kontrola validity struktury
  P.normalize_border(); //normalizace poradi hran
  std::cout << P.is_valid(); //kontrola validity modelu
  std::cout << P.is_closed(); //kontrola uzavrenosti povrchu
  std::ofstream os("/data/puvodni/");
  os << std::fixed << Pcopy << std::flush; //export modelu do OFF formatu
}
```

7.5.2 Načtení původních modelů do databáze a kontrola topologie

Exportované modely budov ve formátu OFF jsou mezitím načteny zpět do databáze a je nad nimi vytvořena topologická struktura. Načtení modelů do databáze provádí vlastní funkce *readOFF*. Modely jsou v databázi uloženy do tabulky jako speciální datový typ *PolyhedralSurface*, který umožňuje provádění analytických operací nad objemovými modely. V databázi se po načtení dat automaticky spustí uložená procedura *Build_topology*, která nad tabulkou sestaví topologický model. Dále se provede kontrola na existenci topologických vztahů dimenze 3, které by znamenaly topologickou chybu. Pokud taková situace nastane, jsou tyto modely z dalšího procesu vyňaty, protože základním předpokladem je topologicky validní výchozí model. Topologická procedura zároveň automaticky generuje databázovou tabulku *models_relations*, která obsahuje dvojice budov s topologickým vztahem a hodnotu dimenze jejich průniku.

```
## readOFF.py
# ze zdrojoveho odresare nacti data do databazove tabulky models
# skala generalizace je 0 m
readOFF("/data/puvodni/", "models", 0)
```

7.5.3 Generování shluků budov

Zjištěné topologické vazby mezi objekty se dále použijí pro hledání skupin shluků, na kterých bude probíhat generalizace. Dekompozice celého modelu na shluky se provádí z důvodu snížení paměťových a výpočetních nároků při generalizaci rozsáhlého vstupního modelu. Z databázové tabulky s topologickými vztahy se vyberou ty dvojice budov, které sdílí společnou plochu (topo_dimension=2).

Tyto dvojice jsou vstupem do vlastní funkce *Cluster_neighbours*, která sestaví seznam disjunktních shluků vstupních budov. Seznam shluků je následně předán zpět do běhu programu v C++.

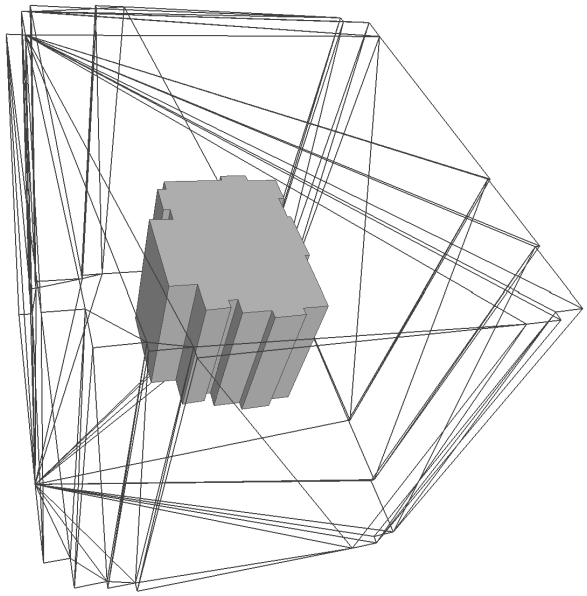
```
## Cluster_neighbours.py
sql="SELECT ID_a, ID_b FROM models_relations WHERE topo_dimension=2;"
tuples = execute(sql)
clusters = ClusterNeighbours(tuples)
```

7.5.4 Generalizace a agregace

V této fázi jsou již k dispozici všechna potřebná data pro spuštění procesu generalizace (řádky 1-3). Vícerozměrný seznam *Clusters* obsahuje seznamy shluků jako odkazy na ID jednotlivých modelů budov. Proces postupuje po jednotlivých shlucích. V následujícím C++ kódu je ukázka implementace pro shluk budov. Na začátku je sestaven strukturní element, krychle o délce hrany, jež je parametrem *elementSize* (ř. 6). Aby bylo možné provádět nad objekty booleovské operace, musí se modely převést na datový typ Nef (ř. 9). Dále je pro celý shluk vypočten hlavní směr rotace v prostoru pomocí vlastní metody *Get_direction*, sestaví se matice afinní transformace, která je aplikována na strukturní element (ř. 10-15).

Algoritmus pokračuje procházením seznamu všech Polyhedron modelů, opět je provedena konverze na typ Nef a na každý model je aplikována operace dilatace. Po aplikaci dilatace je výsledek sjednocen s výsledkem z předešlé iterace. Pokud byly součástí shluku objekty, které spolu přímo sousedily (sdílená plocha) nebo pokud byla jejich vzájemná vzdálenost menší než velikost strukturního elementu, jsou takové objekty sjednocené v jeden objemový model a případná mezera mezi budovami se tím zaplní (ř. 18-29). Operace dilatace (ř. 21) je jednokrokovou operací aplikace Minkowského součtu dvou vektorových množin. Výsledek tohoto mezi kroku znázorňuje obr. č. 7.2. Pro dokončení operace uzavření je nutné provést duální operaci – erozi, která je naopak Minkowského rozdílem a je aplikována již na celý sjednocený model. Minkowského rozdíl je rozdělen do čtyř kroků:

1. Provedení dilatace na dilataci *DoubleDilation* (ř. 31).
2. Doplněk první dilatace *Complement* se získá odečtením první dilatace od dvojnásobné dilatace (ř. 32).
3. Dilatace doplňku *ComplementDilation* je množinou, která obepíná požadovanou erozi (ř. 33).
4. Eroze dilatace *Intermediate* je rozdílem dilatace a dilatace doplňku z předchozího kroku (ř. 34).



Obrázek 7.2: Znázornění dilatace (drátový model) původní geometrie. Zdroj: vlastní tvorba.

Model ve fázi Intermediate tedy představuje výsledek po operaci uzavření. Tam, kde to bylo možné, došlo k vyplnění děr a záhybů. Finální krok, provedení operace otevření, je analogický k výše popsaným krokům, pouze v opačném pořadí (ř. 37-41). Proces generalizace končí konverzí modelu ze struktury Nef zpět do datového objektu Polyhedron *Presult*, na kterém je spuštěna finální fáze, a to sloučení koplanárních ploch pro maximální snížení počtu primitiv definujících model. Sloučení ploch zpracovává vlastní funkce *Merge_faces*. Na konci je model vyexportován do souborového systému jako OFF soubor pojmenovaný podle ID zpracovávaného shluku.

```
1  // Main.cpp
2  std::vector<Polyhedron> Polyhedra;  //Vstupni promenne pro shluk budov
3  double elementSize;
4  int clusterID;
5  ////////////////////
6  Polyhedron Pcube;  //deklarace objektu Polyhedron
7  Build_cube<HalfedgeDS> cube(elementSize); //Sestaveni modelu krychle
8  Pcube.delegate( cube );
9  Pcube.normalize_border();
10 Nef_polyhedron Ncube( Pcube ); //Prevod do datoveho typu Nef
11 double r = Get_Direction(Polyhedra);  //vypocet hlavniho uhlu smeru shluku
12 Aff_transformation_3 rot(  //Matice transformace pro krychli
13   cos(r), -1*sin(r), 0,
14   sin(r), cos(r),    0,
15   0,      0,         1 );
16 Ncube.transform(rot);  //Rotace matice o~vypocteny uhel
17 itemIter=0;
18 Nef_polyhedron Dilation;  //Deklarace Nef objektu pro dilataci
19 for (Polyhedron P : Polyhedra) {  //Pro kazdy model budovy
20   Nef_polyhedron Nobject( P );  //Preved Polyhedron na Nef
21   //Proved dilataci budovy a~krychle
22   Nef_polyhedron PartialDilation = CGAL::minkowski_sum_3(Ncube, Nobject);
23   if (itemIter == 0) {
24     Dilation = PartialDilation; //Prvni iterace, uloz vysledek dilatace
```

```
25    } else {                        //Druha a~dasli iterace, sjednoceni dilataci
26      Dilation = Dilation + PartialDilation;
27    }
28    Dilation.regularization();
29    itemIter++;
30  }
31  // Operace UZAVRENI
32  Nef_polyhedron DoubleDilation = CGAL::minkowski_sum_3(Dilation, Ncube);
33  Nef_polyhedron Complement = DoubleDilation - Dilation;
34  Nef_polyhedron ComplementDilation = CGAL::minkowski_sum_3(Complement, Ncube);
35  Nef_polyhedron Intermediate = Dilation - ComplementDilation;
36
37  // Operace Otevreni
38  Nef_polyhedron DoubleDilation2 = CGAL::minkowski_sum_3(Intermediate, Ncube);
39  Nef_polyhedron Complement2 = DoubleDilation2 - Intermediate;
40  Nef_polyhedron ComplementDilation2 = CGAL::minkowski_sum_3(Complement2, Ncube);
41  Nef_polyhedron Intermediate2 = Intermediate - ComplementDilation2;
42  Nef_polyhedron Final = CGAL::minkowski_sum_3(Intermediate2, Ncube);
43
44  Polyhedron Presult;  //Deklarace typu Polyhedron
45  Final.regularization();
46  Final.convert_to_polyhedron(Presult);  //Konverze z~Nef do Polyhedron
47  Merge_faces(Presult);  //Slouceni koplanarnich ploch
48
49  std::ofstream os(clusterID);
50  os << std::fixed << Presult << std::flush;;  //Exports shluku do OFF souboru
```

7.5.5 Načtení zjednodušených modelů do databáze a kontrola topologie

Po provedení generalizace jsou výstupní modely pomocí metody *readOFF()* načteny zpět do databáze a je nad nimi spuštěna topologická kontrola stejně jako na původní data. I po generalizaci a agregaci může tato kontrola odhalit nevalidní topologickou situaci, což může být příklad původně blízkých, ale disjunktních těles, která však nebyla označena za sousedy a byla zpracovávána odděleně. U těchto případů dojde již jen ke sjednocení, bez procesu generalizace, aby byl výsledný model opět validní. Na konci dojde k uložení relační databázové tabulky *aggregation_tree*, která obsahuje vztahy mezi původními a agregovanými objekty. Vztahy jsou reprezentovány jako dvojice identifikátorů původního a nového modelu. Vztahy jsou zjištěny z předchozího volání Python metody *ClusterNeighbours()*.

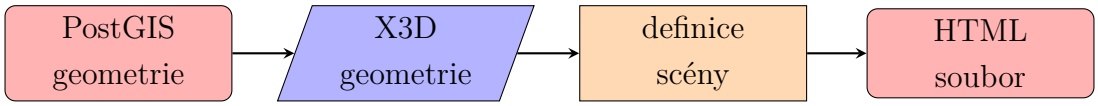
```
## readOFF.py
# ze zdrojoveho odresare nacti data do databazove tabulky
# skala generalizace je 2.0 m
readOFF("/data/generalizace/", "models", 2.0)
```

```
for cluster in clusters:
  for member in cluster:
    sql = "INSERT INTO aggregation_tree (cluster[id], member[id])";
```

7.6 Vizualizace

Databáze jako úložný prostor pro 3D modely umožňuje vytvoření procesu pro vizualizaci uložených dat. Tento proces vyžaduje konverzi geometrie z databázových struktur do formátu specifického XML formátu, takzvaného *Extensible 3D* (X3D). Formát X3D obsahuje definici celé zobrazované scény, což zahrnuje geometrii objektů, definici barevné škály pro jednotlivé objekty nebo povrchy a výchozí místo a směr pohledu na scénu. Scénu v tomto formátu je pak možné snadno zobrazit v moderních internetových prohlížečích za použití knihovny X3DOM. Podmínkou pro bezproblémové zobrazení je podpora skriptovacího jazyka Javascript, standardu HTML5 a grafické knihovny WebGL, což je u moderních prohlížečů standardem. Výhodou pro uživatele je tedy absence nutnosti instalovat dodatečné doplňky (X3DOM, 2017).

Soubor XML lze vytvořit jednorázově nebo je možné scénu dynamicky generovat na serveru a uživatelům dodávat automatické pohledy. Tvorba dynamických scén je již mimo záběr této práce, proto bude popsán pouze jednodušší souborový režim, kdy se popis scény ve formátu XML-X3D vloží do webového HTML souboru, jak je vidět na obr. č. 7.3.



Obrázek 7.3: Schéma procesu vizualizace. Zdroj: vlastní tvorba.

Nadstavba PostGIS pro tento účel poskytuje funkci *ST_AsX3D*, která dokáže binárně uložené geometrie modelů zapsat ve formátu X3D. Kombinací dotazu do databáze a Python skriptu se vybere požadovaná množina modelů, pomocí výše zmíněné funkce se do seznamu geometrií uloží jejich X3D reprezentace. Seznam geometrií je poté postupně volán a X3D data jsou vkládána do HTML souboru mezi značky <shape> a </shape>. Zároveň je nutné nastavit parametr convex na hodnotu false, aby došlo ke správnému vykreslení nekonvexních polygonů.

```
sql = "select ST_AsX3D(geom) as geom from budovy;"
cur.execute(sql)
rows = cur.fetchall()
shapePart = ""
for row in rows:
  newShape = ""
  newShape += "<shape>"
  newShape += "<appearance><material diffuseColor='0.903 0.294 0.309' ></
    material></appearance>"
  newShape += row[0].replace("IndexedFaceSet","IndexedFaceSet convex='false'")
  newShape += "</shape>"
  newShape += "\n"
  shapePart += newShape
```

```
html = partOne + shapePart + partTwo
f = open('html/viz.html', 'w')
f.write(html)
f.close()
```

Struktura HTML souboru obsahuje standardní definici hlavičky, následuje definice X3D scény značkami <x3d> a <scene> a nastaví se výchozí bod pohledu na scénu <viewpoint>. Další značky již určují to, co a jak bude scéna zobrazovat. Každý objekt je do scény zanesen značkou <shape>, která má mimo jiné dvě základní vlastnosti:

- <appearance> definuje barvu modelu.
- <IndexedFaceSet> obsahuje samotná X3D data vygenerovaná databází.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="X-UA-Compatible" content="chrome=1" />
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  <title>Vizualizace generalizace</title>
  <link rel="stylesheet" type="text/css" href="x3dom.css" />
  <script type="text/javascript" src="x3dom.js"></script>
</head>
<body>
<x3d xmlns="http://www.x3dom.org/x3dom" showStat="false" showLog="false"
    x="0px" y="0px" width="1550px" height="600px" altImg="helloX3D-alt.png">
<scene DEF='scene'>
  <viewpoint id="pohled"  position='-742976.0 -1046546.6 270.0'
    centerOfRotation='-742976.0 -1046546.6 250.0' ></viewpoint>
  <shape>
  <appearance>
    <material diffuseColor='0.903 0.294 0.309' ></material>
  </appearance>
  <IndexedFaceSet convex='false' coordIndex="0">
   <Coordinate point='1 1 1'></Coordinate>
  </IndexedFaceSet>
  </shape>
</scene>
</x3d>
</body>
</html>
```

8. Experimentální výsledky

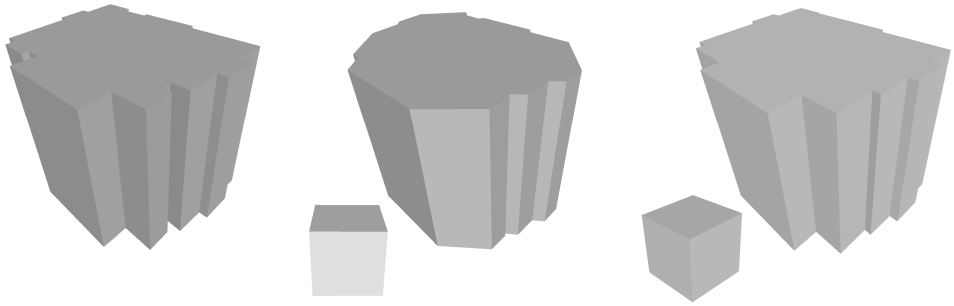
Navržená metoda a její implementace byla otestována na experimentálních datech za účelem ověření toho, zda je navržený postup správný a poskytuje očekávané generalizované modely.

Testování probíhalo nad daty z 3D modelu Prahy, která jsou podrobněji popsána v kapitole č. 6. V průběhu testování bylo experimentováno s velikostí a tvarem strukturního elementu, který je hlavním faktorem výsledné generalizace.

8.1 Tvar strukturního elementu

Jak bylo již diskutováno v kapitole č. 4, pro generalizaci obecných modelů se dá použít strukturní element kulovitého tvaru, protože koule zachovává symetrii ve všech směrech. Tento tvar se však pro generalizace budov nehodí, protože budovy si zachovávají určitou pravidelnost, kterou by kulovitý strukturní element porušil. Koule by navíc musela být modelována velkým počtem plošek, které by aproximovaly její tvar, což by v rohových částech generalizované budovy vyústilo ve výrazné zvýšení počtu ploch. Stejně jako v metodách zpracování 2D obrazu se jako konvoluční okno používá například čtverec, pro účely této práce bylo nejvhodnější použít krychli, která dobře aproximuje pravidelný tvar budov.

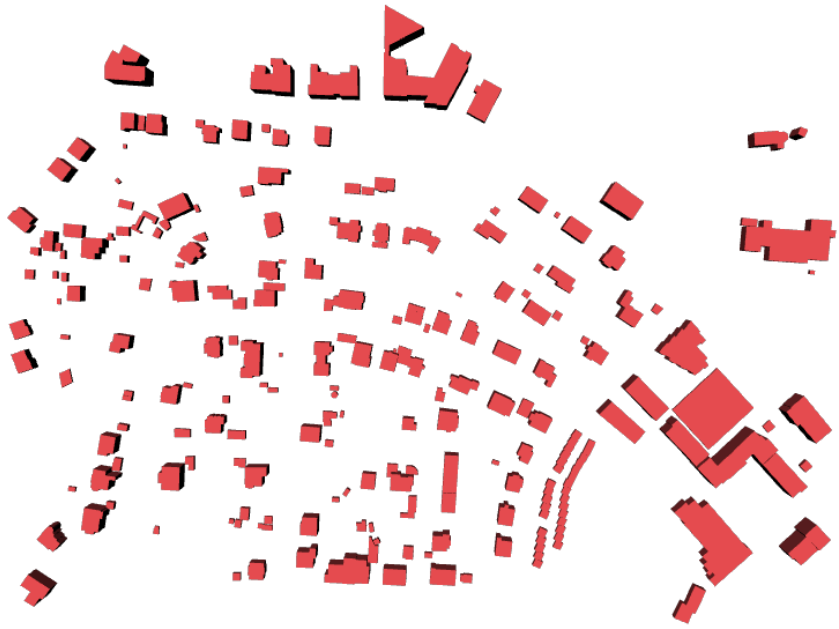
V první verzi implementace byla použita krychle bez dalších prostorových transformací a ukázalo se, že je potřeba provádět i vzájemnou prostorovou orientaci mezi strukturním elementem a generalizovaným objektem, který od elementu částečně přebírá charakteristické tvarové rysy. Na obr. č. 8.1 je zobrazena generalizace původního modelu (vlevo) pomocí neorientované krychle (uprostřed). Je zřejmé, že krychle, která je zarovnaná s osami soustavy souřadnic nedokonale zjednoduší tvar budovy. Nové stěny jsou orientované podle směru souřadnicových os, což však není žádoucí. Při vzájemné orientaci obou objektů je výsledek uspokojivý, charakter tvaru budovy je zachován a zároveň zjednodušen (vpravo). Na základě těchto poznatků byla metoda upravena výpočtem hlavního směru natočení budovy tak, aby strukturní element byl co nejlépe orientován.



Obrázek 8.1: Vliv tvaru strukturního elementu na výslednou generalizaci. Zdroj: vlastní tvorba.

8.2 Výsledky metody

Pro testování byla vybrána lokalita části čtvrti Podolí v Praze. Lokalita byla vybrána z důvodu rozmanitosti typů staveb, obsahuje totiž bloky budov, pravidelné solitérní objekty, ale i budovy se specifickým tvarem. Půdorys lokality je na obr. č. 8.2. Na toto území (3D model) byl aplikován celý navržený proces generalizace v souladu s popsanou metodikou, tedy rekonstrukce původních modelů do topologicky validní struktury. Přestože byla provedena rekonstrukce modelů, při následné topologické kontrole bylo zjištěno, že několik budov sdílí část svého objemu s jinými budovami, což představuje topologickou chybu. Aby byla počáteční zdrojová data validní, byly chybné modely z dat odstraněny.



Obrázek 8.2: Půdorys budov v testovací lokalitě. Zdroj: vlastní tvorba

Fáze generalizace byla provedena ve více iteracích, pokaždé s odlišnou velikostí

strukturního elementu, aby bylo možné mezi sebou jednotlivé scénáře porovnat. Při tomto experimentu bylo empiricky zjištěno, že zdrojové modely jsou tvarově strukturované až v menším měřítku a je tedy nutné použít vysoké hodnoty velikosti strukturního elementu, aby generalizace měla dostatečný efekt. Kombinace operace otevření a příliš velkého strukturního elementu může způsobit, že rozměrově příliš malé objekty budou ze scény úplně odstraněny. Z toho důvodu byla v tomto experimentu použitá pouze operace uzavření (dilatace následovaná erozí), která byla aplikována na 185 shluků budov. Shluky budov byly vyhledány s podmínkou sousednosti dotykem svých hranic. Výsledky počtu ploch, které tvoří modely na jednotlivých úrovních rozlišení, jsou uvedeny v tabulce č. 8.1.

Model	A	B	C	D	E	F
Rozlišení (m)	0	0.1	2	4	8	14
Počet ploch	2946	2052	1987	1947	1921	1911
Podíl	100,0%	69,7%	67,4%	66,1%	65,2%	64,9%
Ovlivněné shluky	–	109	111	112	113	113

Tabulka 8.1: Výsledek generalizace na více úrovních rozlišení.

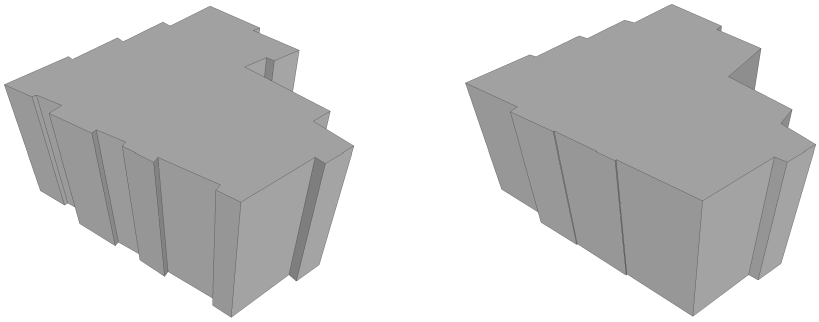
Z výsledků je patrné, že již při prvním stupni generalizace se strukturním elementem o velikosti 10 cm dojde k výraznému zmenšení počtu ploch v testovacím modelu a s dalšími stupni je již tempo poklesu pouze mírné. Je to způsobeno tím, že již na nejmenší škále generalizace dochází k agregaci sousedních objektů a tím pádem k rapidnímu poklesu počtu primitiv. Zároveň je vidět, že vysoká hodnota škály generalizace zapříčinila snížení počtu ploch pouze u 113 z 185 vstupních shluků. Pro potlačení vlivu funkce sloučení koplanárních ploch a vlivu agregace je zajímavé vztáhnout porovnání nikoliv k originálním modelům, nýbrž k prvnímu stupni škály na hodnotě 10 cm, kde je model již agregovaný a obsahuje vyrovnané plochy. Tabulka 8.2 zároveň porovnává jen ty modely, u kterých došlo k odstranění alespoň jedné plochy.

Model	B	C	D	E	F
Rozlišení (m)	0.1	2	4	8	14
Počet ploch	1393	1331	1293	1268	1254
Podíl	100,0%	95,5%	92,8%	91,0%	90,0%

Tabulka 8.2: Výsledek generalizace na více úrovních rozlišení.

8.2.1 Validace výsledků

Snížení počtu ploch a zjednodušení geometrie má jistě za následek deformaci modelu, proto je potřebné provést kontrolu míry odlišnosti výsledných objektů od počátečního stavu. Důležitým faktorem úspěšnosti generalizace je její vizuální stránka. Na obr. č. 8.3 je ukázka výsledku generalizace originálního modelu. Došlo k zjednodušení geometrie vyplněním mezer, zatímco charakter objektu zůstal nezměněn.



Obrázek 8.3: Výsledek generalizace původního modelu (vlevo) na škále 4 m (vpravo). Zdroj: vlastní tvorba

Vizuální porovnání je nicméně subjektivní metoda, proto je vhodné výsledky generalizace kvantifikovat. Pro změření odchylky lze použít podobnou metriku, která se používá při lokální decimaci obecných povrchů, a to měření chyby.

Jako kritérium pro porovnání dvou datových sad se používá souhrnná absolutní chyba, která se vypočítá jako suma všech odchylek nového modelu od původního (Willmott, Kenji, 2005):

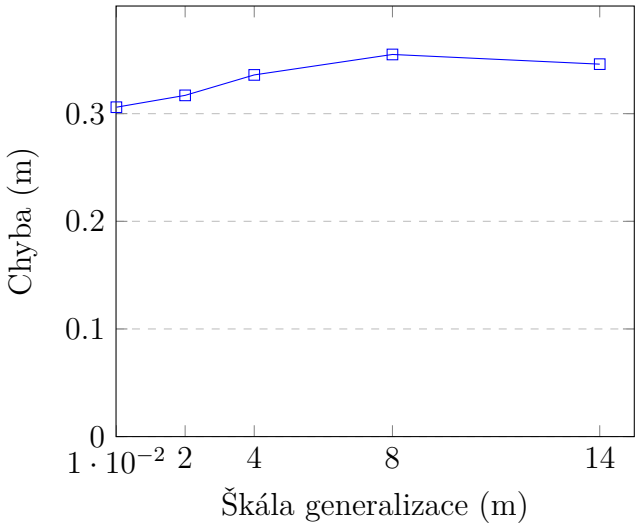
$$E = \sum_{i=1}^{n} |x_i - y_i|$$

Průměrná absolutní chyba modelu je potom

$$M_E = \sum_{i=1}^{n} \frac{|x_i - y_i|}{n}$$

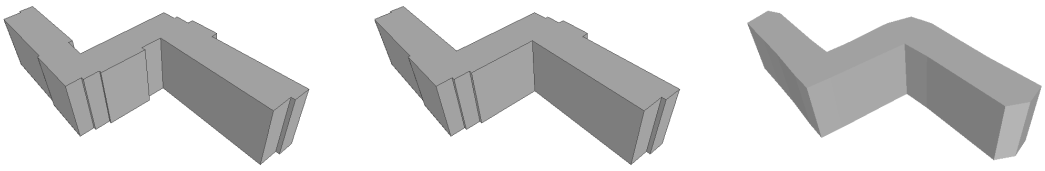
Chyba je v tomto případě nejmenší vzdálenost vrcholů původní geometrie od ploch geometrie nové. Vzdálenost je počítána v $\mathbb{R}^3$ jako vzdálenost bodu k měřenému polygonu. Provede se projekce bodu na rovinu polygonu a zjistí se, zda je bod uvnitř polygonu. Pokud projektovaný bod leží uvnitř, výsledkem je vzdálenost bodu od roviny. Pokud leží vně, je nutné dopočítat vzdálenost k nejbližší hraně v rovině a zároveň se znalostí vzdálenosti bodu od roviny se vypočte 3D vzdálenost (Jones, 1995).

Vypočtená průměrná absolutní chyba pro různé hodnoty škály generalizace je vidět v grafu na obr. č. 8.4. Je patrné, že již v úrovni generalizace na škále 10 cm dochází k nárůstu průměrné odchylky na úroveň 30 cm. Tento náhlý nárůst je způsobený agregací, kdy dochází ke sjednocení sousedních modelů a vyrovnání či odstranění stěn, takže některé původní vrcholy mohou být od nových stěn najednou více vzdálené. Na dalších úrovních již chyba jen mírně stoupá na hladinu 35 cm a na nejvyšší škále dokonce mírně klesne. Hodnota chyby je přesto stále menší než udávaná přesnost původního modelu, která činí 50 cm (Geoportal Praha, 2015).



Obrázek 8.4: Průměrná polohová chyba generalizace. Zdroj: vlastní tvorba

8.3 Srovnání s metodou sjednocení hran



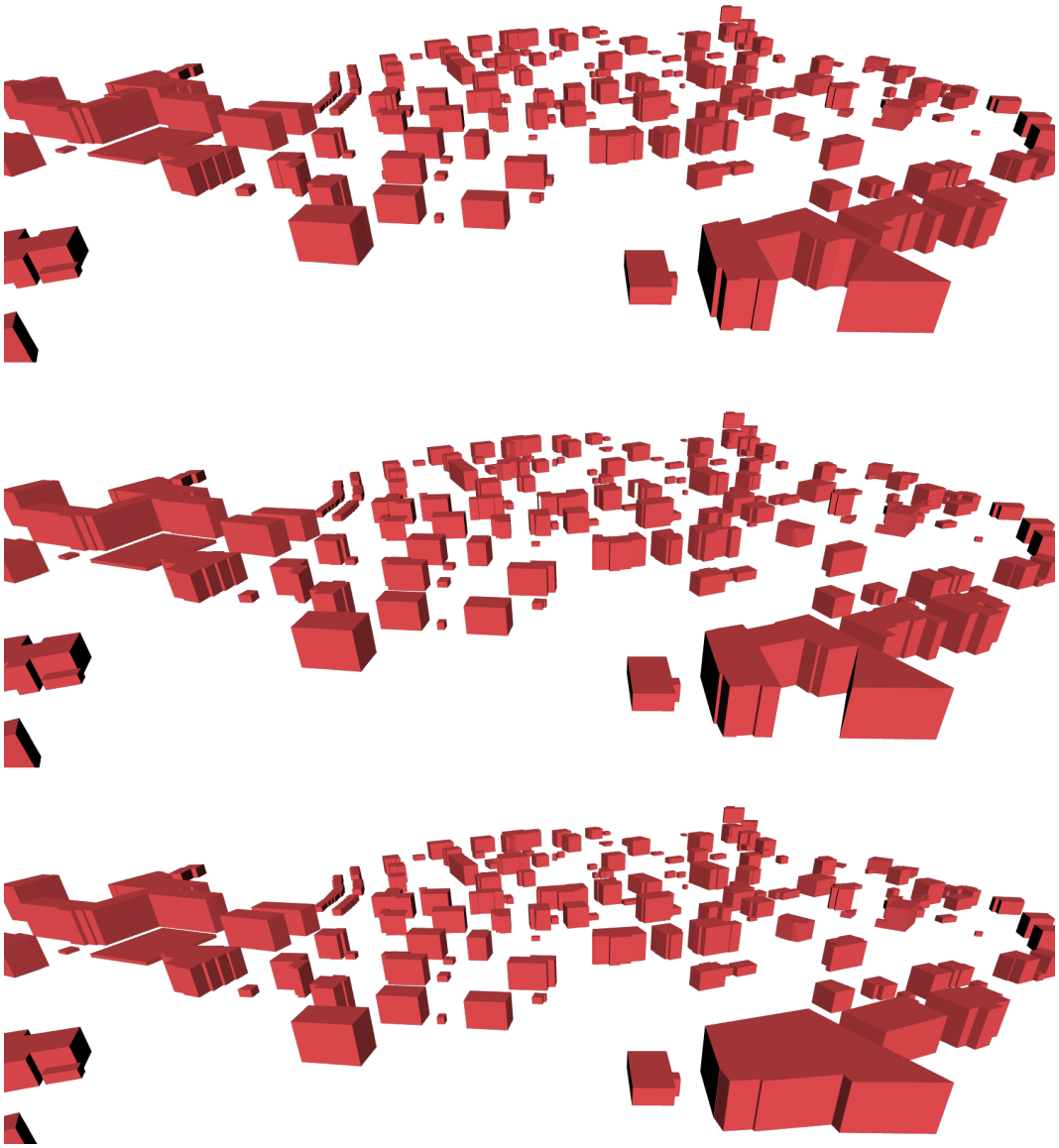
Obrázek 8.5: Porovnání generalizace původního modelu (vlevo) navrženou metodou (uprostřed) s metodou sjednocení hran (vpravo). Zdroj: vlastní tvorba.

Navržená metoda byla experimentálně porovnána s metodou sjednocení hran. Je to jedna z metod popsaných v kapitole 3, která se často používá pro generalizace obecných 3D modelů (Ovreiu, 2012). Její implementace je dostupná v knihovně CGAL. Při srovnání výsledků generalizace na bloku několika budov je vidět, že navržená metoda si vede lépe ve vizuální podobnosti a zachování charakteristického tvaru budovy, jak je patrné z obr. č. 8.5. Metoda sjednocení

hran celkový tvar vystihuje také dobře, ale má problémy na okrajích, kde vytváří mírně zaoblené hrany, kde by naopak měly mít zachovanou pravoúhlost. Oba generalizované modely mají stejný počet ploch.

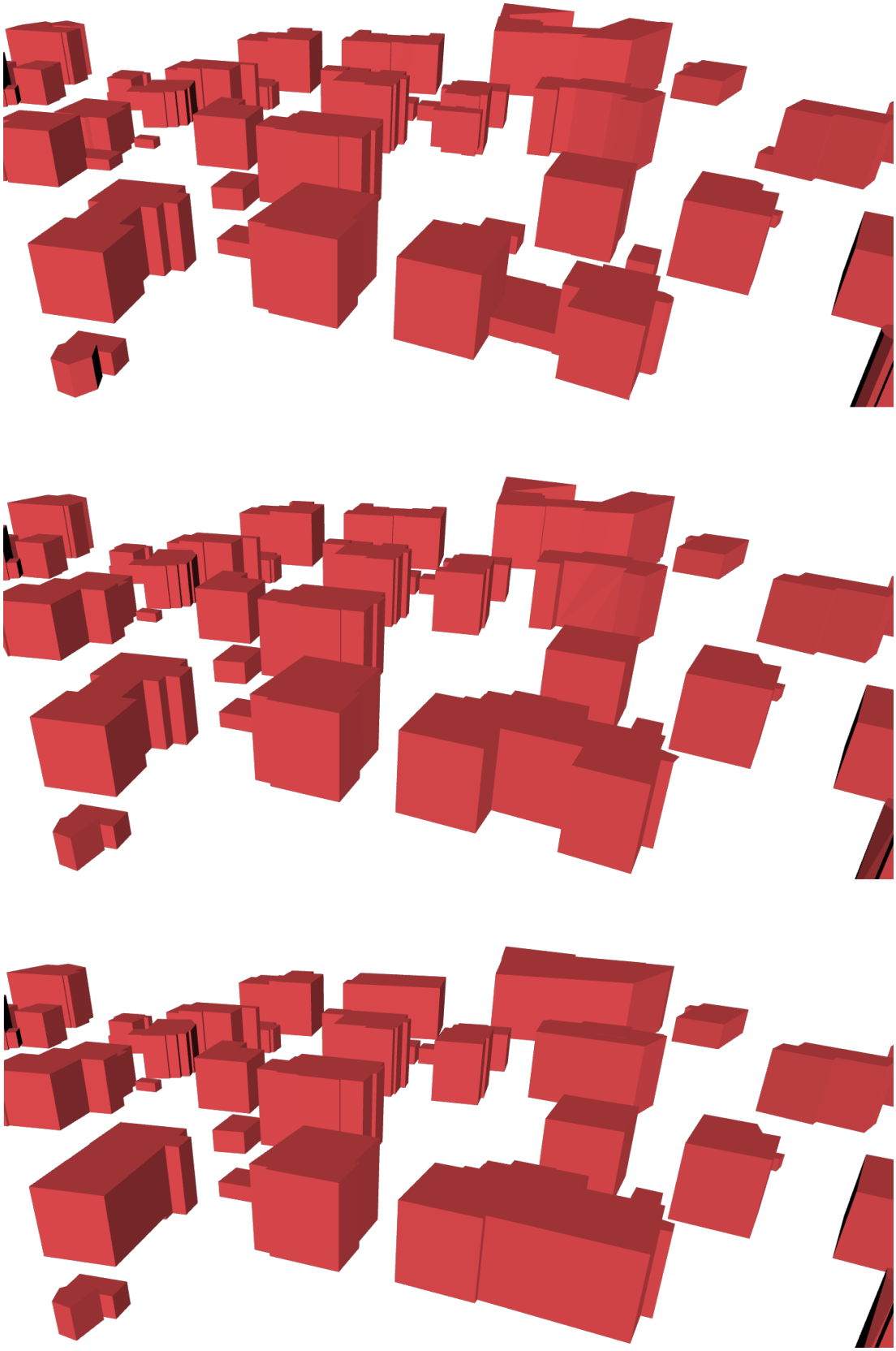
8.4 Vizualizace scény

Níže na obr. č 8.6 je porovnání scén vykreslených s třemi úrovněmi generalizace (0m, 2m a 14m). Při přechodu z první do druhé scény dojde k agregaci bloků budov a výraznému snížení objemu dat na 64.9 %. Třetí scéna ještě dále zjednoduší některé objekty, což vyústí v 60.2 % původního objemu dat. Tyto vizualizace jsou v interaktivní formě k dispozici v digitální příloze této práce.



Obrázek 8.6: Tři úrovně generalizace scény. Shora postupně 0m, 2m, 14m. Zdroj: vlastní tvorba.

Scéna z jiného pohledu a při větším přiblížení je zobrazena na obr. č. 8.7.



Obrázek 8.7: Tři úrovně generalizace scény, detail. Shora postupně 0m, 2m, 14m. Zdroj: vlastní tvorba.

9. Diskuze

Pohled na výsledky v této práci ukazuje, že navržená metoda generalizace 3D modelů budov pomocí operátorů matematické generalizace je schopna dosáhnout generalizace původních 3D modelů při zachování charakteristiky jejich tvaru. Porovnání s klasickou metodou generalizace 3D modelů ukázalo, že takto specifická metoda je pro člověkem vytvořené objekty vhodnější, nežli jiná obecná metoda.

Cílem bylo zároveň celý model zasadit do topologického rámce, který umožní práci s objemovými 3D modely a rozšíří tak dosavadní praxi při generalizaci 3D scén, které řeší hlavně vizuální stránku věci, ne však již validitu scény z pohledu GIS. Přidaná topologie modelu rozšiřuje jeho schopnosti na zjišťování vzájemných vztahů ve 3D, což zatím není v běžně používaných GIS softwarech dostupné.

Při implementaci práce narazila na problém dostupnosti kvalitních 3D datových zdrojů. I přesto, že jsou veřejně k dispozici modely jak pro různé světové metropole, tak i pro samostatné objekty, díky roztříštěnosti formátů není snadné tyto modely mezi sebou porovnat, případně provést jejich konverzi do stejné datové podoby. Data pro generování 3D modelů jsou dnes snímána automaticky či poloautomaticky a stejným způsobem jsou zpracovávána. Algoritmy pro zpracování těchto dat však ještě nejsou natolik dokonalé, aby generovaly validní objemové modely. Data, která byla otestována pro účely této práce, vykazovala dostatečnou kvalitu pro vizualizaci ve webových službách nebo vytváření architektonických studií. Pro analytické účely však tato data vhodná nebyla vzhledem k tomu, že obsahovala velké množství topologických chyb, jako mezery mezi plochami nebo vzájemně se protínající plochy. Pro účely analytického využití dostupných dat proto byla provedena rekonstrukce pomocí jejich otisků do roviny, aby bylo možné vytvořit validní objemový model, který je nutným předpokladem pro další zpracování ve 3D výpočetní geometrii.

Návrh implementace metody se snažil využít co nejaktuálnějších technologií, které budou zároveň dostupné v režimu open-source, aby byla metoda reprodukovatelná a dalo se na ní v budoucnu navázat. Technologie použité při implementaci se snaží kombinovat dva mírně odlišné koncepty pro zpracování dat. Prvním je robustní a algoritmicky schopná knihovna CGAL, která podporuje pokročilé datové reprezentace a operace nad objemovými modely. Druhým konceptem je tradiční,

a to databázový přístup, který nedosahuje výpočetních kvalit knihovny CGAL, ale na druhou stranu poskytuje osvědčené metody ukládání dat, jejich prohledávání. Díky nadstavbě PostGIS je možné v databázi uchovávat i zmíněné objemové modely a usnadnit tak případné analytické využití modelu.

Implementace programu obsahuje návrh několika vlastních funkcí, které obsluhují hlavní části metody. Program jako celek kombinuje jazyk C++ s jazykem Python a pro komunikaci s databází používá dotazovací jazyk SQL. Hlavní metody jsou napsané nad knihovnou CGAL v C++ a Python slouží jako wrapper pro obsluhu databáze. Implementace je snadno přenositelná, nicméně doporučuje se zůstat na Linuxové platformě, kde je instalace a následná kompilace zdrojových kódů přímočařejší.

V průběhu testování metody se ukázalo, že důležitým prvkem metody je tvar strukturního elementu, který má velký vliv na výsledek generalizace. Navržená metoda používá jako strukturní element krychli, která svým tvarem odpovídá tvaru drtivé většiny budov s pravoúhlým charakterem. Tvar krychle by ale nemusel být vhodný pro generalizaci šikmých střech, které by se při použití krychle mohly zploštit a výslednou geometrii místo zjednodušení zahustit. Další vědecká činnost by tedy v tomto ohledu měla pokračovat ve zkoumání vlivu tvaru strukturního elementu na specifické části budovy, případně se pokusit budovu dekomponovat a generalizaci provádět na různých částech budovy odlišným způsobem.

Mezivýsledky práce naznačily, že kromě samotného procesu generalizace je důležitý i postprocessing dat. Po geometrických operacích ve 3D jsou výsledné modely triangulované, což s sebou přináší zvýšení počtu geometrických primitiv oproti původnímu stavu. To by mohlo při zjednodušenému pohledu vést k závěru, že metoda není úspěšná. Výsledná data tedy navíc prošla procesem slučování koplanárních ploch, aby se odstranily důsledky triangulace a model byl porovnatelný se svým původním vzorem. Při finálním zpracování by se daly použít některé přístupy navržené pro generalizaci obecných objektů, a to slučování hran nebo vyrovnání ploch, ovšem nikoliv na celý model, ale pouze lokálně, což může být jedna z dalších možností směřování.

Výsledky generalizace naznačují, že i přes aplikaci na poměrně zjednodušené původní modely lze dosáhnout další úspory dat. Zároveň je při detailním pohledu na tvarově složitější objekty patrné, že pokud by byly k dispozici kvalitní a komplexnější zdrojová data, mohlo by být dosaženo lepšího poměru komprimace dat, než bylo dosaženo v této práci. Zdrojové 3D modely byly zatíženy velkou mírou generalizace a proto se výsledky na první pohled nemusí zdát působivé. Pokud se porovnají původní modely a první stupeň generalizace, který zahrnuje i agregaci, došlo k více než 30% úspoře ploch. V dalších krocích již úspora není tak výrazná

a pohybuje se do 10 %, přesto není na místě to vzhledem k okolnostem označit za neúspěch.

10. Závěr

V této diplomové práci byla navržena metoda generalizace 3D objektů. Vlastní metoda adaptovala existující algoritmus pro generalizaci 3D budov pomocí operátorů matematické morfologie a metodu rozšířila pro použití v 3D GIS zavedením topologie, která je schopná detekovat topologické chyby a vztahy a díky tomu optimalizuje celý proces generalizace hledáním optimálních kandidátů pro případnou agregaci modelu. Implementací metody vzniklo funkční programové řešení jako příspěvek k dosavadní algoritmické knihovně a k open-source komunitě.

Na začátku práce byla provedena rešerše datových struktur a metod generalizace ve 3D, na které potom navázala při přípravě vlastní metody. Přidanou hodnotou metody je rozšíření pomocí prostředků, které umožní její využití v geoinformačních technologiích. Metoda aplikuje využití datových struktur s plnou podporou 3D prostoru pro modelování komplexních objemových těles, připravila rámec pro generalizaci a celý model podpořila topologickou strukturou. Použité 3D datové struktury umožňují využit proces generalizace nejen pro potřeby zjednodušení geometrických těles při vizualizaci, ale hlavně při zpracování geoinformačních analýz nad topologicky validní scénou.

Navržená metoda byla zároveň implementována a byl vytvořen ucelený program, který kombinuje databázový systém pro správu a ukládání dat s robustní knihovnou pro náročné geometrické operace nad 3D objekty. Kvůli usnadnění přenositelnosti a použitelnosti byly použity pouze open-source systémy a nástroje.

Jako úskalí této metody se částečně ukázala dostupnost kvalitních zdrojových dat pro modelování validních objemovými těles, které metoda na vstupu předpokládá. Jedním ze způsobů, jak navázat na tuto práci, by mohly být metody rekonstrukce objemově validních geografických těles. Navržená metoda se dá také mnoha způsoby rozšířit, například použitím ještě obecnějších n-dimenzionálních datových struktur nebo specifickým chováním při generalizaci odlišných geografických objektů.

Výsledky navržené metody přesto ukazují, že jejím použitím lze docílit vizuálně kvalitní 3D generalizace při současném zjednodušení počtu modelovaných primitiv. To může urychlit analytické výpočty nad 3D geoinformačním mode-

lem, pro který je důležité prostorové rozmístění prvků v prostoru, i přes ztrátu některých méně podstatných detailů.

Seznam použité literatury

Biljecki, F. *Level of detail in 3D city models.* PhD thesis, Delft University of Technology, 2017.

Biljecki, F. et al. The Most Common Geometric and Semantic Errors in CityGML Datasets. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences.* 2016, IV-2W1, s. 13–22.

Damen, J., Kreveld, M., Spaan, B. High quality building generalization by extending the morphological operators. *11th ICA Workshop on Generalization and Multiple Representation, Montpellier, France.* 2008, s. 1–12.

Demel, J. *Grafy a jejich aplikace.* Academia, 1. vydání, 2002.

Dong, J. et al. From 1 dimension to N dimensions-fractal in automated cartographic generalization. *Journal of Geographical Sciences.* 2001, 11, 1, s. 86–90.

Donkers, S. et al. Automatic conversion of IFC datasets to geometrically and semantically correct CityGML LOD3 buildings. *Transactions in GIS.* 2015, s. 547–569.

Egenhofer, M. J., Herring, J. A Mathematical Framework for the Definition of Topological Relationships. *Proceedings of Fourth International Symposium on SDH, Zurich, Switzerland.* 1990, s. 803–813.

Egenhofer, M. J., Sharma, J., Mark, D. M. A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis. *Proceedings of Autocarto 11, Minneapolis.* 1993, s. 1–12.

Forberg, A. Generalization of 3D building data based on a scale-space approach. *ISPRS Journal of Photogrammetry and Remote Sensing.* 2007, 62, s. 104–111.

Geoportal Praha. 3D model Prahy, 2015. Dostupné z: <http://www.geoportalpraha.cz/cs/clanek/269/3d-model-prahy>.

Gröger, G., Plümer, L. How to achieve consistency for 3D city models. *GeoInformatica.* 2011, 15, 1, s. 137–165.

Hazewinkel, M. *Encyclopaedia of Mathematics*. Springer, 2. vydání, 2000.

He, S., Moreau, G., Martin, J. Y. Footprint-Based Generalization of 3D Building Groups at Medium Level of Detail for Multi-Scale Urban Visualization. *International Journal on Advances in Software*. 2012, 5, s. 378–388.

Jones, M. W. 3D Distance from a Point to a Triangle. *Technical report CSR-5-95, Department of Computer Science, Swansea University*. 1995.

Kada, M. Automatic Generalisationof 3D Building Models. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 34*. 2002, s. 243–248.

Kada, M. 3D Building Generalization based on Half-Space Modelling. *ISPRS WG II/3, II/6 Workshop Multiple representation and interoperability of spatial data, Hannover, Germany*. 2006, s. 22–24.

Kettner, L. Halfedge Data Structures, 2017. Dostupné z: <http://doc.cgal.org/latest/HalfedgeDS/index.html>.

Kolbe, T. H. Representing and Exchanging 3D City Models with CityGML. In *3D Geo-Information Sciences*. Berlin, Germany: Springer-Verlag, 2009. s. 15–31.

Kolingerová, I., Žalik, B. Reconstructing domain boundaries within a given set of points, using Delaunay triangulation. *Computers and Geosciences*. 2006, 32, s. 1310–1319.

Kothuri, R. K., Ravada, S., Abugov, D. Quadtree and R-tree Indexes in Oracle Spatial: A Comparison using GIS Data. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 2002, s. 546–557.

Lattuada, R. Three-Dimensional Representations and Data Structures in GIS and AEC. In *Large-scale 3D Data Integration*. Boca Raton, United States: CRC Press, 2005. s. 57–86.

Ledoux, H. Automatic conversion of IFC models 2x3 to CityGML LOD, 2014. Dostupné z: <https://github.com/tudelft3d >.

McMaster, R. B., Shea, K. S. *Generalization in Digital Cartography*. Assoc. of American Geographers, 1992.

Najman, L., Talbot, H. *Mathematical Morphology: From Theory to Applications*. ISTE-Wiley, 2010.

Open Geospatial Consortium. CityGML, 2017. Dostupné z: <https://www.citygml.org/>.

Ovreiu, E. *Accurate 3D Mesh Simplification*. PhD thesis, INSA de Lyon, 2012.

Penninga, F. *3D Topography: A Simplicial Complex-based Solution in a Spatial DBMS*. PhD thesis, Delft University of Technology, 2008.

Peterlin, P. Morphological Operations: An Overview, 1996. Dostupné z: <http://www.inf.u-szeged.hu/ssip/1996/morpho/morphology.html>.

Pilouk, M. *Integrated Modelling for 3D GIS*. PhD thesis, Wageningen Agricultural University, 2006.

Pultr, A. *Matematické struktury*. Katedra aplikované matematiky a ITI, MFF Univerzity Karlovy, 2005.

Shen, J., Zhou, T., Chen, M. A 27-Intersection Model for Representing Detailed Topological Relations between Spatial Objects in Two-Dimensional Space. *ISPRS International Journal of Geo-Information*. 2017, 6.

Surynková, P. Přednáška Počítačová geometrie: Zobrazování těles, 2017. Dostupné z: <http://surynkova.info/dokumenty/mff/PG/Prednasky/prednaska_5.pdf>.

Thiemann, F., Sester, M. Segmentation of Buildings for 3D-Generalisation. *Proceedings of the 7th ICA Workshop on Generalisation and Multiple Representation, Leicester*. 2004.

Tuan, A. N. G., Phuoc, T. V., K., D. H. Overview of Three-Dimensional GIS Data Models. *International Journal of Future Computer and Communication*. 2013, 3, s. 1013–1020.

Wikipedia. Mathematical morphology, 2008. Dostupné z: <https://en.wikipedia.org/wiki/Mathematical_morphology>.

Willmott, C. J., Kenji, M. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*. 2005, 30, s. 79–82.

X3DOM. X3DOM Documentation, 2017. Dostupné z: <https://doc.x3dom.org/index.html>.

Zeitouni, K., Cambray, B., Delpy, T. Topological Modelling For 3D GIS. *Proceedings of the Fourth International Conference on Computers in Urban Planning and Urban Management, Melbourne*. 1995.

Zhao, J. et al. Mathematical morphology-based generalization of complex 3D building models incorporating semantic relationships. *ISPRS Journal of Photogrammetry and Remote Sensing*. 2012, 68, s. 95–111.

Zlatanova, S. Topological Relationships and Their Use. In *Encyclopedia of GIS*, 2. Cham, Switzerland: Springer International Publishing, 2017. s. 1–21.

Zlatanova, S., Rahman, A. A., Shi, W. Topological models and frameworks for 3D spatial objects. *Computers and Geosciences*. 2004, 30, s. 419–428.

Zwillinger, D. *Standard Mathematical Tables and Formulae*. CRC Press LLC, 31. vydání, 2003.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

CAD	Computer-aided design
CGAL	Computational geometry algorithms library
codim	kodimenze
CSG	Constructive solid geometry
dim	dimenze
GIS	Geografický informační systém
GML	Geography markup language
GMP	GNU multiple precision (arithmetic library)
GPL	General public license
HDS	Halfedge data structure
HTML	HyperText markup language
ID	identifikátor
IM	Intersection model
LGPL	Lesser general public license
LOD	Level of detail
MIT	Massachusetts Institute of Technology (licence)
MPFR	Multiple precision floating-point reliably
OFF	Object file format
OGC	Open Geospatial Consortium
RAM	Random-access memory
SQL	Structured query language
WebGL	Web graphics library

WKT	Well-known text
X3D	Extensible 3D graphics
X3DOM	Extensible 3D graphics document object model
XML	Extensible markup language

Přílohy

1. CD se zdrojovými kódy a databází 3D modelů.