ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Rovinový řez trojúhelníkovou sítí
Student:	Michael Očenášek
Vedoucí:	Ing. Miroslav Hrončok
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Vytvořte open-source knihovnu v jazyce C, případně C++, obsahující algoritmus provádějící rovinové řezy trojúhelníkovou sítí. Trojúhelníková síť (*triangular mesh*) je typická reprezentace 3D modelu nejen pro 3D tisk (například formát STL). Pro práci se sítí využijte reprezentaci použitou v knihovně ADMesh.

Algoritmus síť rozdělí na několik částí, podle rovinového řezu, díry vzniklé řezem vyplní sítí tak, aby jednotlivé části byly opět 2-manifold – využijte existující svobodné knihovny pro triangulaci polygonů. Pokuste se vyřešit i případy, kdy na vstupu není 2-manifold síť, jak nejlépe to bude možné. Knihovnu otestujte řadou automatických jednotkových a integračních testů.

Knihovnu rozšiřte o program pro příkazovou řádku, který zapouzdří funkcionalitu knihovny pro běžné použití bez nutnosti programování.
Rozšiřte program ADMeshGUI o možnost rovinového řezu.

Seznam odborné literatury

[1] ADMESH CONTRIBUTORS. ADMesh – STL mesh manipulation tool: ADMesh 0.98.1 documentation [online]. 2015 [cit. 2017-2-21]. Dostupný z WWW: http://admesh.readthedocs.io/
[2] VYVLEČKA, David. ADMeshGUI: STL viewer and manipulation tool [online]. 2015 [cit. 2017-2-21]. Dostupný z WWW: https://github.com/admesh/ADMeshGUI

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 28. února 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Bakalářská práce

Rovinový řez trojúhelníkovou sítí

Michael Očenášek

Vedoucí práce: Ing. Miroslav Hrončok

2. srpna 2017

Poděkování

Chtěl bych poděkovat svým rodičům za neochvějnou podporu při studiu i v dobách, kdy jsem si ji nezasloužil.
Mé díky patří také Ing. Miroslavu Hrončokovi, který mi pomohl se všemi problémy, se kterými jsem se na něj obrátil a neváhal přijít s radou i v pozdních nočních hodinách.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. srpna 2017 .....

Odkaz na tuto práci

Očenášek, Michael. *Rovinový řez trojúhelníkovou sítí*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce pojednává o návrhu a implementaci knihovny, která umožňuje provést rovinný řez 3D modelem. K práci s těmito modely využívá jejich reprezentaci v nástroji ADMesh. Dalším cílem této práce je rozšířit o tuto funkci program ADMeshGUI. Zabývá se také mnohými problémy spojenými s prací s trojúhelníkovou sítí a algoritmy, které jsou s rovinným řezem spojeny. Knihovna a testy jsou napsány v programovacím jazyce C++.

Klíčová slova ADMesh, ADMeshGUI, C++, STL, 3D tisk, Poly2Tri, mesh, triangulace polygonu

Abstract

This bachelor's thesis discusses the design and implementation of a library, which allows to make a plane cut of 3D model. To work with these models, it uses their representation in ADMesh. Additional aim of this thesis is extending the ADMeshGUI program of plane cut feature. It also deals with many problems associated with work with a triangular mesh and algorithms that are related to plane cut. The library and tests are written in C++ programming language.

Keywords ADMesh, ADMeshGUI, C++, STL, 3D printing, Poly2Tri, mesh, polygon triangulation

Obsah

Seznam obrázků

Úvod

V posledních letech se technologie 3D tisku dostává do popředí zájmu odborné i laické veřejnosti. Nedílnou součástí procesu vedoucího k finálnímu vytisknutému objektu, je i manipulace s jeho 3D reprezentací. Programy ulehčující jednotlivé kroky v procesu 3D tisku se tím v poslední době stávají velmi důležitými. Před samotným tiskem je, zejména u objektů složitějších tvarů, často nutné 3D objekt rozřezat na několik částí, aby se lépe tisknul. Právě řešení tohoto problému ve formě vytvoření samostatné open–source knihovny StlCut, která bude schopna objekt rozřezat a zacelit řezem vzniklá místa, je hlavní náplní této práce.

Tato práce volně navazuje na bakalářskou práci Davida Vyvlečky, který vytvořil program ADMeshGUI, který převádí ADMesh z příkazové řádky do grafické podoby. Toto uživatelské rozhraní v rámci práce dále rozšiřuji právě o možnost rovinného řezu, kterou v současné době vůbec neobsahuje, což značně snižuje jeho využitelnost v praxi.

Toto téma jsem si zvolil, protože úspěšně spojuje 3D grafiku a programování, což jsou oblasti, které jsou mi velmi blízké.

Kapitola 1

Vymezení pojmů

Tato kapitola slouží k seznámení se s pojmy, které se vyskytují v této práci a jejichž znalost je pro její pochopení klíčová.

1.1 3D modely

3D model je objekt v euklidovském prostoru, jehož tvar je zaznamenán za pomoci jedné z možných reprezentací. Existuje mnoho různých reprezentací, které jsou k uložení objektů využívány. Mezi nejznámější z nich patří hraniční (boundary), objemová (voxel/cell) a reprezentace pomocí konstruktivní geometrie (constructive solid geometry neboli CSG).

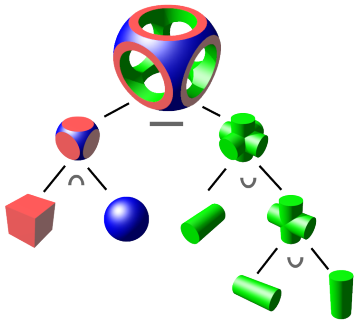
1.1.1 Objemová reprezentace

„V mnoha případech nemáme k dispozici geometrický popis tělesa, ale pouze sadu vzorků v určitém místě povrchu nebo objemu.“ Tato data jsou typicky uspořádána do pravoúhlé trojrozměrné mřížky. *„Při zobrazení těchto modelů se využívá voxelů, které vznikly jako analogie dvourozměrných pixelů.“* Objemová reprezentace se využívá v případech, kdy je vnitřní část modelu stejně důležitá nebo důležitější než vnější tvar – např. při zobrazení orgánů v medicíně, simulacích kapalin či počasí [1].

1.1.2 Konstruktivní geometrie

CSG (Constructiv Solid Geometry), česky konstruktivní geometrie, využívá při tvorbě modelu geometrických primitiv jako je krychle, koule, válec apod. a množinových operací mezi nimi (sjednocení, průnik, rozdíl apod.). Tyto modely pak mohou být znázorněny pomocí binárního stromu, kde listy obsahují primitiva a vnitřní uzly jednotlivé operace. *„Primární výhodou tohoto přístupu je parametrizovatelnost dílčích těles, umožňující přesnou kontrolou*

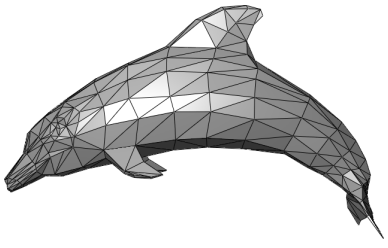
nad rozměry objektu. CSG se tedy hodí pro tvorbu modelů technických součástek, např. pro tisk na 3D tiskárně. Nevýhody spočívají v nutnosti definovat chování množinových operací v hraničních situacích (vznik non-manifold meshe) a velice obtížném vytváření modelů složitějších organických tvarů.“ [2]. CSG strom je znázorněn na obrázku 1.1.



Obrázek 1.1: CSG strom [3]

1.1.3 Hraniční reprezentace

Hraniční reprezentace je nejrozšířenějším způsobem ukládání 3D modelů, využívá se v počítačových hrách, virtuální realitě, filmech a dalších odvětvích. V hraniční reprezentaci se využívá bodů k určení tvaru modelu a jeho plochy. Tyto body jsou propojeny pomocí hran (úseček, případně křivek) a následně vytvářejí n–úhelníky. Tyto n–úhelníky tvoří plochu samotného modelu a nazývají se **facety**. V této práci se zabýváme výhradně modely s facety tvořenými třemi body. Skupina takovýchto bodů, hran a facetů se nazývá **mesh**, v případě, že jsou všechny facety tvořeny třemi body jde o **triangular mesh**, česky **trojúhelníkovou síť**. K uložení trojúhelníkové sítě je možno využít mnoho různých souborových formátů (OBJ, fbx, STL, ma, mel, vrml, dfx, blend, 3DS a jiné). 3D model v hraniční reprezentaci je zobrazen na obrázku 1.2.



Obrázek 1.2: Mesh [4]

Většina formátů byla vyvinuta přímo pro potřeby různých modelovacích programů (blend pro blender, 3ds pro 3D Studio max, ma a mb pro program

Maya atd.). Robustnější z nich umožňují kromě samotného modelu uložit i textury, materiál, kostru apod. V rámci této práce je ovšem důležitý pouze formát **STL**.

1.1.3.1 Facet

Facet je n-úhelník v prostoru R^3. V případě, že $n = 3$ jde o trojúhelník v prostoru, který je tvořen třemi body. Jednotlivé dvojice těchto bodů tvoří hrany facetu.

1.1.4 Reprezentace pomocí STL

Pro účely této práce je zásadní seznámit se s formátem STL (STereoLithography), protože právě s tímto formátem pracuje program ADMesh. STL je jeden z formátů používaných k uložení 3D modelů na počítači. Formát STL vytvořila na konci 80 let 20. století firma 3D Systems a tento formát se díky své jednoduchosti rychle rozšířil. Každý model je v této reprezentaci tvořen množinou facetů. Každý facet obsahuje informaci o pozici 3 vrcholů, které ho tvoří, a navíc jeho normálový vektor. Původní specifikace určovala několik dalších pravidel, ale ne všechna z nich jsou dodržována i v dnešní době. Může se tak stát, že STL soubor z jednoho programu se v jiném programu nezobrazí zcela korektně. Tato pravidla jsou [5]:

- Normála facetu musí směřovat ven z modelu a vrcholy tvořící trojúhelník musí být v pořadí proti hodinovým ručičkám z pohledu z vnějšku modelu. Každý facet definuje povrch objektu a tedy tvoří hranici mezi vnitřní a vnější částí modelu.
- Každý facet musí sdílet dva vrcholy s přilehlými facety.
- Všechny koordináty vrcholů musí být kladné.

Zejména poslední z těchto pravidel se dnes často nerespektuje. Původní pravidla také určovala minimální délku stran trojúhelníku a maximální velikost trojúhelníku. STL tedy ve svojí základní podobě neobsahuje žádné informace o měřítku, barvě, materiálech a podobně, a umožňuje velice přehlednou reprezentaci modelů ve 3D. Existují ovšem specifikace STL, které formát o tyto informace rozšiřují, ale žádná z nich se nestala průmyslovým standardem. STL se ukládá dvěma základními způsoby, a to v ASCII nebo binárně. ASCII STL soubor je čitelný i pro člověka, zejména u základních geometrických objektů jako je např. krychle a obsahuje několik základních slov pro rozlišení jednotlivých prvků modelu. Díky tomu je pro člověka přehlednější, ale z důvodu opakujících se prvků je datově neefektivní. Naproti tomu binární STL soubor je mnohem efektivnější v uložení dat, za cenu ztráty jednoduché čitelnosti.

bitů		
80	ASCII	Hlavička
4	unsigned long integer	počet facetů
4	float	x normály
4	float	y
4	float	z
4	float	x vertexu 1
4	float	y
4	float	z
4	float	x vertexu 2
4	float	y
4	float	z
4	float	x vertexu 3
4	float	y
4	float	z

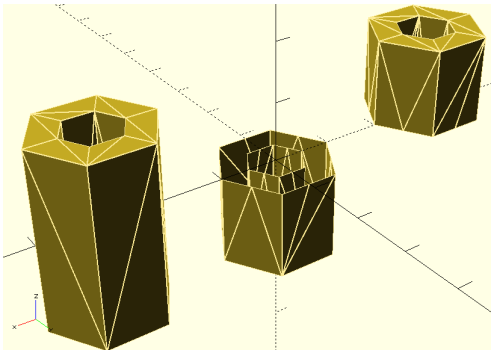
Binární STL, prvky v závorkách se opakují

```
solid name
  facet normal   n1 n2 n3
    outer loop
      vertex     x1 y1 z1
      vertex     x2 y2 z2
      vertex     x3 y3 z3
    endloop
  endfacet
endsolid name
```

ASCII STL, prvky v závorkách se opakují

1.2 Základní idea řezu 3D modelem

Úkolem rovinného řezu 3D modelem je rozdělit model na dvě rozdílné části, kde všechny body jedné části leží nad nebo na rovině a všechny body druhé části pod nebo na rovině. V zásadě jde o rozdělení modelu na trojúhelníky pod rovinou a trojúhelníky nad rovinou, s tím, že některé trojúhelníky (ty, které se vyskytují v místě řezu) je nutno rozdělit na menší. Řez jednoduchým modelem je zobrazen na obrázku 1.3. Pokud je žádoucí po řezu modelem vzniklé díry zaplnit trojúhelníky, je výstupem řezu množina hran, které leží na řezné rovině.



Obrázek 1.3: Řez 3D modelem

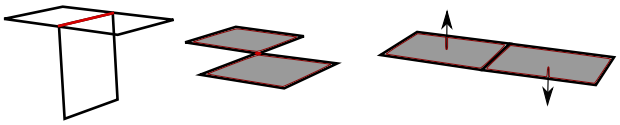
1.2.1 2–manifoldní modely

Vzhledem k tomu, že 3D modely v hraniční reprezentaci jsou tvořeny pouze pomocí stěn a chybí jakákoliv informace o jejich objemu, lze vytvořit takové modely, které nemohou v reálném světě existovat. Modely, které nejsou 2–manifoldní, obsahují problémy, které znemožňují jejich tisk a použití mnohými algoritmy. Mezi takové problémy patří:

- Spojené facety mají opačně orientované normály (jedna směrem vně modelu, druhá směrem do modelu – normála musí vždy vést vně modelu).
- Tři facety mají jednu společnou hranu.
- Dva modely mají společný pouze jeden bod či hranu (místa na modelu s nulovou tloušťkou).

Zjednodušeně lze říci, že korektní model pro 3D tisk musí být rozložitelný do 2D prostoru a zároveň uzavřený (krychle s chybějící stěnou je rozložitelná, ale není korektním modelem pro 3D tisk). S problematikou manifoldnosti modelů se lze blíže seznámit v knize Topological Structures for Geometric Modeling [6]. 2–manifoldnost modelů je potřebná k tomu, aby bylo možné jasně rozlišit

prostor uvnitř a vně modelu, což je nutná podmínka k úspěšnému 3D tisku. Problémy způsobující nemanifoldní objekty jsou zobrazeny na obrázku 1.4.



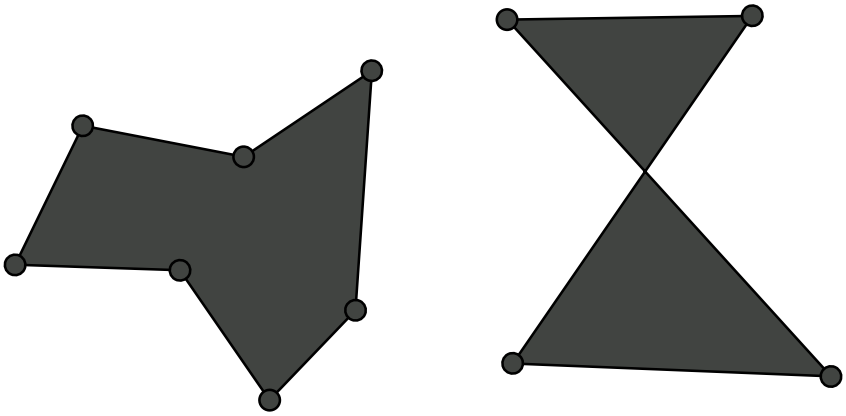
Obrázek 1.4: Problémy vedoucí k nemanifoldním modelům

1.3 Polygon

Polygonem v této práci rozumíme uspořádanou množinu n bodů V_0 až V_{n-1} v prostoru R^2. Po sobě následující dvojice bodů jsou spojeny hranami (V_i, V_{i+1}), $0 \leq i \leq n-2$ a hrana (V_{n-1}, V_0) spojuje první a poslední bod. Každý bod je součástí právě dvou hran.

1.3.1 Jednoduchý polygon

Jednoduchý polygon je polygon, jehož hrany se nikde neprotínají (s výjimkou společných bodů po sobě následujících hran). Pokud není řečeno jinak, výrazem polygon se v tomto textu rozumí jednoduchý polygon. Rozdíl mezi jednoduchým a nejednoduchým polygonem zobrazen na obrázku 1.5.



Obrázek 1.5: Jednoduchý a nejednoduchý polygon

1.4 Triangulace polygonu

Jde o rozklad polygonu na množinu trojúhelníků, které se vzájemně neprotínají. Každý polygon tvořený n body lze rozdělit na přesně $n - 3$ trojúhelníky [7]. Zvláštním případem je triangulace polygonu obsahující díru definovanou dalším polygonem. Příklady triangulace polygonů jsou na obrázku 1.6.

Obrázek 1.6: Triangulace polygonů

1.5 ADMesh

ADMesh je program a knihovna, která umožňuje provádět operace nad STL modely, a jehož počátky sahají do roku 1995. Jde o program pro příkazovou řádku. Existuje verze napsána v jazyku C a verze napsána v jazyku Python. ADMesh je šířen pod open–source licencí GPLv2+ a nabízí operace [8]:

- **Čtení a zápis** – Čtení a zápis binárních i ACII STL souborů a export do OBJ, OFF, VRML a DXF formátů.
- **Translace** – Pohyb modelem podle os x, y, z.
- **Rotace** – Rotace modelu podle os x, y, z.
- **Škálování** – Umožňuje škálovat (měnit velikost) modelu, podle os x, y, z, buď ze středu souřadnic, nebo ze středu modelu.
- **Zrcadlení** – Převrácení modelu dle zadané roviny.
- **Sloučení** – Spojení dvou STL modelů do jednoho.
- **Zaplnění děr** – Vytvoření chybějících facetů.
- **Oprava směru normál** – Přepočítání chybných normál facetů, aby směřovaly vně modelu.
- **Oprava velikosti normál** – Přepočítání chybných normál facetů, aby byly kolmé k facetu a jejich velikost se rovnala 1.
- **Odstranění chybných facetů** – Facety, jež mají dva body stejné, jsou odstraněny.
- **Spojení facetů** – Spojí blízké nepropojené facety podle zadané tolerance.
- **Výpočet objemu**

1.6 ADMeshGUI

ADMeshGUI je grafické uživatelské rozhraní pro program ADMesh, vytvořené v rámci bakalářské práce D. Vyvlečky. Umožňuje otevřít a zobrazit STL modely a provádět nad nimi veškeré operace, které ADMesh nabízí, plus rozdělení STL modelu na jednotlivé uzavřené modely [2].

1.7 Polyline

Polyline v počítačové grafice označuje lomenou čáru v prostoru R^2, která je složená z jednoho nebo více segmentů. Jde o uspořádanou n–tici bodů. Poly2Tri ji využívá k reprezentaci polygonů (obsahuje tedy každý bod polygonu právě jednou).

1.8 Řezná rovina

Řezná rovina v této práci značí rovinu, kterou je prováděn rovinný řez 3D modelem.

1.9 Vektor

Vektor je datový typ v programovacím jazyce C++. Jde o dynamické pole.

Kapitola 2

Analýza a návrh

Tato kapitola se zabývá identifikací problémů spojených s řezem 3D modelu a návrhem jejich řešení. Dále se zabývá algoritmy, které je možno či nutno při řezu 3D modelu využít a grafickým návrhem pro ADMeshGUI.

2.1 Cíle práce

Hlavním cílem této práce je implementace knihovny StlCut v jazyce C++, která umožní rovinný řez trojúhelníkovou sítí reprezentovanou ve formátu STL, rozšíření programu ADMeshGUI o tuto funkcionalitu a vytvoření samostatného programu využívající této knihovny k řezu STL modelem.

2.1.1 Funkční požadavky

- Knihovna bude distribuována jako open–source software.
- Knihovna umožní rovinný řez ASCII i binárním STL souborem.
- Parametry roviny budou nastavitelné uživatelem.
- Výstupem úspěšného řezu budou 2 STL soubory obsahující jednotlivé rozříznuté části modelu.
- Výstupní soubory budou 2–manifold, pokud to platilo o původním STL souboru.
- Program bude obsahovat nápovědu k použití.

2.1.2 Nefunkční požadavky

- K využití programu nebude nutná znalost programování.
- Knihovna bude napsána v jazyce C nebo C++.
- Vytvoření sady testů.

2.2 Souhrn problémů k řešení

Stručný rozpis všech náležitostí, které byly identifikovány jako nutné k řešení v rámci této práce.

2.2.1 Knihovna StlCut

V rámci práce na knihovně je nutno vyřešit následující:

- Navrhnout rozhraní.
- Načíst model a nastavit řeznou rovinu.
- Provést řez modelem (výstupem bude množina hran na řezné rovině a dva STL modely).
- Převést hrany do 2D prostoru a spojit je do polygonů (výstupem budou polygony).
- Provést triangulaci. (výstupem budou trojúhelníky v 2D prostoru).
- Převést trojúhelníky zpět do 3D prostoru.
- Připojit trojúhelníky k STL modelům.
- Uložit modely.
- Navrhnout testy a provést testování.

2.2.2 Program StlCut

Po implementaci knihovny, je nutno využít její funkce a vytvořit samostatný program pro příkazovou řádku umožňující řez modelem. Dále je třeba navrhnout vstupní a výstupní rozhraní pro interakci s uživatelem programu a vytvořit příručku k instalaci.

2.2.3 ADMeshGUI

V programu ADMeshGUI je nutno:

- Přidat tlačítka pro řez modelem.
- Zajistit zobrazení, korektní rotaci a pohyb roviny v závislosti na vstupu.
- Přidat funkci řezu rovinou pomocí knihovny StlCut.

2.3 Návrh rozhraní knihovny

Vzhledem k tomu, že celá knihovna má jeden jediný účel, bylo třeba při návrhu rozhraní zvážit jen několik nutných úkonů:

- Umožnit nastavit volitelné módy knihovny (tichý režim, nezotavování z chyb).
- Umožnit otevření souboru pomocí cesty i předáním na vstupu.
- Umožnit uložení modelů dle zadaného jména, bez zadaného jména nebo je vrátit jako návratovou hodnotu.
- Zajistit možnost provádět řezy jedním modelem opakovaně, bez nutnosti volat opakovaně jinou metodu než pro řez a uložení.
- Provést řez dle roviny zadané jako vstupní parametr.

2.4 Řez 3D modelu

Samotný řez 3D objektu, reprezentovaného triangulární sítí, je relativně jednoduchý problém. Nejdříve je třeba definovat rovinu, podle které se povede řez a připravit si datové struktury pro uložení trojúhelníků a bodů (hraniční body), které leží na rovině, nebo na ní řezem vzniknou. Prvotní návrh počítal s rovinou zadanou ve formě čtyř desetinných čísel a, b, c, d, které by odpovídaly rovině dle obecné rovnice roviny

$$ax + by + cz + d = 0\,. \tag{2.1}$$

Takováto forma zadání roviny ale není uživatelsky intuitivní, obzvlášť v případě využití řezu mimo ADMeshGUI. Ideální by bylo, aby byla při zadání roviny jasná jak její normála, tak vzdálenost od středu souřadnicového systému. Vzdálenost l bodu $[x_1, x_2, x_3]$ od roviny je

$$l = \frac{|ax_1 + bx_2 + cx_3 + d|}{\sqrt{a^2 + b^2 + c^2}}\,. \tag{2.2}$$

Při výpočtu vzdálenosti od středu souřadnicového systému $[0, 0, 0]$ tedy platí, že

$$l = \frac{|d|}{\sqrt{a^2 + b^2 + c^2}}\,. \tag{2.3}$$

Jmenovatel na pravé straně rovnice odpovídá velikosti normály roviny. Z této rovnice je jasné, že v případě, že velikost normály bude vždy rovna jedné, bude d odpovídat vzdálenosti od středu souřadnic. Při zadání roviny tedy stačí takto normalizovat normálový vektor

$$Normalizuj(a,b,c) = \left(\frac{a}{\sqrt{a^2+b^2+c^2}}, \frac{b}{\sqrt{a^2+b^2+c^2}}, \frac{c}{\sqrt{a^2+b^2+c^2}} \right). \tag{2.4}$$

Uživatel tedy může zadat rovinu pomocí normály a vzdálenosti od středu souřadnicového systému, ale v programu se mohou po normalizaci využívat výpočty na základě obecné rovnice roviny. Pozice p bodu $[x, y, z]$ vůči rovině se v případě potřeby vypočítá z rovnice roviny dosazením souřadnic x, y, z,

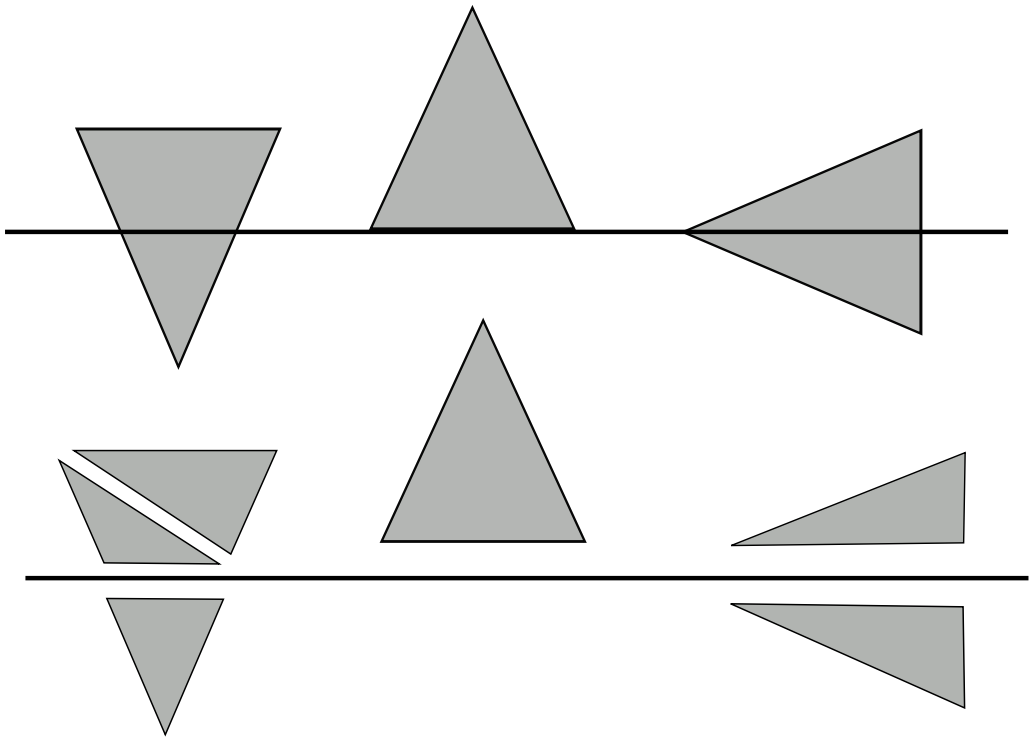
$$p = ax + by + cz + d\,. \tag{2.5}$$

Pokud je p větší než nula, jde o bod nad rovinou, pokud je menší než nula, jde o bod pod rovinou a pokud je p rovno nule, jde o bod ležící přímo na rovině.

Po zadání roviny je nutno zpracovat každý trojúhelník modelu, u každého vyhodnotit body, které trojúhelník definují a určit, zda spadají nad, pod nebo leží na rovině, viz obrázek 2.1. Tento proces má několik možných výsledků:

- Všechny tři body spadají pod nebo nad rovinu, trojúhelník je umístěn do odpovídající datové struktury reprezentující nové modely (nad nebo pod).
- Rovina prochází jedním z bodů a další dva jsou ve stejné skupině, trojúhelník je umístěn na základě pozice ostatních bodů.
- Jeden z bodů spadá do skupiny nad, zatímco ostatní do skupiny pod, či naopak. V tomto případě rozdělí rovina původní trojúhelník na trojúhelník a čtyřúhelník. Trojúhelník je tvořen jedním bodem původního trojúhelníku a dvěma body průsečíku mezi původním trojúhelníkem a rovinou. Čtyřúhelník je tvořen dvěma body průsečíku mezi trojúhelníkem a rovinou a dvěma body původního trojúhelníku. Čtyřúhelník je úhlopříčkou rozdělen na dva trojúhelníky a všechny tři trojúhelníky jsou umístěny do skupin dle pozic jejich bodů z původního trojúhelníku.
- Jeden z bodů leží na rovině a zbývající body jsou v rozdílných skupinách, vznikne jeden nový bod (průsečík), který rozdělí trojúhelník na dva, a ty přiřadíme do opačných skupin.
- Všechny tři body leží na rovině. Tento trojúhelník není uložen ani do jedné ze skupin.
- Dva z bodů leží na rovině a jeden pod nebo nad. Trojúhelník je uložen dle pozice bodu neležícího na řezné rovině.

Pokud vzniknou rozdělením trojúhelníků nové body, je odpovídající hrana ležící na řezné rovině uložena (jako dva body) k dalšímu zpracování. Pokud se objevil trojúhelník s alespoň jednou hranou ležící na řezné rovině (bez toho aby byl předem rozdělen) je uložen celý trojúhelník k dalšímu zpracování (viz další sekce).

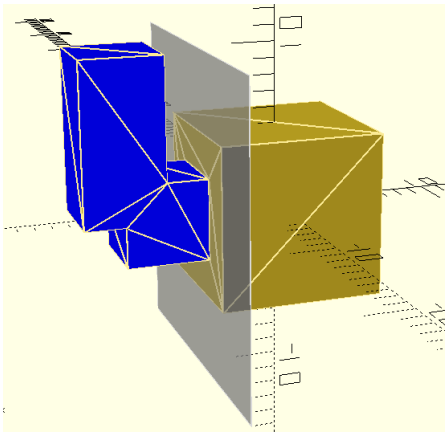


Obrázek 2.1: Rozdělení facetů dle roviny

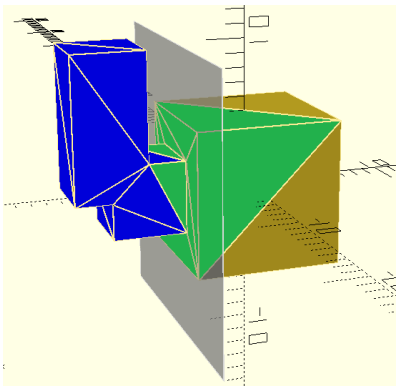
2.4.1 Řez s trojúhelníky na řezné rovině

Při většině řezů modelem je dostatečné provést řez, z uložených bodů identifikovat polygony, provést triangulaci a výsledné trojúhelníky použít pro zacelení díry v obou nově vzniklých modelech. Existují ale případy, kdy takovýto postup nestačí a je nutno, aby polygony obsahovaly rozdílné hrany. Jde o případy, kdy existují trojúhelníky ležící všemi body na řezné hraně. Takovouto situaci zobrazuje obrázek 2.2. Na tomto obrázku je šedivě zobrazena řezná hrana, která rozděluje model na modrou a žlutou část. Pokud by v takovémto případě došlo k triangulaci polygonů ležících na hraně, výsledkem by byla strana žluté krychle s dírou uprostřed v místech, kde se setkává s modrou částí (viz sekce 2.8), a to pro oba výsledné modely. Takový výsledek je nežádoucí, proto je v případě výskytu trojúhelníků na řezné rovině nutno využít jiný postup.

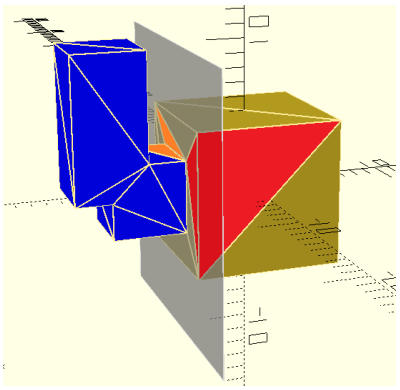
Pro uspokojivé vyřešení řezu s trojúhelníky na řezné rovině je nutné provést pro model nad rovinou i model pod rovinou rozdílné triangulace. Z obrázku 2.2 je jasné, že modrý model potřebuje triangulaci polygonu tvořeného čtyřmi body modrého modelu ležících na řezné rovině a žlutý triangulaci polygonu tvořeného čtyřmi body žlutého modelu (rohy krychle) ležících na řezné rovině. V rámci řezu modelu je proto nutno si uložit všechny trojúhelníky, jež mají na řezné rovině alespoň jednu hranu (na obrázku 2.3 zobrazeny zeleně).



Obrázek 2.2: Specifický případ řezu



Obrázek 2.3: Zvýraznění problémových trojúhelníků

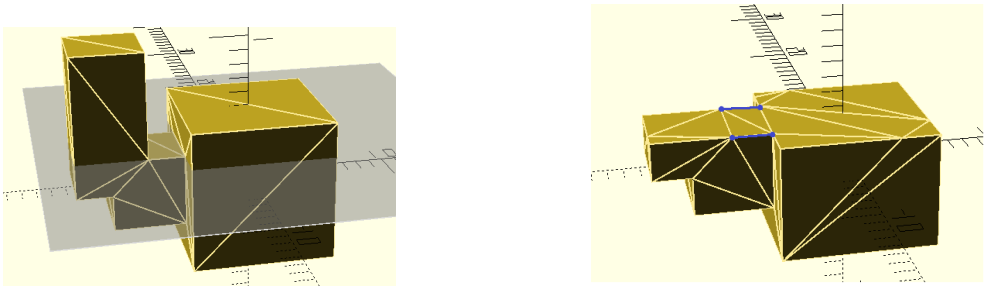


Obrázek 2.4: Identifikace trojúhelníků

Tyto trojúhelníky je nutno po dvojicích projít a pokud mají společnou hranu na řezné rovině, identifikovat tyto případy:

1. Jejich třetí body, neležící na řezné rovině, je každý na jiné straně roviny.
2. Jeden leží na rovině celý a druhý jen jednou hranou, přičemž poslední bod tohoto trojúhelníku leží pod rovinou (obr. 2.4 červeně).
3. Jeden leží na rovině celý a druhý jen jednou hranou, přičemž poslední bod tohoto trojúhelníku leží nad rovinou (obr. 2.4 oranžově).

V případě číslo jedna se body z této hrany uloží k dalšímu zpracování společně s body uloženými v rámci normálního řezu modelu. Naopak body z případu číslo dva a tři se uloží samostatně jako body patřící k části nad rovinou a body patřící k části pod rovinou.



Obrázek 2.5: Řez s pomocí bodů ležících na rovině

Než dojde k triangulaci, je nutno převést body na rovině na polygony ve formě polyline. Pro model ležící nad rovinou, se předají k převedení body získané běžným řezem a k nim se přidají body patřící k části nad rovinou. Pro model ležící pod rovinou, se předají k převedení body získané běžným řezem a k nim se přidají body patřící k části pod rovinou. Obrázek 2.5 zobrazuje řez modelem, na kterém jsou modře označeny body, které byly získány (a využity pouze pro dolní část modelu) právě tímto postupem. Modře jsou vyznačeny také hrany, které vedly k identifikaci těchto bodů.

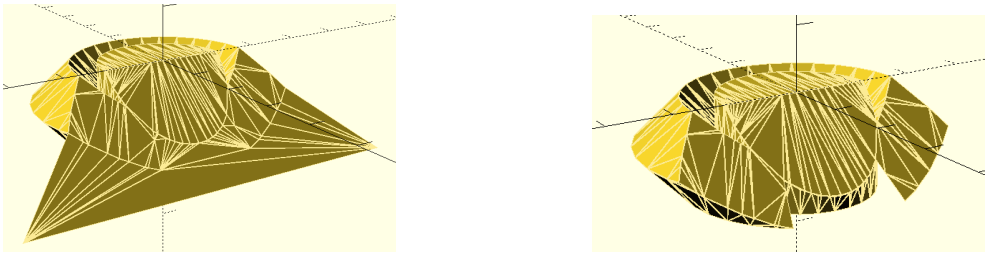
2.5 Triangulace polygonu

Triangulace polygonu je klasickým problémem výpočetní geometrie, který je námětem mnoha prací již po jedno století. Největšího pokroku v jeho řešení bylo dosaženo v 70. a 80. letech 20. století. Triangulace polygonu řeší v zásadě tento problém: Nechť je dána uspořádaná množina n vrcholů tvořící polygon, nalezněte $n-3$ úhlopříček, jež tento polygon rozdělí na $n-2$ trojúhelníků. Zásadním rozdílem mezi mnohými algoritmy řešícími tento problém je krom jejich rychlosti také fakt, zda umožňují triangulovat polygon obsahující díru, tedy další polygon uvnitř, a složitost jejich implementace. Nejpopulárnějšími triangulačními algoritmy jsou [9]:

- Recursive ear clipping algorithm [10].
- Sweep line algorithm [11].
- Incremental randomized algorithm [12].

2.5.1 Zvolené řešení

Vzhledem k tomu, že existuje několik open–source knihoven umožňujících triangulaci polygonů, nezdála se volba vlastní implementace jako nejvhodnější řešení. Ear clipping algoritmus je sice ze tří zmíněných algoritmů nejméně náročný na implementaci, ale sweep line algoritmus je rychlejší [7]. Při hledání



Obrázek 2.6: Neúspěšná a úspěšná triangulace

dostupných řešení třetích stran stojí za zmínku především projekt CGAL (The Computational Geometry Algorithm Library) a knihovna Poly2Tri.

Open–Source projekt CGAL nabízí řešení široké škály problémů souvisejících s výpočetní geometrií, ale právě pro jeho robustnost se mi využití této knihovny nezdálo vhodné. CGAL je projekt, který vznikl v roce 1996 jako konsorcium sedmi evropských a izraelských výzkumných institucí. Dnes je CGAL kolaborací mnoha odborných pracovníků z výzkumných institucí, univerzit a firem po celém světě [13]. Začlenit do práce knihovnu umožňující řešení desítky nesouvisejících problémů by nebylo ideální a navíc by znesnadnilo instalaci výsledného programu z důvodu závislosti na dalších knihovnách.

2.5.2 Poly2Tri

Z těchto důvodů byla nakonec zvolena knihovna Poly2Tri, která se zabývá pouze triangulací polygonů. Je napsaná v jazyce C (a také Python), distribuována pod open–source licencí a využívá sweep line algoritmus. Umožnuje triangulaci polygonů i s dírami a nemá žádný limit ani na počet bodů tvořících polygon, ani na počet děr v polygonu. Její instalace je jednoduchá, na některých Linuxových distribucích ji lze nainstalovat jako dodatečný balíček, případně stačí, stejně jako na systému Windows, stáhnout zdrojové soubory a zkompilovat je. Vstupem pro Poly2Tri je polygon a všechny jeho díry a výstupem je $(n-2)+2m$ trojúhelníků, kde n značí počet všech bodů těchto polygonů a m počet děr [9]. Poly2Tri ovšem garantuje takovýto výstup pouze v případě, že vstup je validní jednoduchý polygon. Pokud je vstupem nejednoduchý polygon, výsledek není definovaný. V některém případě Poly2Tri vytvoří dva nové body (obr. 2.6) a provede triangulaci i s nimi, v jiném případě selže a spadne v důsledku přístupu mimo paměť, vyřešením těchto problémů se zabývá sekce 2.9 a 3.2.8.

2.6 Přechod z 3D do 2D

Při rozříznutí modelu získáme sice množinu bodů, které je nutné dále zpracovat a triangulovat, ale triangulace pomocí Poly2Tri probíhá v dvourozměrném prostoru. Je tedy nutno nějakým způsobem převést body z 3D do 2D. Jako první se jeví myšlenka rotace modelu tak, aby byl řez pokaždé veden rovnoběžně s osou Z. Potom by bylo možné pokaždé zanedbat poslední souřadnici bodu, provést triangulaci a při převedení zpět do 3D by díky rovnoběžnosti s osou Z dostal každý bod stejnou hodnotu souřadnice Z. Vedoucí práce Ing. Hrončok ale poznamenal, že takovéto řešení vnáší do programu potenciální nepřesnosti výpočtů. Rotace modelu vyžaduje násobení desetinnými čísly a mohlo by se stát, že po rotaci, řezu a rotaci modelu zpět do původní polohy, by model byl mírně pozměněn. Tento postup byl tedy zavržen.

Zvolený postup s modelem nijak nepohybuje. Model je zcela identicky rozříznut, ale při převádění řezu na polygony je vypočteno, kterou souřadnici je možno zanedbat. Ta se zvolí na základě porovnání absolutních hodnot souřadnic normálového vektoru řezné roviny. Nejvyšší z nich bude zanedbána (v případě shody je jedna z nich vybrána). S tímto postupem ovšem vzniká nový problém - po triangulaci polygonů je nutno převést trojúhelníky zpět do 3D prostoru. Rovina ale nemusí být v tomto případě rovnoběžná s žádnou z os, je proto nutné si uložit nebo vypočítat poslední souřadnici. Při znovuzískání vynechané souřadnice musí být prohledána množina všech původních bodů ležících v řezné rovině a třetí bod dohledán. Tuto množinu je nutno seřadit tak, aby vynechaná souřadnice byla nejméně důležitou položkou při řazení, aby v ní bylo možno standardně vyhledávat. V případě, že by došlo k chybě a bod se nepodařilo nalézt, je možno hodnotu (obecně ne zcela přesně) dopočítat.

2.7 Převod na polyline

Tento proces obnáší vyhledávání navazujících hran. Z důvodu snahy co nejlépe zpracovat i nemanifoldní modely, se při vyhledávání navazujících hran musí dovolit jistá tolerance v ověření rovnosti jejich společného bodu. Navazující hrany musí být hledány na obě strany. To je nutné například při řezu objektů s chybějící stěnou, kde jedna hrana může polygonu zcela chybět a jednostranné vyhledávání by nemuselo všechny dostupné hrany nalézt.

2.8 Polygony a díry

Po získání jednotlivých polygonů a jejich uložení do pole je nutno rozlišit, které z nich jsou skutečně polygony, a které mají být identifikovány jako díry. Pro přehlednost budou polygony, u nichž o tom ještě nebylo rozhodnuto, označeny jako *potenciální polygony*. Při tomto rozdělení musí být bráno v potaz,

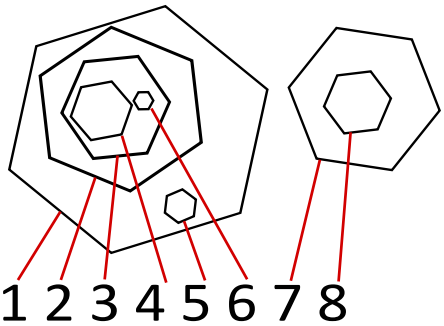
že uvnitř polygonu obsahujícího díru se může nacházet další polygon. Před rozdělením *potenciálních polygonů* je nutno je takto setřídit:

1. Určit plochu *potenciálních polygonů*.
2. Setřídit *potenciální polygony* sestupně podle plochy.
3. Odstranit *potenciální polygony* s nulovou plochou.

Rozdělení na polygony a díry probíhá takto:

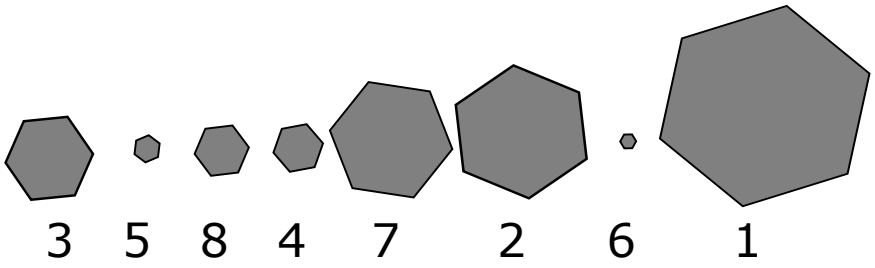
1. Odebrat *potenciální polygon* nebo skončit (když už žádný v poli nezbývá).
2. Určit, jestli je tento *potenciální polygon* uvnitř některého z již identifikovaných polygonů.
3. Pokud ne, uložit *potenciální polygon* jako polygon a přejít na bod 1.
4. Pokud ano, označit *potenciální polygon* jako díru v polygonu.
5. Projít všechny polygony a díry uvnitř polygonu z bodu 4 a zjistit, jestli je *potenciální polygon* uvnitř, pokud ano, změnit jeho označení (z díry na polygon a naopak). Po projití všech polygonů a děr je výsledkem konečné označení jako díra nebo polygon.
6. Uložit *potenciální polygon* podle konečného označení a přejít na bod 1.

Na obrázku 2.7 je znázorněn možný vstup pro tento algoritmus.

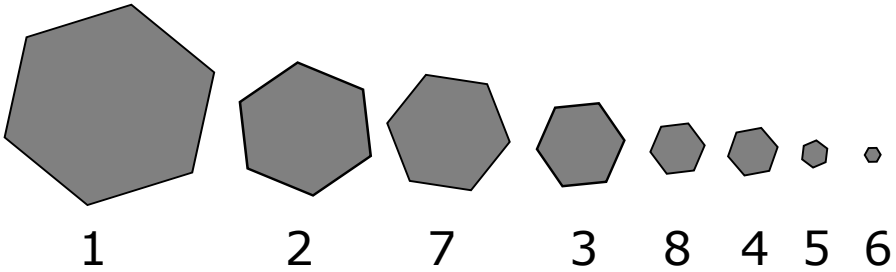


Obrázek 2.7: Vstup algoritmu pro nalezení děr

Na obrázku 2.8 je znázorněno jedno z možných uložení tohoto vstupu v poli. Na obrázku 2.9 je znázorněno toto uložení po setřídění dle plochy.



Obrázek 2.8: Nesetříděné polygony



Obrázek 2.9: Setříděné polygony

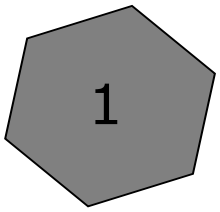
Nejprve se ze setříděného pole odebere *potenciální polygon* číslo 1. Vzhledem k tomu, že pole obsahující polygony, o nichž již bylo rozhodnuto, je prázdné, označí se tento *potenciální polygon* za polygon a uloží se do nového pole (viz obrázek 2.10).

Následuje odebrání *potenciálního polygonu* číslo 2. Vyhodnotí se, že je uvnitř polygonu 1, a je označen za díru. Vzhledem k tomu, že v polygonu 1 není žádný další polygon, je jako díra v polygonu 1 uložen.

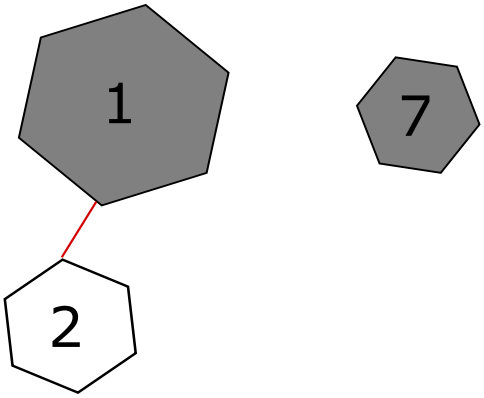
Po odebrání *potenciálního polygonu* číslo 7 se vyhodnotí, že neleží uvnitř žádného již identifikovaného polygonu, a je uložen jako nový polygon viz obrázek 2.11.

Následuje odebrání *potenciálního polygonu* číslo 3. Vyhodnotí se, že je uvnitř polygonu 1, a je označen za díru. Při průchodu všech polygonů a děr uvnitř polygonu 1 se zjistí, že *potenciální polygon* číslo 3 je také uvnitř díry číslo 2, a jeho stav je tedy změněn z díry na polygon. Polygon číslo 1 žádné další polygony ani díry pro porovnání neobsahuje, a *potenciální polygon* číslo 3 je uložen jako polygon uvnitř díry číslo 2. Toto rozdělení zachycuje obrázek 2.12.

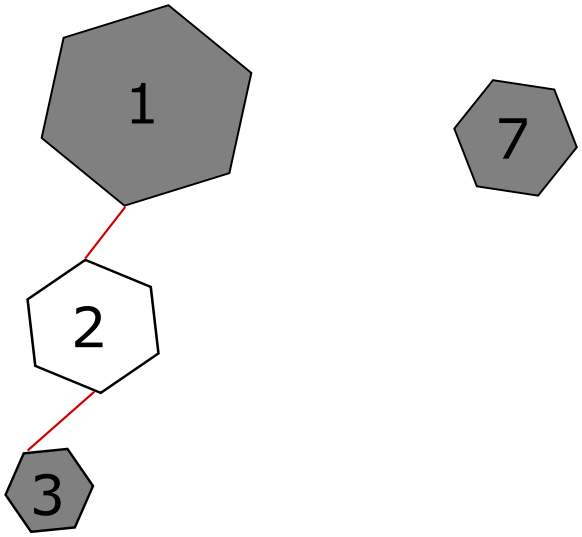
Na obrázku 2.13 je znázorněn výstup po projití všech *potenciálních polygonů*. Výsledek je pak znázorněn na obrázku 2.14, kde šedá plocha označuje prostor, který bude zaplněn trojúhelníky v rámci volání triangulačních funkcí knihovny Poly2Tri.



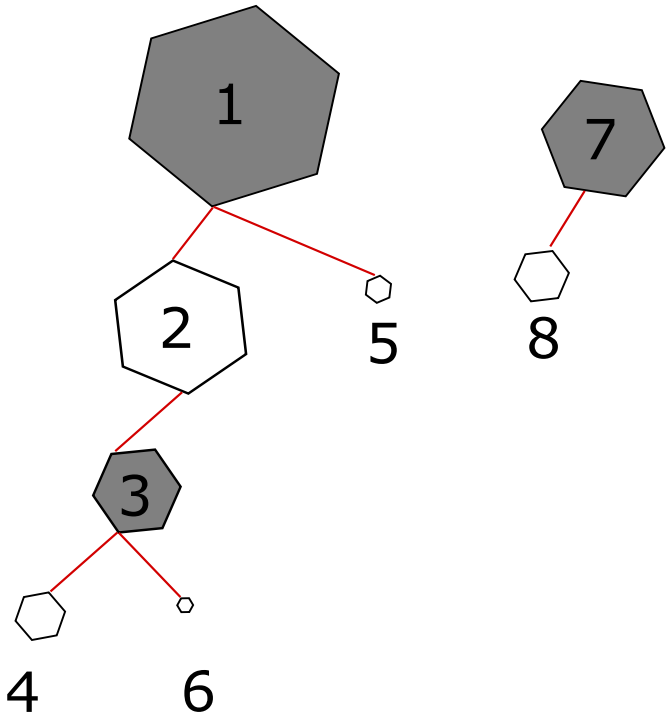
Obrázek 2.10: Postup při hledání děr



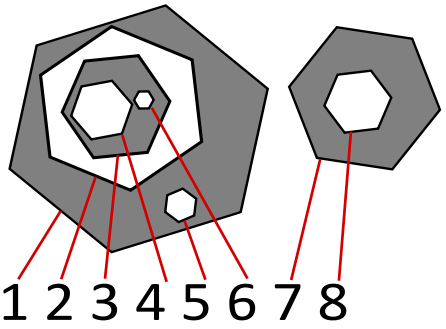
Obrázek 2.11: Postup při hledání děr



Obrázek 2.12: Postup při hledání děr



Obrázek 2.13: Postup při hledání děr

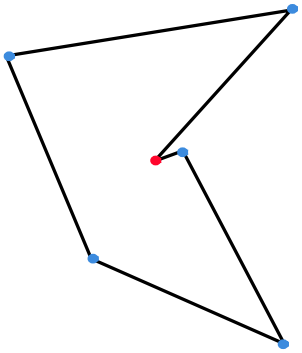


Obrázek 2.14: Výstup algoritmu pro hledání děr

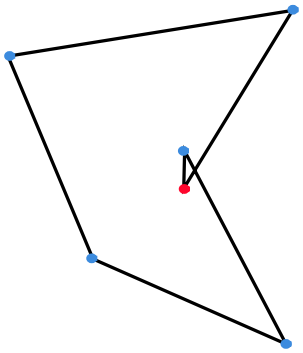
Po tomto rozdělení je možné pomocí Poly2Tri provést triangulaci.

2.9 Kontrola polygonů a triangulace

Program může na vstupu dostat nemanifoldní model a výsledkem řezu tak může být nejednoduchý polygon. Druhou variantou je, že nejednoduchý polygon vznikne v rámci řezu z důvodu nepřesnosti výpočtů. Je tedy nutné provést kontrolu polygonů před triangulací. Jak již bylo zmíněno v sekci 2.5.2, Poly2Tri může v případě nevalidního vstupu vytvořit nové body, což vede k nespráv-



Obrázek 2.15: Očekávaný polygon

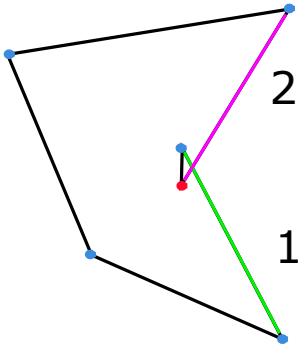


Obrázek 2.16: Skutečný polygon vzniklý kvůli nepřesnosti výpočtu

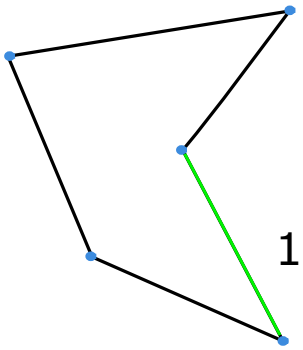
ným trojúhelníkům. Z toho důvodu bylo rozhodnuto, že po triangulaci bude provedena kontrola trojúhelníků.

2.9.1 Kontrola polygonů

Kontrola polygonů je navržena pouze k řešení nejednoduchých polygonů vzniklých nepřesností výpočtu, kde je pozice některého z bodů chybně určena tak, že vznikají protínající se hrany (obr. 2.15 a 2.16). Je nutno projít všechny hrany a zjistit, jestli se protínají. Pokud ano, tak smazat první bod druhé hrany (na obrázku červeně), viz obrázek 2.17 a 2.18.



Obrázek 2.17: Porovnání hran 1 a 2



Obrázek 2.18: Výsledek po smazání chybného bodu

2.9.2 Kontrola triangulace

Kontrola triangulace je postup, při kterém se projdou všechny trojúhelníky, které jsou výstupem triangulace v Poly2Tri, a ověří se, že každý jejich bod je mezi body uloženými na řezné rovině. V opačném případě je trojúhelník smazán. Pseudokódem:

```
for i = 0 to allTriangles
  for j = 0 to 2
    if(vertex j of triangle i is not found as point on border)
      delete triangle i;
```

2.10 Licence

StlCut je dostupný pod licencí GNU GPL verze 2 [14] na adrese `http://github.com/Nathaniell1/STL_CUT`. Tato verze je shodná s licencí, pod kterou je k dispozici knihovna ADMesh, což by mohlo v budoucnu zajistit začlenění knihovny StlCut do knihovny ADMesh. ADMeshGUI je dostupný pod licencí GNU AGPL verze 3 na adrese `http://github.com/admesh/ADMeshGUI`.

2.11 Návrh rozhraní programu

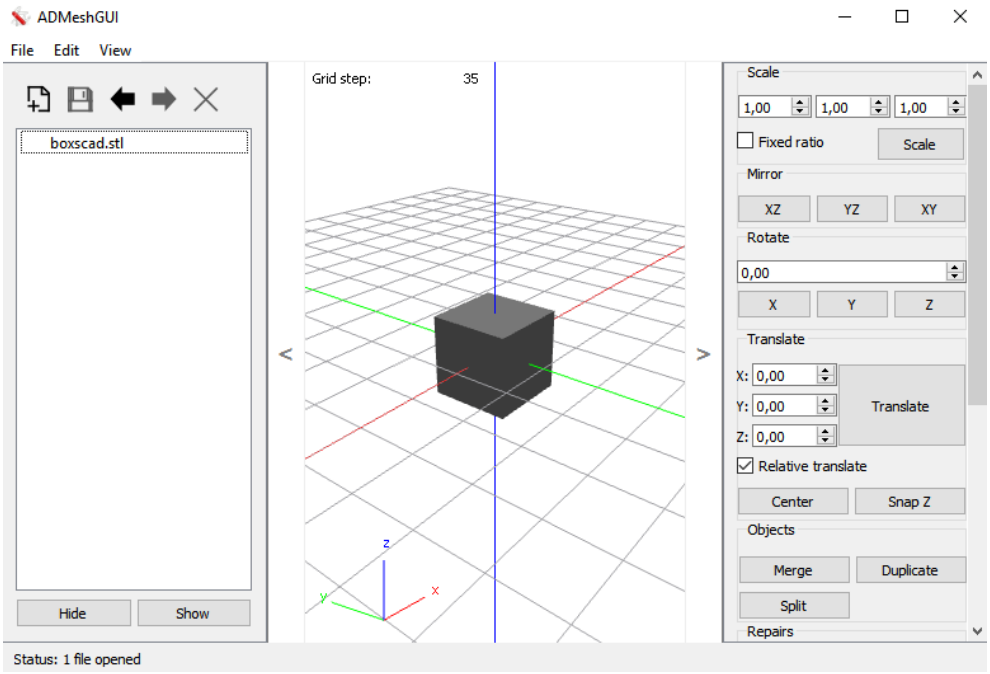
Návrh pro program StlCut byl velice jednoduchý. Je třeba pouze předat informaci o rovině a vstupním souboru a v případě úspěchu od uživatele získat název pro nové soubory. Na vstupu bude očekávána cesta k STL souboru a následně čtyři čísla reprezentující rovinu. Zadání roviny bude nepovinné, v případě absence čísel budou využity výchozí hodnoty. Pro rozličná nastavení chování programu bude využito přepínačů. Po úspěšném řezu vyzve program uživatele k zadání jména nových souborů, nebo umožní uložení pod výchozím jménem. V případě úspěchu i neúspěchu program informuje o výsledku uživatele.

2.12 Testování

Aby se předešlo stavu, kdy implementace jednotlivých testů bude součástí hlavní části knihovny, budou testy a jejich implementace přesunuty do vlastního programu. Testovací program půjde spustit buď s prázdným vstupem, v tom případě proběhnou pouze jednotkové testy, nebo bude přijímat na vstupu cestu k STL souborům. STL soubory rozřeže množstvím předem definovaných rovin a po každém řezu vypíše výsledek integračních testů. Na konci testů program přehledně vypíše, jestli testy proběhly bez problému, nebo ne. Jediným přepínačem, na který bude program reagovat, bude přepínač pro nápovědu (`--help`). Testovací program bude napsán v jazyce C++.

2.13 ADMeshGUI

ADMeshGUI poskytuje grafické rozhraní pro jednoduchou práci s STL modely a umožňuje jejich úpravu. Rozhraní programu je na obrázku 2.19. Pro jeho plné využití při přípravě modelu k 3D tisku ale chybí možnost model rozříznout pomocí roviny na několik částí, což je funkce v 3D tisku často potřebná. Při tisku na 3D tiskárně se rozřezáním modelů předchází nutnosti využití tisknutých podpěr pro ty části modelu, jež výrazně vyčnívají z modelu (typicky pokud je mezi dvěma tisknutými vrstvami v některých bodech rozdíl větší než 45°). Program umožňuje načíst STL modely, prohlédnout si je v několika módech zobrazení (drátový model, plný, plný se zvýrazněnými hranami) a provést nad nimi operace knihovny ADMesh. Grafické rozhraní je rozděleno do tří částí, v levé je seznam modelů, uprostřed je pohled na modely a vpravo je lišta s nabízenými operacemi.

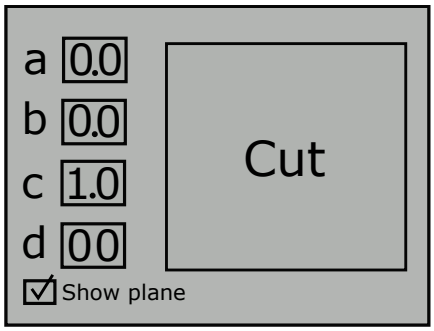


Obrázek 2.19: ADMeshGUI

2.13.1 Úprava GUI

Pro přidání možnosti provedení rovinného řezu je nutno přidat kolonky pro zadání roviny, tlačítko pro její zobrazení a tlačítko pro provedení řezu. Vzhledem k současnému rozdělení operací se jako ideální místo jeví pravá lišta s tím, že možnost řezu přibude až na jejím konci, což logicky odpovídá typickým operacím s modelem. Uživatel si nejdříve nastaví model do pozice, kterou potřebuje,

v případě potřeby provede jeho opravu, a teprve potom ho bude řezat. Návrh GUI pro zajištění rovinného řezu je na obrázku 2.20.



Obrázek 2.20: Návrh rozhraní pro řez

Kapitola 3

Implementace

Tato kapitola se zabývá zdrojovým kódem knihovny. Je zde popsána hlavní třída využitá v programu a blíže popisuje některé metody a problémy, které v nich nastaly.

3.1 Rozhraní knihovny

Na základě analýzy byla knihovna navržena tak, aby k provedení řezu stačilo několik jednoduchých kroků. Jde o:

1. Inicializaci – zavolání konstruktoru třídy `Mesh`.
2. Nastavení volitelných funkcí (nepovinné) – `setOptions(bool silent, bool errorRecovery)`.
3. Předání modelu – `setStl(stl_file file)`, `openStl(char * name)`.
4. Zavolání metody pro řez s řeznou rovinou jako vstupním parametrem – `cut(stl_plane plane)`.
5. Získání výsledku – `save(char* name)`, `getFinalStls()`.

Vstupní model lze definovat pomocí cesty (`openStl`), nebo předat již jako datovou strukturu (`setStl`) definovanou knihovnou ADMesh. Výstupní soubory lze rovnou uložit na disk (`save`) nebo navrátit jako dvojici `stl_file` (`getFinalStls`). Metodou `setOptions` lze nastavit, jestli má knihovna při běhu vypisovat informace na standardní a chybový výstup a jestli se má v případě problému pokusit o nový řez s mírně pozměněnou vzdáleností roviny od středu souřadnicového systému. Metoda `cut` vrací true, pokud řez úspěšně proběhl. Příklad volání knihovních funkcí pro uskutečnění řezu, viz 3.1.

```
Mesh mesh;
mesh.openStl("bunny.stl");
if (mesh.cut(stl_plane(0, 0, 1, 0)))
    mesh.save("half_of_bunny");
    //Saves files half_of_bunny_1.stl and half_of_bunny_2.stl
```

Kód 3.1: Příklad volání knihovních funkci StlCut

3.2 Třída Mesh

Tato třída zajišťuje většinu úkonů potřebných k manipulaci s STL modelem. Nejdůležitější, či implementačně nejzajímavější z nich, jsou:

- **cut** – Hlavní metoda volána uživatelem. Před jejím zavoláním je definován pouze vstupní model. Na konci jejího běhu jsou výstupní modely připraveny k uložení. Vrací true v případě úspěšného řezu.
- **divideFacets** – Rozdělí trojúhelníky modelu podle jejich pozice vůči rovině, vytvoří nové v místě řezu a nově vytvořené hrany uloží.
- **processOnFacets** – Zpracuje původní trojúhelníky s alespoň jednou hranou na řezné rovině.
- **processOnBorder** – Provede řez v případě, že na řezné rovině leží trojúhelníky.
- **createBorderPolylines** – Zpracuje hrany z metody `divideFacets` a převede je na polylines. Také se stará o převod z 3D do 2D souřadnic.
- **calculatePolygonArea** – Slouží k výpočtu plochy polygonu.
- **findHoles** – Porovnává polygony mezi sebou a rozlišuje, který z polygonů je určen k triangulaci a který jen ohraničuje díru v polygonu.
- **triangulateCut** – Využívá funkcí Poly2Tri. Provádí se v ní triangulace.
- **getMissingCoordinate** – Má za úkol převést 2D souřadnice triangulovaných polygonů zpět do 3D.
- **checkPoly2triResult** – Kontroluje, že výstup Poly2Tri neobsahuje trojúhelníky s novými body.

3.2.1 Metoda createBorderPolyline

V rámci této metody dochází k převedení hran ležících na řezné rovině (které jsou uloženy jako dvojice bodů ve vektoru) na polygony. Tyto polygony je nutno najít na základě navazujících bodů jejich hran. Aby se zlepšila schopnost programu provést řez skrz model, kde na sebe jednotlivé trojúhelníky

nenavazují dokonale (obr. 3.1), což je jeden z časných problémů, hledají se navazující hrany se zvyšující se tolerancí. Jako počáteční hodnota bylo zvoleno 10^{-25}.

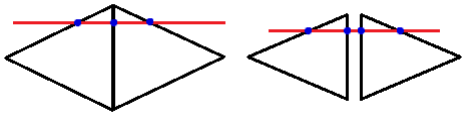
Průběh spojování hran do polygonů:

1. Odeber hranu z vektoru a ulož ji jako část nového polygonu, nebo skonči, když je vektor prázdný.

2. Hledej hranu navazující na hranu z bodu 1.

   - Pokud je tolerance větší než 0.25, ulož část polygonu jako dokončený polygon, zahlas varovaní, resetuj toleranci a pokračuj bodem 1.
   - Nalezena? Přidej ji do nového polygonu z bodu 1.
     - Pokud hrana neuzavřela polygon, přejdi na bod 2.
     - Pokud hrana uzavřela polygon, označ část polygonu z bodu 1 jako dokončený a pokračuj na bod 1.
   - Nenalezena? Zvyš toleranci na minimální rozdíl z minulé série vyhledávání a přejdi na bod 2.

Pro upřesnění, vzhledem k tomu, že polygony jsou ukládány jako polyline, neukládají se navazující hrany, ale pouze jeden z jejich bodů. Polygon ve formě polyline pro Poly2Tri musí obsahovat každý z bodů právě jednou. Při ukládání do polyline také dochází k převodu bodů do 2D. Souřadnice x, y nového bodu se zvolí z původního bodu na základě vynechané souřadnice. Pokud je vynechána souřadnice x, je jako souřadnice x nového bodu v 2D použita souřadnice z, při vynechání souřadnice y, je jako souřadnice y použita souřadnice z.



Obrázek 3.1: Řez navazujícími a nenavazujícími trojúhelníky

3.2.2 Metoda calculatePolygonArea

K výpočtu plochy jednotlivých polygonů je použita Gaussova metoda pro výpočet plochy mnohoúhelníku [15]. Kód metody, viz 3.2.

```
double Mesh::calculatePolygonArea(vector<p2t::Point*> polygon)
{
    int n = polygon.size();
    int j=0;
    double area = 0.0;
    for (int i = 0; i < n; ++i)
    {
      j=(i+1)%n;
      area += polygon[i]->x * polygon[j]->y;
      area -= polygon[j]->x * polygon[i]->y;
    }
    area = abs(area) / 2.0;
    return area;
}
```

Kód 3.2: Metoda pro výpočet plochy polygonů

3.2.3 Metoda findHoles

Tato metoda má za cíl uložit informaci o tom, které polygony lze opravdu označit za polygony a které za díry. Každý z polygonů je do této doby uložen jako vektor bodů. Pro rozlišení jestli jde o polygon, nebo díru, je použito celé číslo. Hodnota -1 určuje polygon, kladné číslo označuje díru a zároveň polygon v kterém se díra vyskytuje. Ke spojení polygonů a těchto čísel je využito datové struktury `pair`, která spojuje tyto data dohromady. Všechny polygony jsou nakonec uloženy v kontejneru z ukázky kódu 3.3.

```
vector< vector< pair <vector<p2t::Point*>,int> > > polygonsWithHoles;
```

Kód 3.3: Datová struktura pro uložení všech polygonů a děr

Každý prvek ve vnějším vektoru obsahuje vektor hlavních polygonů, tj. polygon, jež neleží uvnitř žádného dalšího polygonu, a všechny polygony v něm (polygon číslo 1 z obr. 2.12 a všechny polygonu uvnitř něho jsou tedy uloženy v `polygonsWithHoles[0]` a polygon číslo 7 a polygony v něm jsou uloženy v `polygonsWithHoles[1]`).

Pro zjištění, jestli je jeden polygon uvnitř druhého, je využita pomocná metoda `vertexInPolygon`, která testuje bod jednoho polygonu oproti všem hranám druhého. Tento algoritmus vede polopřímku z tohoto bodu, a pokud protne hranu polygonu, je přepnuto mezi stavem uvnitř a vně polygonu. Po otestování všech hran je výsledkem informace o poloze bodu. Tato metoda je pro jistotu (pokud by měly dva polygony překrývající se bod, ať už z důvodu nepřesných výpočtů nebo z důvodu chyby v modelu, nemusel by algoritmus v takovém případě vrátit správný výsledek) volána na tři body polygonu a výsledek je vybrán pomocí většiny. Algoritmus je popsán v [16] a jeho kód zobrazen v 3.4.

```
bool Mesh::vertexInPolygon( const vector<Point* >& polygon,
                    const double &testx, const double &testy)
{
  int i, j;
  bool in = false;
  //vector from point to the right(to infinity) vs edges
  for (i = 0, j = polygon.size()-1 ; i < polygon.size(); j = i++)
  {
    if( ((polygon.at(i)->y > testy) != (polygon.at(j)->y > testy)) &&
      (testx < (polygon.at(j)->x - polygon.at(i)->x) * (testy - polygon.at(i)->y)
       / (polygon.at(j)->y - polygon.at(i)->y) + polygon.at(i)->x)
       )
    {
       in = !in;
    }
  }
  return in;
}
```

Kód 3.4: Metoda rozhodující jestli bod leží uvnitř polygonu

3.2.4 Metoda triangulateCut

V rámci této metody jsou z vektoru `polygonsWithHoles` postupně vybírány polygony, které jsou vstupem pro konstruktor `CDT` knihovny Poly2Tri. Následně je, v případě, že obsahují polygony označené jako díra, volána metoda `AddHole`, která je jako díru přidá. Na polygony je po přidání všech jejich děr volána metoda `Triangulate`, která provede triangulaci, metoda `checkPoly2triResult`, která odstraní případné trojúhelníky navíc a metoda `createFaces`, která převede tyto trojúhelníky zpět do 3D a vytvoří z nich validní facety z pohledu knihovny ADMesh. Tato metoda je zobrazena v ukázce kódu 3.5.

3.2.5 Metoda processOnFacets

Úkolem této metody je projít všechny trojúhelníky, které ležely alespoň jednou hranou na řezné rovině. Všechny tyto trojúhelníky jsou porovnávány a hledají se takové, které mají shodnou hranu. Pokud se takové naleznou, nastávají tyto situace:

- Oba trojúhelníky mají na rovině právě jednu hranu a jejich body mimo rovinu leží na opačných stranách roviny. Hrana ležící na rovině je přidána k ostatním hranám vzniklým při řezu.
- Jeden trojúhelník leží celý na rovině, druhý právě jednou hranou a jeho bod mimo rovinu leží pod rovinou. Jejich společná hrana je uložena mezi hrany náležící ke zpracování pouze s modelem ležícím pod rovinou.

```
void Mesh::triangulateCut()
{
  map<int, p2t::CDT*> polygons;
  for (int i = 0; i < polygonsWithHoles.size(); ++i)
  {
      for (int j = 0; j < polygonsWithHoles[i].size(); ++j)
      {
          if(polygonsWithHoles[i][j].second==-1) // -1 = polygon
          {
              p2t::CDT* tmp=new  p2t::CDT(polygonsWithHoles[i][j].first);
              polygons.insert ( pair<int,p2t::CDT*>(j,tmp) );
          }
          else // found a hole
          {
              int holeIn=polygonsWithHoles[i][j].second;
              map<int, p2t::CDT*>::iterator it;
              it = polygons.find(holeIn);
              it->second->AddHole(polygonsWithHoles[i][j].first);
          }
      }
      if(polygons.size()>0)
        for (auto k = polygons.begin(); k != polygons.end(); ++k)
        {
            k->second->Triangulate();
            vector<p2t::Triangle*> triangles = k->second->GetTriangles();
            createFaces(triangles);
        }
      for (auto k = polygons.begin(); k != polygons.end(); ++k)
      {
        delete (*k).second;
      }
      polygons.erase(polygons.begin(),polygons.end());
  }
}
```

Kód 3.5: Metoda zajišťující úspěšnou triangulaci polygonů

- Jeden trojúhelník leží celý na rovině, druhý právě jednou hranou a jeho bod mimo rovinu leží nad rovinou. Jejich společná hrana je uložena mezi hrany náležící ke zpracování pouze s modelem ležícím nad rovinou.

Pokud byla nalezena alespoň jedna hrana, která náležela pouze k hornímu či dolnímu modelu, vrací tato metoda true, čímž signalizuje, že je nutno provést rozdílné triangulace pro horní a dolní model.

3.2.6 Metoda processOnBorder

Tato metoda supluje práci metody `cut`, pokud je nutné provést rozdílné triangulace pro horní a dolní část modelu. Volá metody pro vytvoření polygonů, nalezení děr a triangulaci. Před vytvořením polygonů do bodů na řezné rovině

(nalezených při prvotním řezu) přidá body patřící pouze k horní resp. dolní části modelu. Metoda je zobrazena v ukázce kódu 3.6.

```
bool Mesh::processOnBorder()
{
  if (!(topFacets.size()!=0 && botFacets.size() != 0 ))
    return false;
  vector<stl_vertex> borderBackUp = border;
  border.insert(border.end(), botBorder.begin(), botBorder.end());
  if(createBorderPolylines(false))
  {
    findHoles();
    triangulateCut(-1);
  }

  border = borderBackUp;
  border.insert(border.end(), topBorder.begin(), topBorder.end());
  if(createBorderPolylines(false))
  {
    findHoles();
    triangulateCut(1);
  }
  return true;
}
```

Kód 3.6: Metoda zajišťující řez pokud leží trojúhelníky na řezné rovině

3.2.7 Metoda cut

Metoda `cut` slouží k provedení řezu modelem. Z této metody se volají všechny ostatní metody související s provedením řezu. Na začátku dojde k nastavení pozice a proměnných pro obnovení z případné chyby. Potom jsou promazány všechny důležité proměnné pro zajištění možnosti opakovaného volání metody. Následuje nastavení řezné roviny a pak už jsou volány metody pro rozdělení modelu, vytvoření polygonů, nalezení děr a triangulace. Pokud je na konci této metody horní i dolní nový model neprázdný, vrací metoda true. Hlavní část metody `cut` je zobrazena v ukázce kódu 3.7.

3.2.8 Obnovení po chybě

V rámci testování programu byly odhaleny případy, kdy došlo k pádu programu v době triangulace knihovnou Poly2Tri z důvodu čtení mimo vymezenou paměť. Nejednalo se pouze o případy polygonů vzniklých řezem nemanifoldními modely, ale i o polygony, které se po otestování zdály zcela validní. Z tohoto důvodu knihovna obsahuje metodu, která v případě výskytu signálu segmentation fault provede skok zpět v programu, mírně upraví pozici roviny

```
cleanupVariables();
setPlane(plane);
divideFacets();
if(createBorderPolylines())
{
  findHoles();
  triangulateCut();
}

if(topFacets.size() != 0 && botFacets.size() != 0)
  return true;
return false;
```

Kód 3.7: Hlavní metoda knihovny StlCut, starající se o provedení řezu

a pokusí se o celý řez znovu. Pokud při řezu dochází opakovaně k chybám, program vrátí false a vypíše informaci, že tento řez nebyl schopen provést.

Na začátku metody `cut` je nastavena hodnota pro posunutí vzdálenosti roviny od středu souřadnicového systému v případě chyby na 0.00015. Pokud dojde k chybě, program zvýší globální proměnnou udávající počet výskytů chyby, přičte ke vzdálenosti roviny od středu nastavenou hodnotu a pokusí se o řez znovu. Pokud znovu dojde k chybě, hodnota je tentokrát odečtena, při další chybě vynásobena deseti a přičtena a tak dále. Metoda `cut` se tímto postupem pokusí v případě chyb o řez celkem šestkrát.

3.3 Rozhraní a implementace programu

Rozhraní programu StlCut je implementováno v C++ a přesně dle návrhu ze sekce 2.11. Program je typicky volán takto: `./stlcut bunny.stl 0 0 1 0`. Jediným skutečně povinným vstupním parametrem je cesta k STL modelu určenému k řezu. První tři čísla definují normálu roviny a poslední vzdálenost roviny od středu souřadnicového systému. Pokud rovina není zadána, použije se právě hodnota `0 0 1 0`. Při volání programu lze využít tří přepínačů: `-e`, `-s`, `-h`. Přepínač `-e` zajišťuje, že v případě chyby nedojde k obnovení programu. Přepínač `-s` spustí program v tichém módu a poslední přepínač zobrazí nápovědu pro použití programu. Pokud se uživatel pokusí spustit program s neplatným přepínačem, je vypsána informace o chybném přepínači a program se standardně spustí. V případě jinak chybného volání, například s pěti čísly, se program nespustí a zobrazí nápovědu. Ke zpracování přepínačů bylo využito knihovny getopt.

3.4 ADMeshGUI

Práce D. Vyvlečky [2] využívá k vytvoření uživatelského rozhraní frameworku Qt [17], s nímž jsem neměl naprosto žádnou předchozí zkušenost. Po počátečních nezdarech se podařilo přidat tlačítka pro řez dle původního návrhu. Dále bylo třeba udělat množství drobných úprav a vytvořit několik funkcí, propojujících StlCut s ADMeshGUI. Mezi ně patří:

- Funkce umožňující řez všemi vybranými modely.
- Funkce pro vytvoření roviny (aby uživatel měl vizuální reprezentaci, kudy povede řez a její přesun a rotaci na odpovídající místo dle zadaných dat).
- Upravení funkce starající se o vykreslování 3D objektů a upravení fragment shaderu (které umožňují zobrazení částečně průhledné roviny).

Rovina je vytvořena pomocí dvou facetů. Jde vlastně o STL soubor vytvořený přímo v kódu programu. Díky tomu s rovinou lze pracovat jako s ostatními načtenými STL modely. Žádná z původních operací ADMeshGUI nevyžadovala zobrazení dodatečných grafických prvků ve 3D. Z tohoto důvodu bylo implementačně nejjednodušší přidat rovinu podobným způsobem, jakým se přidávají ostatní modely. Důsledkem toho je, že rovina je zobrazena v levém menu jako model s názvem `plane` a lze jí vybrat a skrýt jako ostatní modely. Veškeré operace z pravého menu však byly upraveny tak, aby na řeznou rovinu nefungovaly (rovina má jedinečný atribut který ji identifikuje a zamezuje nad ní provádět operace).

Pokud je zvolen jiný než drátový mód zobrazení (vykresluje pouze hrany modelů), je řezná rovina vykreslena jako částečně průhledný model fialové barvy, viz obrázek 3.2. K zobrazení jednotlivých modelů je v ADMeshGUI použito OpenGL (open graphics library), což je knihovna sloužící pro vykreslování 2D i 3D grafiky. Pro zajištění částečné průhlednosti roviny bylo třeba před jejím vykreslením povolit režim prolínání barev pomocí příkazu `glEnable(GL_BLEND)`. Také bylo nutno změnit univerzální fragment shader používaný pro všechny modely v ADMeshGUI. Shader je program řídící výstup grafické karty při jednotlivých fázích zpracování obrazu. Při zpracování roviny je ve fragment shaderu změněn atribut určující že jde o rovinu na true, a na základě toho se pro rovinu použije jiná barva a nastaví částečná průhlednost viz 3.8.

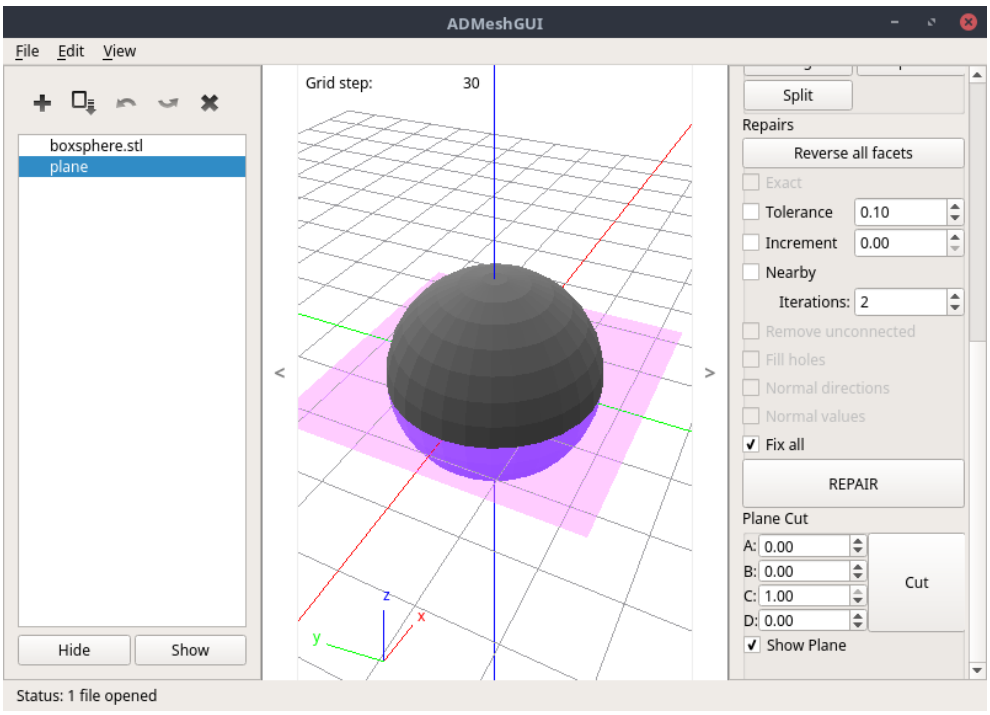
```
if(plane)
  gl_FragColor = vec4(0.5, 0.2, 0.9, 0.4);
```

Kód 3.8: Nastavení barvy roviny ve fragment shaderu

Správné pozice roviny dle zadání uživatele je docíleno jedním posunem a dvěma rotacemi. Vzorec pro rotaci byl nalezen v manuálu programu OpenSCAD [18] a je k vidění v ukázce kódu 3.9.

```
if(objectList[i]->isPlane())
{
    const float PI=3.14159265;
    resetPlane();
    objectList[i]->translate(true, 0, 0, d_cut);
    double length = sqrt(a_cut*a_cut + b_cut*b_cut + c_cut*c_cut);
    double rotY = acos(c_cut/length);
    double rotZ = atan2(b_cut, a_cut);
    objectList[i]->rotateY(rotY*180/PI);
    objectList[i]->rotateZ(rotZ*180/PI);
    objectList[i]->updateGeometry();
    reDrawSignal();
    return;
}
```

Kód 3.9: Nastavení pozice roviny v ADMeshGUI



Obrázek 3.2: ADMeshGUI – úprava rozhraní

Kapitola 4

Testování

Tato kapitola popisuje průběh testování, rozhraní testovacího programu a jeho integrační testy. K automatickému otestování funkčnosti byla vytvořena řada jednotkových testů a čtyři integrační testy.

4.1 Testovací program

Implementace testů je přesunuta do vlastního programu `tests.cpp`. Při absenci STL souborů na vstupu se provedou pouze jednotkové testy. Pokud jsou na vstupu STL soubory, jsou rozříznuty množstvím definovaných rovin a po každém řezu se provedou integrační testy na výsledek. Na konci program vypíše, jestli jednotkové a integrační testy proběhly úspěšně, nebo ne. StlCut je distribuován se složkou obsahující STL soubory k otestování. Standardní testovací volání vypadá takto: `./cuttests ./stl_files/*.stl`.

Na standardní výstup je nejprve vypsán výsledek jednotkových testů, následuje výsledek integračních testů pro jednotlivé modely. Po provedení všech testů je vypsána informace o úspěchu ve všech testech nebo o selhání některého z nich. Výpis je zobrazen v ukázce kódu 4.1.

4.2 Jednotkové testy

Pro většinu metod třídy `Mesh` byly vytvořeny jednotkové testy, které ověřují jejich základní funkčnost. Některé ze složitějších metod jednotkový test postrádají, protože jejich testování by vyžadovalo vytvoření celých STL modelů v kódu programu, funkčnost těchto metod je ověřena pomocí integračních testů. Příklad testu metody `vertexPosition`, která rozhoduje o pozici bodu vůči řezné rovině (viz ukázka kódu 4.2).

```
Testing setRemovedAxis
Testing intersection
Testing sortPolylines
Testing calculatePolygonArea
Testing vertexPosition
Testing checkPoly2triResult
            .
            .
            .
Testing file: ./stl_files/bunny.stl
1.000000 0.000000 0.000000 0.000000
MinMax test was succesfull
Volume test was succesfull
Triangle test was succesfull
Vertex test was succesfull
            .
            .
            .
All unit tests completed without a problem.
All intergration tests completed without a problem.
```

Kód 4.1: Výpis testovacího programu

4.3 Integrační testy

Cílem integračních testů je ověřit, že výstup celého programu je správný. Za tímto účelem byly vytvořeny čtyři integrační testy.

4.3.1 Porovnání minimálních a maximálních souřadnic bodů

První z testů využívá ADMesh funkce `stl_get_size`, která poskytuje informaci o minimálních a maximálních hodnotách souřadnic x, y, z vyskytujících se v modelu. V rámci tohoto testu se porovnají minima a maxima původního modelu s nově vzniklými modely. Postup:

1. Urči minima a maxima souřadnic x, y, z u původního modelu.
2. Urči minima a maxima souřadnic x, y, z prvního modelu.
3. Urči minima a maxima souřadnic x, y, z druhého modelu.
4. Porovnání hodnot jednotlivých minim a maxim souřadnic x, y, z mezi prvním a druhým modelem a výběr menších (minima), respektive větších (maxima) z nich.
5. Tyto hodnoty se porovnají s minimy a maximy původního modelu.
6. Pokud se shodují test dopadl úspěšně, jinak dopadl neúspěšně.

4.3.2 Porovnání objemů

Druhý z testů využívá ADMesh funkce `stl_calculate_volume`, která vypočítá objem modelu. Tento test pouze porovná, že objem původního modelu se shoduje se součtem objemů nových modelů s odchylkou jednoho procenta.

4.3.3 Ověření pozic bodů

Třetí test ověřuje, že každý z bodů v prvním novém modelu leží nad řeznou rovinou nebo na ní. A naopak, že každý z bodů v druhém novém modelu leží pod řeznou rovinou nebo na ní.

4.3.4 Ověření trojúhelníků

Poslední z testů prochází všechny trojúhelníky nově vzniklých modelů. Ověřuje, že jde o trojúhelníky identické s trojúhelníky z původního modelu a nebo, že tyto trojúhelníky mají alespoň jednu z hran na řezné rovině. To ověřuje tak, že body tvořící hrany trojúhelníku hledá mezi body na řezné rovině (ty byly uloženy v rámci výpočtu programu). Tento test tak ověřuje, že nové modely neobsahují žádné nové body mimo ty vytvořené na řezné rovině v důsledku řezu.

Porovnání všech trojúhelníků nových modelů s trojúhelníky původního modelu by bylo v případě lineárního procházení časově velmi náročné. Porovnání dvou trojúhelníků znamená porovnání, že jeho tři body se shodují s třemi body druhého trojúhelníku, což obnáší porovnání, že všechny souřadnice těchto bodů se rovnají. Ve výsledku jde o devět porovnání a to pouze v případě, že body trojúhelníků jsou ve stejném pořadí (trojúhelník tvořený body a, b, c je totožný s trojúhelníkem tvořeným body b, c, a, ale dokázat to programem znamená operace navíc oproti porovnávání dvou trojúhelníků tvořenými body a, b, c). Pro zrychlení tohoto procesu bylo třeba trojúhelníky seřadit. Bylo využito toho, že trojúhelníky neprocházející řeznou rovinou jsou ukládány (v metodě `divideFacets`) beze změn a všechny byly seřazeny podle velikosti souřadnice x jejich prvního bodu. Díky tomu je možné mezi trojúhelníky vyhledávat binárním půlením. Při hledání nějakého trojúhelníku je nalezen první z trojúhelníků, který má s hledaným shodnou hodnotu souřadnice x jeho prvního bodu a následně je lineárně hledáno, dokud se souřadnice x prvních bodů shodují s hledaným trojúhelníkem. Pokud k hledanému trojúhelníku není nalezen identický trojúhelník, otestuje se jestli má trojúhelník hranu na řezné rovině, jejíž body jsou uloženy v paměti (při řezu trojúhelníků se všechny nové body ukládají pro účel převodu z 2D zpět do 3D). V případě, že by nějaký trojúhelník nevyhověl ani jednomu z testů, tento integrační test selže.

```
bool Mesh::t_vertexPosition()
{
  fails.clear();
  int num=1;
  cout<< "Testing vertexPosition" <<endl;
  setPlane(stl_plane(1, 1, -1, 5));
  setRemovedAxis();

  if(!(vertexPosition(getVertex(0,0,10)) == below))
    fails.push_back(num);
  num++;
  if(!(vertexPosition(getVertex(0,0,-10)) == above))
    fails.push_back(num);
  num++;
  setPlane(stl_plane(1, 0, 0, 0));
  setRemovedAxis();
  if(!(vertexPosition(getVertex(1,5,10)) == above))
    fails.push_back(num);
  num++;
  if(!(vertexPosition(getVertex(1,5,-10)) == above))
    fails.push_back(num);
  num++;
  if(!(vertexPosition(getVertex(0,0,0)) == on))
    fails.push_back(num);
  num++;
  if(!(vertexPosition(getVertex(0,0,5)) == on))
    fails.push_back(num);
  num++;

  if(fails.size()>0)
  {
    writeFails();
    return false;
  }
  else return true;
}
```

Kód 4.2: Příklad jednotkového testu

Závěr

Cílem této práce bylo vytvořit knihovnu a program umožňující vést rovinný řez trojúhelníkovou sítí, výsledný řez triangulovat a vytvořit dva nové modely. Z hlediska běžného uživatele je hlavním přínosem rozšíření programu ADMeshGUI o možnost rovinného řezu. Program lze nyní pohodlně využít k většině úkonů spojených s přípravou 3D modelu k tisku.

Na začátku práce jsou uvedeny způsoby reprezentace 3D modelů na počítači a základní pojmy a postupy v této práce využívané. Také je zde představen program ADMeshGUI a samotná knihovna ADMesh.

V kapitole Analýza a návrh jsou identifikovány jednotlivé problémy, které je nutno při řezu modelem vyřešit a teoreticky popsáno jejich řešení. Také jsou identifikovány požadavky na rozhraní knihovny, samotného programu i programu na testování. V části zabývající se programem ADMeshGUI se lze seznámit s rozhraním tohoto programu, s důvodem, proč se modely při přípravě k 3D tisku řezají a je navrženo grafické rozhraní pro rozšíření programu.

V kapitole Implementace jsou popsány finální rozhraní knihovny i programu StlCut. Jsou zde popsány důležité metody, sloužící pro řez modelem i s ukázky kódu. Kapitola také popisuje způsob rozšíření programu ADMeshGUI o možnost rovinného řezu.

Poslední kapitola se zabývá testováním, programem k testování vytvořeným a zejména popisuje jednotlivé integrační testy.

Literatura

[1] ŽÁRA, J.; BENEŠ, B.; SOCHOR, J.; aj.: *Moderní počítačová grafika.* Brno: Computer Press, druhé vydání, 2005, ISBN 80-251-0454-0.

[2] VYVLEČKA, D.: ADMeshGUI: STL viewer and manipulation tool [online]. 2015, [cit. 2017-4-21]. Dostupné z: `https://github.com/admesh/ADMeshGUI`

[3] Wikimedia Commons: CSG Tree. [online], 2016, [cit. 2017-5-11]. Dostupné z: `https://commons.wikimedia.org/w/index.php?title=File:Csg_tree.png`

[4] Wikimedia Commons: Dolphin triangle mesh. [online], [cit. 2017-5-10]. Dostupné z: `https://upload.wikimedia.org/wikipedia/commons/f/fb/Dolphin_triangle_mesh.png`

[5] BURNS, M.: The STL format. *Historical Resource on 3D Printing [online]*, [cit. 2017-4-22]. Dostupné z: `http://www.fabbers.com/tech/STL_Format`

[6] WEILER, K.: *Topological Structures for Geometric Modeling.* Technical report RPI, Center for Interactive Computer Graphics, Rensselaer Polytechnic Institute, 1986. Dostupné z: `https://scorec.rpi.edu/REPORTS/1986-1.pdf`

[7] O'ROURKE, J.: *Computational geometry in C.* New York, NY, USA: Cambridge University Press, druhé vydání, 1998, ISBN 978-0521649766.

[8] ADMesh contributors: ADMesh – STL mesh manipulation tool: ADMesh 0.98.1 documentation [online]. 2015, [cit. 2017-4-25]. Dostupné z: `http://admesh.readthedocs.io/`

[9] LIANG, W.: Poly2Tri: Fast and Robust Simple Polygon Triangulation With/Without Holes by Sweep Line Algorithm. 2005, [cit. 2017-4-24]. Dostupné z: `http://sites-final.uclouvain.be/mema/Poly2Tri/`

[10] TOUSSAINT, G.: Efficient triangulation of simple polygons. *The Visual Computer*, ročník 7(5-6), 1991: s. 280–295, ISSN 0178-2789, doi: 10.1007/BF01905693. Dostupné z: http://link.springer.com/10.1007/BF01905693

[11] CHAZELLE, B.: Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, ročník 6(3), 1991: s. 485–524, ISSN 0179-5376, doi:10.1007/BF02574703. Dostupné z: http://link.springer.com/10.1007/BF02574703

[12] SEIDEL, R.: A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, ročník 1(1), 1991: s. 51–64, ISSN 09257721, doi:10.1016/0925-7721(91)90012-4. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0925772191900124

[13] CGAL Project, The: *CGAL User and Reference Manual*. CGAL Editorial Board, 4.10 vydání, 2017. Dostupné z: http://doc.cgal.org/4.10/Manual/packages.html

[14] Free Software Foundation, Inc.: GNU General Public License. 1991, [cit. 2017-07-30]. Dostupné z: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

[15] BRADEN, B.: The Surveyor's Area Formula. *The College Mathematics Journal*, ročník 17, č. 4, 1986: s. 326–337, ISSN 07468342, 19311346. Dostupné z: http://www.jstor.org/stable/2686282

[16] SUTHERLAND, E. E.; SPROULL, R. F.; SCHUMACKER, R. A.: A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys*, ročník 6, č. 1, Jan 1974: str. 1–55, ISSN 03600300, doi: 10.1145/356625.356626.

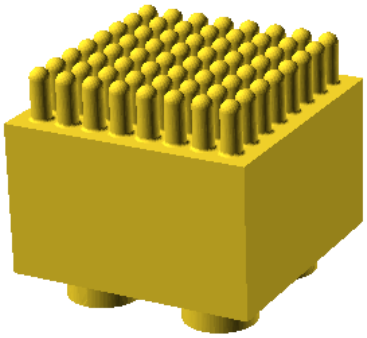
[17] Qt Company, The: Qt project [online]. [cit. 2017-04-15]. Dostupné z: http://www.qt.io

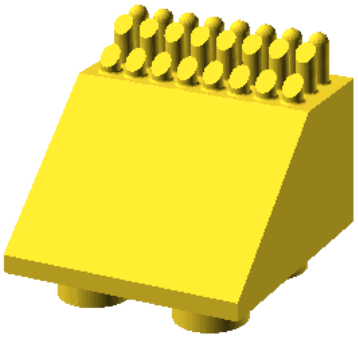
[18] Wikibooks: OpenSCAD User Manual/The OpenSCAD Language — Wikibooks, The Free Textbook Project. [online], 2016, [cit. 2017-7-30]. Dostupné z: https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Transformations

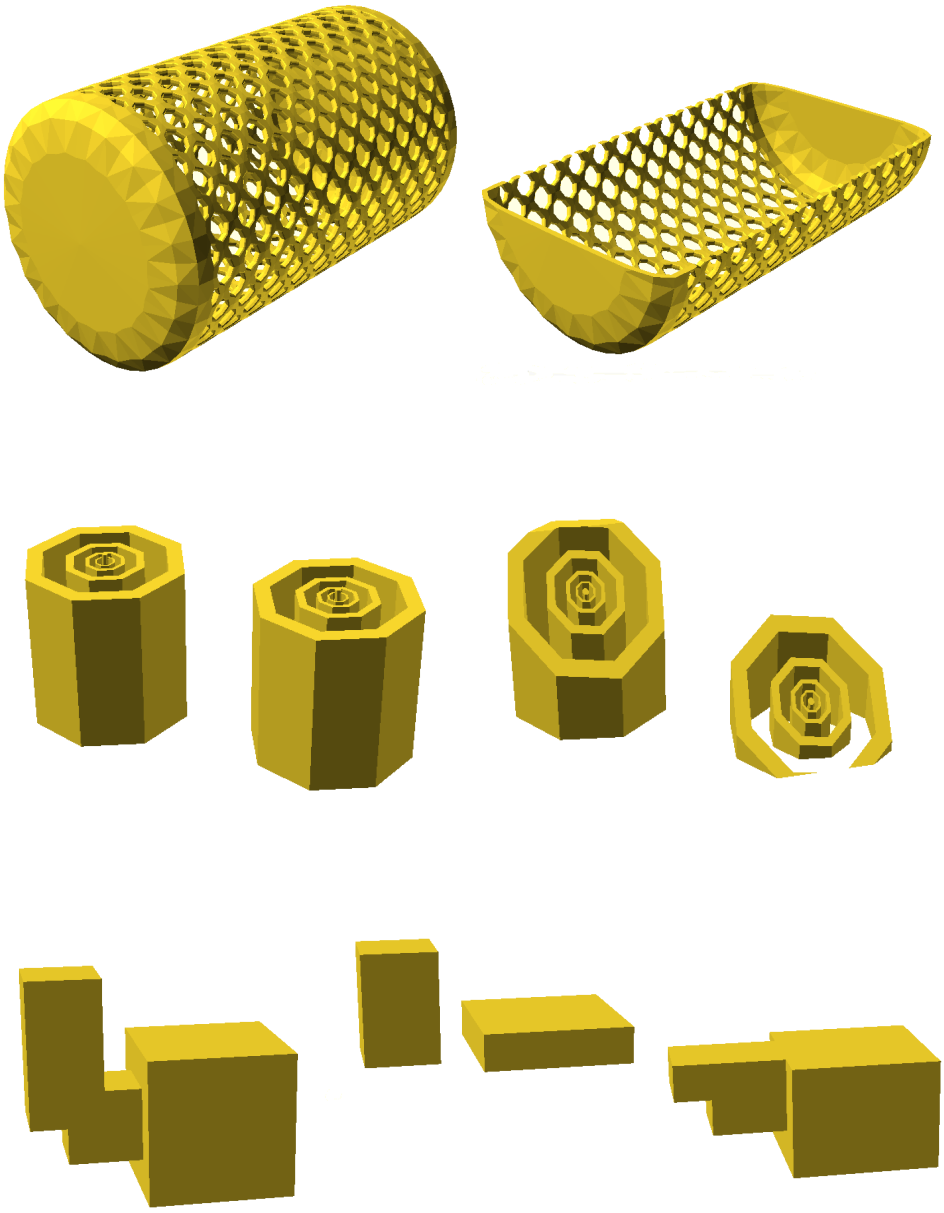
Příloha A

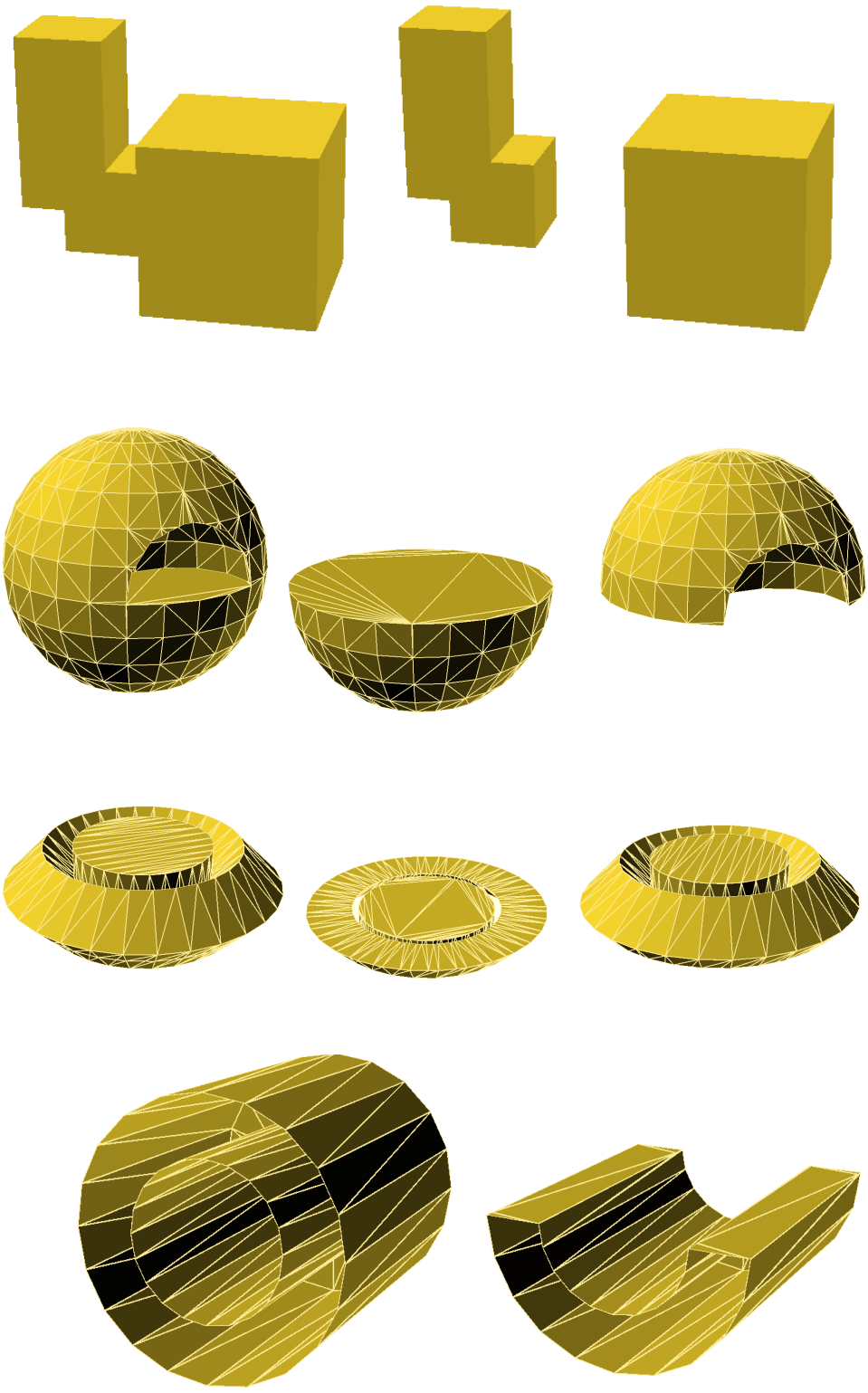
Obrazová dokumentace

Následuje množství obrázků zobrazujících modely rozříznuté pomocí StlCut. Poslední zobrazuje řez nemanifoldním modelem, kterému chybí celá stěna.









Příloha B

Seznam použitých zkratek

GUI Graphical User Interface

CGAL The Computational Geometry Algorithm Library

STL Stereo Lithography

GPL General Public License

OpenGL Open Graphics Library

CSG Constructive Solid Geometry

3D Three–dimensional space

2D Two–dimensional space

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové soubory
  - `stlcut` ..... soubory ke knihovně stlcut
  - `ADMeshGUI` ..... soubory k programu ADMeshGUI
- `Bakalarska_prace`
  - `src` ..... zdrojové soubory k textu
  - `BP_Ocenasek_Michael_2017.pdf` ..... práce ve formátu pdf