**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra aplikované informatiky**

# Implicitní povrchy v počítačové grafice

Bakalářská práce

Autor: Dominik Spilka
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Bruno Ježek, Ph.D.

Hradec Králové duben 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29.4.2022 Dominik Spilka

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Brunu Ježkovi Ph.D. za metodické vedení práce a čas, který mi věnoval.

# Anotace

Bakalářská práce pojednává o možném alternativním způsobu použití implicitních povrchů pro tvorbu trojrozměrných modelů a jejich následné polygonizaci. V rámci této práce jsou navrženy kroky potřebné k možnému modelování pomocí implicitních povrchů, které jsou následně implementovány. Výsledné řešení je poté porovnáno s trojrozměrnými modely vytvořenými standardním postupem pro znázornění odchylek, které díky polygonizaci vznikají. Na základě těchto odchylek jsou navrženy kroky k jejich částečnému nebo úplnému odstranění.

# Annotation

## Title: Implicit Surfaces in Computer Graphics

Bachelor thesis talks about possible alternative usage of implicit surfaces in the creation of three-dimensional models and their polygonization. Steps required for possible modelling using implicit surfaces are outlined, which are then implemented. Resulting solution is then compared to models created using standard means to point out differences created during polygonization. Based on shown differences, solutions are presented to reduce or remove said differences.

# Obsah

# Seznam obrázků

# 1 Úvod

Vývoj hardwarových prostředků, a především grafických subsystémů, které jsou nyní dostupné i pro širokou veřejnost, umožnil dlouhodobě růst složitosti zobrazovaných trojrozměrných scén. Od primitivních modelů, kde lze jednotlivé polygony tvořící 3D scénu rozpoznat pouhým okem, přešla kvalita až do stavu, kde lze polygonálně modelovat i jednotlivé vrásky na obličeji. Tím se přiblížila kvalita zobrazení realitě.

Narůstající složitost s sebou však přináší i řadu negativních vlivů. Díky zvýšenému detailu jsou polygonální sítě vytvářených modelů stále rozsáhlejší, náročnější na výpočetní výkon a práce s nimi se stává pro uživatele obtížnější. V této práci je navrženo alternativní řešení pro tvorbu modelů za pomoci distančních polí definovaných pomocí implicitních funkcí. Ty jsou využity pro matematický popis modelu, ze kterého lze následně extrahovat informace potřebné k jeho polygonizaci.

V práci jsou představeny technologie potřebné k implementaci popisovaného řešení. Teoretické podklady jsou dále využity k implementaci nástroje. Na základě implementace práce je provedeno porovnání modelů vytvořených pomocí polygonizace implicitních ploch s modely vytvořenými standardním postupem. Jsou ukázány chyby způsobené polygonizací a v závěru navrženo řešení vhodné k odstranění těchto chyb.

Téma bylo zvoleno za účelem navržení alternativního přístupu ke tvorbě trojrozměrných modelů, který si za cíl klade urychlení nebo zjednodušení práce grafika při tvorbě konceptuálních nebo i výsledných děl. V případě vhodně navrženého řešení by tento postup umožňoval i méně zkušeným začít efektivně pracovat s modelovacími nástroji, což by mělo za následek zvýšení produktivity v odvětvích počítačové grafiky, jako je například herní nebo animátorský průmysl.

# 2 Modelování implicitních ploch

Tvorba trojrozměrných modelů pomocí implicitních ploch se skládá ze dvou kroků. Model je nejprve definován matematicky. K tomu jsou použita primitiva popsána funkcí, která jsou spolu provázaná logickými operátory. Takový způsob modelování implicitních povrchů byl již navržen autory Reiner et. al. v práci Interactive modelling of implicit surfaces using a direct visualization approach with signed distance function [1]. Modelovací nástroj navržený v této práci se zakládá na podkladech autorů Reiner et. al., zejména z hlediska logiky použití.

Druhým krokem je pak převod vzniklého modelu na polygony. Výsledný model lze již použít v běžném procesu zpracování a rendrovacím řetězci, který je založen na práci s polygonálním 3D modelem. Model musí umožňovat následné texturování a zasazení do existující scény tvořené dalšími polygonálními modely. Z toho důvodu je tedy nutné model po jeho vytvoření exportovat do formátu vhodného pro použití v jiných současně dostupných programech.

## *2.1 Definice implicitního povrchu*

Implicitním povrchem je v této práci myšlena matematická reprezentace trojrozměrného primitiva v 3D prostoru. Jedná se o definici funkce $f[x, y, z]$: $\mathbb{R}^3 \rightarrow \mathbb{R}$, která má parametry x, y a z reprezentující souřadnice bodu 3D prostoru. Funkce vrací vzdálenost bodu od povrchu definovaného tělesa [1].

Pomocí takto definované funkce lze popsat libovolné těleso v prostoru. Pro správné fungování reprezentace je však důležitá i správná signatura funkce. Implicitní funkce kromě vzdálenosti bodu od povrchu tělesa určuje pro vybraný bod podle znaménka také to, zda se nachází uvnitř tělesa, či mimo objem definovaného primitiva. Body, pro které $f[x, y, z] = 0$, pak leží na povrchu tělesa, které funkce popisuje [1].

Při dodržení výše popsaných předpokladů lze pak definovat rozměrnou škálu těles. Například koulová plocha s poloměrem $r$ a středem v počátku soustavy souřadnic je definována funkcí

$$f(x, y, z): x^2 + y^2 + z^2 - r^2 = 0$$

Pro zjednodušení práce s rovnicí a její vhodnější reprezentaci v programovacím jazyce lze tuto funkci ale předefinovat pomocí jednoho vstupního parametru $a$, kde $a = [x, y, z]$ je bodem prostoru $\mathbb{R}^3$ [2]. Za použití tohoto postupu by pak rovnice popisující shodnou kouli v počátku o poloměru $r$ vypadala takto:

$$f(a) = \|a\| - r$$

Stejným způsobem pak lze definovat i jiná trojrozměrná primitiva. Kromě reprezentace uzavřených těles se nabízí i možnost reprezentace nekonečných prostorů, jako je například poloprostor určený rovinou nebo nekonečný válec určený poloměrem a přímkou určující osu válce.

Zvolený přístup má však svá úskalí při popisu primitiv, jejich součástí jsou ostré hrany. Jedním z příkladů tohoto problému je například popis krychle. Taková tělesa lze sice popsat jednou rovnicí splňující požadované podmínky, pro definici složitých těles se ale nabízí vhodnější alternativa.

Jako další způsob definice těles lze kromě jedné složité rovnice použít více jednoduchých rovnic provázaných množinovými operacemi [3]. Například poloprostor lze definovat za pomocí dvou hodnot; normalizovaného vektoru $\vec{n}$, sloužící jako normálový vektor hraniční roviny poloprostoru a jednorozměrné hodnoty $h$, která udává vzdálenost hraniční roviny od počátku soustavy souřadnic [4]. Pro tyto veličiny lze definovat funkci $f(x)$: $\mathbb{R}^3 \rightarrow \mathbb{R}$ ve tvaru

$$f(x) = \vec{x} \cdot \vec{n} - h$$

Kde $\vec{x} \cdot \vec{n}$ je skalární součin bodu a normalizovaného normálového vektoru. Výsledkem funkce je pak vzdálenost bodu $\vec{x} = [x_x, x_y, x_z]$ od roviny kolmé na normálový vektor $\vec{n}$.

Pro popsání krychle o délce strany 1 lze pak použít šest takovýchto poloprostorů popsaných následujícími funkcemi:

$$f_a(x) = x \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} - 0.5$$

$$f_b(x) = x \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} - 0.5$$

$$f_c(x) = x \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - 0.5$$

$$f_d(x) = x \cdot \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} - 0.5$$

$$f_e(x) = x \cdot \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} - 0.5$$

$$f_f(x) = x \cdot \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} - 0.5$$

V jednotlivých hraničních rovinách poloprostorů leží stěny definované krychle. Na přímkách vzniklých protnutím rovin pak leží její hrany. Pro popsání samotné krychle lze poté zjistit hodnotu všech výše uvedených funkcí a vybrat největší vzdálenost bodu od jedné z rovin tvořících hranu krychle. Matematický zápis skládání funkcí pak vypadá následovně:

$$f(x) = \max\left(f_a(x), f_b(x), f_c(x), f_d(x), f_e(x), f_f(x)\right)$$

Krychle popsána výše uvedeným způsobem znázorňuje možnosti skládání funkcí. Díky tomu lze matematicky popisovat i složitá tělesa, pro která by bylo nereálné definovat jednu souvislou funkci. Výsledná hodnota však není euklidovskou vzdáleností od krychle samotné.

Postup skládání funkcí odpovídá množinové operaci průniku, kdy množina všech bodů ležících uvnitř daného poloprostoru je definována jako $M = \{x, f(x) < 0\}$. Alternativní zápis pomocí množinových operací pro množiny $A, B, C, D, E, F$ odpovídající prostorům popsaným funkcemi $f_a, f_b, f_c, f_d f_e, f_f$ vypadá následovně:

$$M = A \cap B \cap C \cap D \cap E \cap F$$

Podobným způsobem lze přidružit i další množinové operace vhodným matematickým funkcím. Pro funkce $f_a, f_b$ popisující dvě množiny bodů $A, B$ ležících uvnitř dvou různých těles lze například definovat jejich sjednocení jako

$$A \cup B = \{x: \min(f_a(x), f_b(x)) < 0\}$$

nebo množinový rozdíl jako

$$A - B = \{x: \min(-f_a(x), f_b(x)) < 0\}$$

Použitím těchto postupů je poté možné vytvořit složitá tělesa za pomoci jednoduchých primitiv popsaných matematickými funkcemi a jejich následným provázáním množinovými operacemi. Takto vytvořené popisy modelů pak vytváří distanční pole.

Distanční pole je definováno jako prostorové pole skalárních hodnot vzdáleností od povrchu geometrie [5]. Každý prvek tohoto pole reprezentuje minimální vzdálenost od definovaného povrchu v libovolném směru. Prakticky to znamená, že z dané pozice se můžeme o danou hodnotu posunout libovolným směrem, aniž by došlo k protnutí povrchu objektu. Pokud je pak povrch orientovaný, lze v distančním poli využít kladných a záporných hodnot k rozlišení prostorů ležících uvnitř a vně povrchu [6]. Při použití matematických funkcí je distanční pole souvislé (hodnotu lze určit v libovolných souřadnicích), alternativně však může být distanční pole diskrétní.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| +5 | +2 | +1 | +2 | +5 | +5 | +2 | +1 | +2 | +5 |
| +4 | +1 | -1 | +1 | +2 | +2 | +1 | -1 | +1 | +4 |
| +5 | +2 | +1 | -1 | +1 | +1 | -1 | +1 | +2 | +5 |
| +2 | +1 | -1 | -1 | -1 | -1 | -1 | -1 | +1 | +2 |
| +1 | -1 | -1 | +1 | -1 | -1 | +1 | -1 | -1 | +1 |
| -1 | -1 | -2 | -1 | -2 | -2 | -1 | -2 | -1 | -1 |
| -1 | +1 | -1 | -1 | -1 | -1 | -1 | -1 | +1 | -1 |
| -1 | +1 | -1 | +1 | +1 | +1 | +1 | -1 | +1 | -1 |
| +1 | +2 | +1 | -1 | -1 | -1 | -1 | +1 | +2 | +1 |
| +4 | +5 | +2 | +1 | +1 | +1 | +1 | +2 | +5 | +4 |

**Obrázek 1. Dvojrozměrné diskrétní distanční pole.**
**Zdroj: https://prideout.net/blog/distance_fields/[6]**

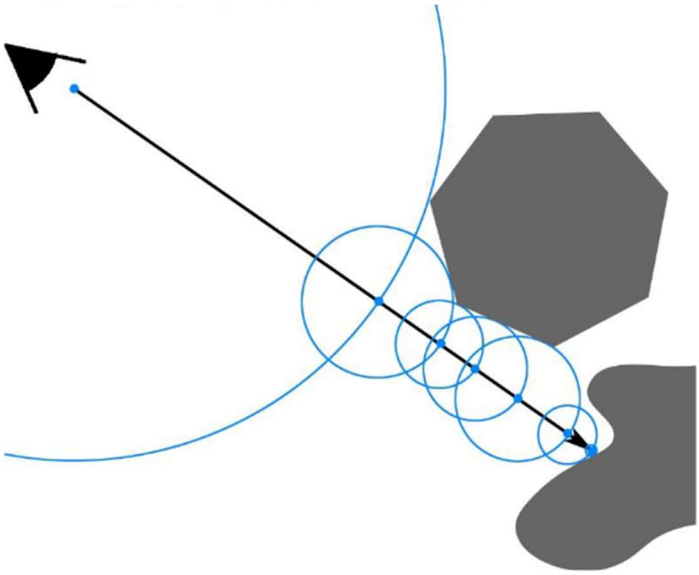
## 2.2 Vykreslování distančních polí

Matematicky definovaná pole lze zobrazit několika způsoby. Již v tomto kroku se nabízí možnost rozklad povrchu na polygony, například trojúhelníky, a následné vykreslení získané sítě polygonů (mesh). Pokud se ale jedná čistě o možnost náhledu na scénu, existují i další postupy. Jedním z takových postupů je algoritmus sphere tracing, také známý jako raymarching [7]. Jedná se o per pixel algoritmus založený na určení vzdálenosti bodu od povrchu modelovaného objektu, díky které detekuje kolizi paprsku, vycházejícího z oka pozorovatele daným pixelem průmětny, s povrchem objektu.

Paprsek definujeme jako polopřímku vycházející z bodu $r_0$ a směřující ve směru vektoru $r_d$. Nejprve je zjištěna vzdálenost bodu $r_0$ od zadaného povrchu. Jedná se o maximální možnou vzdálenost kroku, o který se lze v prostoru posunout, aniž by došlo ke střetu s povrchem.

Po nalezení maximální možné vzdálenosti je určen nový bod $r_1$, který je výsledkem posunutí se po polopřímce paprsku o zjištěný krok. Pro bod $r_1$ je pak znovu zjištěna vzdálenost od povrchu, jež je použita jako nová velikost následujícího kroku podél paprsku.

Stejný postup je opakován do chvíle, než je pro libovolný bod $r_n$ ležící na paprsku zjištěna vzdálenost od povrchu rovna nule. V takovém případě pak bod $r_n$ leží na povrchu tělesa a je tudíž vykreslen do průmětny. Alternativně pak může nastat situace, kdy vzdálenost bodu $r_n$ překročí arbitrárně určenou hranici. V takovém případě se předpokládá, že paprsek minul povrch tělesa.



**Obrázek 2. Reprezentace Raymarching Algoritmu**

**Zdroj: Signed Distance Fields in Real Time Rendering[8]**

Zápis algoritmu v pseudokódu vypadá následovně:

```
Pro každý pixel:
      vzorkovanýBod = počátekPaprsku;
      krok = SDF(vzorkovanýBod);
      while(krok > 0)
            krok = SDF(vzorkovanýBod);
            vzorkovanýBod += krok * směrPaprsku;
```

Vzorkovaný bod v tomto zápise reprezentuje právě zpracovávaný bod $r_n$ daného kroku algoritmu. Ten je na začátku inicializován na hodnotu $r_0$. V každém kroku cyklu je pak určen součtem dvou vektorů. Prvním z nich je předchozí vzorkovaný bod ($r_{n-1}$), druhým je pak normalizovaný vektor směr ($r_d$) vynásobený zjištěnou vzdálenosti. Výpočet vzdálenosti od povrchu je reprezentován voláním funkce SDF.

Díky zjištění vzdálenosti vzorkovaného bodu od scény nedochází k opomenutí žádného z těles a zároveň lze efektivně zpracovat rozsáhlé prázdné prostory. Tento postup má však i svá úskalí, zejména v místech, kde se paprsek pohybuje velmi blízko povrchu paralelně s ním. V takových místech se paprsek pohybuje velmi pomalu a je tedy vysoká náročnost na výpočet. Kromě maximální vzdálenosti od scény je proto vhodné určit i maximální počet kroků, které má algoritmus provést[9].

Výpočet tohoto algoritmu lze provádět ve fragmentovém shaderu vykreslovacího řetězce (GPU pipeline). Pro zobrazení scény jsou rendrovací pipelině předány pouze dva trojúhelníkové polygony pokrývající celou plochu průmětny (obrazovky). Barva pixelů vzniklých rasterizací těchto polygonů je určena fragment shaderem, s implementovaným Raymarching algoritmem, což zajišťuje správné vykreslení celé scény modelované pomocí distančního pole [9]. Tento postup je běžně používaný i ve scéně takzvaných 4kb demo, kde je cílem reprezentovat komplexní trojrozměrnou scénu za pomoci nanejvýš 4 kb úložného prostoru. Příkladem je webový nástroj Shadertoy, který slouží jako platforma pro tvorbu a sdílení fragment shaderů obsahujících kód nutný k vyobrazení jedné konkrétní statické, nebo dynamické scény [10].

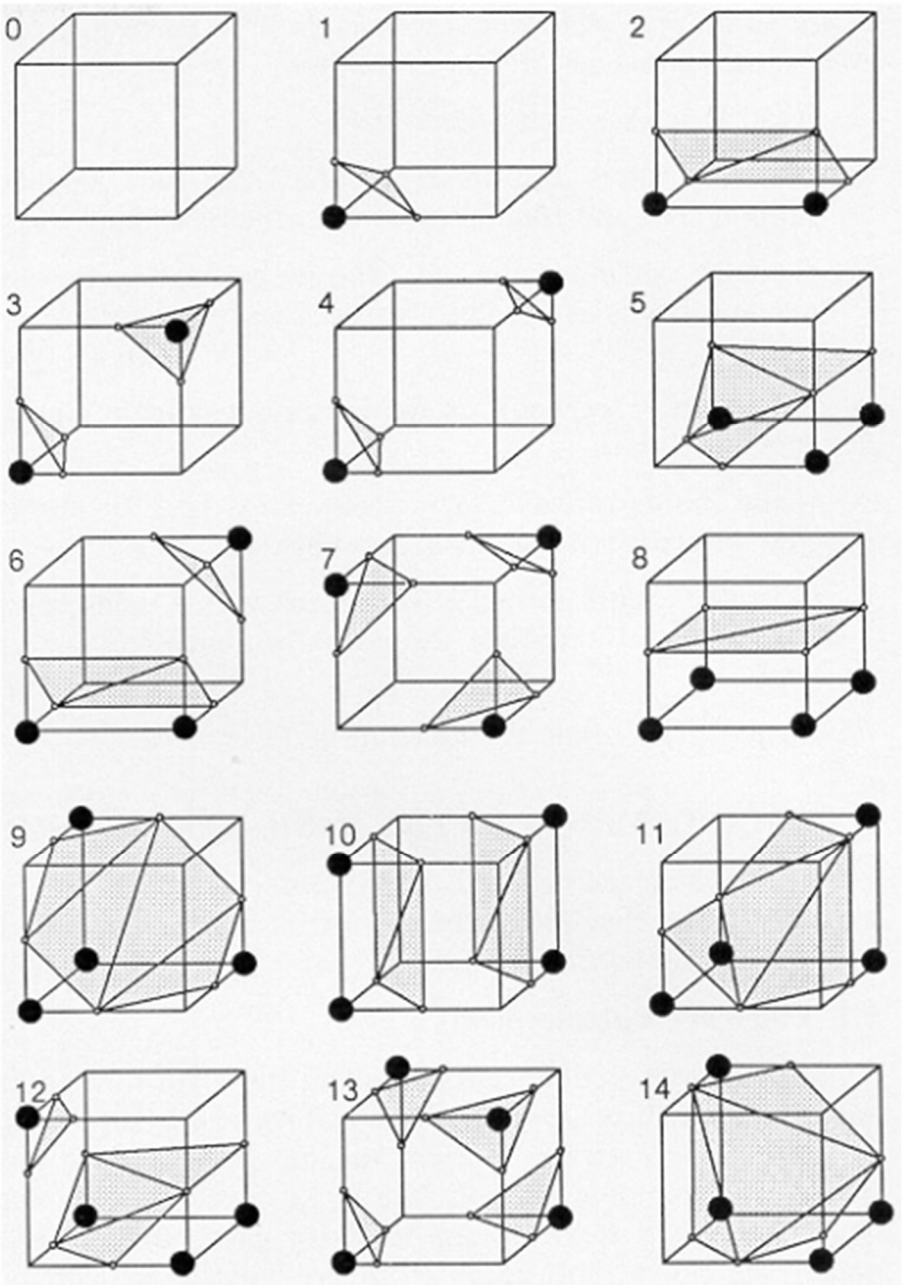
## 2.3 Polygonizace

Autoři Lorensen a Cline ve svém článku Marching Cubes: A high resolution 3D surface construction algorithm navrhují způsob zobrazení volumetrických dat uživateli [11]. Jedná se o data, která nesou informaci o měřené veličině v závislosti na pozici v prostoru. V případě autorů Lorensen a Cline jde o medicínská data získaná například pomocí počítačové tomografie nebo magnetické resonance.

Cílem autorů je umožnit zobrazení isoploch těchto volumetrických dat. Isoplochy reprezentují body ve 3D prostoru o konstantní hodnotě, podobně jako například vrstevnice ve 2D prostoru. Za tímto účelem byl autory navržen algoritmus známý jako Marching Cubes.

Průběh Marching Cubes algoritmu je rozdělen do několika kroků. Prvním z nich je rozdělení prostoru do trojrozměrné mřížky složené ze shodných krychlí. Jednotlivé krychle sdílí každou svou stranu s šesti sousedními krychlemi mřížky. Tím vznikne definice bodů, konkrétně vrcholů jednotlivých krychlí, pro které budou vzorkována volumetrická data.

Druhým krokem je vzorkování prostoru. V tomto kroku dochází k měření hodnoty v každém z určených bodů. Tyto hodnoty jsou pak využity k účelům polygonizace isoplochy, která má být zobrazena.




**Obrázek 3. Vzory polygonizace Marching Cubes algoritmu**
**Zdroj: Marching Cubes: A high resolution 3D surface construction algorithm [11]**

Ve třetím kroku je pro každou krychli mřížky rozhodnuto, zda jí isoplocha prochází. Přítomnost isoplochy lze detekovat díky hodnotám vzorků ležících v rozích zpracovávané krychle. Pokud některé vzorky mají vyšší hodnotu, než je hodnota isoplochy a zbylé vzorky mají hodnotu nižší, prochází isoplocha krychlí. Pokud mají

všechny vzorky hodnotu vyšší či nižší, než je hodnota isoplochy, pak isoplocha krychlí neprochází.

Poslední krok Marching Cubes algoritmu je vytvoření finálního polygonálního meshe. Pro vzorky, které byly ve třetím kroku zvoleny jako klíčové, je zjišťováno, kterými hranami isoplocha prochází. Takové hrany lze poznat podle toho, že vzorky v jejich vrcholech splňují podmínku, kde hodnota jednoho je vyšší než hodnota isoplochy, zatímco hodnota druhého je nižší. Taková hrana protíná isoplochu. Ve hranách protínajících isoplochu jsou vytvořeny vertexy, které jsou nakonec propojené do polygonů. Těmi je tvořen výsledný mesh.

Autoři ve své práci poukazují na fakt, kdy každá krychle zpracovávaná v rámci Marching Cubes algoritmu spadá do jednoho z 256 vzorů [11]. Tyto vzory se odlišují tím, které vzorky v nich jsou považovány za vnitřní (mají hodnotu vyšší, než hodnotu isoplochy) a které jsou vnější (jejich hodnota je nižší, než hodnota isoplochy). Při použití vhodné symetrie lze těchto 256 vzorků omezit na 14 variant výsledných polygonů, znázorněných na obrázku 3.

Marching Cubes algoritmus lze bez nutnosti úprav použít pro řešení polygonizace distančních polí. Distanční pole je ve své podstatě pouze sbírka volumetrických dat, která určují vzdálenost jednotlivých bodů od definovaného tělesa. Postup polygonizace je tedy stejný, jako polygonizace isoplochy o hodnotě 0 z těchto volumetrických dat.

Výsledný mesh vytvořený pomocí Marching Cubes algoritmu lze pak dále vylepšit pomocí interpolace vytvářených vertexů. Primitivní implementace algoritmu totiž umisťuje vytvářené vertexy do středů jednotlivých hran krychle. Jelikož jsou ale hodnoty vzorků dané hrany známé, lze tyto informace využít. Pozici vytvářeného vertexu lze po délce hrany interpolovat tak, aby jeho výsledná pozice vždy odpovídala povrchu. Tím lze odbourat převážnou část nepřesností, které při polygonizaci vznikají a vytvořit tak hladší povrch přesněji reprezentující očekávaný model.

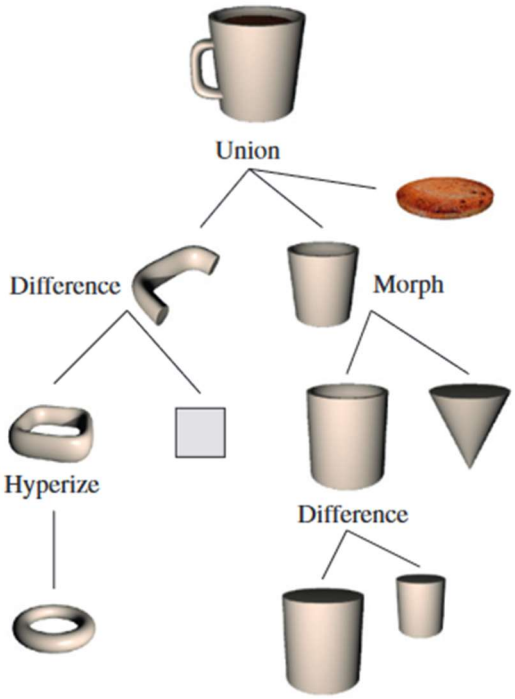
# 3 Cíl práce

Cílem práce je navrhnout alternativní postup pro tvorbu trojrozměrných modelů a porovnat možnosti jeho použití s již existujícími postupy. Účelem navrhovaného nástroje je nahradit nutnost tvorby samotného polygonálního modelu (mesh) za tvorbu komplexního matematického popisu složeného z primitiv, který je následně do podoby meshe exportován. Kladeny jsou následující otázky; Jaká je uživatelská náročnost takového nástroje? K jakým typům nepřesností dochází v případě použití navrhovaného způsobu? Jak lze případné nepřesnosti potlačit, nebo případně zcela odstranit?

V práci byly představeny teoretické podklady vhodné k vytvoření zmiňovaného nástroje. Dále budou tyto podklady využity k implementaci nástroje sloužícího pro modelování implicitních povrchů. Výsledný nástroj pak dále musí umožňovat i extrakci meshe z modelu implicitních povrchů. Výsledky exportu budou vizuálně zhodnoceny. Cílem zhodnocení je podotknout konkrétní problémy, které při exportu vznikají. Na základě těchto pozorování, spolu se zjištěními vycházejícími z implementace nástroje, budou navržena možná řešení pro další rozvoj modelování implicitních povrchů a jejich polygonizace.

# 4 Popis řešení

Modelovací nástroj použitý k demonstraci řešení této práce byl vyvinut v programovacím jazyce Java. Základním zobrazovacím nástrojem je knihovna LWJGL[12] určená pro práci s dvojrozměrnou a trojrozměrnou polygonální grafikou Knihovna byla využita zejména pro možnosti implementace shaderů. Kód samotných shaderů je pak psaný v jazyce GLSL.




**Obrázek 4. Znázornění stromové struktury vedoucí k výslednému modelu. Zdroj: Interactive modelling of implicit surfaces using a direct visualization approach with signed distance functions [3]**

## *4.1 Fungování aplikace*

Po spuštění aplikace je uživatel uvítán prázdnou scénou. Uživatel má možnost do scény přidávat operátory a tělesa. Scéna je reprezentována stromovou strukturou, kde operátory fungují jako kořen a větve stromu. Primitiva poté mohou existovat pouze jako listy.

Uživatel má možnost ze scény vybrat libovolné těleso nebo operátor a aplikovat nad ním transformace posunu, rotace a škálování. Úpravy provedené na operátoru se projeví na všech jeho potomcích.

Rozpracovanou scénu lze v libovolném kroku uložit a znovu načíst. Scény jsou uloženy jako stromová struktura ve formátu XML. V libovolném momentě je dále možné scénu exportovat do polygonálního meshe. Ten je exportován ve formátu .obj obsahující definici vrcholů a polygonů. Na jednotlivá tělesa nelze aplikovat materiály ani způsoby vyhlazování.

## *4.2 Reprezentace scény*

### 4.2.1 Primitiva a operátory

Pro reprezentaci scény byla zvolena stromová struktura, kde každý prvek stromu může obsahovat až dva potomky. Základním stavebním prvkem prvku stromu je abstraktní třída Element definující společné vlastnosti. Těmi jsou název a funkce prvku, jeho transformace v prostoru, rodič a seznam potomků. Jednotlivé prvky stromu poté z této třídy dědí a definují vlastní rozšiřující vlastnosti.

Kromě definice těchto vlastností nabízí pak třída Element funkcionalitu pro zobrazení tělesa pomocí programovacího jazyka GLSL. Třída dokáže sestavit transformační matici tělesa na základě parametrů, kterými jsou posun, rotace a škálování ve směrech x, y a z. Dále pak dokáže sestavit inicializační hodnoty pro uniformní hodnoty fragment shaderu a výslednou funkci pro vykreslení sebe sama.

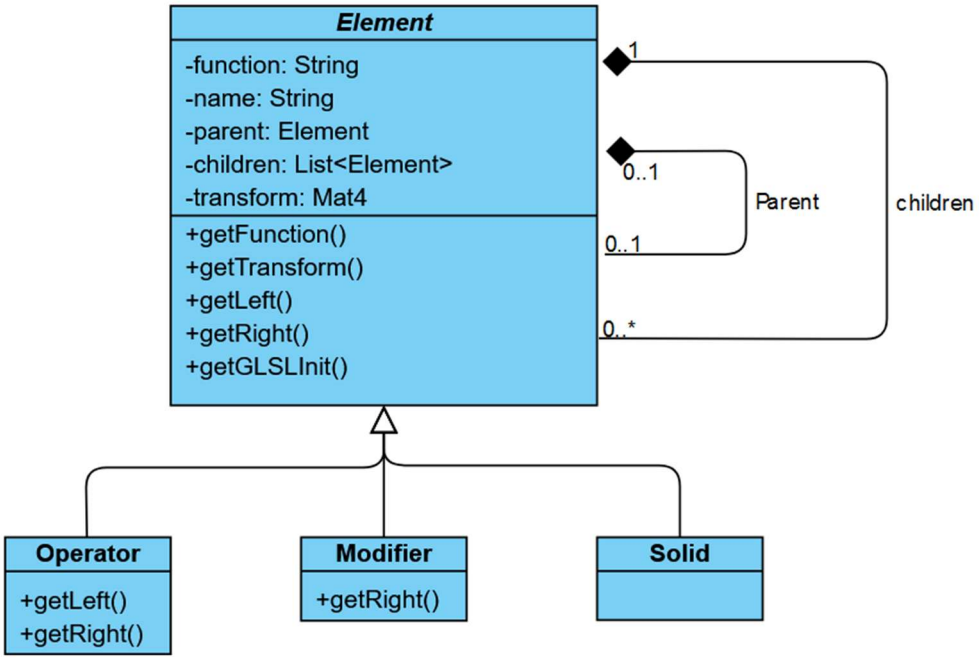
Uniformní hodnoty shaderu jsou takové hodnoty, které se v rámci vykreslení jednoho snímku, resp. v rámci jednoho volání metody draw, nemění. Jedná se o informace, které nejsou závislé na právě rozrastrovaných pixelech, jako například parametry zobrazení, čas od spuštění, nebo pozice kamery. V případě třídy Element se jedná o transformační matici.

Nejjednodušším potomkem třídy Element je třída Solid. Jedná se o kořenový prvek obsahující definici samotného primitiva zadaného pomocí implicitního povrchu. Tento prvek nenabízí žádnou dodatečnou funkcionalitu oproti rodičovské třídě

Element. Omezuje však dostupnou funkcionalitu, zejména z důvodu, že nesmí mít potomky.

Třída Modifier existuje pro upřesnění definice modifikátoru jiného prvku. Jedná se o pokročilé transformace jako zkroucení nebo ohnutí. Tato třída může v rámci stromu obsahovat pouze jednoho potomka.

Operator představuje jednu z implementovaných množinových operací. Tato třída se ve stromové struktuře smí nacházet kdekoliv, kromě listových prvků. Třída má vždy právě dva potomky, které díky svému GLSL předpisu zpracovává jako zvolená množinová operace.




**Obrázek 5. Reprezentace struktury pomocí UML Class Diagramu.**
**Zdroj: Vlastní práce**

### 4.2.2 Instancování

Pro tvoření nových instancí tříd byl v této práci zvolen návrhový vzor Factory. Tento návrhový vzor umožňuje jednoduchou tvorbu nových objektů stejné třídy s odlišnými hodnotami. Zvolen byl, jelikož definice jednotlivých prvků stromu je odlišná pouze v názvu a GLSL předpisu. Dvě různá primitiva jsou pak instancemi jedné třídy Solid, která je inicializována s různými názvy a předpisy.

Implementace tohoto návrhového vzoru se skládá ze čtyř abstraktních továren. Hlavní z nich je obecná pro libovolný element. Zbylé tři odpovídají každé ze tříd dědících z abstraktního Elementu. Konkrétní prvky jsou poté instancovány vždy z konkrétní implementace jedné z továren. V konkrétních implementacích je objektu přiřazen předpis funkce v jazyce GLSL a parametry jako počátek a výchozí rozměry.

Využití tohoto návrhového vzoru nabízí versatilní možnosti implementace dodatečných těles, modifikátorů a operátorů. Pro nové těleso je nutné pouze implementovat novou konkrétní továrnu, která jej vytvoří s konkrétním GLSL předpisem. Ukládání a načítání scény

Jako způsob persistence scény napříč spuštěními programu byl zvolen zápis scény do souboru ve formátu XML. Výsledný soubor obsahuje jeden kořenový prvek Scene, který obsahuje jako jediného potomka kořenový prvek stromu. Jednotlivé prvky jsou poté reprezentovány tagem Element, který v sobě obsahuje název prvku, jeho předpisovou funkci a transformace. Poslední hodnotou ukládanou do prvku Element je pole jeho potomků v tagu Children, kde jsou pro prvky typu Modifier a Operator uloženi jeho potomci.

Ukládání a načítání XML souboru probíhá rekursivně. Při načtení scény z XML souboru dojde ke znehodnocení stávajících informací o scéně a následné instancování jednotlivých prvků dle struktury XML. Výsledný kořenový prvek je poté nastaven jako kořen stromu aktivní scény.

## 4.3 Vykreslení scény

Vykreslení scény probíhá pomocí jednoho fragment shaderu psaného v programovacím jazyce GLSL. Do kódu shaderu jsou jako uniformní hodnoty dynamicky vkládány transformační matice těles. Dále je pak stejným způsobem zadána pozice, pohledový a up vektor kamery. Nakonec je pak do kódu shaderu vložena funkce pro vykreslení scény.

### 4.3.1 Raymarching aloritmus

Pro vykreslení scény samotné je ve fragment shaderu využita základní implementace raymarching algoritmu dle kapitoly 3.2. Po nalezení bodu scény je dále proveden ještě jeden výpočet viditelnosti z nalezeného bodu směrem ke zdroji světla umístěného ve scéně.

Zjištěná hodnota slouží k implementaci základního osvětlení scény. Pokud je nalezen průsečík s povrchem blíže, než je zdroj světla, vykreslovaný pixel leží ve stínu. Nakonec je pro každý bod zjištěn normálový vektor. Na základě skalárního součinu normálového vektoru s vektorem směrem ke zdroji světla je pak určeno množství odlesku. Vkládání uniformních hodnot a vykreslovací funkce

Pro správné fungování zobrazovací pipeliny v programu je nutné, aby byl zdrojový kód fragment shaderu dynamicky upravován. K takové úpravě dochází v případě, že se změní složení scény. Jedná se o operace, které do scény vkládají nebo z ní odstraňují elementy.

Ve chvíli, kdy dojde k přidání tělesa do scény, proběhne nejprve znehodnocení původního kódu shaderu. Kód je poté sestaven znovu s novou distanční funkcí. Ta je vkládána do GLSL kódu na konkrétní místo určené komentářem. Nakonec jsou pak do stejného kódu vloženy definice uniformních hodnot. Nový kód je zkompilován do programu fragment shaderu a zapojen do pipeline namísto původního.

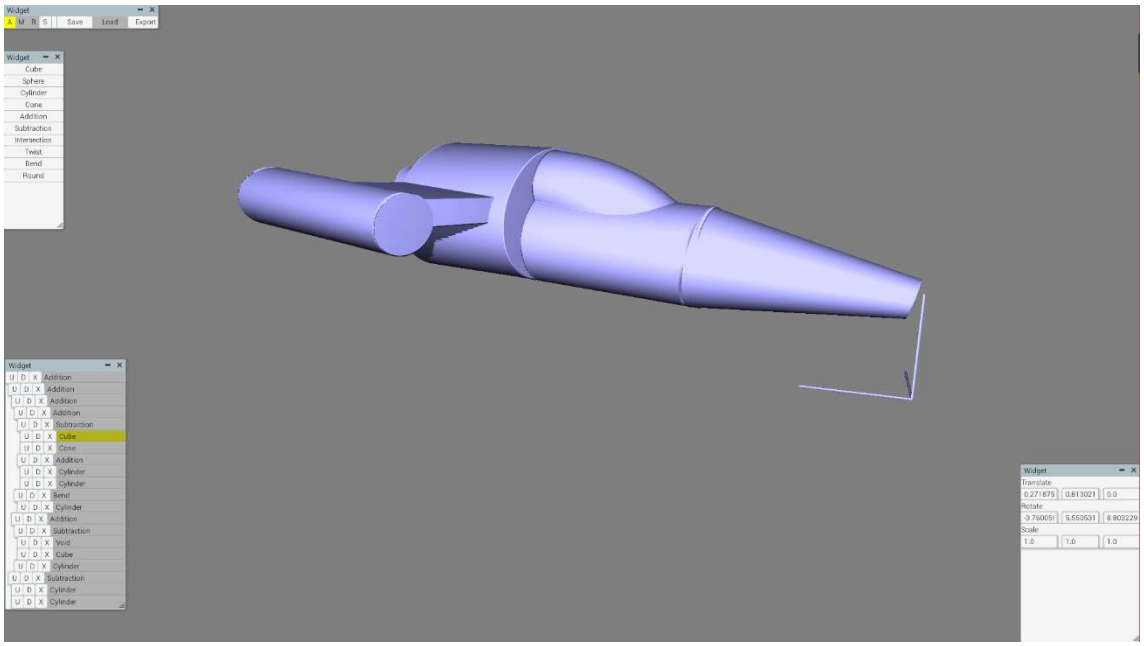
Uniformní hodnoty jsou použity pro předání transformačních matic jednotlivých těles do programu shaderu. Jedná se o nejčastěji editované informace o elementech. V případě, že by transformace těles byly zakódované do funkce popisující prostor,

bylo by nutné program fragment shaderu znehodnotit při libovolné manipulaci s tělesy scény. Při použití uniformních hodnot je však možné jejich hodnotu dynamicky měnit mezi jednotlivými snímky.

Stejným způsobem je v shaderu definovaná i pozice kamery. Ta se skládá ze tří údajů; pozice, pohledový vektor a up vektor. Jednotlivé informace jsou zapsány do připravených uniformních proměnných. Shader na základě těchto informací vypočte right vektor kamery a tyto informace použije k vykreslení scény.

## 4.4 Uživatelské rozhraní

Pro zajištění ovládání aplikace byla zvolena knihovna LEGUI[13]. Jedná se o knihovnu založenou na práci s widgety zobrazenými pomocí knihovny LWJGL přímo ve vykreslované scéně. Za pomocí knihovny lze definovat vlastní komponenty uživatelského rozhraní a za běhu programu dynamicky zobrazovat, upravovat a skrývat jednotlivé widgety.




**Obrázek 6. Hlavní okno aplikace**

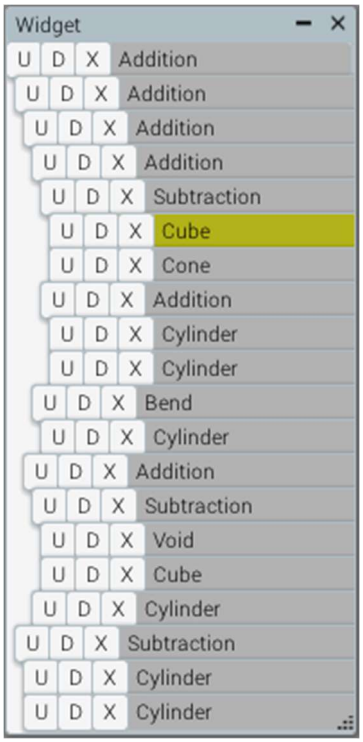
**Zdroj: Vlastní práce**

Uživatelské rozhraní aplikace je rozděleno do čtyř nezávislých widgetů. Centrálním prvkem je hlavní widget sloužící k přepínání módů ovládání aplikace. Tyto módy jsou přidávání těles, a jejich translace, rotace a škálování. Dále pak nabízí uložení a načtení scény a její export do .obj souboru.


**Obrázek 7. Detail widgetu módů. Zdroj: Vlastní práce**

**Význam zástupných znaků: A – Add Element, M – Move, R – Rotate, S – Scale**

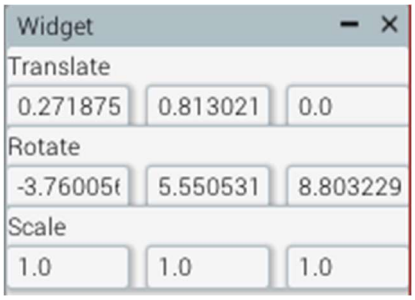
Druhým prvkem je widget reprezentující strukturu scény. V tomto widgetu je graficky znázorněna stromová struktura těles a operátorů. Za pomocí ovládacích prvků lze jednotlivá tělesa ve struktuře přesouvat, nebo je odstraňovat. Libovolný element lze v tomto widgetu označit za aktivní.




**Obrázek 8. Detail widgetu položek scény.**

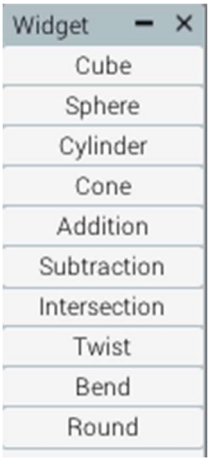
**Tlačítka U a D slouží ke změně pozice elementu ve stromu. Tlačítkem X jej lze ze scény odstranit. Žlutě označený element je právě aktivní. Zdroj: Vlastní práce**

Po zvolení aktivního elementu lze jeho pozici v prostoru ovládat buď za pomocí volby módu a následného tažení, nebo přímým zadáním hodnot do widgetu transformací. V tomto widgetu je zobrazeno celkem devět vstupních polí reprezentujících translaci, rotaci a škálování ve směru os x, y a z.




**Obrázek 9. Obrázek widgetu transformací. Zdroj: Vlastní práce**

V případě, že je zvolen mód přidávání těles, zobrazí se nakonec widget s nabídkou dostupných elementů. Ty lze přidat do scény jako potomka právě zvoleného elementu, nebo záměnou za něj. Toto chování je kontextuální na základě právě zvoleného elementu. Pokud je například aktivním elementem operátor, zvolením nového operátoru ve widgetu přidání dojde k jeho nahrazení za aktivní element.




**Obrázek 10. Detail widgetu pro přidání elementu. Zdroj: Vlastní práce**

### 4.4.1 Způsob použití

Uživateli je nabízena možnost zvolit mód translace, rotace, nebo škálování jednotlivých těles, nebo jejich skupin. Po volbě módu pak uživatel zvolí element scény, který má být transformován.

Pro provedení transformace je nejprve nutné zvolit vodící osu. Toho uživatel dosáhne držením patřičné klávesy X, Y, nebo Z. Po zvolení osy je možné toto těleso transformovat relativně k této ose tažením kurzoru myši vodorovně po obrazovce. Tažení vlevo představuje transformaci v negativním směru osy, tažení vpravo pak transformaci v pozitivním směru.

Ovládání kamery lze odemknout držením klávesy shift. Při držení této klávesy lze otáčet pohled do scény držením levého tlačítka myši. Pravým tlačítkem lze poté kameru v prostoru přemístit. Toto přemístění probíhá relativně k současnému směru pohledu kamery.

## *4.5 Polygonizace*

Cílem je z distančního 3D pole definovaného implicitní funkcí vytvořit polygonový model. K tomu je použit algoritmus Marching Cubes, který vytvoří seznam povrchových polygonů. Pro export je zvolen vhodný vektorový formát.

### 4.5.1 Popis formátu .obj

Pro uložení výsledného polygonového modelu byl zvolen formát Wavefront .obj [14]. Jedná se o jednoduchý otevřený formát, který umožňuje definici více těles v jednom souboru.

Reprezentace tělesa v souboru se skládá ze seznamů hodnot. Každá hodnota v seznamu má specifický prefix odpovídající danému seznamu. Dále jsou pak uvedeny jednotlivé informace. Hodnoty v seznamu jsou odděleny novým řádkem.

```
# Nový objekt
o Cube

# Souřadnice v prostoru
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
...
# Souřadnice textury
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
...
# Normálové vektory
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
...

# Polygony
f 1/1/1 5/2/1 7/3/1 3/4/1
f 4/5/2 3/4/2 7/6/2 8/7/2
f 8/8/3 7/9/3 5/10/3 6/11/3
...
```

**Obrázek 11. Příklad struktury souboru Wavefront .obj**

**Zdroj: Vlastní práce**

Nejprve jsou v souboru uvedeny všechny vertexy modelu. Seznam vertexů je definován prefixem v. Pro každý z vertexů je nutné mít definovanou pozici v podobě souřadnic x, y, z. Ke každému vertexu lze uvést ještě w hodnotu pro uložení homogenních souřadnic. Pokud však není uvedena, je její výchozí hodnota 1.0.

Dále je možné definovat seznam souřadnic textury. Jednotlivé hodnoty mají prefix vt a nesou informace o souřadnicích u, v, w do textury. Tento seznam není povinný, není tedy v současné implementaci vytvářen.

Třetím seznamem je seznam normál jednotlivých vertexů. Hodnoty nesou předponu vn a obsahují souřadnice x, y, z normálových vektorů. Tyto vektory nemusí být normalizované. Seznam normálových vektorů není nutné definovat. Pokud není definován, normály vertexů jsou vypočteny z jednotlivých polygonů. Takto zapsané polygony musí pak jednotlivé vertexy seřadit v kladném směru po jejich obvodu.

Nakonec je zapisován seznam polygonů. Hodnoty seznamu polygonů mají prefix f a obsahují indexy do seznamů vertexů, textury a normál. Každá hodnota může obsahovat libovolný počet indexů. Současná implementace exportuje model do trojúhelníkových polygonů, proto každá hodnota obsahuje tři indexy.

Zápis indicií je v případě polygonů specifický tím, že pro každý bod polygonu lze určit jeho souřadnice v prostoru, souřadnice v textuře a normálový vektor, zapsané v tomto pořadí. Jednotlivé hodnoty jsou pro každý vrchol odděleny lomítkem (/). Pro definici polygonu jsou však povinné pouze souřadnice v prostoru. Nedefinované informace lze jednoduše vynechat.

Výsledný export tedy obsahuje seznam vertexů následovaný seznamem polygonů. Ostatní informace nejsou v současné implementaci exportovány.

### 4.5.2 Zjištění hodnot z distanční funkce

Zjištění hodnot z distanční funkce je řešeno pomocí compute shaderu. Ten byl zvolen, jelikož reprezentace scény pro zobrazení a pro kalkulaci je v GLSL kódu shodná a nedochází díky tomu k nepřesnostem. Využití jazyka GLSL pak dále umožňuje znovu využít již existující část implementace elementů.

Pro danou instanci compute shaderu je při každém běhu připraven buffer floating pointových hodnot. Úkolem compute shaderu je pro každý bod prostoru kalkulovat jeho vzdálenost od povrchu a zapsat na příslušnou pozici v bufferu. Pro řešení byly alternativně navrhovány booleovské hodnoty, kdy pravdivá hodnota odpovídá bodu uvnitř tělesa, zatímco nepravdivá odpovídá bodu vně. Booleovské hodnoty mají nižší datový objem než floating pointové hodnoty, ve výsledném řešení byly však zvoleny floating pointové hodnoty. Použitím booleanovských hodnot by totiž došlo ke ztrátě možnosti interpolace výsledných vertexů.

Po naplnění bufferu jsou hodnoty dále zpracovány Marching Cubes algoritmem. Konkrétní postup algoritmu je popsán v kapitole 3.3. V tomto řešení je použit algoritmus s interpolací vertexů po hraně pravidelně vzorkované krychle, což zvyšuje přesnost výsledného modelu v relaci k distanční funkci. Vypočtené vertexy jsou ukládány do mapy klíčů a hodnot, kde jsou jako klíč použity jejich souřadnice.

Díky tomu pak nedochází k problémům, kdy dva vertexy okupují stejný bod prostoru a povrch díky tomu není plynulý. Jako hodnota je pak pro každý vertex uložena jeho reprezentace v .obj.

K vypočteným vertexům jsou poté do oddělené mapy zapsány i jednotlivé nalezené polygony. Jejich hodnoty se skládají ze tří klíčů do mapy vertexů. Klíče do mapy vertexů jsou ukládány proto, aby na jejich základě bylo možné určit správnou hodnotu indexu ve výsledném .obj souboru.

Po zpracování všech vzorků jsou data zapsána do souboru. Výsledný mesh je ukládán jako jedno těleso, nezávisle na tom, zda je jeho povrch souvislý.

# 5 Zhodnocení výsledků

Za pomocí navrhovaného a implementovaného nástroje lze provádět trojrozměrné modelování a následné exportování výsledku do polygonálního meshe. Dále budou zhodnoceny výsledky implementace nástroje. Ty budou porovnány s možnostmi, které nabízí volně dostupný modelovací nástroj Blender.

Zhodnocení výsledků je rozděleno na dvě hlavní skupiny. Nejprve jsou zhodnocena jednoduchá tělesa. U jednoduchých těles jsou porovnány zejména vlastnosti jako počet polygonů, nebo jejich odlišnost od očekávaného vzhledu. Důraz je pak kladen na chyby nebo nedostatky způsobené polygonizací.

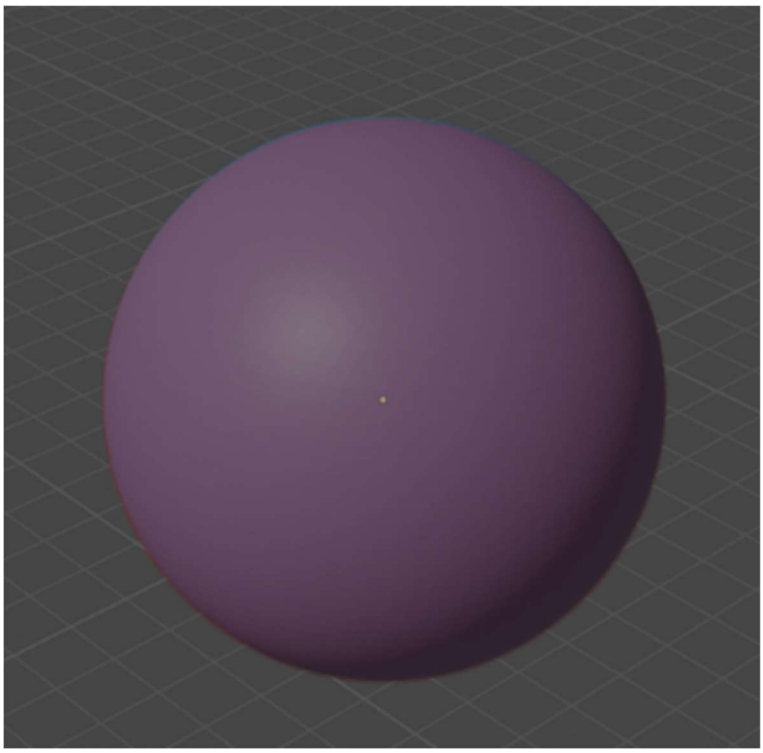
U složitých modelů je klíčová jejich použitelnost ve scéně. U těchto porovnání již není kladen důraz na přesnost jejich exportu, jelikož tyto problémy jsou shodné s problémy vznikajícími u primitiv. I přes tyto problémy však mohou být výsledné modely použity například ve stylizovaných scénách, proto budou výsledky porovnány zejména opticky. Dále je pak zkoumána i jednoduchost práce s nástrojem.

## 5.1 Jednoduchá tělesa

### 5.1.1 Přesnost

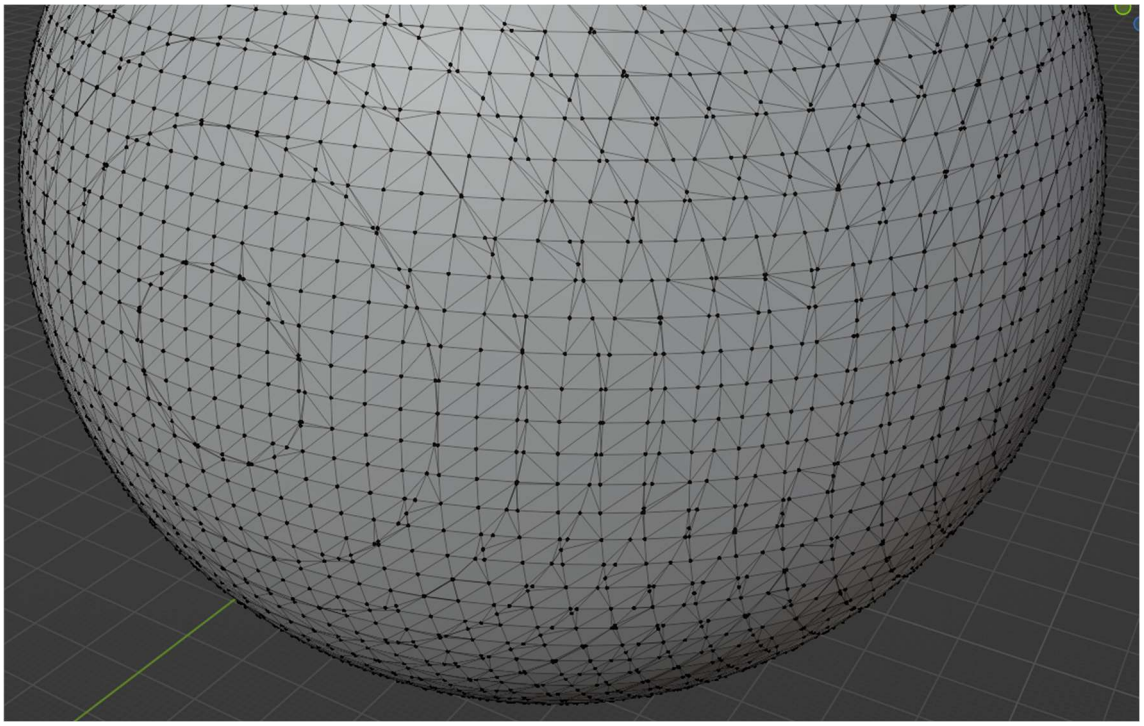
V navrhovaném řešení byly nabízeny dva způsoby polygonizace těles. Oba způsoby využívají jako základ Marching Cubes algoritmus, první z nich jej však využívá v jeho základní podobě, zatímco druhý způsob použití zahrnuje navíc interpolaci výsledných vertexů v závislosti na jejich vzdálenosti od očekávaného povrchu. Vzhledem k zaměření této podkapitoly na porovnání přesnosti výsledných meshů s očekávanými primitivy bude pro porovnání použit pouze druhý způsob.

Marching Cubes algoritmus nenaráží na žádný problém při exportu zaoblených těles. Koule či elipsoidy prakticky přesně odpovídají očekávaným primitivům.



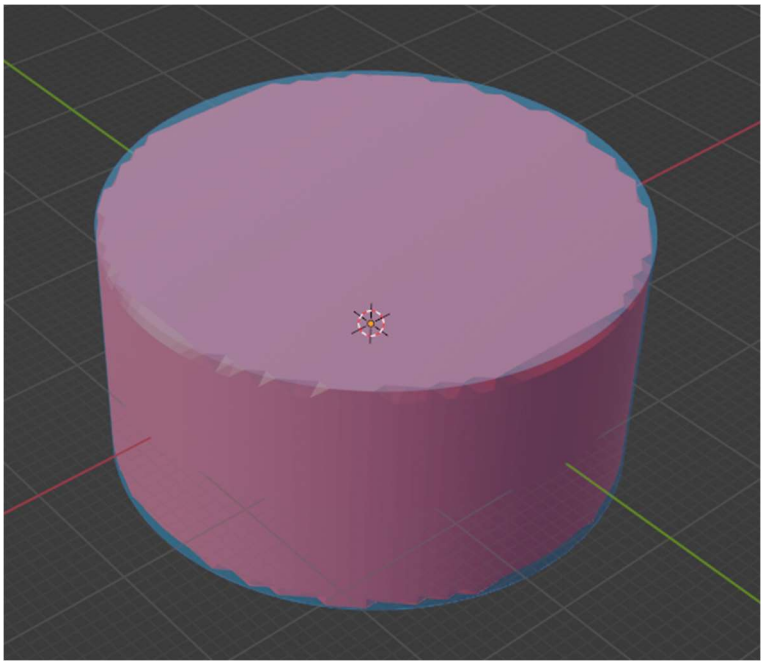
**Obrázek 12 Porovnání exportované kulové plochy s UV koulí vytvořenou v programu Blender. Zdroj: Vlastní práce**

Problém však nastává v meshi samotném. Jelikož je objekt exportovaný za pomocí uniformní vzorkovací mříže, po interpolaci jednotlivých vertexů nejsou jeho polygony uniformní velikosti. Tento nedostatek může tvořit problémy při texturování povrchu, přestože moderní nástroje si s takovýmto meshem poradí.



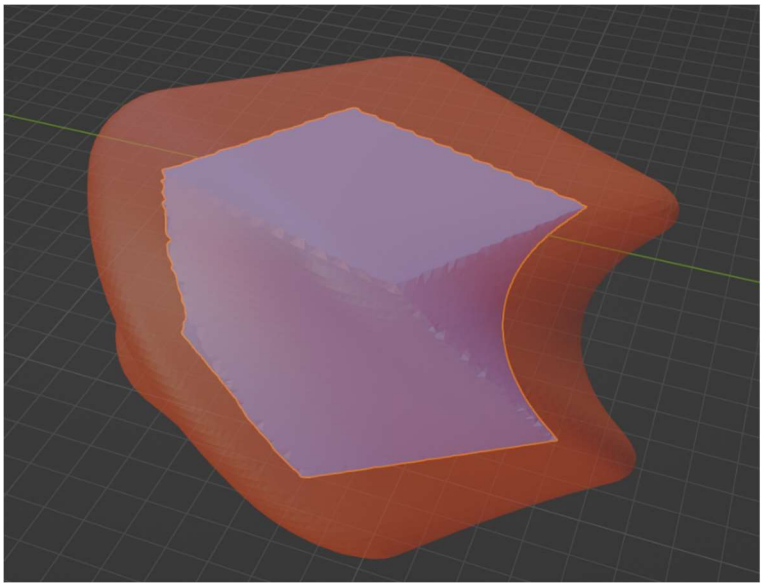
**Obrázek 13. Detail struktury meshe exportované kulové plochy. Viditelné jsou nerovnoměrné polygony vznikající použitím Marching Cubes algoritmu. Zdroj: Vlastní práce**

Velmi podstatným problémem Marching Cubes algoritmu jsou však ostré hrany. Vzhledem k omezenému množství možných vzorků tohoto algoritmu nelze očekávat úplnou přesnost při exportu objektů, které takové hrany obsahují. Problém není příliš znatelný u hran, které odpovídají směrům vzorkovací mříže, v místech, kde se orientace hrany liší, je však problém velmi viditelný.



**Obrázek 14. Porovnání exportovaného válce s válcem vytvořeným v programu Blender. Nepřesnosti vznikající exportem lze pozorovat na ostrých hranách po obvodu podstav válce. Zdroj: Vlastní práce**

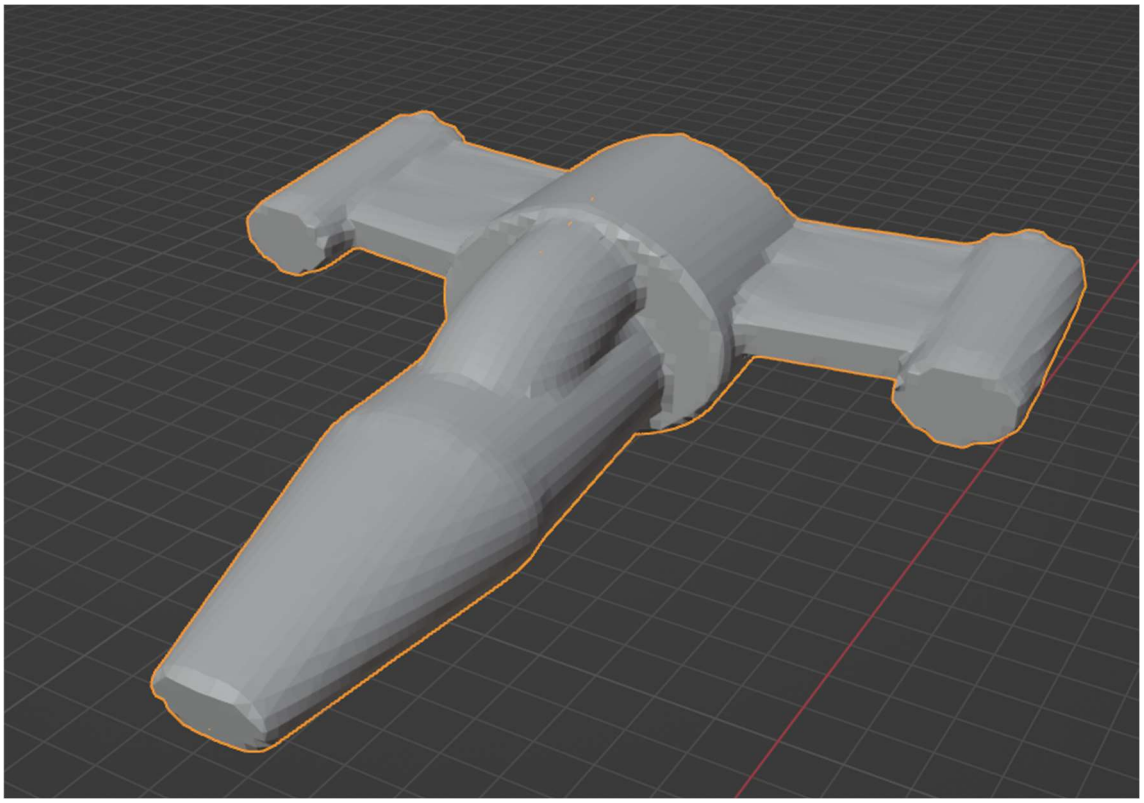
Problém chyb na ostrých hranách lze obejít pomocí zaoblení takových hran ještě před exportem. Pokud je velikost zaoblení větší než vzdálenost mezi jednotlivými vzorky, povrch se v takových místech chová jako hladký. Díky tomu se při exportu chová stejně, jako v případě koule nebo elipsoidu. Výsledný objekt však neodpovídá objemu původního tělesa, jelikož distanční funkce je upravena o velikost zaoblení plošně.



**Obrázek 15. Porovnání ostré zakřivené krychle s objemem po přidání zaoblení. Oranžový prostor znázorňuje zvětšení objemu tělesa.**
**Zdroj: Vlastní práce**

## *5.2 Složité scény*

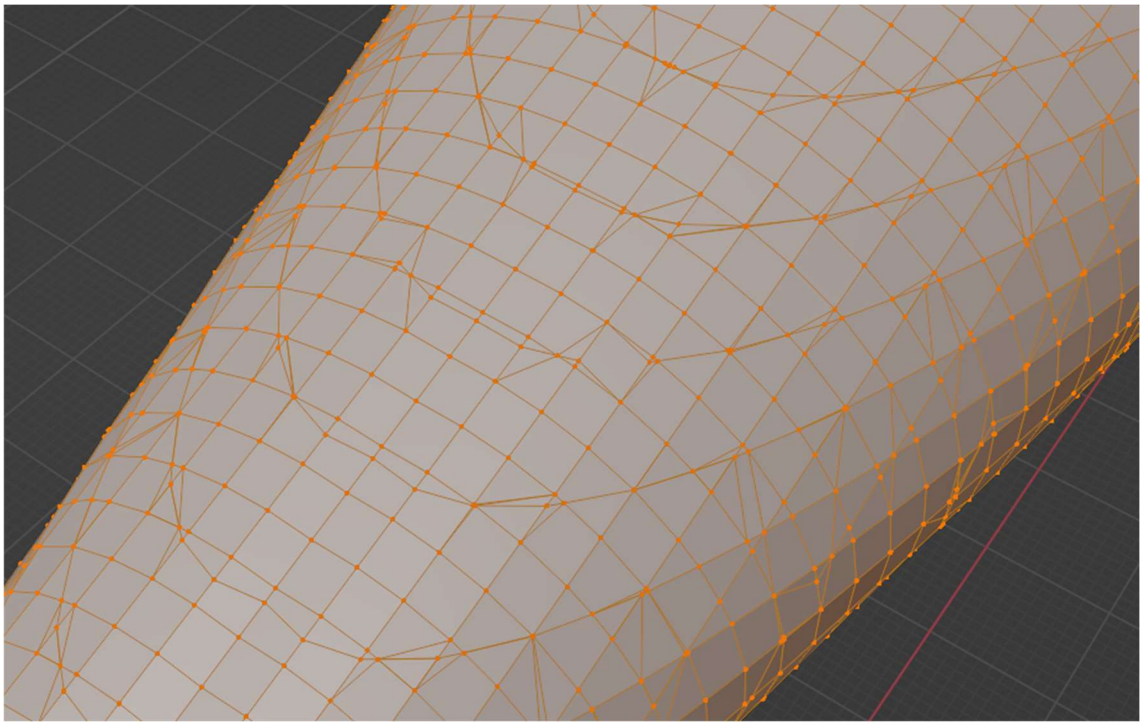
Stejné problémy, které lze sledovat u primitivních těles, lze pak samozřejmě pozorovat i na objektech složitějších. Pokud je však zvolena správná velikost vzorku, spolu s interpolací vrcholů dochází ke vzniku relativně reprezentativních modelů. Takto vytvořené modely zabírají kratší dobu pro prototypování a tvorbu a poskytují polygonální síť s uniformní velikostí. Tím je umožněna dodatečná práce s modelem ve vertexově orientovaném nástroji.



**Obrázek 16. Jednoduchý model vesmírné stíhačky vytvořený pomocí implicitních ploch. Zdroj: Vlastní práce**

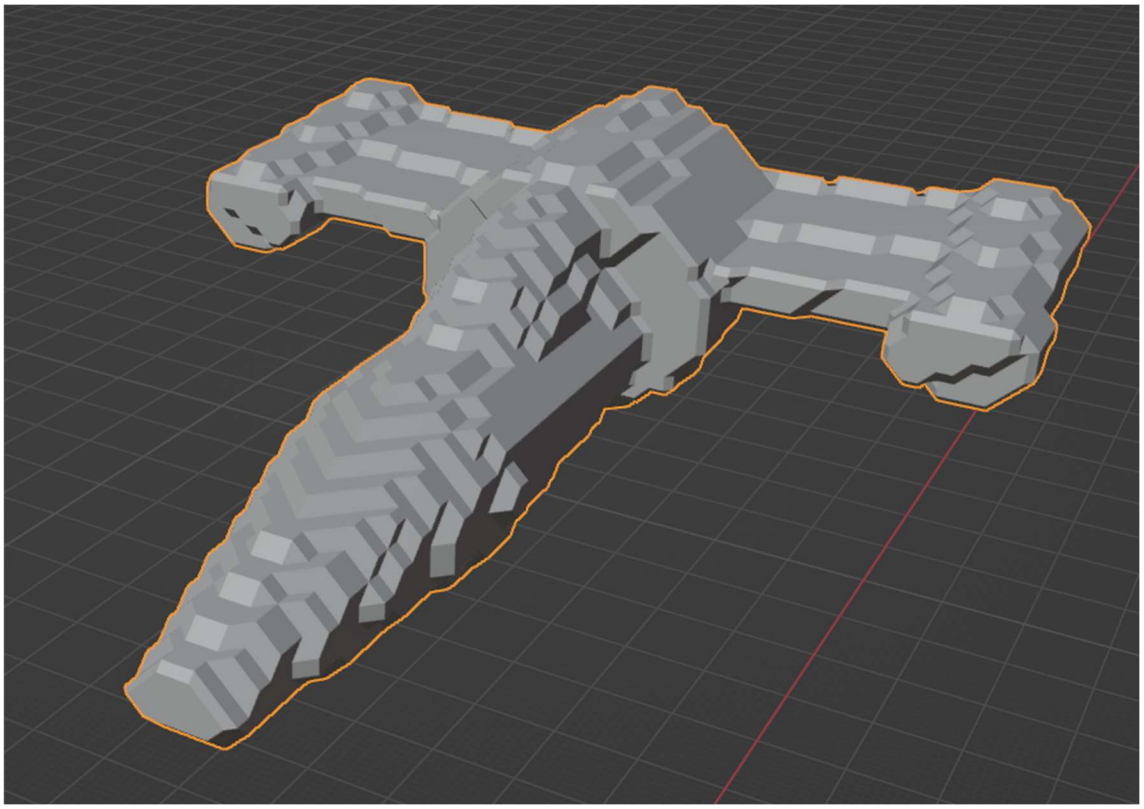
Na obrázku N lze pozorovat jeden z takto vytvořených modelů. Jedná se o hrubý model skládající se z devíti primitiv dále upravených modifikátory sjednocení a odečtení. Model samotný byl pak zrcadlen podél osy x pro zajištění souměrnosti. Exportovaný mesh se skládá z 11860 vertexů a 22494 trojúhelníků.

Takový mesh by však mohl v určitých případech způsobovat potíže při nadcházející práci z důvodu použití Marching Cubes algoritmu. Jelikož při exportu modelu dochází nejprve k dohledání konkrétního vzorku v seznamu možných výsledků a následné interpolaci po hranách tohoto vzorku, na modelu samotném vznikají exportovací artefakty. Ty lze pozorovat na obrázku N. Možnosti odstranění těchto artefaktů, nebo alternativně možnosti využití takového meshe, je nutné dále prozkoumat.



**Obrázek 17. Přiblížený pohled na polygonální strukturu meshe. Pozorovat lze nerovnoměrné polygony způsobené použitím Marching Cubes algoritmu. Zdroj: Vlastní práce**

Druhou alternativou pro použití modelu je jeho exportování bez interpolace vertexů. Takto vyexportovaný model lze pozorovat na obrázku N. Výsledný model na první pohled vypadá velmi chaoticky, jelikož vzdálenosti vertexů od sebe jsou napříč celým meshem uniformní. Při správné volbě velikosti vzorku však mohou takto tvořené modely působit stylizovaně. Nejedná se však o vhodný způsob použití implementovaného nástroje, jelikož výsledný model je příliš nekonzistentní se zobrazovaným distančním polem.



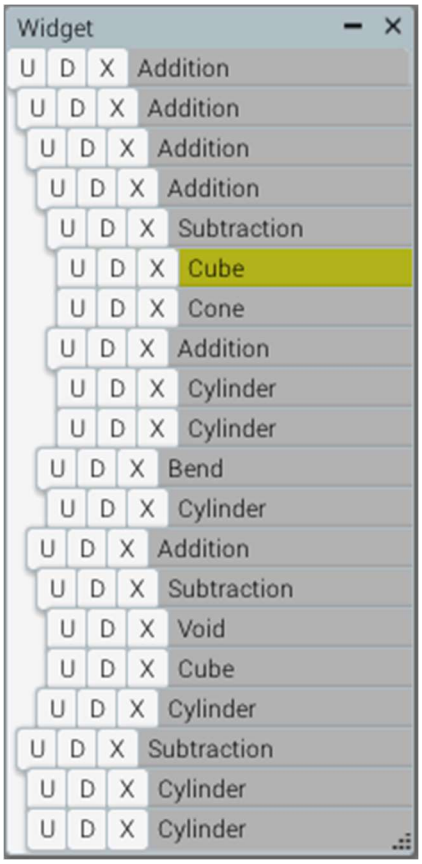
**Obrázek 18. Model vesmírné stíhačky exportovaný bez interpolace vertexů. Zdroj: Vlastní práce**

## 5.3 Ovládání aplikace

Při implementaci aplikace byly pro její ovládání využity shodné nebo podobné ovládací vzory, jako jsou použité ve volně dostupném modelovacím nástroji Blender. Práce s modelem si pak klade za cíl být pro uživatele, kteří mají s tímto nástrojem zkušenosti, velmi intuitivní. Samotná manipulace s tělesy je tedy po krátké navykací době bezproblémová. Práce s implicitními povrchy však vytváří několik problémů zejména pro pokročilejší použití.

Prvním z těchto problémů je složitost scény. Jelikož výsledný model je skládán z primitiv slučovaných do jednoho modelu za pomocí operátorů, byl pro vizuální reprezentaci zvolen obecně používaný vzor stromu. Ten umisťuje kořen stromu na nejvyšší pozici v seznamu a jednotlivé podúrovně pak zleva postupně odsazuje. Tento vzor lze sledovat například při zobrazení adresářové struktury souborového systému. Takováto struktura funguje velmi dobře při práci pouze s několika

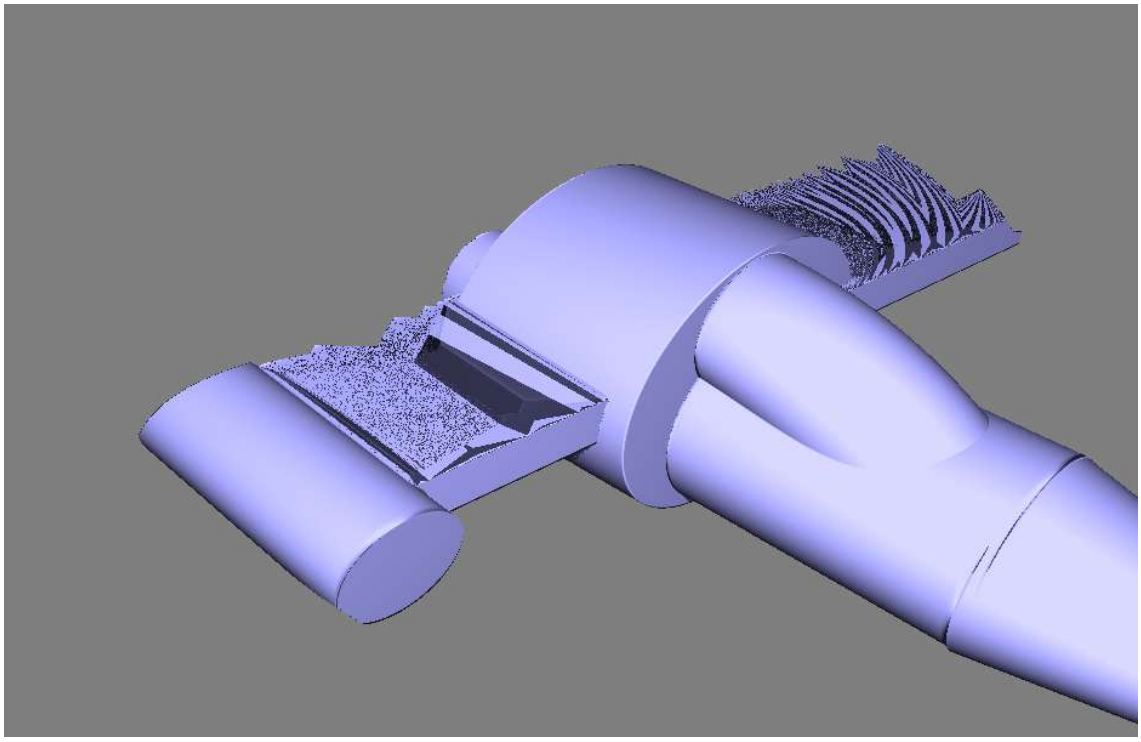
úrovněmi i při větším množství položek na každé úrovni, nebo alternativně s více úrovněmi za předpokladu, že na každé z nich se nachází právě jedna položka, která zobrazuje své potomky. V současné chvíli je však v aplikaci využíváno velké množství úrovní, kde každá zobrazuje své potomky a zároveň se na každé úrovni nachází až dvě položky. Zvolená reprezentace pak díky tomu není vhodná pro práci s modelem, jelikož s narůstajícím množství primitiv ve scéně začíná být velmi nepřehledná.




**Obrázek 8. Detail widgetu položek scény. S rostoucím množstvím položek lze pozorovat ztrátu přehlednosti stromové struktury.**
**Zdroj: Vlastní práce**

Složitost scény pak s rostoucím množstvím primitiv ve scéně způsobuje i rostoucí náročnost na hardwarové prostředky. Ta pak při dosažení stropu vede ke zpomalování práce s aplikací, jelikož její ovládání přestává být plynulé. Tento problém lze částečně eliminovat snížením vykreslovací přesnosti raymarchingového algoritmu. Tím lze oddálit problém komplexity scény, nejedná se

však o úplné odstranění problému. Snížení přesnosti navíc vede ke tvorbě zobrazovacích artefaktů. Například u tenkých těles může docházet k tomu, že paprsek zcela mine jeho povrch a do výsledného obrazu se pak takové těleso nepropíše celé, nebo vůbec.



**Obrázek 19. Znázornění vykreslovacích artefaktů u tenkých těles.**
**Zdroj: Vlastní práce**

Druhým podstatným problémem je pak interakce jednotlivých objektů ve scéně. Při použití modifikátorů, jako je průnik nebo odečítání, dochází k tomu, že některá tělesa ve scéně nejsou vykreslována. Například u průniků, pokud spolu dvě různá primitiva nepokrývají žádný společný prostor, ani jedno z nich se ve scéně nezobrazí. Uživatel je pak nucen operátor průniku dočasně nahradit za jiný, například za sčítání, pozici těles upravit a operátor nakonec přepnout zpět.

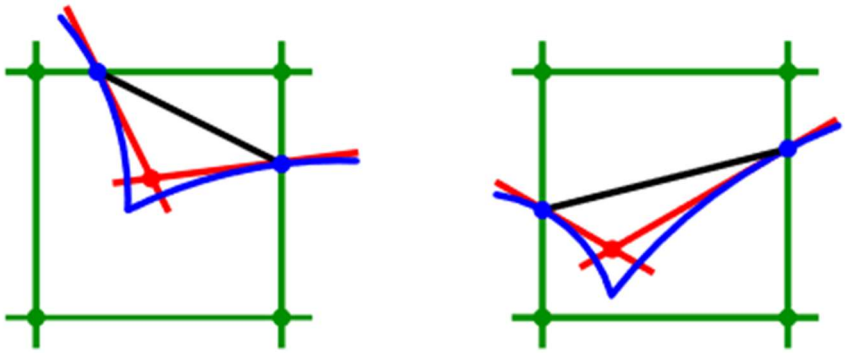
V neposlední řadě pak přichází v úvahu i uniformní velikost vzorku Marching Cubes algoritmu. Tato uniformnost je v některých případech vhodná pro následující práci s modelem, zejména v případě, kdy se model skládá z velkého množství zakřivených ploch. Pokud jsou však součástí modelu rovné plochy, není nutné, aby byla hustota

meshe v těchto místech tak vysoká, jako u hran a nerovných povrchů. V úvahu pak přichází následná redukce meshe například v místech, kde se vertexy nachází přesně na spojnici sousedních vertexů.

# 6 Možnosti rozšíření stávajícího řešení

## *6.1 Extrakce ostrých hran modelu*

Problematikou extrakce informací o vlastnostech objektu z volumetrických dat se zabývají již autoři Kobbelt et. al. v příspěvku Feature sensitive surface extraction from volume data [15]. Chybovost modelu je v této práci přičítána pevné velikosti vzorku Marching Cubes algoritmu, který se nepřizpůsobuje exportovanému tělesu. V místech, kde se v distančním poli nachází ostré hrany, pak dochází ke ztrátě informací o této vlastnosti, jelikož algoritmus vzorkuje pouze sousední body tělesa, které pak jednoduše spojí do jednoho trojúhelníku.



**Obrázek 20. Znázornění práce s tečnými prvky. Modrá křivka znázorňuje původní povrch tělesa. Červené přímky jsou tečné prvky ve vzorkovaných bodech. Jejich průsečík pak odpovídá odhadované pozici hrany povrchu. Zdroj: Feature sensitive surface extraction from volume data [15]**

Jako řešení této ztráty autoři navrhují kromě pozice bodu sbírat informace i o normále povrchu v tomto bodě. Zjištěním normály jednotlivých bodů lze pak vytvořit lineární lokální aproximaci povrchu uvnitř daného vzorku (tečný prvek).

Průsečík tečných prvků zpracovávaných vzorků pak aproximuje umístění hrany a jeho přidání do výsledného meshe způsobí korektní extrakci hrany distančního pole.

## 6.2 Odstranění přebytečných vertexů

U modelu obsahujícího rovné plochy dochází při polygonizaci distančního pole ke vzniku nadměrného množství vertexů, které však o povrchu modelu nenesou zcela žádné informace, jelikož se nachází v rovině s body, se kterými sousedí. Tento problém lze vyřešit dvěma různými způsoby.

Jules Bloomenthal ve své knize Polygonization of implicit surfaces navrhuje první ze dvou řešení.[16] Jedná se o zamezení samotné tvorby takovýchto vertexů. Namísto použití uniformní vzorkovací velikosti Marching Cubes algoritmu je možné velikost vzorku měnit adaptivně v místech, kde se nachází vlastnosti modelu, které vyžadují vyšší preciznost. Tento přístup vede k úbytku vzorků zpracovávaných Marching Cubes algoritmem a tím i k úbytku exportovaných bodů, zachovává si však výslednou preciznost v místech modelu, kde je vyšší hustota vertexů potřebná.

Druhým způsobem řešení tohoto problému je následný post-processing vytvořeného meshe a tím způsobená dodatečná redukce vertexů po jeho vyexportování. Jako vhodný nástroj k této redukci se nabízí geometry shader, o který byla vykreslovací pipeline rozšířena v hardwaru podporujícím DirectX® 10 [17]. Na rozdíl od již dlouhodobě dostupného vertex shaderu, sloužící k provádění per-vertex kalkulací, umožňuje geometry shader provádět nad modelem úpravy, které mění jeho výslednou geometrii. Výsledný buffer, který vznikne za pomoci tohoto shaderu, lze pak použít v pozdějších fázích vykreslování, nebo jej extrahovat zpět a zpracovat pomocí CPU.

## 6.3 Uživatelské rozhraní

Jako největší problém uživatelského rozhraní aplikace se jeví grafická reprezentace stromové struktury scény. Aplikace v tuto chvíli zobrazuje strukturu scény jako běžně používaný top-down strom. Pro zvýšení přehlednosti se však nabízí volba alternativní vizualizace využívající komplexnější reprezentaci.

Autoři Pavlopoulos et. al. ve článku A reference guide for tree analysis and visualization řeší problémy reprezentace dat ve stromové struktuře pro zobrazení informací získaných z velkoobjemových studií evoluce [18]. Článek prezentuje souhrn dostupných open-source technologií a možnosti jejich použití pro vizualizaci dat skládajících se až z tisíců uzlů. Přestože od implementovaného nástroje není takováto datová hustota očekávána, lze k reprezentaci scény využít některého z navrhovaných řešení.

Typy vizualizace se skládají ze dvou významných skupin. První z nich je reprezentace stromu ve dvourozměrném prostoru. Nástroje jako Dendroscope nebo HyperTree jsou volně dostupné a zaměřeny pro tyto účely. Standardními nástroji pro manipulaci se stromem je v takovýchto nástrojích přiblížení/oddálení a posun pomocí kurzoru myši, což je funkčnost, kterou zde implementovaná aplikace v současnosti nenabízí.

Jako druhý způsob řešení se pak nabízí reprezentace stromu ve trojrozměrném prostoru. Taková řešení se oproti dvourozměrným stromům vyznačují daleko vyšší infomační hustotou. Nastává zde však alternativní otázka jednoduchosti použití. Manipulace s takto reprezentovaným stromem se stává náročnější z hlediska nutnosti pochopení druhého trojrozměrného prostoru a způsobu, jakým jeho obsah vyjadřuje obsah scény. V závěru je však nutno znovu podotknout nižší objem dat potřebných k reprezentaci ve zde vytvořeném nástroji, která s velkou pravděpodobností povede k volbě dvojrozměrných stromů jako vhodnější alternativy.

I při reprezentaci scény pomocí tradičně známých způsobů lze však práci se stromem výrazně zjednodušit. Nabízí se implementace možností skrytí potomků u vybraných uzlů. V takovém případě by se jednalo o shodný způsob práce se stromem, jaký využívají nástroje určené k práci se souborovými systémy. Tento postup však nebyl v prvotním návrhu využit vzhledem k faktu, že způsobuje ztrátu některých informací o scéně a potřebu častější interakce se stromem pro jejich zobrazení.

# 7 Závěr

V práci byly prozkoumány a navrženy možnosti tvorby trojrozměrných modelů matematickým popisem. Toho bylo dosaženo využitím implicitních povrchů a množinových operátorů. Návrh byl opodstatněn nedostatky práce s trojrozměrnými modely založenými na editaci polygonálního meshe. Jedná se o nedostatky jako vysoké množství polygonů, nebo nároky na výpočetní výkon. Práce s implicitními povrchy má sloužit jako alternativa k polygonálnímu modelování. Za tímto účelem byl popsán způsob extrakce polygonálního modelu z výsledného matematického popisu.

V první části byl představen teoretický podklad. Popsán byl způsob definice primitivů pomocí matematických funkcí známých jako implicitní povrchy. Primitiva byla dále seskupena pomocí množinových operátorů jako sjednocení nebo průnik. Jako způsob zobrazení takto popsaných modelů bylo navrženo použití Raymarching algoritmu. Pro extrakci výsledné polygonální sítě byl pak zvolen algoritmus Marching Cubes.

Navržené postupy byly dále implementovány ve druhé části práce. Popsán zde byl způsob použití knihoven LWJGL a LEGUI pro implementaci nástroje pro modelování implicitních povrchů. Tento nástroj byl implementován v programovacím jazyce Java. Popsána byla struktura aplikace a očekávané použití. V neposlední řadě byl představen způsob extrakce polygonální sítě a její uložení do formátu Wavefront .obj.

Výsledné modely byly ve třetí části vizuálně zhodnoceny. Nejprve byla exportovaná tělesa porovnána s možnostmi nabízenými volně dostupným nástrojem Blender. Na těchto srovnáních byly pozorovány chyby Marching Cubes algoritmu. Zjištěny byly dva podstatné problémy extrahovaných sítí. Prvním z nich je nepřesnost algoritmu Marching Cubes v částech modelu, kde se nachází ostré hrany, nebo jiné detailní vlastnosti. Druhým z nich je nerovnoměrnost exportovaného meshe a jeho vysoká hustota v rovných plochách, kde je výsledný počet polygonů příliš vysoký.

Dále byl proveden export složitějšího modelu složeného z více primitiv. Na něm byla prezentována přítomnost stejných chyb, které byly pozorovány na jednoduchých tělesech. Zhodnoceno bylo i jeho možné použití v praxi.

Nakonec byla věnována pozornost i uživatelskému rozhraní aplikace. Jako největším nedostatkem uživatelského rozhraní byla zjištěna reprezentace stromové struktury scény. S rostoucím počtem primitiv ve scéně byl pozorován úbytek přehlednosti současně implementované reprezentace stromu.

V závěru byly navrženy kroky vhodné k odstranění výše zmíněných problémů. Řešení si zakládají na třech teoretických podkladech. Pro extrakci ostrých hran z polygonální sítě bylo navrženo použití vylepšené implementace Marching Cubes algoritmu navržené autory Kobbelt et. al. Možnost použití tohoto algoritmu při práci s vytvářeným modelem je však nutné dále prozkoumat.

Pro zredukování nepotřebných polygonů byly navrženy dva postupy. Prvním z nich je použití dynamické velikosti vzorkovací mříže Marching Cubes algoritmu navržené J. Bloomenthalem. Druhým je pak post-processing výsledného meshe pomocí geometry shaderu. Zde je nutné další šetření s ohledem na to, který z navržených způsobů se jeví jako vhodnější k použití.

Pro zpřehlednění stromové struktury scény bylo navrženo využití pokročilejších nástrojů pro vizualizaci stromů. Jedná se o nástroje jako Dendroscope a HyperTree, které dokážou lépe reprezentovat stromovou strukturu ve dvou rozměrech. Způsob nasazení těchto nástrojů je nutné dále zkoumat.

# 8 Seznam použité literatury


[1] J. F. Blinn, „A generalization of algebraic surface drawing", *ACM Trans. Graph. TOG*, roč. 1, č. 3, s. 235–256, 1982.

[2] G. Turk, „Modelling with Implicit Surfaces that Interpolate", *ACM Trans. Graph.*, roč. 21, č. 4, s. 19.

[3] T. Reiner, G. Mückl, a C. Dachsbacher, „Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions", *Comput. Graph.*, roč. 35, č. 3, s. 596–603, čer. 2011, doi: 10.1016/j.cag.2011.03.010.

[4] I. Quilez, „Inigo Quilez :: fractals, computer graphics, mathematics, shaders, demoscene and more". https://iquilezles.org/www/articles/distfunctions/distfunctions.htm (viděno 27. srpen 2020).

[5] P. G. Kry a A. Bunt, *Graphics Interface 2014*. CRC Press, 2020.

[6] „Distance Fields", *The Little Grasshopper*. https://prideout.net/blog/distance_fields (viděno 22. duben 2022).

[7] J. C. Hart, „Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces", *Vis. Comput.*, roč. 12, č. 10, s. 527–545, pro. 1996, doi: 10.1007/s003710050084.

[8] M. Mroz, „Signed Distance Fields in Real-Time Rendering", Diplomová práce, FH Technikum Wien, Philadelphia, 2017.

[9] C. Loop a J. Blinn, „Real-time GPU rendering of piecewise algebraic surfaces", in *ACM SIGGRAPH 2006 Papers*, New York, NY, USA, čvc. 2006, s. 664–670. doi: 10.1145/1179352.1141939.

[10] P. Jeremias a I. Quilez, „Shadertoy: learn to create everything in a fragment shader", in *SIGGRAPH Asia 2014 Courses on - SA '14*, Shenzhen, China, 2014, s. 1–15. doi: 10.1145/2659467.2659474.

[11] W. E. Lorensen a H. E. Cline, „Marching cubes: A high resolution 3D surface construction algorithm", *ACM SIGGRAPH Comput. Graph.*, roč. 21, č. 4, s. 163–169, srp. 1987, doi: 10.1145/37402.37422.

[12] *LWJGL/lwjgl3*. Lightweight Java Game Library, 2022. Viděno: 28. duben 2022. [Online]. Dostupné z: https://github.com/LWJGL/lwjgl3

[13] SpinyOwl, *LEGUI*. 2022. Viděno: 28. duben 2022. [Online]. Dostupné z: https://github.com/SpinyOwl/legui

[14] „Wavefront OBJ File Format", 23. leden 2020. https://www.loc.gov/preservation/digital/formats/fdd/fdd000507.shtml (viděno 28. duben 2022).

[15] L. P. Kobbelt, M. Botsch, U. Schwanecke, a H.-P. Seidel, „Feature sensitive surface extraction from volume data", in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, srp. 2001, s. 57–66. doi: 10.1145/383259.383265.

[16] J. Bloomenthal, „Polygonization of implicit surfaces", *Comput. Aided Geom. Des.*, roč. 5, č. 4, s. 341–355, lis. 1988, doi: 10.1016/0167-8396(88)90013-1.

[17] C. DeCoro a N. Tatarchuk, „Real-time mesh simplification using the GPU", in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, New York, NY, USA, dub. 2007, s. 161–166. doi: 10.1145/1230100.1230128.

[18] G. A. Pavlopoulos, T. G. Soldatos, A. Barbosa-Silva, a R. Schneider, „A reference guide for tree analysis and visualization", *BioData Min.*, roč. 3, č. 1, s. 1, úno. 2010, doi: 10.1186/1756-0381-3-1.

# 9 Přílohy

1) Zdrojový kód

# *Příloha č. 1: Zdrojový kód*

Odkaz: https://gitlab.com/tepo99/dominik-spilka-bakalarska-prace-2022

Struktura:
Zdrojový kód aplikace lze najít v adresáři src/main/java. V adresáři src/main/resources se nachází zdrojové kódy pro fragment a compute shader.
Aplikace je postavená na návrhovém vzoru MVC. Aplikace je spuštěna instancováním třídy RenderController volaným při spuštění. Jednotlivé vlastnosti scény jsou popsány třídami ve složce Model. Instancování elementů probíhá pomocí tříd ve složce Factory. Složka Services obsahuje logické třídy aplikace například pro ukládání a načítání scény, nebo pro práci s maticemi. Ve složce View se nachází třídy popisující prvky uživatelského rozhraní.

Univerzita Hradec Králové
Fakulta informatiky a managementu

# Zadání bakalářské práce

| | |
|---|---|
| **Autor:** | **Dominik Spilka** |
| Studium: | I1800227 |
| Studijní program: | B1802 Aplikovaná informatika |
| Studijní obor: | Aplikovaná informatika |
| **Název bakalářské práce:** | **Implicitní povrchy v počítačové grafice** |
| Název bakalářské práce AJ: | Implicit surfaces in computer graphics |

**Cíl, metody, literatura, předpoklady:**

Cíl práce:

Prozkoumat přístupy a techniky modelování a zobrazení implicitních povrchů.

Postup prací:

1. Prozkoumat principy a metody pro definici modelů definovaných implicitní funkcí v 3D prostoru
2. Vytvořit přehled používaných metod editace a zobrazení těchto modelů
3. Navrhnout a implementovat řešení pro převod implicitních povrchů na polygonální model se zaměřením na možnosti využití programovatelných grafických karet
4. Navrhnout metriku srovnání a pro vybrané scény provést testování a porovnání s klasickou renderovací pipeline
5. Zhodnotit dosažené výsledky


W. E. Lorensen a H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm", *SIGGRAPH Comput. Graph.*, roč. 21, č. 4, s. 163–169, srp. 1987, doi: 10.1145/37402.37422.

T. Reiner, G. Mückl, a C. Dachsbacher, "Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions", *Computers & Graphics*, roč. 35, č. 3, s. 596–603, čer. 2011, doi: 10.1016/j.cag.2011.03.010.

L. P. Kobbelt, M. Botsch, U. Schwanecke, a H.-P. Seidel, "Feature sensitive surface extraction from volume data", in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, srp. 2001, s. 57–66, doi: 10.1145/383259.383265.

M. Mroz, "Signed Distance Fields in Real-Time Rendering", Diplomová práce, FH Technikum Wien, Philadelphia, 2017.

J. Bloomenthal, *Polygonization of Implicit Surfaces*. 1988.


| | |
|---|---|
| Garantující pracoviště: | Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu |
| Vedoucí práce: | Ing. Bruno Ježek, Ph.D. |
| Oponent: | Ing. Milan Košťák |
| Datum zadání závěrečné práce: | 4.5.2020 |