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

16. května 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 16. května 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 umožnuje 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 je napsána v programovacím jazyce C++ a testování je prováděno za pomoci programovacího jazyka Python.

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 plane cut of 3D models. 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 is written in C++ programming language and testing is done with use of Python.

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 (constructiv solid geometry neboli CSG).

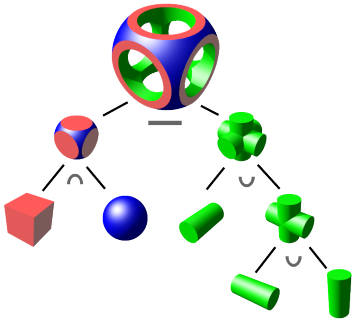
1.1.1 Objemová reprezentace

„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*

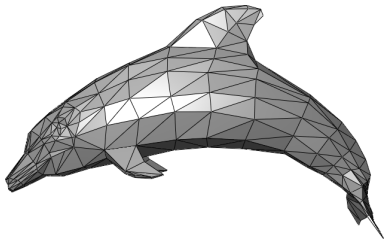
nad rozměry objektu. CSG se tedy hodí pro tvorbu modelů technických součástek, např. pro tisk na 3D tiskárně. Nevýhody spočívají v nutnosti definovat chování množinových operací v hraničních situacích (vznik non-manifold meshe) a velice obtížném vytváření modelů složitějších organických tvarů.“[2]. CSG 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 troj nebo více úhelníky. Tyto n–úhelníky tvoří plochu samotného modelu a nazývají se **facety**, při dalších zmínkách o nich v tomto textu budeme mluvit výhradně ve smyslu trojúhelníků. Skupina takovýchto bodů, hran a facetů se nazývá **trojúhelníková síť**, neboli **mesh**. 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 z nich byla vyvinuta přímo pro potřeby různých modelovacích programů (blend pro blender, 3ds pro 3D Studio max, ma 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**.

1.1.4 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 umožnuje 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	$n_1\ n_2\ n_3$ ⎫
outer loop	⎪
vertex	$x1\ y1\ z1$ ⎪
vertex	$x2\ y2\ z2$ ⎪
vertex	$x3\ y3\ z3$ ⎪
endloop	⎪
⎩ endfacet	⎭

endsolid name

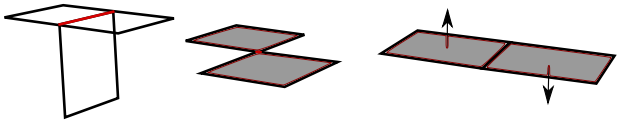
ASCII STL, prvky v závorkách se opakují.

1.1.5 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). Pokud se chce čtenář s tímto problémem blíže seznámit, věnuje se mu kniha 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 1.3.



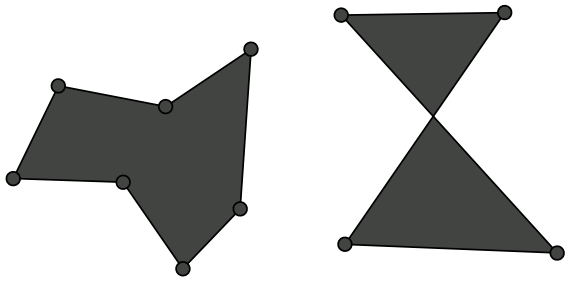
Obrázek 1.3: Problémy vedoucí k nemanifoldním modelům

1.2 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 a hrany se mohou protínat pouze v jednotlivých bodech.

1.2.1 Jednoduchý polygon

Jednoduchý polygon je polygon, jehož hrany se nikde neprotínají (s výjimkou jednotlivých bodů). Pokud není řečeno jinak, výrazem polygon se v tomto textu rozumí jednoduchý polygon. Rozdíl mezi jednoduchým a nejednoduchým polygonem zobrazen na obrázku 1.4.



Obrázek 1.4: Jednoduchý a nejednoduchý polygon

1.3 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 v C a v Pythonu. ADMesh je šířen pod open–source licencí GPLv2+ a nabízí operace [7]:

- **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**

1.4 ADMeshGUI

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

1.5 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.

1.6 Rovinný řez

Rovinným řezem 3D modelem se rozumí takový proces, jehož výsledkem je rozdělení modelu na dvě oddělené části (v případě, že rovina modelem prochází) v závislosti na poloze roviny.

1.6.1 Řezná rovina

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

Kapitola 2

Cíl práce

Cílem této práce je implementace knihovny StlCut v jazyce C++, která umožní rovinný řez trojúhelníkovou sítí reprezentovanou ve formátu STL a zároveň rozšíření programu ADMeshGUI o tuto funkcionalitu.

2.1 Funkční požadavky

- Knihovna bude distribuována jako open–source software.
- Knihovna umožní rovinný řez ASCII i binárním STL souborem.
- Parametry roviny budou nastavitelné uživatelem.
- Knihovna bude využitelná i jako samostatný program.
- Výstupem programu budou 2 STL soubory obsahující jednotlivé rozříznuté části modelu.
- Tyto soubory budou 2–manifold, pokud to platilo o původním STL souboru.
- Program bude obsahovat nápovědu k použití.
- Knihovna bude napsána v jazyce C++.
- Součástí knihovny bude i sada testů.

2.2 Nefunkční požadavky

- K využití programu nebude nutná znalost programování.

Kapitola 3

Analýza a návrh

Analýza se zabývá identifikací problémů, na které narazíme při práci s 3D modely a zvoleným řešením. Dále se zabývá algoritmy, které je možno či nutno při řezu 3D modelu využít a grafickým návrhem pro ADMeshGUI.

3.1 Řez 3D modelu

Samotný řez 3D objektu, reprezentovaného triangularní 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:

$$ax + by + cz + d = 0 \tag{3.1}$$

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 D bodu (x_1, x_2, x_3) od roviny je:

$$D = \frac{|ax_1 + bx_2 + cx_3 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{3.2}$$

Při výpočtu vzdálenosti od středu souřadnic tedy platí:

$$D = \frac{|d|}{\sqrt{a^2 + b^2 + c^2}} \tag{3.3}$$

Jmenovatel v tomto výpočtu 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 normalizovat normálový vektor.

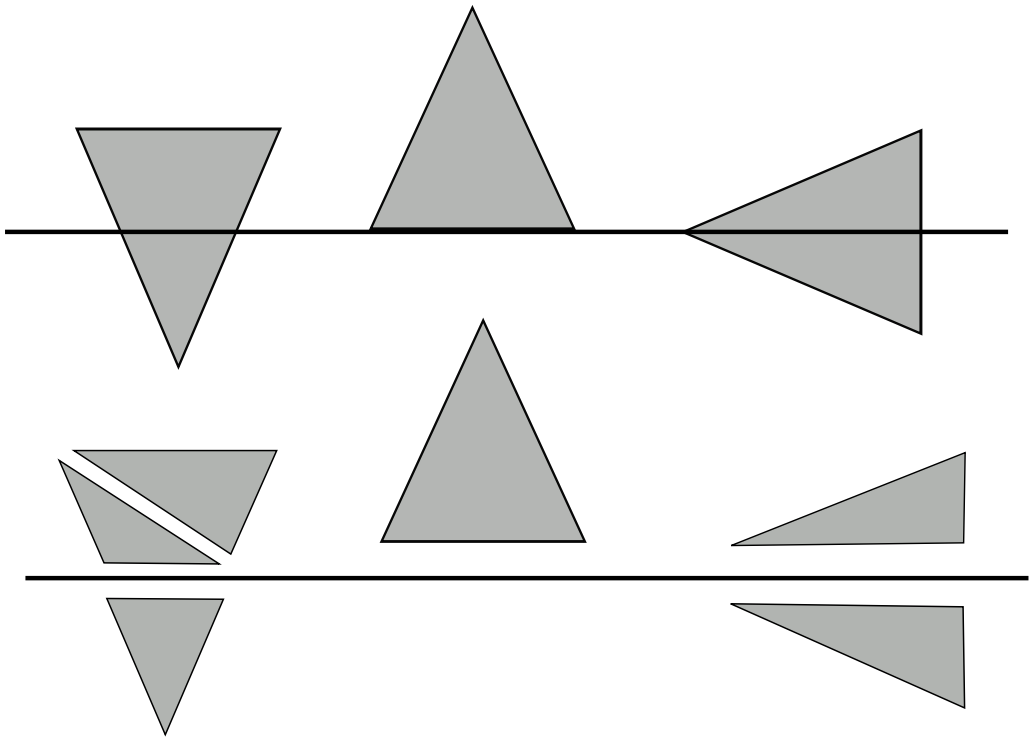
$$Normalize(x_1, x_2, x_3) = \left(\frac{x_1}{\sqrt{x_1^2 + x_2^2 + x_3^2}}, \frac{x_2}{\sqrt{x_1^2 + x_2^2 + x_3^2}}, \frac{x_3}{\sqrt{x_1^2 + x_2^2 + x_3^2}} \right) \tag{3.4}$$

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.

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 3.1. Tento proces má několik možných výsledků:

- Všechny tři body spadají pod nebo nad rovinu, trojúhelník je umístěn do odpovídající datové struktury (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. Je vytvořen nový trojúhelník, který tvoří první z bodů a další dva body průsečíku mezi trojúhelníkem a rovinou. Ten rozdělíme na dva trojúhelníky a umístíme je do správné skupiny.
- 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, trojúhelník je ignorován.

Z hraničních bodů je následně nutno vytvořit hrany a z nich navazující polygony. Hraniční body jsou přidávány vždy po dvou, každá takováto dvojice tedy tvoří hranu a za předpokladu, že model byl 2–manifoldní, tvoří tyto hrany polygon (s jednou výjimkou, když řez vede po hraně špičaté části modelu – pro představu řezná rovina ležící přesně na vrcholu střechy). Vezme se tedy první hrana a hledá se hrana následující, dokud nevznikne polygon. Takovýto polygon se uloží jako polyline, protože to vyžaduje funkce Poly2Tri. Takto se pokračuje, dokud jsou k dispozici hrany.



Obrázek 3.1: Rozdělení facetů dle roviny.

3.1.1 Problémy

V případě, že se dva body nacházejí na řezné rovině, je tato hrana uložena dvakrát a je nutno tyto duplicity z dat vymazat. Největším problémem jsou ale nepřesnosti v kalkulacích s desetinnými čísly. Kvůli nim mohou vzniknout pří výpočtu průsečíku dva body, které by měly být identické, ale přitom se liší. Při hledání navazujících hran je tedy nutno počítat s jistou tolerancí a tu zvyšovat, dokud nenalezneme navazující hranu nebo nedojdeme ke zvolené toleranci. V takovém případě je nutno posoudit, zda došlo k chybě a pokračovat s další hranou jako novým polygonem (tento postup umožňuje řez i modelem, který není 2–manifoldní, ale výsledek je velmi nejistý).

3.2 Triangulace polygonu

Triangulace polygonu je klasickým problémem výpočetní geometrie, který je námětem mnoha prací již po jedno století a největší pokrok zaznamenal 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 [8]:

- Recursive ear clipping algorithm [9]
- Sweep line algorithm [10]
- Incremental randomized algorithm [11]

3.2.1 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 z třech algoritmů nejméně náročný na implementaci, ale sweep line algoritmus je rychlejší [12]. Při hledání 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. CGAL je 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.

3.2.2 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 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 [8].

3.3 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 koordinát bodu, provést triangulaci a při převedení zpět do 3D by díky rovnoběžnosti s osou Z dostal každý bod stejnou hodnotu Z–koordinátu. Vedoucí práce Ing. Hrončok ale poznamenal, že takovéto řešení vnáší do programu další nepřesnost 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 koordinátu 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 koordináty musí být prohledána množina všech původních bodů ležících v řezné rovině a třetí bod dohledán. Tato množina musí být seřazena tak, aby vynechaná souřadnice byla nejméně důležitá při řazení, aby v ní bylo možno vyhledávat. V případě, že by došlo k chybě a bod se nepodařilo nalézt, bude hodnota dopočtena.

3.4 Polygony a díry

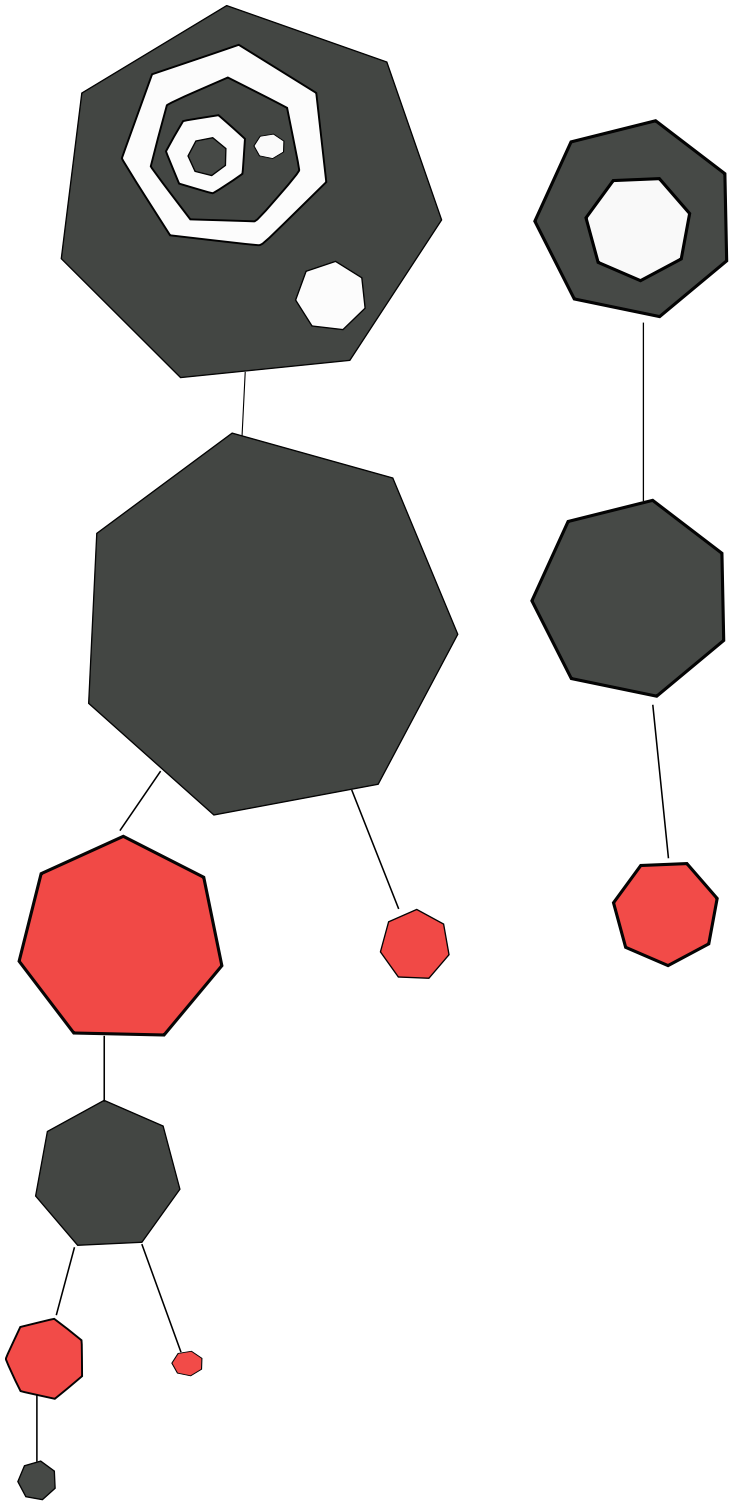
Po získání jednotlivých polygonů je nutno rozlišit, které z nich jsou skutečně polygony, a které mají být identifikovány jako díry. Pro přehlednost budou polygony, u nichž o tom ještě nebylo rozhodnuto, označeny jako *polygony*. K rozdělení byl zvolen tento postup:

1. Určit plochu *polygonů*.
2. Setřídit *polygony* sestupně podle plochy.
3. Odstranit *polygony* s nulovou plochou.
4. Odebrat *polygon*, nebo skončit (když už žádný nezbývá).
5. Určit, jestli je tento *polygon* uvnitř některého z polygonů (X).
6. Pokud ne, uložit ho a označit jako polygon a přejít na bod 4.

7. Pokud ano, označit *polygon* jako díru, projít všechny polygony a díry uvnitř polygonu X a zjistit, jestli je *polygon* Y uvnitř, pokud ano, změnit jeho označení (z díry na polygon a naopak) a v případě, že jde o díru, zapamatovat si, ve kterém polygonu se nachází.

8. Uložit *polygon* Y podle konečného označení a přejít na bod 4.

Výstup tohoto rozdělení je graficky znázorněn na obrázku 3.2.



Obrázek 3.2: Znázornění rozdělení polygonů na polygony (šedě) a díry (červeně).

Po tomto rozdělení je možné pomocí Poly2Tri provést triangulaci. O to se postará Poly2Tri funkce *triangulate* a následně funkce *getTriangles* vrátí pole trojúhelníků. Ty je nutné převést zpět do formy STL facetů a dopočítat vynechanou koordinátu. Následně se uloží oba vzniklé modely.

3.5 Testování

K automatickému otestování funkčnosti bylo zvoleno porovnání objemu původního modelu se součtem objemů vzniklých modelů. K výpočtu objemu může být využito funkce ADMeshe *stl_calculate_volume*.

3.6 Licence

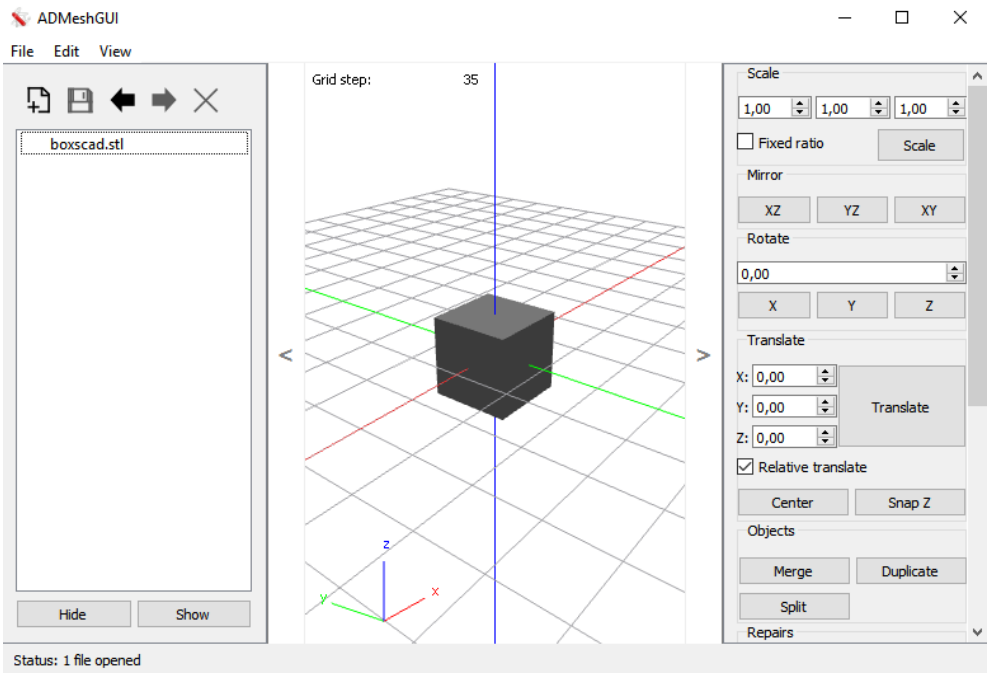
StlCut je dostupný pod licencí GNU GPL verze 2 [14] na adrese `http://github.com/Nathaniell1/STL_CUT`. Tato verze je shodná s licencí, pod kterou je k dispozici knihovna ADMesh, což by mohlo v budoucnu zajistit začlenění knihovny StlCut do knihovny ADMesh. ADMeshGUI je dostupný pod licencí GNU AGPL verze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

3.7 ADMeshGUI

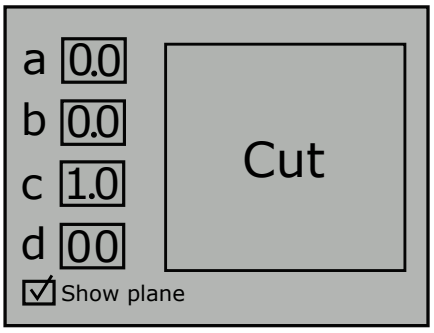
ADMeshGUI poskytuje grafické rozhraní pro jednoduchou práci s STL modely a umožnuje jejich úpravu. Rozhraní programu je na obrázku 3.3. Pro jeho plné využití při přípravě modelu k 3D tisku ale chybí možnost model rozříznout pomocí roviny na několik částí, což je funkce v 3D tisku často potřebná. Při 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 umožnuje načíst STL modely, prohlédnout si je v několika módech zobrazení (drátový model, plný, plný se zvýrazněnými hranami) a provést nad nimi operace ADMeshe. Grafické rozhraní je rozděleno do tří částí, v levé je seznam modelů, uprostřed je pohled na modely a vpravo je lišta s nabízenými operacemi.

3.7.1 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 3.4.



Obrázek 3.3: ADMeshGUI.



Obrázek 3.4: Návrh rozhraní pro řez.

Kapitola 4

Implementace

V této kapitole se blíže seznámíme se zdrojovým kódem knihovny. Popíšeme si hlavní třídu využitou v programu, blíže si rozebereme některé metody a problémy, které v nich nastaly.

4.1 StlCut

Knihovna a samotný program pro řez STL modelem byl pojmenován StlCut.

4.1.1 Vstup

Program přijímá na vstupu stl soubor a 4 čísla reprezentující rovinu (normálu (a, b, c) a vzdálenost od středu souřadnicového systému d). Pokud není rovina definována, použijí se výchozí hodnoty 0 0 1 1.

4.1.2 Výstup

Výstupem programu jsou, v případě že došlo k rozříznutí modelu, 2 STL soubory obsahující části modelu po rozdělení řeznou rovinou.

4.1.3 Třída Mesh

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

- **cut** – Tato metoda rozděluje facety dle řezné roviny. Nejdůležitějším úkolem je najít body, které leží na řezné rovině nebo při řezu vznikly. Tyto body budou po zpracování vstupními daty pro triangulaci polygonů.
- **createBorderPolylines** – Tato metoda zpracuje data z metody cut a převede body na jednotlivé polylines. Také se stará o převod z 3D do 2D souřadnic.
- **calculatePolygonArea** – Tato metoda slouží k výpočtu plochy polygonu.
- **findHoles** – Tato metoda porovnává polygony mezi sebou a rozlišuje, který z polygonů je určen k triangulaci a který jen ohraničuje díru v polygonu.
- **triangulateCut** – Tato metoda využívá funkcí Poly2Tri a v ní se provádí samotná triangulace.
- **setMissingCoordinate** – Tato metoda má za úkol převést 2D souřadnice triangulovaných polygonů zpět do 3D.

4.1.4 Create Border Polyline

V rámci této metody dochází k převedení hran ležících na řezné rovině v polygony. Tyto polygony je nutno najít na základě navazujících bodů jejich hran, ale kvůli nepřesnosti ve výpočtech s desetinnými čísly při výpočtu těchto bodů, je nutno hledat shodu s jistou tolerancí. Jako počáteční hodnota bylo zvoleno 1^{-24}. V případě, že funkce projde všechny hrany a nenajde navazující hranu, je jako tolerance zvolena nejnižší hodnota z rozdílu mezi hranami z předešlého běhu algoritmu, která se ještě o 0.5% zvětší, a hledá se znovu. V případě, že hrana není nalezena do tolerance 0.5, je uložena jako polygon tak jak je, program zahlásí varování, že mesh je zřejmě poškozená, resetuje toleranci a pokračuje dál. Poly2Tri využívá pro uložení polygonů polyline. Tyto body obsahují pouze koordináty x a y. Právě v této metodě dochází k zanedbání jedné z os a převedení bodu do 2D prostoru.

4.1.5 Calculate Polygon Area

K výpočtu plochy jednotlivých polygonů je použita Gaussova metoda pro výpočet plochy mnohoúhelníku [15].

```
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;
}
```

4.1.6 Find holes

V rámci této metody bylo nutno nějakým způsobem ukládat informace o tom, které polygony lze opravdu označit za polygony a které za díry, a zároveň rozlišit, které z těchto polygonů mají všechny své body uvnitř jiného polygonu. Každý z polygonů je uložen jako vektor bodů. Všechny polygony jsou tedy ve vektoru vektoru bodů. Při ukládání polygonu a jeho děr a vnitřních polygonů, by bylo zřejmě nejlepší využít stromu. Ale z důvodu, že ukládání těchto polygonů bylo už takto složité, zvolil jsem další vektor s tím, že kvůli absenci stromové struktury budu při hledání správné pozice polygonu uvnitř jiného polygonu muset projít všechny vnitřní polygony (při použití stromu by stačilo postupovat po vnitřních polygonech, které by přidaný polygon obsahovaly). Při zpětném pohledu ani tohle řešení rozhodně na přehlednosti nepřidalo. V rámci této metody jsou polygony ukládány do tohoto kontejneru:

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

Nejvnořenější vektor tvoří polygon. Pár tvoří vektor a číslo. Číslo určuje, buď že se jedná o polygon, nebo o index nejmenšího z vektorů, ve kterých se tento polygon (tedy v tomto případě díra) nachází. Další vektor obsahuje tyto polygony (každý polygon, který neleží uvnitř žádného dalšího polygonu je první prvek v těchto vektorech) a poslední (vnější) vektor obsahuje tyto vektory obsahující skupiny polygonů.

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. Tato metoda je pro jistotu (pokud by měly 2 polygony, ať už z důvodu nepřesných výpočtů nebo z důvodu chyby v modelu, překrývající se bod, nemusel by algoritmus vrátit správný výsledek) volána na 3 body polygonu a výsledek je vybrán pomocí většiny. Algoritmus je popsán v [16].

```
bool Mesh::vertexInPolygon (int nvert, const vector<Point* >& vertex,
                                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 = nvert-1; i < nvert; j = i++)
  {
    if ( ((vertex.at(i)->y > testy) != (vertex.at(j)->y > testy)) &&
     (testx < (vertex.at(j)->x - vertex.at(i)->x) * (testy-vertex.at(i)->y)
      / (vertex.at(j)->y - vertex.at(i)->y) + vertex.at(i)->x) )
    {
       in = !in;
    }
  }
  return in;
}
```

4.1.7 Triangulate Cut

V rámci této metody jsou z vektoru *polygonsWithHoles* postupně vybrány hlavní polygony (polygon, který není uvnitř žádného dalšího polygonu je označen jako hlavní polygon), které jsou vstupem pro konstruktor *CDT* knihovny Poly2tri. Následně je, v případě, že obsahují díru, volána metoda *AddHole*, která je přidá jako díru. Po projití všech polygonů uvnitř hlavního polygonu je volána metoda *Triangulate*, která provede triangulaci a metoda *createFaces*, která převede tyto trojúhelníky zpět do 3D a vytvoří z nich validní facety z pohledu knihovny ADMesh.

```
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 (map<int, p2t::CDT*>::iterator k = polygons.begin(); k != polygons.end(); ++k)
        {
            k->second->Triangulate();
            vector<p2t::Triangle*> triangles = k->second->GetTriangles();
            createFaces(triangles);
        }
      for (map<int, p2t::CDT*>::iterator k = polygons.begin(); k != polygons.end(); ++k)
      {
        delete (*k).second;
      }
      polygons.erase(polygons.begin(),polygons.end());
  }
}
```

4.2 Testy

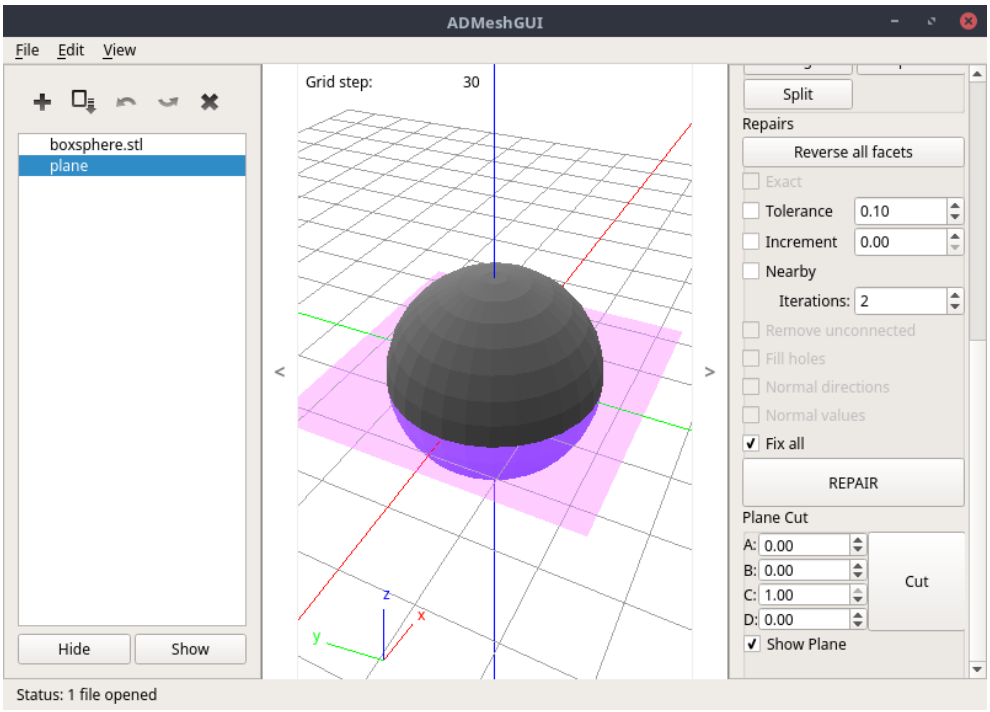
Testy byly původně napsány jako vlastní metoda třídy Mesh a byly tedy součástí knihovny. Byl také vytvořen testovací script, který opakovaně pouštěl program StlCut na zadané modely. Na návrh Ing. Hrončoka byl místo skriptu vytvořen testovací program využívající programovací jazyk Python a framework pytest. Z důvodu nedostatku zkušeností s prací v Pythonu mi vedoucí práce s s vytvořením testů významně pomohl. Tento test zavolá StlCut na každý STL soubor v nadřazené složce stl_files a za využití knihovny ADMesh, která je implementována i v Pythonu, zavolá její funkci na výpočet objemu původního modelu a modelů nově vzniklých. Výsledkem je přehledný výpis výsledků testů.

4.3 ADMeshGUI

Práce D. Vyvlečky [2] využívá k vytvoření uživatelského rozhraní frameworku Qt [17], s nímž jsem neměl naprosto žádnou 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 a pracuje se s ní jako s ostatními načtenými STL modely. Žádná z původních operací ADMeshGUI nevyžadovala zobrazení dodatečných grafických prvků ve 3D. Z toho důvodu bylo implementačně nejjednodušší přidat rovinu podobným způsobem, jakým se přidávají ostatní modely, a rovina je tedy zobrazena v levém menu jako model s názvem *plane* a lze jí vybrat (nakliknout) a skrýt jako ostatní modely. Veškeré operace z pravého menu však byly upraveny tak, aby na řeznou rovinu nefungovaly. Pokud je zvolen jiný než drátový mód zobrazení, je řezná rovina vykreslena jako částečně průhledný model fialové barvy, viz obrázek 4.1.



Obrázek 4.1: ADMeshGUI – úprava rozhraní.

Kapitola 5

Zhodnocení

Program StlCut splnil cíl této práce a zajišťuje řez STL modely, ale existuje několik nevyřešených problémů, které program obsahuje. Při řezu modelů, jejichž objem se pohybuje v řádu jednotek a menších a jež obsahují množství detailů, může docházet k výskytu chybně propojených polygonů z důvodu nepřesných výpočtů. Tyto problémy lze velmi často vyřešit posunem řezné roviny nebo zvětšením modelu. Při řezu nemanifoldním modelem dochází k tvorbě chybně propojených trojúhelníků, což ale vychází z chyb v modelu samotném a jeho nekompatibilitě s použitými algoritmy. Pokud uživatel povede řez modelem tak, že na řezné rovině bude ležet celý polygon, jež ale nebude triangulovaný (půjde tedy o spojené hrany, pro něž platí, že třetí bod obou trojúhelníků, jichž jsou hrany součástí, leží oba pod nebo nad řeznou rovinou), dojde k triangulaci tohoto polygonu. Takový výsledek je ovšem logicky nevyžádanou úpravou a vede k vytvoření **nemanifoldního** modelu. Tento problém nebyl zvážen při návrhu programu. K řešení by bylo nutno speciálně ukládat a zpětně prohledávat trojúhelníky obsahující hrany na řezné rovině a výrazně pozměnit a zpomalit některé metody.

Závěr

Cílem této práce bylo vytvořit knihovnu a program umožňující vést rovinný řez trojúhelníkovou sítí, výsledný řez triangulovat a vytvořit dva nové modely. Myslím, že tento cíl byl úspěšně splněn. Hlavním přínosem této práce z hlediska běžného uživatele je rozšíření programu ADMeshGUI o možnost rovinného řezu. Program lze nyní pohodlně využít k většině úkonů spojených s přípravou 3D modelu k tisku. Díky této práci jsem se naučil lépe dohledávat a pracovat s odbornými zdroji a také získal zkušenosti v oblasti pokračování vývoje cizího softwaru. Jako další možné rozšíření knihovny ADMesh a program ADMeshGUI by mohla být knihovna pro zadělávání děr v modelu na základě tvaru v okolí díry, tato funkcionalita by mohla být prospěšná k úpravě nedokonalých modelů získaných pomocí 3D skeneru.

Literatura

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

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

[3] CGS Tree. [online], [cit. 2017-5-11]. Dostupné z: `https://upload.wikimedia.org/wikipedia/commons/8/8b/Csg_tree.png`

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

[5] The STL format. *Historical Resource on 3D Printing [online]*, [cit. 2017-4-22]. Dostupné z: `http://www.fabers.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] CONTRIBUTORS, A.: ADMesh – STL mesh manipulation tool: ADMesh 0.98.1 documentation [online]. 2015, [cit. 2017-4-25]. Dostupné z: `http://admesh.readthedocs.io/`

[8] 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/`

[9] 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`

[10] 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`

[11] 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`

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

[13] The Computational Geometry Algorithm library. [cit. 2017-4-26]. Dostupné z: `http://www.cgal.org/`

[14] GNU General Public License [online]. 1991, [cit. 2017-05-13]. 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 project [online]. [cit. 2017-04-15]. Dostupné z: `www.qt.io`

Příloha **A**

Seznam použitých zkratek

GUI Graphical user interface

CGAL The Computational Geometry Algorithm Library

STL Stereo Lithography

Příloha **B**

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