Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Nástroj pro virtuální modelování soch pomocí haptického zařízení

Plzeň 2016 Jan Dvořák

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2016

Jan Dvořák

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce panu Doc. Ing. Liboru Vášovi, Ph.D., za odborný dohled a věcné rady k technické i teoretické části této práce, a paní Prof. Dr. Ing. Ivaně Kolingerové, která tuto práci zprostředkovala.

Abstract

The goal of this thesis was to design and implement an intuitive tool for haptic scultping of 3D objects with the ability to insert missing parts and to define which areas of objects cannot be modified.

The first part compares pros and cons of available devices, libraries and software which can be used to implement the tool.

The second part describes triangle meshes and voxels, the most used data structures for 3D object representation in this area. Methods for haptic rendering and object modification are also described.

The third part describes implemented methods and data structures for object modification.

The performance of haptic libraries and modification tools is compared by experiments in the end of this thesis.

Abstrakt

Cílem této práce bylo navržení a implementace intuitivního nástroje pro virtuální modelování 3D objektů pomocí haptického vstupního zařízení s možností doplnění chybějících částí a určení, jaké oblasti objektů nelze modifikovat.

V první části práce jsou porovnány klady a zápory dostupných zařízení, knihovnen a programů, které lze při návrhu nástroje využít.

Druhá část popisuje trojúhelníkové sítě a voxely, dvě nejužívanější datové struktury pro reprezentaci 3D objektů v tomto typu aplikací. Dále se věnuje přístupům pro haptické renderování a modifikaci objektů.

Třetí část popisuje modifikační přístupy a datové struktury, které byly v rámci této práce naimplementovány.

V závěru jsou pomocí experimentů porovnány výkony dostupných haptických knihoven, a také implementovaných modifikačních nástrojů.

Obsah

1 Úvod

Slovo haptika vzniklo z Řeckého *haptesthai*, což v překladu znamená dotýkat se [23]. V současné době můžeme najít její využití v mnoha oborech, například lékařské simulace, robotika, či herní průmysl.

Haptické modelování je obor, který si získal velkou popularitu díky nedávnému růstu zájmu o 3D tisk. Oproti normálnímu modelování umožňuje cítit hmatovou odezvu při manipulaci s objekty, a proto máme v jistém ohledu větší kontrolu nad tím, jaké změny provádíme. Úskalím haptického modelování je, že abychom docílili reálného vjemu dotyku, musíme zvládat generovat síly s frekvencí 1KHz. To znamená, že je třeba všechny výpočty a modifikace vykonávat podstatně rychleji než grafický rendering.

I přes vyšší zájem o toto odvětví haptiky existuje jen málo softwarových řešení a naprostá většina z nich je proprietární. Cílem této práce byl tedy návrh volně šířitelné aplikace, která umožňuje intuitivní editaci 3D modelů (broušení, řezání atp.), navíc jsme měli ještě specifický požadavek na modelování soch, kde část sochy je „pevná“ (historická), která se editovat nesmí, a část je „doplněná“. Pokud je nám známo, tak tuto věc žádný existující software neumožňuje.

2 Dostupný hardware

2.1 Novint Falcon

Obrázek 2.1: Zařízení Novint Falcon, Zdroj: `http://www.novint.com/index.php/novintfalcon`

Prvním ze dvou zařízení dostupných v haptické laboratoři na Katedře informatiky a výpočetní techniky je přístroj s názvem *Novint Falcon* vyráběný firmou *Novint Technologies inc.* Disponuje třemi stupni volnosti pohybu a třemi haptickými stupni volnosti (síla je generována ve třech osách). K PC se připojuje pomocí rozhraní USB.

Firma také vyrábí nástavec s názvem *Pistol Grip*, který lze jednoduše na zařízení namontovat. Díky svému tvaru je spíše uzpůsoben k ovládání počítačových her, než k 3D modelování.

Haptické rozmezí	101 mm × 101 mm × 101 mm
Polohové rozlišení	400 dpi
Silové schopnosti	$< 8{,}9$ N
Rozhraní	USB 2.0

Tabulka 2.1: Technické specifikace zařízení Novin Falcon, Zdroj: `http://www.novint.com/index.php/novintxio/41`

2.2 Geomagic Touch

Obrázek 2.2: Zařízení Geomagic Touch

Haptické zařízení *Geomagic Touch* (dříve známé pod názvem *Sensable Phantom Omni*) je v současné době vyráběno a prodáváno firmou *3D Systems* specializující se na 3D tisk, 3D skenování a 3D modelování. Disponuje šesti pohybovými a třemi haptickými stupni volnosti.

Hlavní nevýhodou tohoto zařízení je konektivita. K PC se připojuje pomocí síťového kabelu a je třeba spárování pokaždé, když se znovu připojí. Na druhou stranu tvar zařízení a jeho počet stupňů volnosti pohybu tento nedostatek převažují.

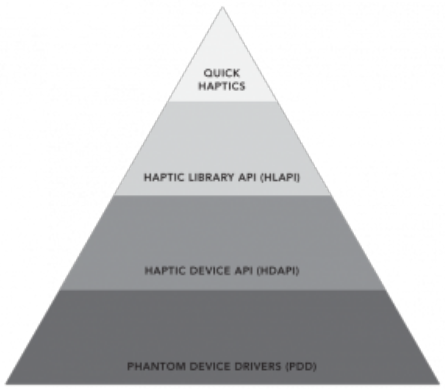
Haptické rozmezí	160 mm × 120 mm × 70 mm
Polohové rozlišení	450 dpi
Silové schopnosti	< 3,3 N
Rozhraní	RJ45

Tabulka 2.2: Technické specifikace zařízení Geomagic Touch, Zdroj: `http://www.geomagic.com/en/products/phantom-omni/specifications/`

3 Dostupné knihovny

3.1 OpenHaptics

Jedná se o sadu nástrojů pro práci s haptickými zařízeními Phantom (mezi které patří i zařízení Geomagic Touch) psaných v jazyce C++. Jedná se o 3 knihovny a 1 ovladač zařízení. Je navržena vrstveně, což znamená, že knihovna/nástroj z vyšší vrstvy využívá pouze knihovny/nástroje z nižší vrstvy. Sada je k dispozici zdarma pro akademické účely, ovšem ke zdrojovým kódům přístup není.



Obrázek 3.1: Pyramida znázorňující rozdělení sady OpenHaptics. [4]

3.1.1 QuickHaptics Micro API

QuickHaptics je knihovna nejvyšší úrovně. S její pomocí lze napsat jednoduché haptické aplikace v několika řádcích. Psaní komplexnějších programů v ní však takřka nelze, jelikož nemáme přístup ani k datovým strukturám, ani ke grafické smyčce.

3.1.2 Haptic Library API

Haptic Library API (dále již HLAPI) je vysokoúrovňová knihovna určená převážně pro programátory znalé práce s *OpenGL*. Není třeba počítat síly aplikované zařízením, stačí pouze namapovat haptický prostor na ten grafický (tzn. provést transformace souřadnic zařízení na sořadnice grafiky) a označit jaká grafická primitiva z grafické smyčky se zobrazí i hapticky (ohraničením daných příkazů mezi `hlBeginFrame()` a `hlEndFrame()`). HLAPI

totiž počítá detekce kolizí pomocí depth (resp. feedback) bufferu. Umožňuje také vyvolání události při dotyku objektů, nebo generování vlastních sil pomocí níže uvedené knihovny *HDAPI*.

3.1.3 Haptic Device API

Jedná se o knihovnu nejnižší úrovně. S její pomocí můžeme na zařízení aplikovat vlastní síly a můžeme je jednoduše měnit. Na druhou stranu je třeba vlastní implementace detekce kolizí haptického kurzoru s objekty, či výpočet sil při jakékoliv interakci.

3.2 Chai3D

Chai3D [11] je multiplatformní open-source C++ knihovna původem ze Stanfordovy univerzity. Slouží k haptice, vizualizaci a interaktivní simulaci v reálném čase [1]. Umožňuje rychlý vývoj jak jednoduchých aplikací, tak i složitých, využívajících nízkoúrovňový přístup, kde je potřeba rychle vypočítat sílu, kterou aplikujeme na zařízení. Hlavními výhodami jsou otevřenost kódu, množství ukázkových programů, ze kterých lze vycházet a také podpora voxelů (viz. podkapitola 5.2). Na druhou stranu má až přehnanou funkcionalitu (např. zvuky při tření o předmět), což se může projevit na výkonu aplikace.

3.3 H3DAPI

Další multiplatformní open-source knihovna se jmenuje *H3DAPI*. Pro haptické renderování využívá C++, Python a X3D [2]. Tuto knihovnu se mi bohužel v laboratoři nepovedlo zprovoznit.

4 Existující nástroje pro haptické modelování

4.1 Řešení od firmy 3DSystems

Firma 3DSystems vyvíjí tři produkty zabývající se otázkou haptického modelování:

- Geomagic Freeform
- Geomagic Sculpt
- Cubify Sculpt

Geomagic Freeform je více-účelová platforma určená k návrhu modelů pro 3D tisk. Má podporu pro velkou škálu formátů a datových struktur. Přestože není primárně určen pro haptické modelování, plně ho podporuje. Více informací na `http://www.geomagic.com/en/products/freeform/overview/`.

Geomagic Sculpt je nástroj pro 3D modelování pomocí myši a haptického zařízení. Výstupem programu jsou 3D modely, které lze rovnou vytisknout v 3D tiskárně. Další informace lze najít na `http://www.geomagic.com/en/products/sculpt/overview/`.

Cubify Sculpt je program primárně určený pro haptický design pro 3D tisk. Více informací na `http://cubify.com/products/sculpt`

Bohužel všechny tyto nástroje jsou proprietární, takže nemáme přístup ke kódu. Také je nutné uvést, že z informacích uvedených na jednotlivých webových stránkách nelze přesně určit hlavní rozdíly mezi danými programy.

4.2 Anarkik3D Design

Jediný zde uvedený SW, který není vyvíjen firmou 3D Systems. Funkčností se velice podobá programu Cubify Sculpt, je však určena spíše pro umělce, než designéry, a proto slibuje intuitivnější ovládání. Bohužel i tento program je proprietární. Vice informací na `http://www.anarkik3d.co.uk/`

5 Používané datové struktury

5.1 Trojúhelníkové sítě

Trojúhelníkové sítě patří mezi nejužívanější datové struktury pro reprezentaci 3D objektů. Jsou reprezentovány množinou vektorů V a množinou trojúhelníků T [9].

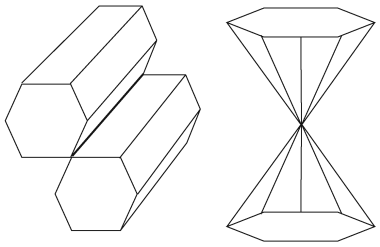
Abychom mohli správně vykreslit trojúhelníky, musíme také znát jejich normálový vektor. Ten pro trojúhelník $t = \{v_a, v_b, v_c\}$ vypočeteme pomocí vzorce [5]:

$$\vec{n}_t = (v_b - v_a) \times (v_c - v_a)$$

Hlavní výhodou trojúhelníkových sítí je jejich nižší paměťová náročnost, jelikož uchováváme informace pouze o povrchu objektu. Na druhou stranu je manipulace s nimi mnohem složitější, jelikož při výraznější změně topologie (například díra v objektu) je třeba množiny vrcholů a trojúhelníků přepočítat.

5.1.1 Manifoldní sítě

Mnoho algoritmů vyžaduje, aby modifikovaná síť byla manifoldní. To znamená, že v dané síti neexistuje hrana ve které by sousedily více jak dva trojúhelníky, a hrany přilehlé k vrcholu tvoří pouze otevřený vějíř, nebo uzavřený vějíř[3]. Na obrázku 5.1 lze vidět, jak vypadá porušení těchto omezení. Manifoldnost je také jeden z požadavků pokud chceme daný objekt vytisknout 3D tiskárnou.

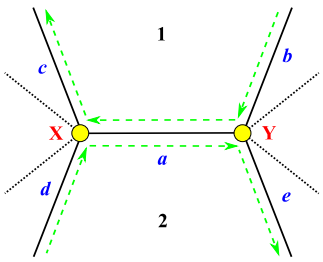


Obrázek 5.1: Příklady sítí, které nejsou manifoldní, Zdroj: [3]

5.1.2 Reprezentace konektivity

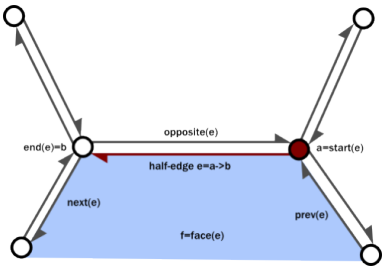
Většina metod pro manipulaci částí objektů vyžaduje pro značné urychlení znalost konektivity modifikované sítě. Například pokud chceme nalézt vrcholy, jenž se nachází do zadané vzdálenosti od určeného vrcholu, můžeme místo iterace nad celou množinou užít algoritmus prohledávání do šířky (resp. jakýkoliv jiný algoritmus pro prohledávání v grafu). Existuje spousta datových struktur, které jsou schopny poskytnout dané prostředky, uvedu zde tři asi nejzajímavější.

Jedna z nejstarších datových struktur sloužících k tomuto účelu se nazývá **Okřídlená hrana** (*Winged edge*)[6]. Ke každé křídlové hraně jsou přiloženy odkazy na dva body, které tato hrana spojuje, dvě plochy, které v této hraně sousedí, předchozí a následující hrany z pohledu první přilehlé plochy, a předchozí a následující hrany z pohledu druhé přilehlé plochy. Množina těchto struktur se přiloží k množině trojúhelníků a vrcholů. Dále je potřeba, aby si každý vrchol a každá plocha uchovávaly odkaz na jednu z přilehlých hran.



Obrázek 5.2: Znázornění ukládaných dat v rámci křídlové hrany. Převzato z: `http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/model/winged-e.html`

Další struktura, která stojí za zmínku se nazývá **Polohrana** (*Half-edge*). Jako polohranu bychom mohli označit stranu hrany přilehlou k jedné ploše. Uchovává odkaz na bod na jejím začátku (resp. konci, záleží na implementaci), přilehlou plochu, následující polohranu a polohranu na opačné straně hrany[18]. Stejně jako u křídlové hrany je potřeba aby si každy vrchol a každá plocha uchovávaly odkaz na jednu z přilehlých polohran. Narozdíl od předchozí struktury však vyžaduje, aby síť byla manifoldní.

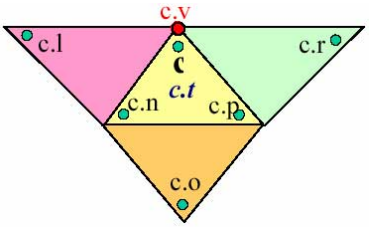


Obrázek 5.3: Znázornění ukládaných dat v rámci polohrany. Zdroj: `https://fgiesen.wordpress.com/2012/02/21/half-edge-based-mesh-representations-theory/`

Další struktura, která lze využít, se nazývá **Rohová tabulka** (*Corner Table*)[21]. Narozdíl od předchozích, neobsahuje informace o hranách (alespoň ne přímé), ale o rozích jednotlivých trojúhelníků. Jejich princip závisí na tom, aby byly trojúhelníky definovány proti směru hodinových ručiček. Pokud je tato podmínka splněna, můžeme pro každý trojúhelník s indexem i a vrcholy v_1, v_2 a v_3 očíslovat jeho rohy následovně:

- roh přilehlý k v_1: $3i + 0$
- roh přilehlý k v_2: $3i + 1$
- roh přilehlý k v_3: $3i + 2$

Poté lze informace o konektivitě sítě uchovávat pomocí dvou polí. V prvním poli se pro každý vrchol nachází vždy jeden libovolný přilehlý roh. Druhé pole uchovává pro každý roh číslo jeho ekvivalentu na opačné straně hrany, naproti které tento roh leží. I tato datová struktura vyžaduje, aby daná trojúhelníková síť byla manifoldní.

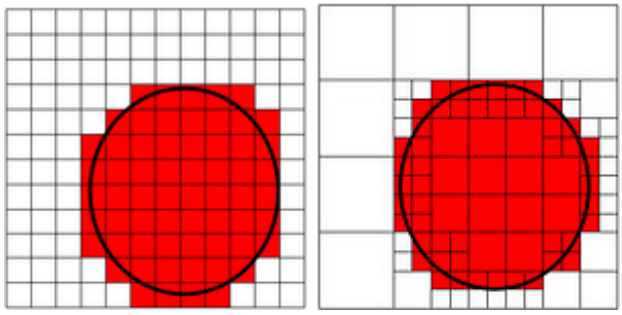


Obrázek 5.4: Znázornění ukládaných dat v rámci rohové tabulky. Převzato z: `http://www.cc.gatech.edu/~jarek/edgebreaker/`

5.2 Voxely

Další často využívaná struktura pro haptické simulace je voxel. Slovo Voxel vzniklo spojením slov *Volume element* [13]. Jedná se v podstatě o trojrozměrný ekvivalent pixelu. Voxely bývají často využívány pro vizualizaci lékařských či vědeckých dat.

Jsou tvořeny číselnou hodnotou - ohodnocením (např. barva, hustota, teplota). Voxely většinou shromažďujeme do mřížky, která má definované rozlišení (např. $512 \times 512 \times 512$ voxelů). Pokud není třeba vysokého rozlišení, můžeme mřížku implementovat pomocí jedoduchého pole. V opačném případě je třeba pro jejich uložení využít efektivnější struktury, jako je třeba **Octree** [19].



Obrázek 5.5: Znázornění paměťové úspory *Octree* pomocí jejího dvojrozměrného ekvivalentu *Quadtree*. Zdroj: `http://www.tomshardware.com/reviews/voxel-ray-casting,2423-4.html`

Kromě výše zmíněné paměťové náročnosti je další nevýhodou voxelů „hrubost“ povrchu objektů. Proto se využívají algoritmy jako **Marching Cubes** [17], které daný objekt převedou na trojúhelníkovou síť, a ta je poté vykreslována.

Oproti trojúhelníkovým sítím mají voxely jednu velkou výhodu - snadnou manipulaci. Pokud bychom chtěli zásadně změnit topologii trojúhelníkové sítě (rytí díry naskrz, zalepování děr, roztahování částí objektů do větších vzdáleností), musíme řešit průniky trojúhelníků (pokud hrany jednoho trojúhelníka pronikají druhým, nejspíše byla stěna zatlačena skrz a měl by se v objektu vytvořit tunel), jejich dělení a posuvy. Pokud však chceme stejnou funkcionalitu pomocí voxelů, stačí pouze měnit jejich ohodnocení, čímž určujeme zda jsou vidět a pokud jsou, tak z jaké části.

6 Používané haptické metody

6.1 Haptické renderování

Haptické renerování je proces, který řeší kdy a jaké síly se na zařízení generují. Je také důležité, aby toto rozhodování probíhalo co nejrychleji, neboť aby se hmatová odezva zdála plynulá, je třeba hapticky vykreslovat s frekvencí kolem 1 KHz [23]. Abychom toho docílili, musíme detekovat kolize a určit velikost sil dostatečně rychle (rychleji než grafické renderování). Existují dva výrazné směry co se týče způsobů, jak tyto síly určit.

6.1.1 Penalty based methods

Tyto metody počítají síly na základě hloubky vniknutí. Ta je určena jako vzdálenost mezi reálným bodem zařízení a nejbližším bodem na povrchu objektu.

V současné době jsou již tyto metody méně používané, jelikož jsou spojovány s některými limitacemi [22]. Například pokud je objekt tenký, v jistém bodě průniku se jako nejbližší bod na povrchu vypočte bod na druhé straně objektu a místo tlačení ovladače zpět k povrchu budeme „vystřeleni“ druhou stranou ven.

6.1.2 Constraint based methods

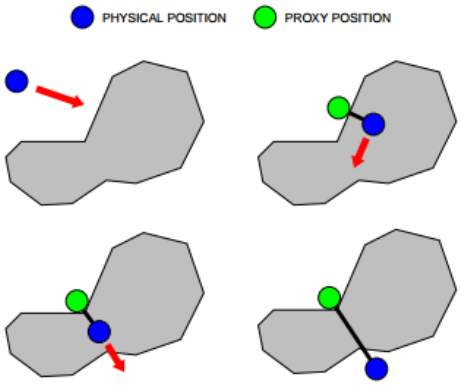
Principem těchto metod je, že je zařízení v prostoru realizováno dvěma body, jeden reprezentuje fyzické souřadnice zařízení, druhý reprezentuje tzv. proxy - bod, kde by zařízení bylo, kdyby nemohlo proniknout dovnitř objektu. Dokud není zařízení v kolizi s nějakým objektem, jsou souřadnice obou těchto bodů identické. Pokud však dojde ke kolizi, fyzický bod pronikne dovnitř objektu a proxy zůstane na povrchu a snaží se po porvchu pohnout tak, aby vzdálenost mezi nimi byla co nejmenší. Nikdy však neprojde skrz na druhou stranu. Výsledná generovaná síla je simulací pružiny mezi těmito dvěma body.

Nejznámější algoritmy založené na tomto principu jsou:

- *God-object* [36]
- *Finger - Proxy* [22]

Rozdíl mezi nimi je v reprezentaci objektu, který zůstane na povrchu. Algoritmus *God-object* jej reprezentuje jako nekonečně malý bod v prostoru, zatímco *Finger - Proxy* používá konečně veliký objekt.

Knihovna Chai3D využívá pro své haptické renderování právě algoritmus *Finger - Proxy*.



Obrázek 6.1: Finger-Proxy algoritmus Zdroj: `http://www.chai3d.org/download/doc/html/chapter16-haptics.html`

6.2 Detekce kolizí

6.2.1 Kolize s jedním trojúhelníkem

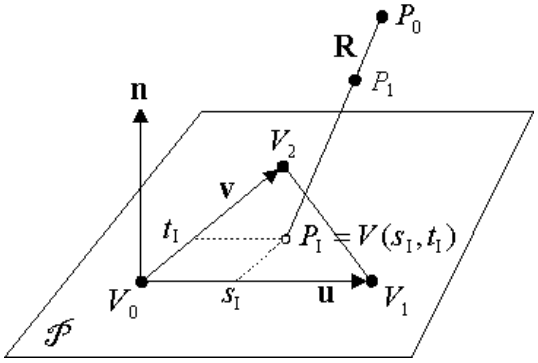
Pokud chceme detekovat kolize nad trojúhelníkovou sítí, musíme umět rozpoznat, zda bod zařízení neprošel nějakým z trojúhelníků (tzn. protnutí spojnice mezi předchozí a současnou pozicí zařízení a daného trojúhelníku). Toho se dá docílit výpočtem barycentrických souřadnic průsečíku [28].

Principielně to funguje takto: Pokud známe bod P_1, ve kterém daná úsečka či přímka protne rovinu trojúhelníka, lze souřadnice tohoto bodu převést na souřadnice v prostoru generovaném pomocí vektorů u a v vytvořených jako

spojnice jednoho vrcholu trojúhelníka V_0 s ostatními vrcholy.
Poté můžeme tedy pozici průsečíku s rovinou zapsat následovně:

$$P_1 = V_0 + t_1 v + s_1 u$$

Platí, že pokud jsou t_1 a s_1 kladná, a $t_1 + s_1 < 1$, bod leží uvnitř trojúhelníku.



Obrázek 6.2: Výpočet průsečíku s jedním trojúhelníkem Zdroj: `http://geomalgorithms.com/a06-_intersect-2.html`

6.2.2 Brute force

Jakmile víme, jak určit zda došlo ke kolizi s jedním trojúhelníkem, musíme navrhnout způsob, jak projít množinou trojúhelníků a určit, které z nich s daným zařízením kolidují.

Nejjednodušší způsob je projít množinu celou a otestovat každý trojúhelník. Nicméně jelikož výpočet barycentrických souřadnic vyžaduje vícero vektorových součinů, je při velkém množství trojúhelníků tento způsob velice neefektivní.

6.2.3 S využitím ohraničujících oblastí

Možným vylepšením výše uvedené metody je obalit trojúhelníky do ohraničujících oblastí a testovat, zda bod zařízení leží uvnitř, či vně této oblasti [24]. Až ve chvíli, kdy leží uvnitř, můžeme testovat podrobněji.Nejznámější z ohraničujících oblastí jsou *AABB* (*Axis-Aligned Bounding Box*), *OBB* (*Oriented Bounding Box*) a *Bounding sphere*.

Z výše zmíněných je nejjednodušší na implementaci *AABB*. Jedná se o osově zarovnaný kvádr, který se dá zkonstruovat nalezením největších a nejmenších souřadnic obalovaného objektu v každé ose. Kontrola, zda daný bod leží uvnitř je pak provedena pouhým zjištěním, zda leží v daném rozmezí souřadnic.

Dalšího zefektivnění detekce kolizí můžeme docílit, pokud dané ohraničující oblasti rozdělíme do nějaké stromové struktury (např. [7]). Tyto struktury je však po každé modifikaci potřeba rekonstruovat.

6.2.4 Procházkový algoritmus

Tento přístup je od všech dříve uvedených naprosto rozdílný. Využívá znalosti o konektivitě trojúhelníkové sítě, aby putoval po trojúhelnících směrem k bodu, kde se protíná přímka určující směr pohybu zařízení s povrchem objektu [27]. Startovní trojúhelník je buď'to vybírán náhodně, nebo lze použít ten, se kterým došlo ke kolizi naposledy.

Výhodou tohoto algoritmu je, že jakmile danou síť zmodifikujeme, není potřeba žádného přepočtu. Navíc jeho paměťová složitost je $O(1)$, bereme-li v podtaz, že informace o konektivitě sítě jsou již uloženy v modifikované trojúhleníkové síti.

7 Manipulace s objekty

7.1 Trojúhelníkové sítě

7.1.1 Posun trojúhelníkem

Jedná se asi o výpočetně nejméně náročný ze zde uvedených přístupů, jelikož nevyžaduje žádné složité prohledávání trojúhelníkové sítě, abychom zjistili, jakými vrcholy se bude pohybovat, a ani není složité určit, jak budou posouvány. Při zdetekované kolizi získáme trojúhelník, na kterém k dané kolizi došlo. K pozicím jeho tří vrcholů poté přičítáme vektor, který je roven spojnici pozice haptického kurzoru v předchozím kroku s tou současnou.

Tato metoda se nejvíce hodí pro modifikace jednoduchých sítí, u kterých chceme vytvořit hrubý detail. Naopak, pokud bychom ji chtěli použít na složitější objekty, pohyb pouze třemi vrcholy by byl velice nepraktický.

7.1.2 Posun n vrcholy

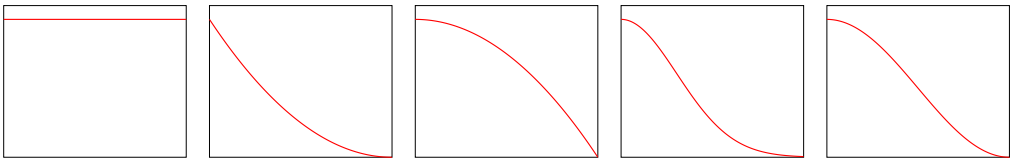
Další možností je k posunu vybrat všechny vrcholy, které se nachází v uživateli předem dané vzdálenosti od bodu kolize zařízení s objektem. Častá složitost trojúhelníkových sítí a požadavek na co nejrychlejší haptickou odezvu vyžadují, aby byl kladen důraz na efektivitu algoritmu, který dané vrcholy vyhledává. Řešením může být například dělení prostoru, či užití některé z výše uvedených datových struktur pro získání konektivity v síti a následovné prohledávání v grafu.

Je vhodné také určit koeficienty, které stanoví, jak moc se tyto vrcholy budou pohybovat. Potřebujeme tedy použít funkci parametrizovanou zadaným poloměrem r, která daný koeficient vypočte ze vzdálenosti vrcholu od bodu kolize. Na funkci jsou kladeny tyto požadavky:

- Musí zobrazit interval $< 0, r >$ do intervalu $< 0, 1 >$ (nechceme, aby body „ustřelovaly“)
- Musí být nerostoucí (s rostoucí vzdáleností od středu by se vrcholy neměly pohybovat více)

Dané požadavky splňují například tyto funkce:

- $f_1(d) = 1$
- $f_2(d) = (1 - \frac{d}{r})^a, a \geq 1$
- $f_3(d) = 1 - (\frac{d}{r})^a, a \geq 1$
- $f_4(d) = e^{(-\frac{d^2}{2(\frac{r}{u})^2})}$, u je p-percentil normálního normovaného rozdělení, $p \to 100\%$
- $f_5(d) = \frac{1}{2}\cos(\frac{d\pi}{r}) + \frac{1}{2}$



(a) (b) (c) (d) (e)

Obrázek 7.1: Porovnání funkcí pro výpočet koeficientů

První uvedená funkce(7.1a) způsobí, že se všechny vybrané vrcholy budou pohybovat stejně, nezávisle na poloze. Efekt, který vytvoříme touto funkcí, částečně připomíná metodu *Posunu trojúhelníkem*. Na rozdíl od ní ale lze použít i na sítě s větším množstvím vrcholů, aniž by ztratil na své praktičnosti.

Druhá a třetí funkce jsou obě polynomy a zápisem se liší pouze osou, po které je musíme posunout, abychom dosáhli výše zmíněných požadavků. Rozdíl je naopak jasně znatelný, pokud porovnáme jejich efekt na modifikovanou síť. Funkce f_2(7.1b), polynom posunutý po ose x, při modifikaci vytváří špičaté hrboly. S rostoucím exponentem roste i její špičatost. Na druhou stranu funkce f_3(7.1c), polynom posunutý po ose y, vytváří zaoblené hrboly a platí, že čím větší exponent a, tím oblejší tyto hrboly jsou. Speciálním případem u obou funkcí je pak pokud $a = 1$. To jsou obě funkce shodné.

Ve čtvrtém případě(7.1d) se jedná o Gaussovskou funkci s parametry $\mu = 0$ a $\sigma = \frac{r}{u}$. Sigma bylo odvozeno ze vztahu pro výpočet kvantilů normálního rozdělení:

$$r = \mu + \sigma u \tag{7.1}$$
$$\frac{r}{u} = \sigma \tag{7.2}$$

Tato funkce působí na síť jako f_3, to znamená vytváří oblé hrboly. Její nevýhodou je, že i body na samém okraji výběru budou mít vždy nenulový, i když velice malý koeficient.

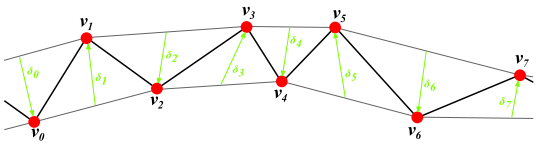
Funkce f_5(7.1e) je kosinus s přizpůsobenou amplitudou a frekvencí, posunutý po ose y. Má podobný tvar jako f_4, ovšem řeší problém s nenulovými koeficienty na samém okraji výběru.

7.1.3 Laplaceovská editace povrchu

Nevýhodou dříve zmíněných přístupů bylo, že pokud bychom pracovali se sítí s vysokým detailem, byl by tento detail při modifikaci značně narušen. Řešení nabízí tzv. metoda Laplaceovské editace povrchu[26]. Pro každý vrchol vypočteme tzv. Laplaceovské souřadnice (značíme δ), které určují, jak moc se daný vrchol liší od svého okolí (viz obrázek 7.2). Vzorec vypadá takto:

$$\delta_i = v_i - \frac{1}{d_i} \sum_{j \in N_i} v_j,$$

kde d_i je stupeň vrcholu v_i, a N_i je množina indexů jeho sousedů.



Obrázek 7.2: Znázornění Laplaceovských souřadnic ve dvojrozměrném prostoru

Proces modifikace vypadá následovně: Vybereme jeden vrchol, který je nejblíže k bodu kolize se zařízením. Tento vrchol se bude pohybovat společně s haptickým kurzorem. Dále nalezneme vrcholy, jejichž pozice budeme určovat (např. všechny vrcholy v poloměru). Je třeba aby byl detail sítě po modifikaci co nejméně pozměněný, což znamená, aby Laplaceovské souřadnice byly takřka stejné. Abychom docílili tohoto kritéria, řešíme soustavu lineárních algebraických rovnic $Ax = b$, odvozenou od výše uvedeného vztahu pro výpočet delta souřadnic, kde hodnoty spojené s vrcholy, jejichž souřadnice nejsou dopočítávány, jsou převedeny na pravou stranu (to znamená, že řádky vrcholů, jejichž souřadnice jsou dány a nesousedí s žádnými dopočítávanými, jsou nulové). A je řídká matice rozměrů $m \times n$, kde m je počet vrcholů, jejichž pozice se přepočítávají a n je počet všech vrcholů dané sítě. Jelikož pomocí

Eulerovy charakteristiky lze dokázat, že v průměru má každý vrchol sítě 6 a méně sousedů, můžeme zkonstatovat, že matice A je řídká. Platí také, že $m < n$, jelikož vždy existuje alespoň jeden vrchol, jehož souřadnice nebudou propočítávány (ten, se kterým pohybujeme haptickým kurzorem), což znamená že daná soustava je pře-určená. V tom případě je potřeba užít metody nejmenších čtverců a počítat místo toho soustavu $A^T Ax = A^T b$. Výsledná matice nové soustavy je řídká, symetrická a pozitivně definitní. Můžeme ji tedy řešit například pomocí metody Choleského rozkladu[31]. Důležité je také zdůraznit, že v průběhu modifikace se mění pouze pravá strana rovnice, což znamená, že rozklad matice stačí spočítat pouze jednou.

7.1.4 Vytváření děr

Pokud používáme jednu z metod, kdy je vrcholy posouváno, je nasnadě velice intuitivní řešení vytváření děr - pokud budeme zařízením dostatečně tlačit vůči objektu, může dojít k tzv. sebe-protnutí, což znamená, že jeden, nebo více vrcholů prošlo skrz druhou stranu objektu, a některé z trojúhelníků dané sítě se protínají s jinými objekty té samé sítě. A právě převedení sebe-protnutí na díru v objektu (neboli oprava sítě) se jeví velice intuitivně (tlačíme, dokud se neprotlačíme skrz). Tomuto problému se částečně věnuje ve své práci slečna Skorkovská[25]. Jak uvádí, přístupy, které tento problém řeší, se dělí na dvě hlavní kategorie: globální a lokální.

Globální přístupy řeší všechna sebe-protnutí, které v dané době na síti proběhly, najednou. Většinou využívají nějakou pomocnou datovou strukturu, na kterou danou síť převedou, aby se později převedla zpět. Jako příklad lze uvést třeba přístup [20], který danou síť převede na volumetrickou reprezentaci, s tím, že určí, jaké voxely jsou uvnitř, a jaké vně objektu. Poté je objekt zpět převeden na trojúhelníkovou síť pomocí Marching cubes algoritmu [17]. Tento přístup je sice přesnější, co se týče opravy sebe-protnutí, nicméně daná síť může kvůli převodu částečně ztrácet svůj detail.

Lokální přístupy řeší jednotlivá sebe-protnutí zvlášť, bez ovlivnění globálního detailu sítě. Buď to znovu využívají nějakou pomocnou datovou strukturu (voxely [8], BSP [10]), ale tentokrát užitou pouze lokálně na místo průniku, nebo upravují přímo geometrii dané sítě [35].

7.1.5 Doplňování materiálu

Tento problém je podobný, jako vytváření děr. Tentokráte ovšem místo sebe-protnutí řešíme protnutí dvou sítí mezi sebou. Dalším rozdílem je, že pokud nám nevadí, že uvnitř sítě leží vrcholy, jež jsou neviditelné, a že i přes dotyk, nejsou tyto dvě sítě propojené, nemusíme opravu vůbec řešit, jelikož tyto dva objekty se při pohledu budou zdát sloučené.

Další věcí, kterou je třeba řešit, je jaký tvar bude mít doplňovaný materiál, a jak se bude generovat. Jednou z mnoha možností je například užití koule. Koule může být generována z pravidelného dvacetistěnu tím, že trojúhelníky, které tvoří jeho stěny, rozdělíme na čtyři stejně velké části a vrcholy, které tímto vzniknou, posuneme, aby jejich vzdálenost od středu byla rovna poloměru koule[16]. Čím více iterací dělení, tím více se generovaný objekt podobá kouli. Tuto kouli pak můžeme škálovat do požadované velikosti.

7.1.6 Vyhlazování

Jedná se o proces, kdy se chceme částečně zbavit hrubých detailů (zuby, ostré hrany, šum atp.). Jedním z přístupů řešících tento problém se nazývá Laplaceovské vyhlazování[15] a využívá Laplaceovských souřadnic, které byly zmíněny už při popisu Laplaceovské editace povrchu. Samotné vyhlazování probíhá tak, že vrcholy posuneme směrem k průměru sousedních vrcholů. Míru posunu určuje koeficient λ, který lze získat pomocí funkcí uvedených u přístupu posunu více vrcholů pouze s tím, že výsledný interval $< 0, 1 >$ je třeba zúžit na $< 0, x >$, kde $x < \frac{1}{2}$. Pokud bychom daný interval nezúžili, detail by se místo vyhlazení obrátil (body, které se v rámci svého okolí jevily jako prohlubně budou nyní vystouplé).

Pokud tento algoritmus iterujeme dostatečně dlouho, narazíme na jeho hlavní nedostatek - síť se při vyhlazování smršťuje. Řešení v podobě metody $\lambda|\mu$ navrhl G. Taubin[29] - v rámci iterace je síť nejdříve vyhlazena způsobem jako u předchozí metody, posléze jsou však vrcholy ještě posunuty s koeficientem μ a platí: $0 < \lambda < -\mu$.

Obě tyto metody ovšem mají dva společné problémy - rychlost konvergence a numerická stabilita. Určité zlepší proto nabízí další přístup - Implicitní vyhlazování [12]. Ten na celý problém pohlíží zcela naopak. Místo toho, aby na danou síť v každém iteračním kroku aplikoval posuny vrcholů směrem k průměru jejich sousedů, hledá pomocí řešení soustavy lineárních rovnic

takový stav sítě, na jež kdyby se aplikovalo zhrubění, vznikl by stav, ve kterém je síť nyní. Maticově se daná soustava dá zapsat následovně:

$$(I + \lambda L)X^{i+1} = X^i,$$

kde L je matice sestavená ze vztahů pro výpočet Laplaceovských souřadnic, X^i je stav sítě v současném kroku a X^{i+1} je nově vypočítávaný stav.

7.1.7 Zachování existujících částí povrchu

Abychom zabránili nechtěné modifikaci některých částí objektu, je třeba explicitně určit, které vrcholy je možné posouvat, a které jsou statické. Jednou z možností, jak tyto vrcholy určit, je načtení jejich indexů ze souboru. Je ovšem velice těžké takové soubory vytvářet bez nějakého pomocného nástroje. I zde je možno využít výhod haptického kurzoru. Můžeme například při dotyku s objektem vzít trojúhelník, se kterým došlo ke kolizi a jeho vrcholy označit (resp. odznačit).

Zavedení statických vrcholů s sebou přináší několik dalších pozitiv. Například pokud vybíráme vrcholy v daném poloměru pomocí prohledávání sítě do šířky, můžeme tento proces značně urychlit, jelikož není třeba expandovat vrcholy označené za statické. Vycházíme z toho, že nechceme, aby při pohybem jedné modifikovatelné oblasti nebyly ovlivněny ostatní, které s touto nejsou nijak propojeny.

7.2 Voxely

Jak již bylo zmíněno výše, manipulace voxelů je velmi jednoduchá. Pokud je například reprezentujeme binárními hodnotami, pak odebrání resp. doplnění materiálu znamená pouze nastavení hodnoty na *„0“* (nevykresluj), resp. *„1“* (vykresluj).

8 Popis implementace

V rámci předmětu KIV/PRJ5 jsem porovnal knihovnu HLAPI s knihovnou Chai3D a voxely s trojúhelníkovými sítěmi. Naměřené hodnoty lze nalézt v sekci 9.1. Výsledkem měření bylo, že jsem se rozhodl ve své práci využít knihovnu Chai3D, jelikož umožňuje mnohem rychlejší vývoj aplikací, neboť spoustu úloh už řeší sama, a proto se programátor nemusí tolik zaobírat haptickým a grafickým vykreslováním a může rovnou začít řešit problémy s modifikací objektů. Navíc i přes nadbytečnou funkcionalitu stále poskytuje větší výkon než HLAPI a je také důkladněji zdokumentována. Co se týče reprezentace 3D objektů, nakonec jsem se rozhodl užít trojúhelníkových sítí, jelikož knihovna Chai3D je pro ně primárně určena, a také kvůli velkému množství testovacích objektů, které byly k nalezení na internetu.

Práce je psána v jazyce C++ a vyvíjena v prostředí Visual Studio 2015. Výsledný program jsem nazval *Haptic Sculpting Tool*, či zkráceně *HST*, což ve volném překladu do českého jazyka znamená *Nástroj pro haptické sochání*.

Program je rozdělen do čtyř logických částí:

- `ui` - Třídy spojené s uživatelským rozhraním (haptickým i grafickým)
- `tool` - Implementace modelovacích nástrojů
- `util` - Pomocné třídy
- `fixer` - Třídy pro opravu trojúhelníkových sítí poskytnuté slečnou Ing. Věrou Skorkovskou

8.1 Uživatelské rozhraní

O hlavní funkcionalitu uživatelského rozhraní se stará třída `UIController`. K popisu celého prostředí aplikace využívá instanci třídy `cWorld`, což je kořenový uzel grafu scény, jenž uchovává informace o všech objektech, kameře, světlech a haptickém kurzoru. `UIController` obsahuje grafickou i haptickou vykreslovací smyčku, a také metodu pro zpracování události stisku klávesy na klávesnici, která je hned po haptickém zařízení nejdůležitějším prostředkem pro ovládání.

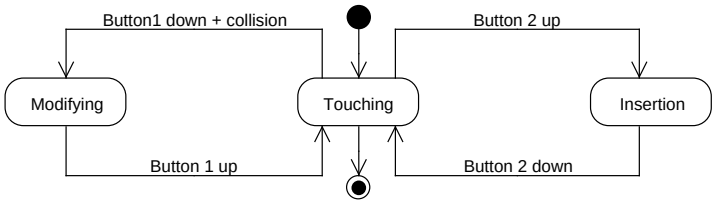
8.1.1 Podporovaná zařízení

Práce sice byla vyvíjena na míru zařízení *Geomagic Touch*, fungovat by však měla na všech zařízeních, která uvádí knihovna *Chai3D* jako podporovaná. Částečně otestováno bylo i zařízení *Novint Falcon*, a na žádné nedostatky jsem nenarazil. Nicméně protože nástroj pro pohyb s objekty využívá rotaci haptického kurzoru, doporučuji tuto aplikaci využívat se zařízením s šesti stupni pohybu.

8.1.2 Virtuální haptické zařízení

Jelikož ne každý, kdo přijde do styku s touto aplikací bude mít k dispozici kompatibilní haptické zařízení, modifikoval jsem knihovnu *Chai3D* tak, aby když se jí nepodaří zinicializovat žádné fyzické zařízení, použije virtuální, ovládané klávesnicí. To má tři stupně pohybové volnosti, což znamená, že při pohybu s objekty není schopné s nimi rotovat. Pro testovací účely ale naprosto postačí.

8.1.3 Stavy aplikace



Obrázek 8.1: Stavový diagram aplikace

Aplikace se v rámci svého běhu může nacházet ve třech různých stavech: dotýkání, modifikace a vkládání materiálu. Tyto stavy byly zavedeny, abychom mohli aplikovat různá omezení v různých situacích, které mohou nastat. Při startu se aplikace nachází ve stavu dotýkání. V tomto stavu může uživatel ohmatávat zobrazované objekty a ovládat program pomocí klávesnice (zapnutí resp. vypnutí režimu celé obrazovky, ukládání a načítání objektů, výběr mezi objekty a mezi modifikačními nástroji atp.). Pokud uživatel drží tlačítko č. 1 na haptickém zařízení (v případě *Geomagic Touch* se jedná o tmavě šedé tlačítko blíže ke špičce haptického pera) a zároveň se dotkne právě zvoleného objektu, aplikace přejde do stavu modifikace. V tomto stavu se aplikuje pohyb haptického kurzoru na vybraný objekt. Je velice

nevhodné měnit v této fázi modifikační nástroj, či načítat nový objekt ze souboru. Vstup z klávesnice je tedy ignorován (kromě ovládání virtuálního zařízení). Ve chvíli, kdy uživatel pustí tlačítko, provedou se poslední úpravy na objektu a aplikace znovu přejde stavu dotýkání. Stav vkládání materiálu se od modifikace liší pouze dvěma detaily, a to tím, že je ovládán druhým tlačítkem, a že pro přechod do tohoto stavu není potřeba se daného objektu dotknout.

8.1.4 Grafická smyčka

Aplikace pro grafický výstup využívá knihoven `OpenGL` a `freeglut`. Funkce překreslení je volána pokaždé, kdy grafické vlákno nemá nic na práci (neboli je ve stavu idle). Jako její nejdůležitější část bych označil těchto několik řádek:

```
//render world
graphicMutex->acquire();
camera->renderView(windowW, windowH);
graphicMutex->release();

// swap buffers
glutSwapBuffers();
```

O samotné vykreslení celého prostředí aplikace se stará objekt kamery, jelikož však užíváme *double buffering*[30], je ještě nutné zavolat funkci pro výměnu bufferů. Také je potřeba ošetřit metodu vykreslení mutexem, abychom zabránili vykreslování objektů ve chvíli, kdy jsou kvůli probíhající modifikaci v nekonzistentním stavu.

8.1.5 Haptická smyčka

Haptická simulace běží ve vlákně s vysokou prioritou. O rozhraní aplikace a haptického vstupu se stará instance třídy `cToolCursor` s názvem `hapticCursor`. Průběh haptické smyčky záleží na stavu, v jakém se aplikace nachází. Společné kroky všech stavů jsou přepočet globálních pozic všech uzlů grafu scény, aktualizace stavu zařízení, přepočet sil a aplikování sil. Přepočtené síly se ovšem aplikují na zařízení pouze ve stavu dotýkání. V ostatních stavech jsou vypočtené síly přepsány vlastními hodnotami. I tak je ovšem potřeba vykonat přepočet sil, jelikož v rámci tohoto kroku se detekují kolize a je přepočítávána pozice proxy objektu.

Detekce kolizí je řešena tak, že každý vykreslovaný objekt vlastní tzv. *detektor*. Ten je na základě dat předaných haptickým kurzorem schopen říct,

zda došlo ke kolizi, či ne. Výchozí detektor využívá **Brute force** algoritmus (kontrola všech trojúhelníků), je proto nutné explicitně určit, že bude využíván jiný. Knihovna *Chai3D* nabízí ještě detektor využívající **strom AABB** (viz podkapitolu 6.2.3), který je používán v rámci této aplikace. Jeho nevýhodou však je to, že při jakékoliv změně geometrie objektu je potřeba celý strom znovu přepočítat, neboť není dynamický.

Ve stavu modifikace a vkládání materiálu jsou, jak již bylo nastíněno, vypočtené síly přepsány vlastními. Při vkládání materiálu jsou síly zcela vynulovány. Pokud jsme ve stavu modifikace, je na zařízení aplikována síla simulující efekt pružiny. Výhodou takto generované síly je, že haptický kurzor má tendenci se vracet do výchozího bodu modifikace, což poskytuje uživateli možnost danou změnu vrátit zpět. Efekt pružiny se dá simulovat následovně:

$$\vec{F} = k(\vec{c} - \vec{p}),$$

kde $\vec{c}$ je střed pružiny, $\vec{p}$ je současná pozice haptického kurzoru a k je tuhost pružiny.

8.2 Reprezentace trojúhelníkových sítí

8.2.1 Souborový formát Wavefront OBJ

Pro souborový vstup (resp. výstup) využívá aplikace zjednodušeného formátu Wavefront OBJ[34]. Zjednodušený je tím, že obsahuje informace pouze o pozicích vrcholů a o plochách. Ostatní údaje (např. normály, textury a materiály) byly vypuštěny, jelikož by další kontrola vstupních parametrů zpomalovala proces načtení. Program je schopen načíst i OBJ soubor, který tyto informace obsahuje, pouze jsou tyto informace ignorovány. Soubory tohoto formátu vypadají takto:

```
#Toto je komentář

#Následující řádky definují pozice vrcholů.
#Řádek začíná znakem v~a následují tři čísla s~pohyblivou řádovou čárkou.
v~0.001 2.578 3.14159

#Pokud za těmito třemi hodnotami následují další, jsou ignorovány.
#Některé programy tyto hodnoty používají např. pro stanovení barev vrcholů.
v~0.23 2.47 -0.1 255 255 0
.
.
.
#Následují definice ploch
#Řádek začíná znakem f a následují tři a více hodnot značících indexy vrcholů
#Indexy počítány od 1
f 1 5 6
f 1 10 54 6 45
.
.
.
```

8.2.2 Třída `cMesh`

Chai3D využívá pro reprezentaci trojúhelníkových sítí třídu `cMesh`. Ta obsahuje základní údaje potřebné pro správné vykreslení sítě, jako například seznam vrcholů, jejich pozice a normály, seznam trojúhelníků, detektor kolizí, údaje o barvách a materiálu atd. Pro naše potřeby je ovšem nedostačující, jelikož je nejspíše určena hlavně pro nemodifikovatelné objekty. Soudím to podle toho, že například co se týče přepočtu normál, poskytuje pouze metodu pro přepočet pro všechny trojúhelníky, což je například pokud modifikujeme malé množství trojúhelníků na složité síti velmi nepraktické. Naštěstí jsou její data přístupná a modifikovatelná, takže bylo možno všechny požadované metody do-implementovat v rámci mé obalovací třídy(viz sekci 8.2.6).

8.2.3 Třída `CornerTable`

Další nevýhodou třídy `cMesh` je to, že neposkytuje dostatečná data pro zjištění konektivity sítě. Obsahuje sice pole hran, nicméně je nutné iterovat celým tímto polem, abychom nalezli všechny přilehlé hrany (resp. všechny sousední vrcholy) daného vrcholu, neboli časová složitost tohoto algoritmu je $O(|E|)$, kde $|E|$ je počet hran.

Proto jsem k dané reprezentaci přidal ještě **Rohovou tabulku** (*Corner Table*) [21], která je schopna ten samý úkol vyřešit se složitostí $O(|N|)$, kde $|N|$ je počet sousedních vrcholů. Je to způsobeno tím, že další sousední vrchol lze nalézt v konstantním čase. Je tedy jednoduché jimi iterovat:

```
nextIncidentCorner = opposite[prev(corner)]
nextIncidentVertex = getVertexFromCorner(nextIncidentCorner)
```

Navíc ze všech struktur zmíněných v sekci 5.1.2 je dle mého názoru nejjednoduší, jelikož její data jsou uložena pouze ve dvou polích - `incidentCorner` a `opposite`. V poli `incidentCorner` se pro každý vrchol nachází jeden přilehlý roh. Pole `opposite` zase udržuje ke každému rohu index jejich ekvivalentu na opačné straně incidentní hrany. Zbylé struktury ještě musejí uchovávat další data, která umožní spojit trojúhelník s jedním přilehlým rohem a naopak. Toho díky již dříve zmíněnému důmyslnému očíslování rohů není třeba a například pokud chceme určit, v jakém trojúhelníku se daný roh nachází, stačí pouze jeho index celočíselně vydělit třemi.

Původním autorem mnou užité implementace rohové tabulky je vedoucí mé práce, pan Doc. Ing. Libor Váša Ph.D., ovšem bylo ji potřeba přepsat z jazyka `C#` do `C++` a doplnit ji o metodu pro vyhledávání vrcholů v daném poloměru s přihlédnutím k jejich statičnosti (nechceme vybrat takové vrcholy, se kterými nelze pohybovat). Stačilo však pouze doplnit stávající metodu pro vyhledávání o novou podmínku, zda se daný expandovaný vrchol nachází v množině statických.

Použití této datové struktury znamená, že modifikované trojúhelníkové sítě **musí být manifoldní**.

8.2.4 Laplaceovské souřadnice

Laplaceovské souřadnice, které jsou využívány Laplaceovskou editací povrchu a Laplaceovským vyhlazováním (více viz. sekce 7.1), jsou ukládány

v poli trojrozměrných vektorů. Metody pro jejich výpočet se nachází ve třídě `LaplacianUtil`, která je součástí skupiny pomocných tříd.

8.2.5 Množina statických vrcholů

To, jaké části jsou „pevné“ určuje tzv. množina statických vrcholů. S vrcholy, které se v této množině nacházejí nelze pohybovat. Jedinou výjimkou je vytváření děr. Pro jejich ukládání program využívá datové struktury `unordered_set`[odkaz], která pro přístup využívá rozptylovou funkci. Proto určení, zda daný vrchol lze posouvat, má průměrnou časovou složitost $O(1)$.

Hodnoty jsou nejdříve načítány ze souboru, nicméně lze je měnit pomocí nástroje pro výběr statických (resp. pohyblivých) vrcholů, jenž bude popsán v sekci 8.3.5. Pro přehlednost jsou statické a pohyblivé vrcholy barevně rozlišeny.

Pro účely této aplikace jsem navrhl jednoduchý textový formát *SVS* (zkratka výrazu *Static Vertex Set*, neboli v překladu *Množina statických vrcholů*) pro načítání a ukládání dat. Soubory v tomto formátu obsahují na každé své řádce nenulové číslo, které značí index daného statického vrcholu. Je tedy důležité, aby tato čísla odpovídala počtu vrcholů používané trojúhelníkové sítě.

8.2.6 Třída `Mesh`

Jedná se o obalovací třídu všech datových struktur uvedených v této sekci. Každému zobrazovanému objektu náleží právě jedna instance této třídy. Doplňuje metody, které poskytuje třída `cMesh` o následující: výpočet normál pouze pro vrcholy ovlivněné modifikací, přepočet rohové tabulky a laplaceovských souřadnic a oprava sítě. S její pomocí je již možno definovat objekty, jejichž geometrie se bude často měnit. Pokaždé, když je modifikace objektu dokončena, stačí zavolat metodu `update(vector<int> affectedVertices, bool shouldRepair)`, které předáme seznam ovlivněných vrcholů a příznak, zda chceme na daném objektu opravit sebe-protnutí a vytvořit tak díry.

8.2.7 Výstup do souboru

Poté co, uživatel zmodifikuje načtené objekty, může je všechny uložit do souboru. Jelikož se ukládají dohromady, je potřeba přepočítat indexy vrcholů v nově vzniklém objektu, který bude definován vytvářeným souborem. Vzorec pro výpočet nového indexu pro vrchol j, který je součástí sítě i, je následující:

$$index_{ij} = base_i + j$$

$$base_i = \sum_{k=1}^{i-1} n_k, base_1 = 0$$

nebo

$$base_i = base_{i-1} + n_{i-1}, base_1 = 0,$$

kde n_i je počet vrcholů objetku s indexem i.

8.3 Nástroje pro modifikaci

Třídy implementující nástroje pro modifikaci trojúhelníkové sítě jsou všechny až na jedinou výjimku (nástroj pro vkládání materiálu) potomky třídy `Tool`, která definuje rozhraní mezi manipulací sítě a UI. Metody, které poskytuje jsou:

- `bool processKeyEvent(char key)`
- `std::string getInfo()`
- `void prepare(cCollisionEvent* collisionEvent, cVector3d intersectionCenter)`
- `void modify(cTransform cursorTransform)`
- `void finalize()`

Metoda `processKeyEvent` slouží k předání události stisku klávesy nástroji pro další zpracování. Díky ní může být každý nástroj ovládán jinými klávesami. Jediné omezení je, aby dané klávesy nekolidovaly s těmi, jež už má uživatelské rozhraní nadefinováno k vlastnímu ovládání (například vypnutí programu, či uložení objektů do souboru). Metoda `getInfo` slouží k zobrazení textové informace o daném stavu nástroje.

První metoda, která je spjata s manipulací objektů, se nazývá `prepare`. Je volána těsně poté, co program přejde do stavu modifikace. V jejím rámci většinou nástroje připravují svá data, aby následné modifikování bylo co nejplynulejší. Její parametry jsou událost kolize, která obsahuje informace jako trojúhelník, se kterým došlo ke kolizi, pole vrcholů, nebo trojúhelníků, a pozice bodu, ve kterém k dané kolizi došlo.

Metoda `modify` slouží k samotné modifikaci sítě, a to tak, že se vemou v podtaz předpřipravené hodnoty a nová transformace haptického kurzoru (pozice a natočení), a pomocí nich se na síť aplikují změny. Na tuto metodu je kladen požadavek, aby běžela co nejrychleji, neboť chceme, aby proces modifikace probíhal plynule.

Posledním krokem před přechodem zpět do stavu dotýkání je právě zavolání metody `finalize`. Zde se obvykle určuje, které vrcholy byly danou modifikací ovlivněny, a posléze je volána metoda `repair` třídy `Mesh` nad objektem reprezentujícím upravenou trojúhelníkovou síť.

8.3.1 Pohyb trojúhelníkem

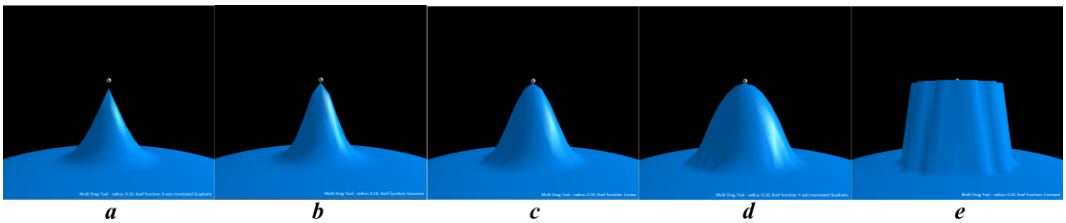
Jedná se o první nástroj, který jsem implementoval. V rámci přípravy získá trojúhelník, se kterým došlo ke kolizi a uloží si indexy jeho vrcholů, které nejsou statické. V rámci procesu modifikace pak vypočítává jejich pozice pomocí následujícího vzorce:

$$p_i = p_{i-1} + (x_i - x_{i-1}),$$

kde p_i je nová pozice vrcholu, p_{i-1} pozice vrcholu v předchozím kroku, c_i je současná pozice haptického kurzoru a x_{i-1} je jeho pozice v předchozím kroku.

8.3.2 Pohyb n vrcholy

Tento nástroj lze přizpůsobit dvěma parametry: poloměr (je v rozsahu $< 0, 1 >$), a funkcí pro výpočet koeficientů určujících, o kolik budou vrcholy posunuty. Původně jsem byl rozhodnut vybrat pouze jednu z funkcí ze sekce 7.1.2, ovšem došel jsem k názoru, že každá má své typické vlastnosti rozdílné od ostatních (viz obrázek 8.2) a uživateli je tak dána šance alespoň částečně ovlivnit, jakým způsobem budou vrcholy posouvány. V případě polynomiálních funkcích jsem stupeň polynomu určil 2, protože je to nejnižší celočíselný exponent větší než 1 (kdybych zvolil 1. stupeň, obě funkce by byly totožné).



Obrázek 8.2: Porovnání funkcí pro výpočet koeficientů. Funkce ***a*** je kvadratická funkce posunutá po ose x, ***b*** je Gaussovská funkce, ***c*** je kosinus, ***d*** je kvadratická funkce posunutá po ose y a ***e*** je konstantní funkce.

V rámci předpřípravy jsou pomocí rohové tabulky právě vybrané síťě nalezeny nestatické vrcholy, které jsou v zadaném poloměru od bodu kolize. K tomu je potřeba naleznout nejbližší vrchol na trojúhelníku. Následně jsou předvypočteny koeficienty a je možno začít s modifikací. Vzorec pro přepočet pozic je velmi podobný předchozí metodě, pouze je doplněn o koeficient:

$$p_i = p_{i-1} + k\,(x_i - x_{i-1})$$

$$k = f_r(\|p_0 - c\|),$$

kde k je daný koeficient, f_r je funkce pro jeho výpočet, p_0 je pozice daného vrcholu v počátečním stavu, c je bod, ve kterém došlo ke kolizi a $\|p_0 - c\|$ je eukleidovská vzdálenost[32] těchto dvou bodů.

Pokud je vybráno příliš mnoho vrcholů, může se stát, že proces modifikace nebude dostatečně rychlý a výsledkem bude zvláštní trhání zařízením. Je to způsobeno tím, že po dobu, kdy se přepočítávají pozice se na zařízení aplikuje ta samá síla, takže kurzor je protlačen skrz centrum pružiny až na druhou stranu. V té chvíli se přepočtou síly a na zařízení je působeno silou s opačným směrem. Výsledkem pak může být dokonce házení haptickým kurzorem ze strany na stranu. Tento problém se dá řešit jedině urychlením modifikace, nicméně vždy se dá najít množství vrcholů, které tuto nechtěnou vlastnost způsobí.

8.3.3 Laplaceovská editace povrchu

Implementace tohoto přístupu se nachází ve třídě `LaplacianDragTool`. Jelikož základem této metody je nalezení řešení soustavy lineárních rovnic, je využito externí knihovny s názvem *Eigen*[14], která je určena pro řešení úloh lineární algebry. Tuto knihovnu jsem vybral, jelikož byla již přiložena k *Chai3D*.

Prvním krokem přípravy je nalezení vrcholů, které se nacházejí v zadaném poloměru, podobně jako tomu bylo i u předchozího přístupu. Následně je potřeba najít všechny vrcholy, které sousedí s těmi modifikovanými, ale ony samotné se modifikovat nebudou. Poté určíme matici A. Nejdříve projdeme polem modifikovaných vrcholů a pro každý vrchol v nastavíme 1 na pozici $[i_v, j_v]$, kde i_v je index v poli všech vrcholů sítě, který odpovídá danému vrcholu, a j_v je index v poli modifikovaných vrcholů. Poté pro každý vrchol w, který sousedí s v nastavíme $-\frac{1}{n_w}$ na pozici $[i_w, j_v]$, kde n_w je počet přilehlých vrcholů w. Výsledná matice popisuje tzv. přeurčenou soustavu, a je proto nutné ještě určit matice A^T a A^TA, abychom pak mohli řešit $A^TAx = A^Tb$.

Dále je potřeba určit pravou stranu. Ta se skládá ze dvou částí - první obsahuje informace spjaté s nemodifikovanými vrcholy, druhá informace o vrcholu, se kterým bude pohybováno a jeho pozice je parametrem této soustavy. Je třeba říct, že levá strana je stejná jak pro x-ové, tak i pro y-ové a z-ové souřadnice bodů. Pravá strana bohužel stejná není, a proto je třeba sestavit její tři různé verze. Uvedu zde postup jenom pro x-ové souřadnice, pro ostatní je podobný. Nejdříve nastavím $b_x[v] = \delta_x[v]$ pro všechny modifikované vrcholy v. Poté nastavím $b_x[w] = \delta_x[w] - p_x[w]$ pro všechny nemodifikované vrcholy w, které sousedí s modifikovanými, kde $p_x[w]$ je jejich x-ová souřadnice. Dalším krokem je pro každý vrchol u, který sousedí s w udělat následující: Pokud tento vrchol není modifikovaný, přičteme k $b_x[w]$ hodnotu $\frac{1}{n_w}p_x[u]$. Pokud patří mezi modifikované, musíme naopak přičíst k $b_x[u]$ hodnotu $\frac{1}{n_u}p_x[w]$.

Dále je potřeba naplnit vektor s, který bude odpovídat vrcholu, kterým bude taháno. Ten se konstruuje velice jednoduše: Nastavíme -1 na řádce, která tomuto vrcholu odpovídá, a poté na všech řádkách odpovídajících jeho sousedům vložíme $\frac{1}{n}$, kde n je počet sousedů těchto vrcholů. Posledním krokem pak je přičíst Laplaceovské souřadnice k daným b vektorům na řádce, která odpovídá tahanému vrcholu.

Matice A^TA je symetrická, pozitivně definitní a řídká, program tedy pro řešení soustav využívá instanci třídy `SimplicialLLT` s názvem `solver`, která nad danou maticí provede tzv. *Choleského rozklad*. Jelikož, jak již bylo zmíněno, se levá strana soustavy nemění, je tento rozklad proveden v rámci předpříprav s výpočetní složitosí $O(n^3)$, kde n je počet řádek v matici A^TA, poté již jsou pouze dosazovány různé pravé strany a nalezení řešení má složitost $O(n^2)$.

V rámci modifikace se tedy děje následovné: Nejdříve je posunut tahaný bod: $v_i = v_{i-1} + (x_i - x_{i-1})$, poté určíme pravé strany pro všechny osy (pro zjednodušení znovu uvedu pouze výpočet pro osu x): $r_x = A^T b_x + A^T(v_{xi}s)$. Měnit se bude pouze $A^T(v_{xi}s)$, zbytek je možno vypočítat již v předpřípravě. Pro každou osu pak vyřešíme rovnice pomocí metody `solver.solve()` a výsledkem budou vypočtené nové souřadnice modifikovaných bodů.

Důležité je po provedené modifikaci znovu přepočíst Laplaceovské souřadnice, neboť kvůli užití metody nejmenších čtverců se jejich hodnoty změnily, byť jen nepatrně. Pokud bychom tak neučinili, můžeme při další modifikaci přijít o vzniklý modifikovaný tvar. Například pokud bychom měli lidskou sochu, při první modifikaci bychom zvedli její pravou ruku a při druhé bychom celým modelem posunuli byť jen o malý kus stranou, postava by se přesunula na své místo, ovšem její ruka by byla znovu dole.

Stejně jako metoda posunu n vrcholy, i Laplaceovská editace povrchu trpí na trhání haptickým perem, pokud je počet modifikovaných vrcholů příliš veliký.

8.3.4 Transformační nástroj

Tento nástroj přijde vhod, pokud potřebujeme modifikovaný objekt posunout po prostoru aplikace, či s ním libovolně rotovat. Jako jediný z implementovaných nástrojů pohlíží na objekty jako celky, jelikož upravuje pouze transformační matici daného objektu. Ta určuje jeho polohu v rámci daného prostředí a bývá aplikována na vrcholy, pokud potřebujeme určit jejich globální souřadnice (například při detekci kolizí).

Modifikace probíhá takto: Nechť A je transformační matice objektu, a B transformační matice haptického kurzoru. Podobu A v i-tém kroce manipulace lze vypočítat ze vzorce:

$$A_i = (A_0 \cdot B_0^{-1})B_i$$

Z daného vztahu lze vidět, že pokud se zařízení vrátí do své výchozí polohy, bude B_i rovno B_0, což způsobí, že se $B_0^{-1}B_i$ vykrátí, a i výsledná matice objektu bude totožná s tou výchozí.

8.3.5 Výběr statických/modifikovatelných vrcholů

Společně s nástrojem pro Laplaceovské vyhlazování (viz podkapitolu 8.3.6) provádějí veškerou modifikaci již v metodě `prepare`, neboť narozdíl od zbytku vyžadují pouze informaci o haptické kolizi, ne o tom, jakým směrem se zařízení pohybovalo následně. Proto je jako poslední krok této metody změněn stav uživatelského rozhraní z modifikace zpět na ohmatávání objektů. I když se v rámci aplikace jeví jako dva různé typy nástrojů (*Statix Vertex Select Tool* a *Movable Vertex Select Tool*), z pohledu implementace je to pouze jeden, a daný rozdíl je reprezentován příznakem `selectMovable`.

Nedělá nic víc, než že vezme všechny tři vrcholy trojúhelníka, se kterým došlo ke kolizi, označí je jako statické (resp. modifikovatelné) a obarví je podle jejich nového stavu. Geometrie, ani topologie sítě nejsou tímto nástrojem nijak ovlivněny.

8.3.6 Laplaceovské vyhlazování

Ze všech metod pro vyhlazování uvedených v podkapitole 7.1.6, jsem se nakonec rozhodl implementovat tu nejjednodušší, neboť je kladen důraz na co největší interaktivitu procesu a její nedostatky nás až tolik netrápí. Uživatel může ovlivnit to, jaká část objektu bude modifikována, pomocí poloměru určujícím maximální vzdálenost od bodu kolize se zařízením, jaké mohou vrcholy dosáhnout, aby byly vybrány. V základní verzi tohoto algoritmu se počítá s tím, že jsou koeficienty posunu stejné pro všechny vybrané vrcholy. Pro zvýšení intuitivnosti procesu jsem se však rozhodl použít podobného principu jako u metody posunu n vrcholů a určovat tyto koeficienty pomocí jejich vzdálenosti.

V rámci úpravy jsou nejdříve nalezeny všechny modifikovatelné vrcholy v zadaném poloměru, a poté jsou určeny koeficienty posunu λ. Pro jejich výpočet jsem použil fukci f_3 zmíněnou v kapitole 7.1.2 (kvadratický polynom posunutý po ose y) s tím, že jsem ji vynásobil konstantou 0.25. Požadavek na tuto konstantu byl, aby se nacházela v intervalu $(0, 0.5)$. Čím menší je, tím méně se daný objekt smršťuje, ovšem samotné vyhlazování je velice zdlouhavý proces. Proto jsem se rozhodl pro kompromis a zvolil přesnou polovinu tohoto intervalu. Jakmile jsou vypočteny koeficienty, můžeme modifikované vrcholy posunout pomocí následujícího vzorce:

$$p_i = p_{i-1} - \lambda\delta_{i-1},$$

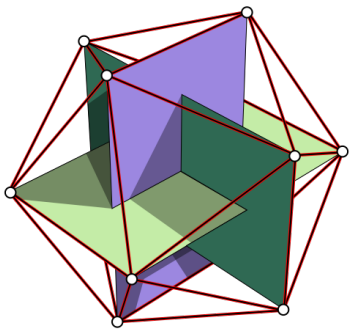
kde p_i je nová pozice vrcholu, p_{i-1} jeho předchozí, a δ_{i-1} je Laplaceovská souřadnice před modifikací. Index u Laplaceovské souřadnice napovídá, že je bude potřeba po každém kroku této modifikace znovu přepočítat.

8.3.7 Vkládání materiálu

Tento nástroj se od ostatních liší tím, že jako jediný není potomkem třídy `Tool`. Důvodem je, že pro vkládání materiálu není třeba dotyku s objektem. V důsledku toho má sice třída `MatetialInsertTool`, ve které je tento nástroj implementován, stejně pojmenované metody, ovšem předávané parametry jsou rozdílné:

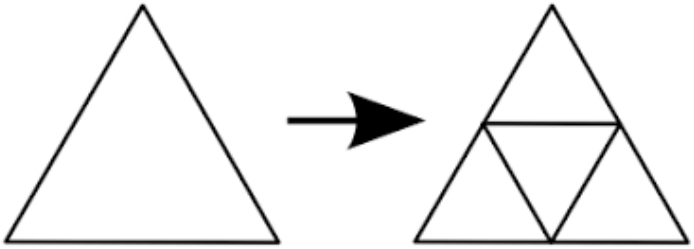
- `void prepare(cVector3d center)`
- `void modify(cVector3d cursorPosition)`
- `void finalize()`

V metodě `prepare` je vytvořena koule, kterou budeme do objektu vkládat. Postup její konstrukce jsem implementoval dle [16]. Prvním krokem je vygenerování pravidelného dvacetistěnu. Pokud je délka jeho hrany rovna 2, dají se polohy jeho vrcholů popsat pomocí cyklických permutací trojice $(0, \pm 1, \pm\phi)$, kde ϕ je zlatý řez, jelikož jsou to také vrcholy tří na sebe kolmých zlatých obdélníků[33]. Takto vytvořené body je ještě potřeba posunout tak, aby jejich spojnice se středem byla velikosti 1, ale měla stejný směr (neboli vektory, které popisují jejich polohy je potřeba znormalizovat).



Obrázek 8.3: Znázornění konstrukce dvacetistěnu pomocí tří zlatých obdélníků. Zdroj [33]

Následně je nutné všechny jeho trojúhelníky rozdělit na čtyři stejně velké (viz obrázek), a vrcholy vzniklé touto operací posunout stejně, jako jsme provedli v předchozím kroku. Toto dělení opakujeme, dokud výsledný objekt není dostatečnou aproximací jednotkové koule.



Obrázek 8.4: Rozdělení trojúhelníků na čtyři stejně veliké části. Zdroj [16]

V rámci metody `modify` je prováděno pouze škálování koule tak, aby byl její poloměr roven vzdálenosti mezi původní pozicí haptického kurzoru a jeho současnou.

Posledním krokem je překopírování vrcholů a trojúhelníků koule do právě modifikovaného objektu a přepočtení informací. Pokud ještě nebyl načten žádný objekt, vytvoří se nový, což uživateli umožňuje jednoduše navrhovat jednoduché předměty tím, že danou kouli začne různě deformovat.

8.4 Oprava děr

Tato funkční část se nachází ve složce `fixer`. Autorkou celé této části je slečna *Ing. Věra Skorkovská*, která se tímto problémem zabývala ve své rigorózní práci [25]. Z pohledu této aplikace je nejdůležitější třída `MeshFixer` a zejména pak její metoda `repairMesh(cMesh *mesh, CornerTable **ct, unordered_set<int> affectedTriangles)`, která je volána na konci modifikace. Jako parametry se předávají upravená trojúhelníková síť, která může obsahovat sebeprotnutí, její rohová tabulka, a množina trojúhelníků, které byly danou modifikací ovlivněny.

Všechny ovlivněné trojúhelníky jsou zkontrolovány, zda nezpůsobují sebeprotnutí a pokud ano, je vždy vytvořena jedna díra. To znamená, že je potřeba tuto metodu volat, dokud je stále co opravovat. Přístup slečny Skorkovské je lokální, což znamená, že při úpravě je ovlivněna pouze ta část sítě, které se oprava opravdu týká. Proces vytváření děr je urychlen tím, že pokud je nalezeno první sebeprotnutí trojúhelníků, další prohledávání sítí probíhá v okolí tohoto místa, čehož je docíleno užitím předané rohové tabulky. Po nalezení všech trojúhelníků, které utvářejí právě opravovanou díru, jsou tyto trojúhelníky smazány a triangulací je docíleno obnovení manifoldnosti sítě.

Bohužel, tento přístup trpí na numerické nepřesnosti, což občas způsobuje, že je objekt opraven špatně. Například, jednou za čas se stalo, že modifikovaná trojúhelníková síť zcela zmizela. Někdy oprava dokonce způsobí pád programu. Rozhodl jsem se tedy opravu trojúhelníkových sítí uvést pouze jako experimentální vlastnost programu, která se dá povolit, pokud se při kompilaci definuje konstanta `MESH_REPAIR`. Pro zjednodušení jsem vytvořil speciální *Release* konfiguraci projektu Visual Studia s názvem *Release MESH_REPAIR*, jíž kompilací lze vytvořit program, který má tuto funkcionalitu povolenu.

9 Experimenty

V rámci této práce jsem provedl dvoje měření. První proběhlo již v minulém semestru v rámci *KIV/PRJ5* a porovnával jsem v něm výkon dostupných knihoven a datových struktur pomocí FPS grafické smyčky. V druhém měření jsem porovnal mezi sebou mnou implementované metody manipulace trojúhelníkových sítí. Obě měření proběhla v haptické laboratoři na Katedře informatiky na stroji s 64 bitovým procesorem Intel Core 2 Duo E6600, 4 GB RAM a grafickou kartou NVIDIA GeForce GTX 295.

9.1 Porovnání knihoven a struktur

Testovanou veličinou bylo průměrné FPS grafiky dané aplikace. Jelikož u implementace pomocí HLAPI je velký rozdíl mezi rychlostí při normálním běhu a při simulaci pružiny, rozhodl jsem se hodnotit tyto části běhu odděleně.

Nejdříve jsem mezi sebou porovnal implementace trojúhelníkových sítí (HLAPI vs. Chai3D).

Počet trojúhelníků	Počet vrcholů	FPS_{HLAPI}	FPS_{Chai3D}
4422	2213	57,9960	58,0299
69666	34835	6,94731	9,9721
212574	106289	—	2,7161

Tabulka 9.1: Porovnání FPS při obyčejném ohmatávání.

Počet trojúhelníků	Počet vrcholů	FPS_{HLAPI}	FPS_{Chai3D}
4422	2213	58,0669	58,7764
69666	34835	41,8079	10,7906
212574	106289	—	3,2452

Tabulka 9.2: Porovnání FPS při simulaci pružiny.

V rámci porovnání knihoven, si o trochu lépe vedla knihovna Chai3D, ovšem rozdíl není tak výrazný. Výsledky HLAPI s nejdetailnějším modelem nejsou uvedeny, jelikož při měření nefungovalo haptické renderování (nakonec se ukázalo, že to bylo způsobeno implicitně nastaveným maximálním počtem vykreslovaných vrcholů, a že tato hodnota lze navýšit).

Dále jsem změřil FPS u implementace pomocí voxelů. FPS_{ob} znamená FPS při obyčejném ohmatávání, FPS_{pr} znamená FPS při simulaci pružiny.

Rozlišení mrížky	FPS_{ob}	FPS_{pr}
$64 \times 64 \times 64$	58,2347	57,7830
$128 \times 128 \times 128$	57,3902	51,4356
$256 \times 256 \times 256$	50,0036	56,0002
$512 \times 512 \times 512$	23,9234	34,1777

Tabulka 9.3: Naměřené hodnoty FPS pro voxelovou implementaci

Měření ukázalo, že voxelová reprezentace dosahuje o něco horších výsledků, než trojúhelníková (předpokládáme podobnost detailu u voxelové mřížky $256 \times 256 \times 256$ a nejméně detailní trojúhelníkovou sítí). Nutno podotknout, že kvůli jednoduchosti implementace (žádná optimalizace paměti) se detailnější voxelové objekty nepodařilo zobrazit (program selhal na nedostatek paměti).

Naměřené FPS jsou v rámci grafické renderovací smyčky. Haptickou smyčku nebylo možno změřit, jelikož v *HLAPI* jsem k ní kvůli uzavřenosti kódu neměl přístup. Nicméně rychlost grafiky s haptikou úzce souvisí.

Výsledky porovnání knihoven mne velmi překvapily, jelikož jsem očekával, že knihovna *Chai3D* bude pomalejší, kvůli její vyšší funkčnosti. Další výsledky již byly dle očekávání - Voxely byly opravdu náročné na paměť, simulace pružiny byla rychlejší než obyčejné ohmatávání.

9.2 Porovnání implementovaných nástrojů

Tentokrát nebylo měřeno FPS, ale doba výpočtu. Objekt, na kterém byly testy provedeny byla koule vygenerovaná *nástrojem pro vkládání materiálu*, ovšem byla speciálně vytvořena s vyšším detailem (163842 vrcholů, 327680 trojúhelníků). Porovnávané byly tyto přístupy:

- *Posun n vrcholy*
- *Laplaceovská editace povrchu*
- *Laplaceovské vyhlazování*

Zbylé přístupy byly z experimentu vyřazeny záměrně - *posun trojúhelníku* a *transformační nástroj* kvůli své jednoduchosti, *nástroj pro výběr statických vrcholů* kvůli tomu, že danou síť geometricky nijak neupravuje a *vkládání materiálu* kvůli zcela rozdílnému přístupu modifikace.

Nejdříve byly porovnány časy výpočtů metody `prepare`. Množství zpracovávaných vrcholů bylo určeno pomocí poloměru, kterým se parametrizují všechny testované přístupy.

r	Počet vrcholů	$t_{n\_vrcholu}$ $[ms]$	$t_{lap\_editace}$ $[ms]$	t_{lap_vyhl} $[ms]$
0,05	1901	7,460	43,804	828,546
0,1	7146	30,348	322,010	867,097
0,3	60086	346,943	13889,789	1316,414

Tabulka 9.4: Porovnání časů výpočtu metod `prepare`

Nejlépe si vedla metoda *Posunu n vrcholy*, což bylo zcela očekávané, jelikož kromě nalezení modifikovaných bodů a výpočtu koeficientů nic nedělá. Výsledkově naprosto propadla metoda *Laplaceovské editace povrchu*, která pro největší testovaný poloměr dosáhla dokonce necelých čtrnácti vteřin. Nicméně i to se dalo očekávat, jelikož se v rámci přípravy konstruuje matice s přibližně 60000 sloupci a 160000 řádky. I tento výsledek ovšem není úplně špatný, jelikož takto proběhne pouze jednou v rámci jedné modifikace. V případě *Laplaceovského vyhlazování*, je ovšem dosažený výsledek dosti nevyhovující, jelikož narozdíl od ostatních nástrojů, se metoda `prepare` volá pravidelně. Je to způsobeno tím, že abychom mohli detekovat novou kolizi zařízení, musí se pokaždé přepočítat kolizní strom (a to i nad vrcholy, které nebyly vůbec ovlivněny modifikací). Ostatně, pokud bychom od výsledného času odečetli tento postup, dosáhl by tento nástroj podobných výsledků jako *Posun n vrcholy*.

Dále byl měřen výpočetní čas metody `modify`. Tentoktáte již byly porovnány pouze první dva nástroje, jelikož *Laplaceovské vyhlazování* vykonává veškerou modifikaci už v metodě `prepare`.

r	Počet vrcholů	$t_{n\_vrcholu}$ $[ms]$	$t_{lap\_editace}$ $[ms]$
0,05	1901	0,035	2,313
0,1	7146	0,145	16,885
0,3	60086	1,562	231,767

Tabulka 9.5: Porovnání časů výpočtu metod `modify`

Z výsledků lze vidět, že metoda *Posunu n vrcholů* si vedla výborně, a i pro největší měřený počet vrcholů je stále blízko splnění požadavku na 1 KHz haptické odezvy pro pocit plynulé modifikace. Metoda *Laplaceovské editace povrchu* na tom byla znovu hůře kvůli výpočtu soustav lineárních rovnic, a pro největší měřený počet vrcholů by zcela určitě bylo cítit trhání haptickým perem. Nicméně zbylé hodnoty jsou ještě docela v mezích normy, jelikož při modifikaci není simulován dotyk zařízení s objektem, ale tah pružiny, který nemá tak vysoké nároky na rychlost haptické odezvy.

Porovnání výpočetních časů metody `finalize` nebylo třeba, jelikož oba nástroje v jejím rámci vykonávají stejné - aktualizace dat o objektu. Pro testovaný objekt trvala tato metoda v průměru přibližně $850ms$, což je i přibližně hodnota, o kterou se v první části měření lišilo *Laplaceovské vyhlazování*.

10 Závěr

V rámci této práce jsem se seznámil se čtyřmi knihovnami pro práci s haptickými zařízeními, pochopil základní principy haptického renderování a dostal povědomí nejčastěji používaných struktur a algoritmů. Dále jsem pochopil základy práce s *OpenGL* a vývojovým prostředím *Microsoft Visual Studio*. S pomocí knihovny *Chai3D* jsem navrhl a implementoval šest různých nástrojů modifikace objektů (a jeden pomocný), z nichž jsem tři podrobil testům.

Nejlépe si v měřeních vedl nástroj *Posunu n vrcholů*, který je sice jednoduchý, zato ale velmi mocný. I když si zbylé dva nástroje vedly o poznání hůře, stále si myslím, že je jejich užití velice intuitivní, a že je lze bez žádných limitací dále používat.

Co se týče nedostatků výsledného programu, musím vyjmenovat dvě věci - opravu děr v síti a přepočet celého kolizního stromu objektu po sebemenší změně geometrie. První problém není dle mého názoru trvalý, a myslím si, že je jen otázka času doladit tuto funkcionalitu tak, aby nemusela být označena za experimentální. Přepočet kolizního stromu však znamená velké zpomalení běhu programu a do budoucna by bylo nejspíše třeba naimplementovat zcela jiný přístup pro detekci kolizí. Nabízí se například implementace procházkového algoritmu pana Soukala [27] využívající **Rohovou tabulku**, která je už v této práci implementována. Další možností, jak programu dodat pocit plynulejší editace objektů, by bylo přesunutí modifikační logiky do samostatného vlákna (v současné době se nachází v haptickém vlákně).

Literatura

[1] *Chai3D - About* [online]. 2015. [cit. 13.12.2015]. Dostupné z: `http://www.chai3d.org/concept/about`.

[2] *Open Source Haptics - H3D.org* [online]. 2012. [cit. 13.12.2015]. Dostupné z: `http://www.h3dapi.org/`.

[3] *Mesh Processing (Advanced Methods in Computer Graphics) Part 1* [online]. [cit. 16.4.2016]. Dostupné z: `http://what-when-how.com/advanced-methods-in-computer-graphics/mesh-processing-advanced-methods-in-computer-graphics-part-1/`.

[4] *OpenHaptics Toolkit Overview* [online]. 2015. [cit. 13.12.2015]. Dostupné z: `http://www.geomagic.com/en/products/open-haptics/overview/`.

[5] *Calculating a Surface Normal* [online]. 2013. [cit. 14.12.2015]. Dostupné z: `https://www.opengl.org/wiki/Calculating_a_Surface_Normal/`.

[6] Baumgart, B. G. Winged edge polyhedron representation. Technical report, DTIC Document, 1972.

[7] Bergen, G. v. d. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*. 1997, 2, 4, s. 1–13.

[8] Bischoff, S. – Kobbelt, L. Structure preserving CAD model repair. In *Computer Graphics Forum*, 24, s. 527–536. Wiley Online Library, 2005.

[9] Botsch, M. et al. Geometric modeling based on triangle meshes. In *ACM SIGGRAPH 2006 Courses*, s. 1. ACM, 2006.

[10] Campen, M. – Kobbelt, L. Exact and Robust (Self-) Intersections for Polygonal Meshes. In *Computer Graphics Forum*, 29, s. 397–406. Wiley Online Library, 2010.

[11] Conti, F. The CHAI libraries. Technical report, Dublin, Ireland, 2003.

[12] Desbrun, M. et al. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, s. 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[13] Foley, J. – Dam, A. v. – Feiner, S. Computer graphics: principles and practice. *Systems programming series Show all parts in this series*. 1990.

[14] Guennebaud, G. – Jacob, B. – others. Eigen v3. http://eigen.tuxfamily.org, 2010.

[15] Herrmann, L. R. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*. 1976, 102, 5, s. 749–907.

[16] Kahler, A. *Creating an icosphere mesh in code* [online]. 2009. [cit. 23.4.2016]. Dostupné z: `http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html`.

[17] Lorensen, W. E. – Cline, H. E. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, 21, s. 163–169. ACM, 1987.

[18] McGuire, M. The half-edge data structure. *Website: http://www. flipcode. com/articles/article halfedgepf. shtml*. 2000.

[19] Meagher, D. J. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

[20] Nooruddin, F. S. – Turk, G. Simplification and repair of polygonal models using volumetric techniques. *Visualization and Computer Graphics, IEEE Transactions on*. 2003, 9, 2, s. 191–205.

[21] Rossignac, J. 3D compression made simple: Edgebreaker with ZipandWrap on a corner-table. In *Shape Modeling and Applications, SMI 2001 International Conference on.*, s. 278–283. IEEE, 2001.

[22] Ruspini, D. C. – Khatib, O. The haptic display of complex graphical environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, s. 345–352. ACM Press/Addison-Wesley Publishing Co., 1997.

[23] Salisbury, K. – Conti, F. – Barbagli, F. Haptic rendering: introductory concepts. *Computer Graphics and Applications, IEEE*. 2004, 24, 2, s. 24–32.

[24] Sjöberg, H. – Ylinenpää, O. Collision Detection for Haptic Rendering, 2009.

[25] Skorkovská, V. Modeling of Erosion Impact on Geometric Objects. Technical Report DCSE/TR-2015-06, University of West Bohemia in Pilsen, Department of Computer Science and Engineering, 10 2015.

[26] Sorkine, O. – Cohen-Or, D. – Lipman, Y. Laplacian Surface Editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, s. 175–184, New York, NY, USA, 2004. ACM. doi: 10.1145/1057432.1057456. Dostupné z: http://doi.acm.org/10.1145/1057432.1057456. ISBN 3-905673-13-4.

[27] Soukal, R. – Purchart, V. – Kolingerová, I. Surface point location by walking algorithm for haptic visualization of triangulated 3D models. *Advances in Engineering Software*. 2014, 75, s. 58–67.

[28] Sunday, D. *Intersection of Rays, Planes and Triangles (3D)* [online]. 2012. [cit. 15.12.2015]. Dostupné z: http://geomalgorithms.com/a06-_intersect-2.html.

[29] Taubin, G. Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, s. 852–857. IEEE, 1995.

[30] Urquhart, D. *OpenGL Buffering* [online]. 2010. [cit. 19.4.2016]. Dostupné z: http://www.swiftless.com/tutorials/opengl/smooth_rotation.html.

[31] Wikipedia. *Cholesky decomposition — Wikipedia, The Free Encyclopedia* [online]. 2016. [cit. 19.4.2016]. Dostupné z: https://en.wikipedia.org/wiki/Cholesky_decomposition.

[32] Wikipedia. *Euclidean distance — Wikipedia, The Free Encyclopedia* [online]. 2016. [cit. 18.4.2016]. Dostupné z: https://en.wikipedia.org/wiki/Euclidean_distance.

[33] Wikipedia. *Regular icosahedron — Wikipedia, The Free Encyclopedia* [online]. 2016. [cit. 23.4.2016]. Dostupné z: https://en.wikipedia.org/wiki/Regular_icosahedron.

[34] Wikipedia. *Wavefront .obj file — Wikipedia, The Free Encyclopedia* [online]. 2015. [cit. 19.4.2016]. Dostupné z: https://en.wikipedia.org/wiki/Wavefront_.obj_file.

[35] Zaharescu, A. – Boyer, E. – Horaud, R. Topology-adaptive mesh deformation for surface evolution, morphing, and multiview reconstruction. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*. 2011, 33, 4, s. 823–837.

[36] Zilles, C. B. – Salisbury, J. K. A constraint-based god-object method for haptic display. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, 3, s. 146–151. IEEE, 1995.