VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

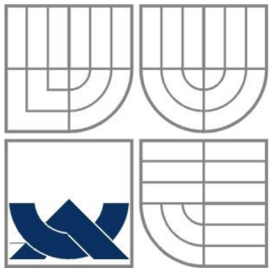
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE ZUBŮ NA 3D POČÍTAČOVÉM POLYGONÁLNÍM MODELU ČELISTI

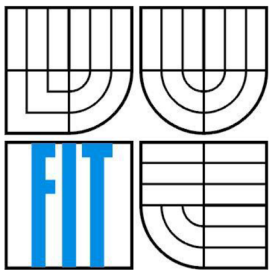
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Hulík Rostislav
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE ZUBŮ NA 3D POČÍTAČOVÉM POLYGONÁLNÍM MODELU ČELISTI

TOOTH DETECTION ON JAW 3D COMPUTER POLYGONAL MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Hulík Rostislav
AUTHOR

VEDOUCÍ PRÁCE — Kršek Přemysl, Ing., Ph.D.
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce se zabývá návrhem a implementací algoritmů pro detekci zubů na 3D polygonálním modelu čelisti. Postupně zde popisuji možné způsoby detekce tvarů na počítačovém modelu, včetně jejich využití na konkrétním 3D modelu čelisti. Dále zde probírám problematiku filtrace takto získaných segmentů, aby byly spolehlivě odděleny regiony zubního oblouku a zbytku čelisti. Všechny tyto poznatky poté použiji pro tvorbu cílové aplikace. Zde je možnost výběru ze dvou uživatelských nabídek, jedné jako jednoduchého uživatelského prostředí, kde demonstruji možné použití v praxi, druhé pak testovací, která umožňuje demonstraci použitých filtrů a jejich vliv na detekci. Program je psaný v jazyce C++ ve vývojovém prostředí MS Visual Studio 2005 a přeložen pod jeho interním kompilátorem. Testován byl pod OS MS Windows XP a MS Windows Vista.

Klíčová slova

detekce, zuby, 3D, model, polygonální, čelist, křivost, střední, Gaussova, minimální, maximální, vyhlazení, Laplace, Watershed, filtr, MDSTk, Open Scene Graph, MS Visual Studio 2005

Abstract

This document discusses a concept and implementation of algorithms for teeth detection on 3D polygonal jaw model. At first, I describe possible shape detection techniques on three dimensional computer model, including utilization just in my case. After it, I analyze filtration problems of obtained model segments in order to reliably separate regions of tooth-arch from the rest of jaw. I included all these pieces of knowledge into a final application. Here, it is possible to choose from two user interfaces, first, classic user interface which demonstrates possible utilization in real world and second, testing interface which demonstrates functions of each detection and filtration method used in program and their effect on detection. Application is written in C++ language in development environment of MS Visual Studio 2005 and compiled with MS VS internal compiler. Tested was under MS Windows XP and MS Windows Vista OS.

Keywords

Detection, tooth, teeth, 3D, model, polygonal, jaw, curvature, mean, Gauss, minimal, maximal, smoothing, Laplace, watershed, filtration, MDSTk, Open Scene Graph, MS Visual Studio 2005

Citace

Hulík Rostislav: Detekce zubů na 3D počítačovém polygonálním modelu čelisti. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Detekce zubů na 3D počítačovém polygonálním modelu čelisti

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Přemysla Krška, Ph.D., a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Rostislav Hulík
13. 4. 2008

Poděkování

Zde bych chtěl poděkovat panu Ing. Přemyslu Krškovi, Ph.D. za jeho výborné vedení, ochotu, připomínky a odbornou pomoc při řešení této problematiky. Dále děkuji panu Ing. Michalu Španělovi za jeho připomínky k práci, které řešení této práce značně urychlily.

Obsah

Úvod

V této práci se budu postupně zabývat návrhem algoritmu pro detekci zubů na počítačovém modelu čelisti. Tento postup může být užitečný například ve stomatologii, kde je program schopný z naskenovaného sádrového modelu uloženého v stl formátu graficky abstrahovat zubní oblouk a výrazně tak urychlit orientaci na modelu. I když se zde zabývám konkrétně detekcí zubů, navrhované postupy s jinými parametry je možné použít i v kterémkoli jiném odvětví detekce tvarů na 3D modelu.

V první kapitole seznamuji s cílem této práce, dále pak popisuji možné řešení z teoretického hlediska, tzn. popisuji zde strukturu 3D polygonálních modelů, styl jejich uložení v paměti a možnou práci s nimi. Dále se zde zabývám možností detekce tvarů na hraničním modelu, ve kterém nejsou použitelné informace o složení jednotlivých částí. Jako poslední předkládám způsob vyhlazení modelu, který bude potřebný v dalších výpočtech této práce (viz druhá kapitola).

V druhé kapitole poté řeším možnosti použití postupů z předchozí kapitoly v praxi, konkrétně v mém případě detekci zubů na modelu, tzn. způsoby detekce a rozdělení modelu na jednotlivé segmenty a následné experimentování s jejich filtracemi. Nakonec předkládám návrh automatického algoritmu pro detekci zubů.

V předposlední kapitole rozebírám konkrétní implementaci programu. To znamená objektový návrh aplikace spolu s detailním popisem všech důležitých funkcí použitých při samotném výpočtu. Nakonec zde zveřejním výsledky jednotlivých experimentů s detekcí zubního oblouku včetně demonstračních obrázků.

1 Rozbor problematiky

Tato kapitola podrobně rozebírá teoretický základ všech technik použitých při řešení detekce zubů na polygonálním modelu čelisti.

Popisuji zde termíny jako polygonální model, křivost povrchu, možnosti využití této křivosti aj. Jako poslední věc zde zmiňuji vyhlazení modelu, které bude nutné k přesnější detekci oblouku.

1.1 Cíl práce

Hlavním cílem této práce je detekovat jednotlivé zuby (případně zubní oblouk) na 3D polygonálním modelu čelisti s následným uložením těchto informací do externího souboru.

1.2 Modelování objektů

1.2.1 3D model

3D počítačový model je matematická reprezentace trojrozměrného objektu. Ta může být následně vykreslena na dvourozměrnou plochu pro zobrazení (např. na monitoru PC), nebo použita pro simulační výpočty, kde zobrazení ani nemusí být vyžadováno. Pro popis modelů se využívá více metod rozdílných ve vlastním způsobu popisu modelu. Patří mezi ně konstruktivní geometrie, šablonování, dekompoziční přístup, hraniční reprezentace a implicitní plochy [1].

1.2.1.1 Konstruktivní geometrie

Konstruktivní geometrie využívá k popisu modelu informace o primitivech, ze kterých je daný model složen. Mezi tato primitiva se řadí základní a lehko matematicky popisovatelné objekty, jako krychle, koule, jehlany, elipsoidy aj. Dále jsou u modelu definované booleovské vztahy, které platí mezi jednotlivými primitivy.

1.2.1.2 Šablonování

Tento způsob modelování těží z invariance spline křivek a ploch vůči lineárním transformacím. Samotný způsob modelování totiž využívá pohybu dvojrozměrného profilu po určité dráze, který ve výsledku tvoří požadovaný objekt. Šablonování se též hodí pro tvorbu primitiv v konstruktivní geometrii.

1.2.1.3 Dekompoziční modely

Pro modelování dekompozičním způsobem byl zaveden speciální termín pro základní stavební jednotku - voxel. Voxel, neboli volume pixel, zastupuje stejnou roli ve 3D grafice jako jeho

dvojrozměrný předek. Model je tedy složen ze základních stavebních jednotek, a to v celém svém objemu. Na rozdíl od předchozích způsobů modelování je zde možné uchovávat informaci o složení celého modelu (hustota aj.), ne pouze jeho hraničních oblastí.

1.2.1.4 Implicitní plochy

Popis modelu pomocí implicitních ploch se skládá z kostry a potenciálního pole okolo každého jejího bodu. To následně odděluje vnitřní prostor od vnějšího. K jednotlivému ovlivňování potenciálních polí se využívá konkrétní směšovací funkce (v dnešní době většinou konečná).

1.2.1.5 Hraniční reprezentace

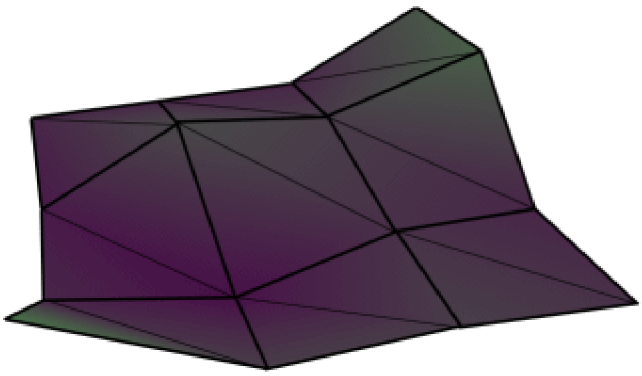
Hraniční reprezentace je posledním způsobem, kterým se zde budu zabývat, ale naopak jediným, který následně více rozvedu, protože se do něho řadí i polygonální modelování použité v mé práci. Při této reprezentaci se k popisu modelu využívá jen popis jeho povrchu. To znamená, že nemáme žádné informace o složení vnitřních částí modelu. Jediná naše znalost je právě tvar povrchu, který může být popsán například drátovým, polygonálním, nebo spline modelem.

1.2.2 Polygonální modelování

Jak jsem se již zmiňoval v předchozí kapitole, polygonální modelování využívá pro popis objektů definici tvaru jeho povrchu. Základní stavební jednotkou každého takového modelu je vrchol (vertex) s informací o umístění v prostoru, případně dalšími atributy. Spojením dvou vrcholů rovnou úsečkou vznikne hrana. S těmito informacemi jsme již schopni sestavit drátový model.

Pro polygonální modelování však musíme pokračovat ještě dále. Spojením tří hran nám vznikne nejjednodušší polygon Eukleidovského prostoru - trojúhelník. Samozřejmě u polygonální reprezentace je možné sestavovat libovolné mnohoúhelníky, v praxi se však používají většinou trojúhelníky či čtyřúhelníky.

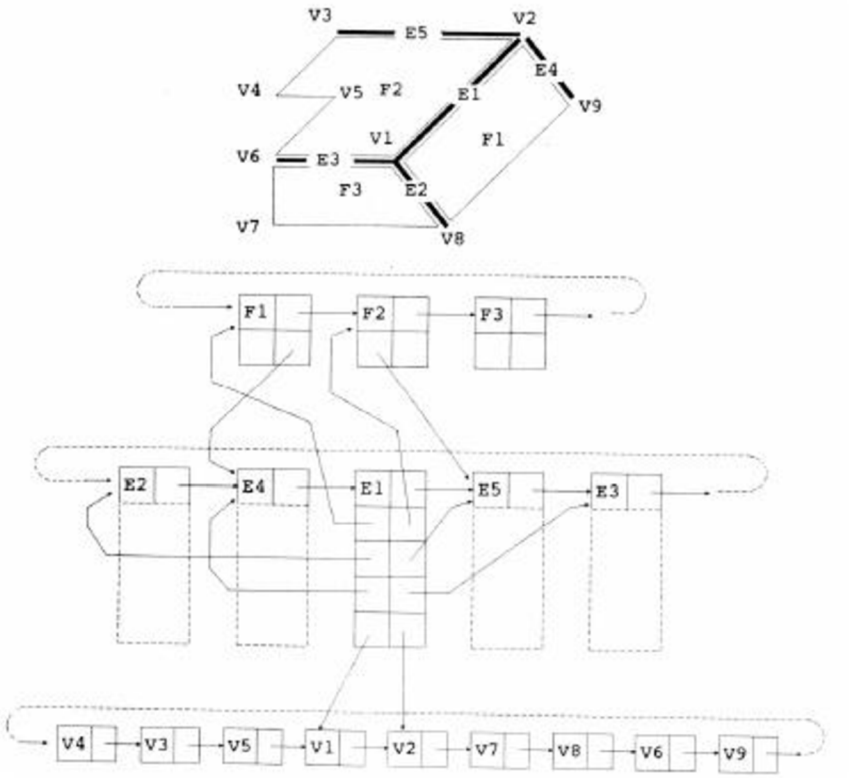
Zde bych však vyzvedl důležité vlastnosti trojúhelníků pro polygonální modelování. Všechny tři body trojúhelníku (jako jediného mnohoúhelníku) vždy leží v jedné rovině, což je důležitá informace pro operace s modelem, pro jeho zobrazování, stínování aj. A vědomost, že z trojúhelníku lze vytvořit jakýkoli jiný mnohoúhelník, z něj dělá ideální útvar pro polygonální modelování.



Obrázek 1.2.1 – Hraniční reprezentace pomocí trojúhelníků [2]

1.2.3 Datová struktura polygonálního modelu

Model je v poslední době nejčastěji uložen datovou reprezentací „okřídlená hrana“. Tato struktura v sobě zahrnuje tři lineární seznamy pro vrcholy, hrany a mnohoúhelníky. Nejvíce informací pro model však obsahuje právě struktura seznamu hrana (proto „okřídlená hrana“).



Obrázek 1.2.2 – Okřídlená hrana, datová struktura

Zatímco prvek seznamu vrcholů obnáší pouze ukazatel na další (předchozí) prvek, u stěn se navíc přidává ukazatel na registrovanou hranu. Seznam hran poté obsahuje následující informace [3]:

- ukazatel na další hranu v seznamu
- ukazatel na levou přilehlou stěnu
- ukazatel na pravou přilehlou stěnu
- ukazatel na levou přilehlou hranu ve vrcholu 1
- ukazatel na pravou přilehlou hranu ve vrcholu 1
- ukazatel na levou přilehlou hranu ve vrcholu 2
- ukazatel na pravou přilehlou hranu ve vrcholu 2
- ukazatel na počáteční vrchol hrany
- ukazatel na koncový vrchol hrany

Datová struktura má, jak je vidět z definice, velmi vysokou paměťovou náročnost, její topologie však dovoluje poměrně rychle řešit otázky typu:

- hledej stěny přilehlé k vrcholu
- hledej vrcholy hrany
- hledej přilehlé stěny hrany
- hledej hrany či vrcholy, je-li zadána stěna aj.

Pokud přihlédneme k výše popsanému, ideální datovou strukturou pro moji práci je právě okřídlená hrana ve spojení s trojúhelníkovým polygonálním modelem. Další kapitola se bude zabývat detekcí různých útvarů na polygonálním modelu.

1.3 Možnosti získávání informací o modelu

Tato kapitola se zabývá možnostmi získávání informací o polygonálních modelech. Především zde popíši možnost detekce různých dílčích útvarů na modelu a na konci zde rozeberu problematiku výpočtu křivosti na modelu.

V mnoha případech je nutné daný model segmentovat na menší jednotky. Například po skenování určitého fyzického útvaru dostaneme jediný polygonální model, který je poté žádoucí rozdělit podle určitých kritérií. V případě rozkladových (dekompozičních) modelů je tato operace daleko snazší, existuje mnoho informací o vnitřní struktuře, jako je například hustota, materiál ze kterého se daný objekt skládá aj.

U polygonálního modelu však tato možnost odpadá, zde je pouze možné analyzovat povrch. A nejčastějším způsobem této analýzy je výpočet křivosti povrchu.

1.3.1 Křivost povrchu

Křivost vyjadřuje index, o který se povrch odchyluje od roviny. Existuje však více definicí těchto křivostí. Nejprve vysvětlím pojem křivost v určitém směru [4][5].

1.3.1.1 Křivost v daném směru

Pro rovinný řez plochy $r = r(u, v)$ je poloměr křivosti r v bodě P určen jako:

$$r = \frac{Edu^2 + 2Fdudv + Gdv^2}{|Ldu^2 + 2Mdudv + Ndv^2|}\cos(\Theta)$$

kde θ je úhel, který svírá rovina křivky s normálou v bodě *P* plochy, *E*, *F*, *G* jsou základní veličiny plochy prvního řádu a *L*,*M*,*N* jsou základní veličiny plochy druhého řádu.
Pro naše případy a pro výpočet hlavních křivostí však budeme potřebovat řez normálový. K tomu může sloužit **Meusnierova věta** [6]:

$$r = R_n\cos(\Theta)$$

Meusnierova věta udává vztah mezi normálovým a rovinným poloměrem křivosti. Rovinný řez má v bodě P poloměr křivosti r, jehož přímka je pravoúhlým průmětem úsečky poloměru křivosti R_n normálového řezu, který má v daném bodě s rovinným řezem společnou tečnu. Pro normálovou křivost $\frac{1}{R}$ v daném bodě poté platí:

$$\frac{1}{R} = \frac{Ldu^2+2Mdudv+Ndv^2}{Edu^2+2Fdudv+Gdv^2} = \frac{\varepsilon}{R_n}$$

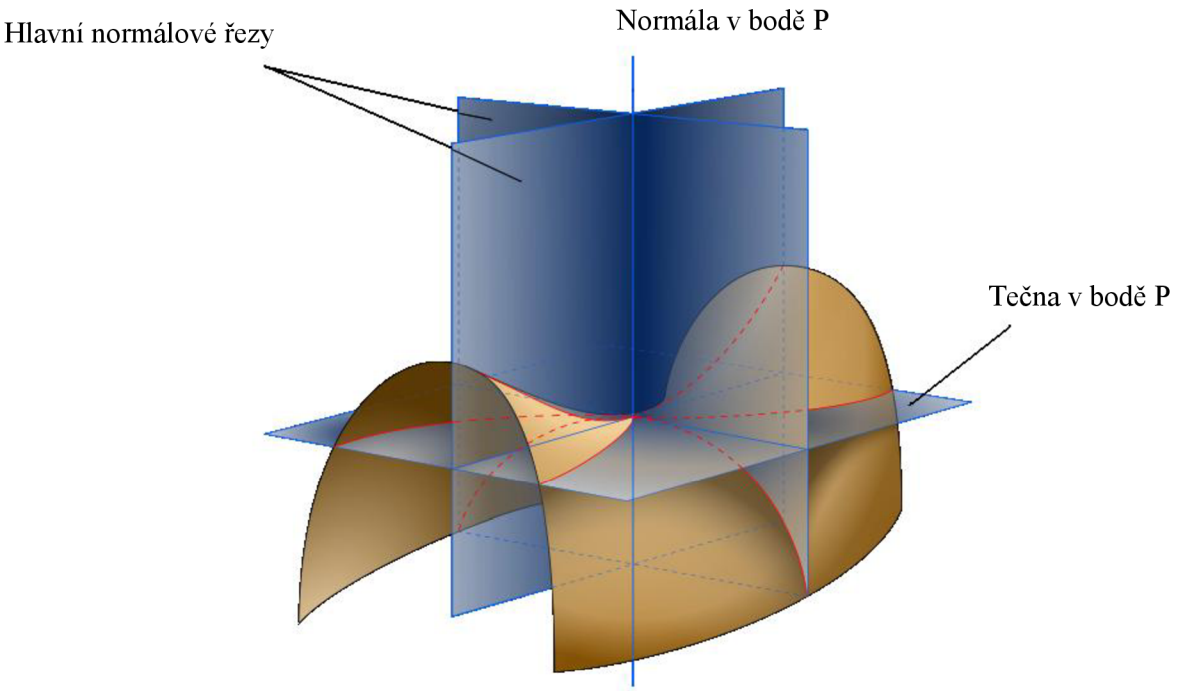
kde ε = +-1.¨

1.3.1.2 Hlavní křivosti (minimální a maximální)

V bodě P poté existuje normálový řez, který má v daném bodě maximální křivost ze všech normálových řezů. Podobně lze též najít normálový řez s křivostí nejnižší. Tyto dvě křivosti se pak nazývají hlavní křivosti a jejich řezy hlavní normálové křivosti (jsou na sebe vždy kolmé).

Maximální a minimální poloměry křivostí R lze určit vyřešením následující rovnice:

$$(EG - F^2)\frac{1}{R} - (EN - 2FM + GL)\frac{1}{R} + (LN - M^2) = 0$$



Obrázek 1.3.1 – Hlavní křivosti [7]

Jednoduše řečeno, maximální a minimální křivost je obrácená hodnota z poloměru kružnice, kterou jsme schopni vepsat do křivky průmětu plochy v obou normálových řezech s tím, že jedna je maximální možná, druhá naopak minimální, která existuje. Normálové řezy jsou na sebe kolmé.

1.3.1.3 Gaussova a střední křivost

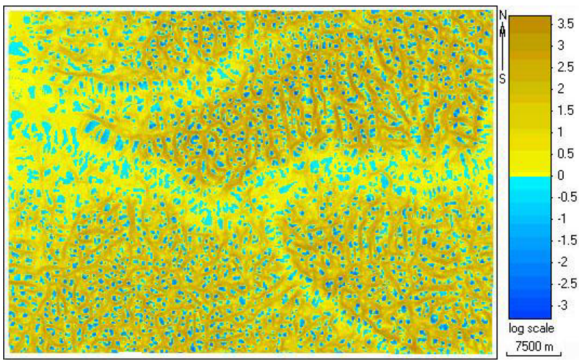
S vědomostí výpočtu minimální a maximální křivosti jsme schopni lehce spočítat další dva typy křivostí, totální (Gaussovu) a střední. Gaussova křivost (K) je definována jako součin hlavních křivostí (neboli součin obrácených hodnot z hlavních poloměrů křivostí), střední (H) naopak jako průměr maximální a minimální křivosti:

$$H = \frac{k_1 + k_2}{2}$$

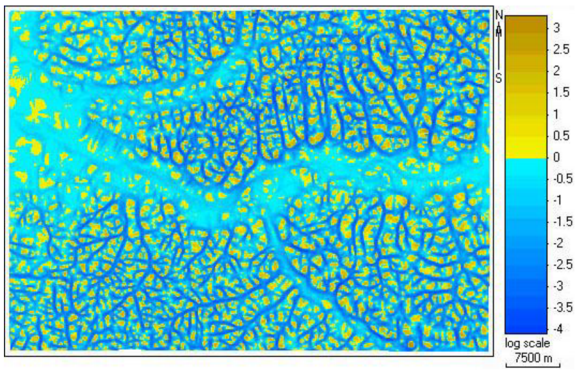
$$K = k_1 k_2$$

kde k_1 a k_2 jsou hlavní křivosti.

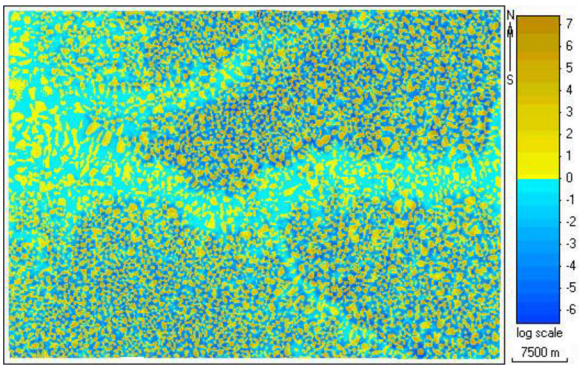
Na závěr přikládám několik obrázků demonstrujících rozdíly mezi čtyřmi zmiňovanými křivostmi [8].



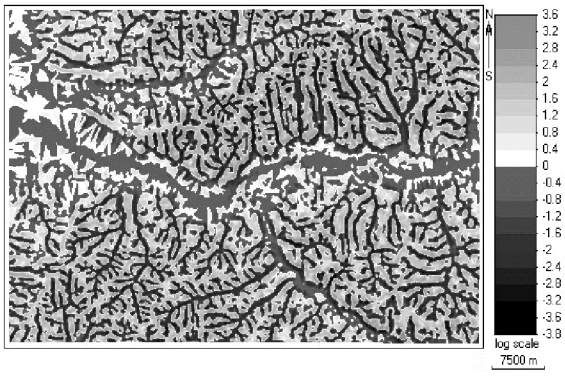
Obrázek 1.3.2 – Maximální křivost



Obrázek 1.3.3 – Minimální křivost



Obrázek 1.3.4 – Gaussova křivost



Obrázek 1.3.5 – Střední křivost

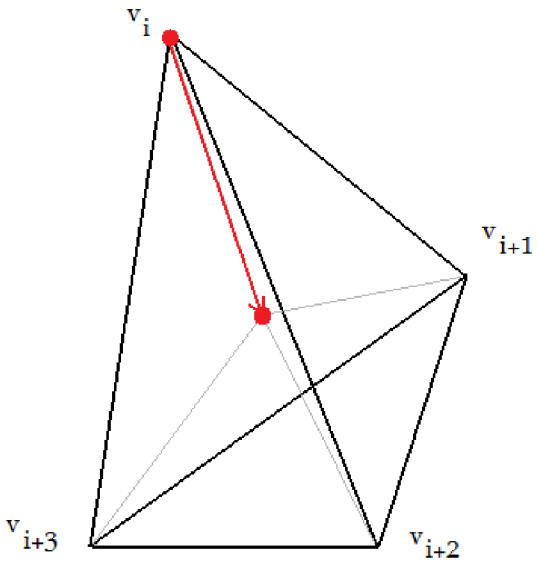
1.4 Vyhlazování

V následující kapitole krátce popíši způsob vyhlazování modelu. Budu se zde zabývat pouze nejjednodušším Laplaceovým vyhlazováním, protože pro můj účel (vysvětleno v následující kapitole) plně postačuje.

1.4.1 Laplaceovo vyhlazování

Laplaceovo vyhlazování je nejjednodušším vyhlazovacím algoritmem. Pracuje na principu posunování vrcholů, topologii tělesa tedy nemění (tzn. počet vrcholů zůstává stejný). Pozice každého z vrcholů je tedy posunuta v závislosti na pozici vrcholů okolních. Tento přístup pracuje výborně pro povrchy složené pouze z trojúhelníků, případně pouze z čtyřúhelníků. Jakmile však máme model složený z obou výše zmiňovaných prvků, metoda nemá ideální průběh. V mé práci však pracuji pouze s trojúhelníkovými modely, takže výše zmiňovaná nevýhoda je bezvýznamná.

Samotný algoritmus pracuje na principu posunu vrcholů do těžiště jeho sousedů. Pro výpočet těžiště tělesa je však nutné znát rozložení jeho váhy. Při Laplaceově vyhlazování se tato situace řeší přiřazením vah k jednotlivým vrcholům. Touto metodou se dá jednoduše obejít i zmiňovaná nevýhoda s kombinovaným troj-/čtyřúhelníkovým modelem. Pro můj případ, kdy od této metody očekávám rychlost a jednoduchost na trojúhelníkovém modelu, však postačí, vezmeme li váhu všech vrcholů rovnou jedné [9][10].



Obrázek 1.4.1 – Laplaceovo vyhlazení

Pro každý bod modelu poté těžiště spočítáme jednoduchým spočítáním průměru všech tří souřadnic (váha jednotlivých vrcholů = 1):

$$v_x = \frac{\sum_{i=0}^{n} v_{xi}}{n},$$
$$v_y = \frac{\sum_{i=0}^{n} v_{yi}}{n},$$
$$v_z = \frac{\sum_{i=0}^{n} v_{zi}}{n}.$$

Při výše uvedeném algoritmu sice dochází k nejznámějšímu nežádoucímu účinku Laplaceova vyhlazování, smršťování modelu, ale pro můj účel je tento problém zanedbatelný. Vyhlazený model se bude používat pouze pro spočítání křivosti, která je nezávislá na jeho velikosti.

2 Návrh řešení

2.1 Úvodem

V této kapitole nastíním návrh struktury výsledné aplikace a budu zvažovat možnost použití externích knihoven. Následně popíši konkrétní použití teoretických znalostí z kapitoly 1, konkrétně pohyb po 3D polygonálním modelu a výpočet křivosti.

Následně, již ve spojení s experimentováním na modelu, rozeberu možné různé postupy segmentace modelu, filtrace segmentů aj.

2.2 Externí knihovny

Po doporučení panem dr. Krškem jsem usoudil, že pro budoucí aplikaci bude nejlepší kombinace dvou externích modulů – Medicine Data Segmentation toolkit (dále jen MDSTk) [11] a Open Scene Graph (OSG) [12].

MDSTk je soubor nástrojů pro práci s 2D/3D obrázky původně určený pro medicínské účely, konkrétně segmentaci medicínských dat. Z MDSTk je pro mou práci nejdůležitější modul Vector Entity, který disponuje velkým množstvím funkcí pro práci s polygonálním modelem. Modely jsou dodávané ze scanneru ve formátu .stl, který je Vector Entity schopný bez problémů načíst. Navíc očekávám, že bude nutné tento model různě procházet kvůli výpočtům, pročež je Vector Entity přímo stvořený.

OSG naopak v mém programu zastupuje pouze zobrazovací funkci. Tento soubor knihoven je nadstavbou OpenGL a určen pro vývoj nejnovějších grafických aplikací. Pro moji práci je důležité, že disponuje poměrně jednoduchým API a navíc obsahuje vestavěný viewer, který velice elegantním způsobem vykresluje dané modely na výstup.

Oba dva zmiňované soubory jsou navíc pod volně šiřitelnou licencí.

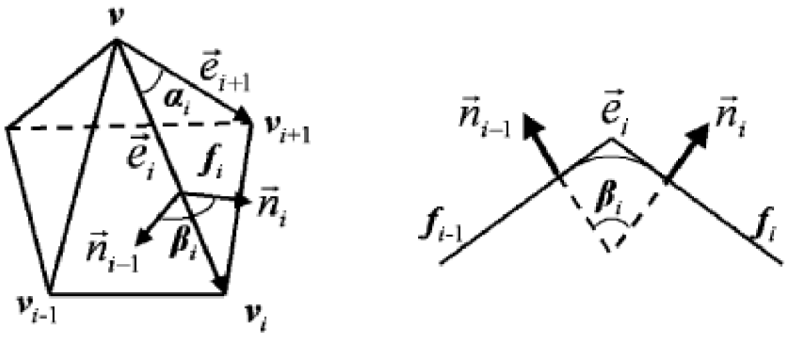
2.3 Výpočet křivosti na polygonálním modelu

První dílčí činností pro detekci zubů na 3D polygonálním modelu bude (po načtení samotného modelu) spočítání křivosti pro každý vrchol. V kapitole 1 jsem prezentoval různé způsoby výpočtů křivostí, vše se však vztahovalo na modely spojité. Z teoretického hlediska polygonální model nemá vůbec žádnou křivost, protože na stěnách je křivost nulová a v hranách nemá řádně definované zakřivení. Pokud si ale představíme trojúhelníkový model jako lineární aproximaci nějakého zakřiveného povrchu, křivost je možné spočítat.

Pro můj případ bude potřeba vypočítat jak minimální a maximální křivost, tak pokud možno také Gaussovu a střední. S těmito křivostmi bude nutné experimentovat a nalézt nejvhodnější kombinaci pro můj účel.

2.3.1 Odvození aproximace

Polygonální model se skládá [13], jak již jsem se zmínil, ze seznamu vrcholů $V = \{v_i\}_i \subset R^3$, seznamu hran $E = \{e_j = (v_{j1}, v_{j2})\}_j$ a v poslední řadě seznamu stran $F = \{f_k = (v_{k1}, v_{k2}, v_{k3})\}_k$. Nechť $v \in V$ je vrchol modelu M a $v_1, \dots, v_n$ jsou vrcholy s ním sousedící. Definuji poté strany $\vec{e}_i = v_i - v$ a úhel α_i mezi dvěma sousedícími stranami. Trojúhelník mezi e_i a e_{i+1} nazývám $f_i = (v, v_i, v_{i+1})$.



Obrázek 2.3.1 – Popis modelu

Nyní můžeme definovat integrál Gaussovy křivosti $\bar{K}$ a integrál střední křivosti $\bar{H}$ s ohledem na obsah S připojený k vrcholu v:

$$\bar{K} = \int_S K = 2\pi - \sum_{i=1}^{n} \alpha_i \quad \text{a}$$
$$\bar{H} = \int_S H = \frac{1}{4}\sum_{i=1}^{n} \|e_i\| \beta_i.$$

Pro zpětné zderivování těchto integrálů ve vrcholu *v* budeme předpokládat, že křivosti jsou kolem našeho vrcholu rovnoměrně rozmístěné:

$$K = \frac{\bar{\bar{K}}}{S} = \frac{2\pi - \sum_{i=1}^{n} \alpha_i}{\frac{1}{3}A} \quad \text{a}$$
$$H = \frac{\bar{H}}{S} = \frac{\frac{1}{4}\sum_{i=1}^{n} \|e_i\| \beta_i}{\frac{1}{3}A}.$$

A v rovnicích značí součet obsahů všech trojúhelníků napojených na vrchol *v*. Pomocí výše zmíněných rovnic lze již spolehlivě napsat algoritmus na výpočet křivostí. Co se týče minimální a maximální křivosti, dají se použít převodní vztahy z kapitoly 1.3.1.3.

2.4 Využití výpočtů křivosti

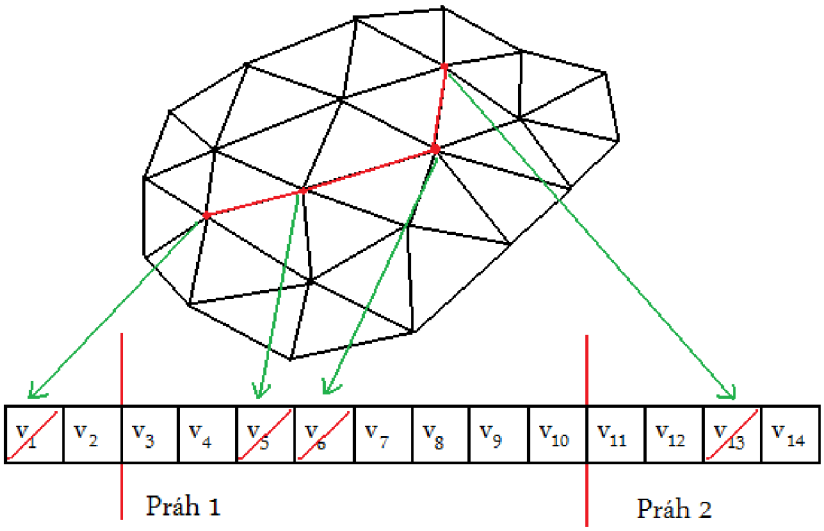
Samotný výpočet křivosti však určitě nestačí k detekci tvarů na modelu. Po provedení výpočtu následuje další důležitý úkol, detekce křivek. Podle křivosti je totiž nutné na modelu vyznačit křivky oddělující jednotlivé regiony. V mé práci jsem se zaměřil na dva algoritmy detekce křivek.

2.4.1 Vlastní vyhledávání

Po konzultaci s Ing. Krškem Ph.D. a Ing. Španělem jsem implementoval vyhledávací algoritmus založený na seřazeném seznamu vrcholů podle křivosti. Zde, po určení dvou prahů, vyberu vrchol s největší křivostí. Z vrcholu zkontroluji okolí a vyberu souseda s největší křivostí. Následně kontroluji, zda zmíněný vrchol splňuje podmínku druhého prahu, a pokračuji na něj. Toto opakuji do té doby, než narazím na vrchol, který má všechny sousedy menší než práh 2. Zde linka končí a vybírám další počáteční bod s největší křivostí. Při tomto postupu použité vrcholy postupně ze seznamu mažu, aby se linky nemohly křížit (viz obrázek 2.4.1).

Navržený algoritmus má však dvě zásadní nevýhody. Model čelisti se zuby má značně nerovnoměrně rozloženou křivost, proto nastane situace, že v blízkosti zubů (kde je koncentrace nejvyšších křivostí) se detekovaná linka „zamotá“ a může pokračovat do spirály, či jiných útvarů, což je zcela jistě nežádoucí jev. Daná situace se dá sice ošetřit například odebíráním nejbližšího okolí linky ze seznamu, ale nikdy to nebude mít dokonalý efekt.

Druhým, zásadním problémem, je nerozdělení modelu do regionů. Linky jsou sice detekovány, ale nerozdělí model uzavřenými oblastmi (ideální pro semínkové vyplňování v konečně fázi). I když se i toto dá výrazně omezit přidáním ještě dvou prahů pro maximální a minimální délku linky, nikdy tento problém v metodě nezmizí.

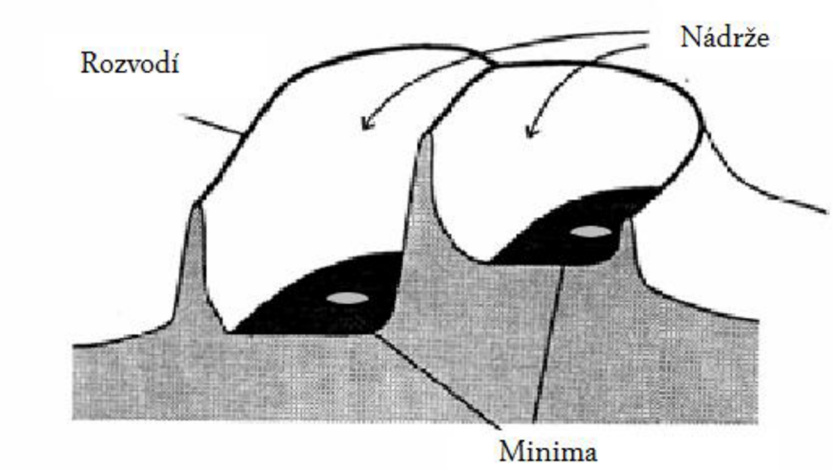


Obrázek 2.4.1 – Popis algoritmu

Po mnoha víceméně neúspěšných pokusech a modifikacích jsem tuto metodu opustil a implementoval upravený algoritmus Watershed, který popíši v následující kapitole.

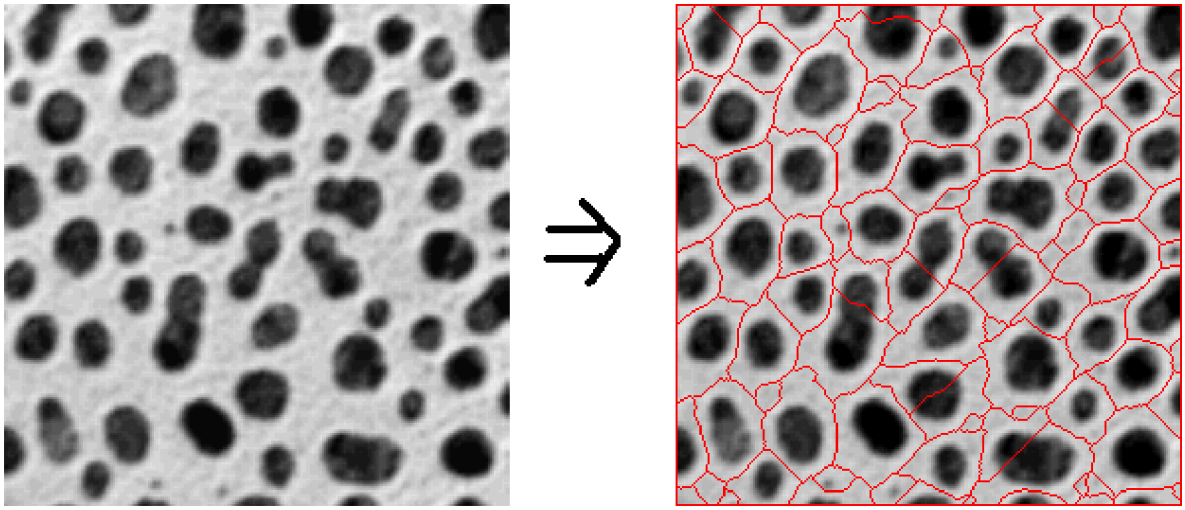
2.4.2 Metoda Watershed

Následující algoritmus je použitý ve finálním programu, proto ho zde popíši podrobněji. Metoda Watershed [14] je častým postupem pro segmentaci modelu do různých částí. Ve své podstatě je analogická se zaplavováním oblastí tekutinou.



Obrázek 2.4.2 – Analogie

Dané oblasti budeme všechny ve stejném čase zaplavovat vodou (či jinou tekutinou, viz níže). Pokud se dvě nádrže spojí, nestane se z nich jedna, jak bychom očekávali v uváděné analogii, nýbrž zde vznikne hranice mezi dvěma regiony. V mém programu jsem metodu Watershed užil ne na výškové rozdíly, jak jsem uváděl doposud, ale na údaje křivosti.



Obrázek 2.4.3 – Příklad použití

Zde však vyvstává jeden základní problém metody Watershed. Pokud je použita na model s velkým množstvím lokálních minim, vznikne nepoužitelné množství regionů, proto jsem zde musel provést několik úprav v implementaci základního algoritmu.

První hlavní změnou je úvodní počet regionů. Klasická metoda začíná nejnižšími hodnotami a pokračuje s tím, jak voda přibývá. V mé verzi existuje minimální křivost, po kterou se vše vyplní jako jeden region. Tím vznikne velice malé množství počátečních regionů s velkou rozlohou, které však nenarušují křivky zubů. Tato hranice se pohybuje okolo 0.4 střední křivosti.

Další odlišností je způsob zakládání nových segmentů. To je pozdrženo do té doby, dokud to nebude naprosto nutné. Pokud program narazí na bod, jímž by začal nový region, zvýší jeho křivost tak, že ho posune až na konec seznamu. Tím se docílí toho, že nové regiony jsou vytvářeny tehdy, až opravdu jiná možnost nezbývá.

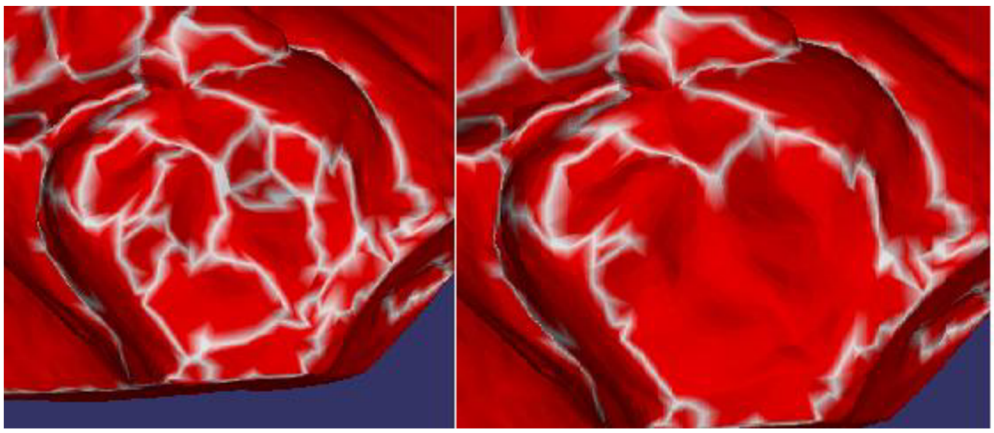
2.5 Filtry segmentů na modelu

Samotná metoda Watershed však jistě k detekci zubů nestačí. I když díky zmíněným modifikacím se dosáhne průměrně 150 segmentů na model, stále je to číslo poměrně vysoké a hlavně nám neříká nic o umístění zubů. Segmenty se sice budou soustřeďovat okolo zubů, ale například i na patře, kde je značný počet nerovností. Proto po segmentaci modelů používám řadu filtrů pro snížení počtu segmentů na modelu.

Tyto kapitoly jsou doplněny o obrázky z již hotového programu (kapitola 3), protože i jejich návrh probíhal s ohledem na potřebné postupy po implementaci předešlých metod.

2.5.1 Spojování malých segmentů

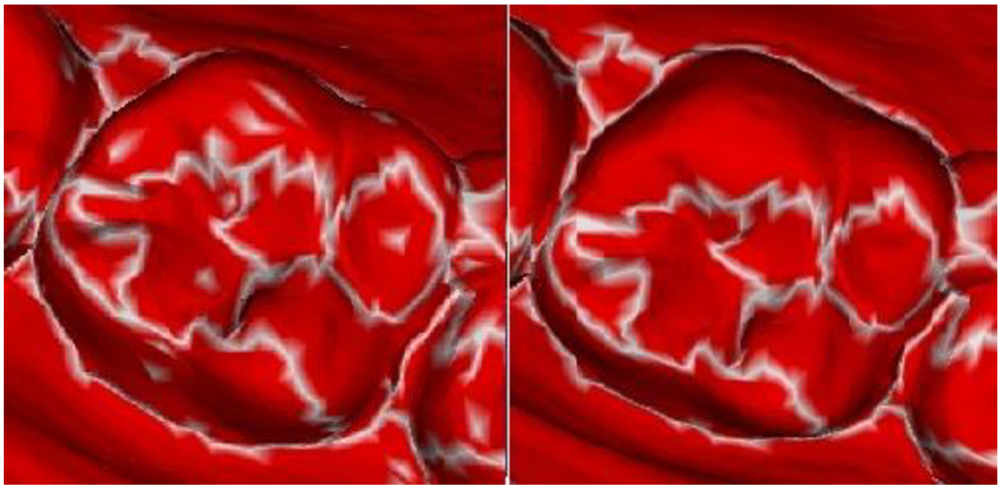
Hned při prvním pohledu na segmentovaný model je zřejmé, že první filtrací by mohlo být právě pospojování malých regionů. Tato metoda by měla postupně projít všechny segmenty a ty, jež nesplňují velikostní limit, spojit do jednoho. To, ke kterému se region připojí, může záviset na mnoha aspektech, pro příklad uvádím spojení s jiným malým segmentem nebo se segmentem, který má s mazaným nejdelší hranici, případně spojení zcela náhodné. Tyto metody je potřeba prozkoumat experimentálně na modelu (viz kapitola Implementace).



Obrázek 2.5.1 – Před a po filtru

2.5.2 Mazání samotných hranic

Modifikovaný Watershed způsobuje též jeden nežádoucí efekt, a to osamocené hraniční vrcholy uprostřed segmentů. To způsobuje odkládané vytváření nových regionů. Proto bude potřeba implementovat funkci, která tyto vrcholy smaže. Tato metoda nebude složitá, ale výrazně se zlepší celkový vzhled a použitelnost segmentovaného modelu.

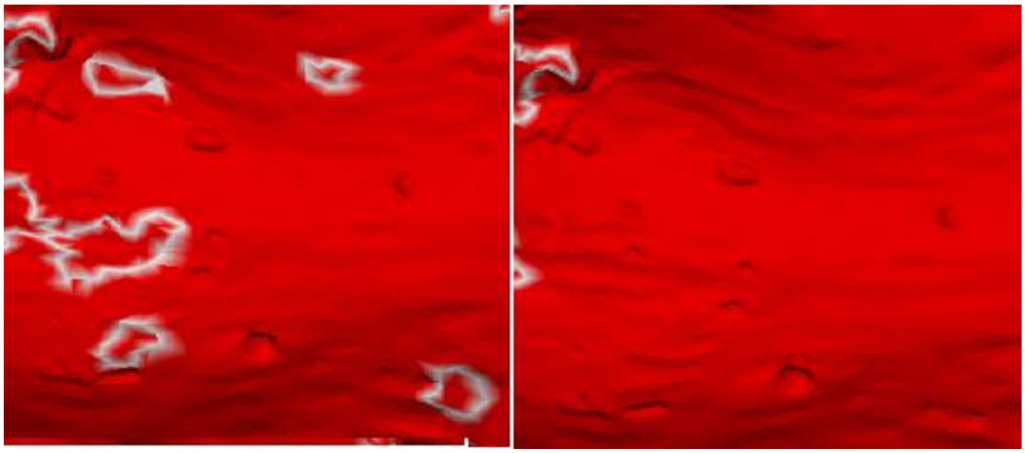


Obrázek 2.5.2 – Před a po filtru

2.5.3 Spojování samostatných segmentů

Spojování samotných segmentů je jedna z dalších metod zmenšení počtu segmentů na modelu. Při pohledu na segmentovaný model totiž velké množství segmentů leží uprostřed jiných. Tyto jsou pro nás bezvýznamné, protože se zcela jistě jedná o lokální minima křivosti. Proto další filtr bude mít za úkol tyto regiony odstranit, respektive sloučit je s tím, který je obklopuje.

Tuto metodu jsem však ještě modifikoval o počítání délky hranice regionů. Pokud má totiž region A např. 350 hraničních vrcholů s regionem B a 2 s regionem C, je pro můj účel též bezvýznamný, ale metoda by ho nesloučila. Samozřejmě tato modifikace způsobí, že je metoda schopna být použita vícekrát po sobě.

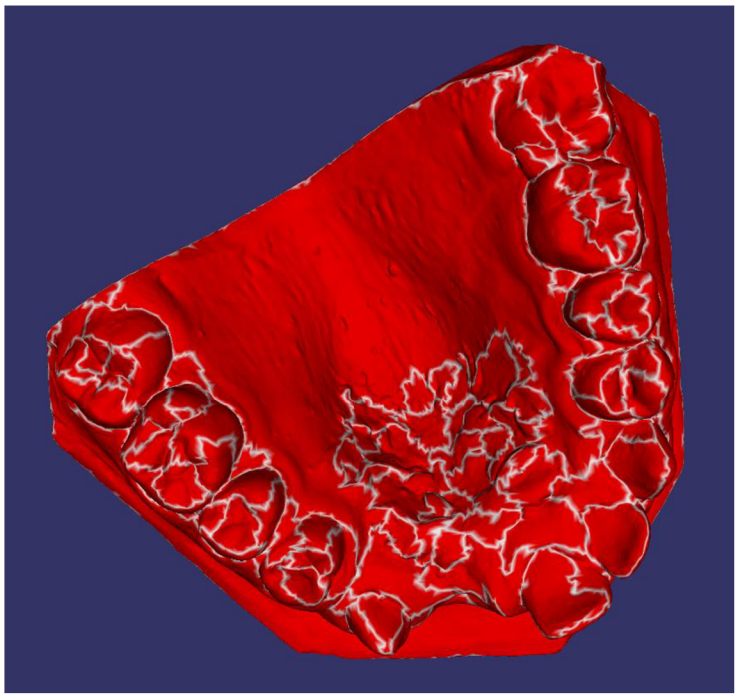


Obrázek 2.5.3 – Před a po filtru

2.5.4 Vyhlazení modelu

Segmentovaný model po aplikování všech zmíněných filtrů je možné zhlédnout na obrázku 2.5.4.

Je zřejmé, že se počet segmentů zredukoval na číslo pohybující se okolo 100, přičemž na jeden zub vychází průměrně 4.6 regionu. To již zcela postačuje pro manuální detekci zubů, kterou uživatel provede semínkovým vyplňováním jednotlivých segmentů. Tato metoda bude určitě přesnější, než plně automatická detekce, ta je však zajisté pro uživatele nejvýhodnější. O návrh automatického vyhledávání se proto pokusím v následujících kapitolách.

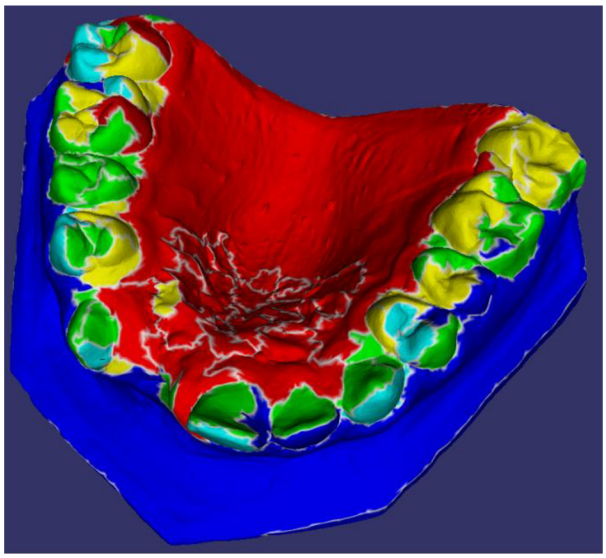


Obrázek 2.5.4 – Po výše zmíněných filtrech

Po konzultaci s Ing. Španělem, který mi doporučil pokračovat silným vyhlazením modelu, jsem do výpočtů zahrnul Laplaceovo vyhlazení, přesněji 20 iterací tohoto vyhlazení. Tím způsobím citelnou změnu křivosti po celém modelu. Výsledkem bude, že zmizí lokální nerovnosti, které mají za následek segmentaci např. na patře. Důležité globální nerovnosti (zuby) by však měly být zachovány. Samozřejmě má to i své stinné stránky, křivost se značně zvýší v částech zabroušení modelu (spodní část), kde se vlivem vyhlazení naopak model „zakulatí“. To ale problém nepředstavuje, tyto regiony jsem již vyloučil předchozími metodami. Vyhlazení je nutné provést na odděleném modelu tak, aby bylo možné použít informace o hranách z modelu nevyhlazeného a informace o křivostech z upraveného modelu.

Po spojení těchto dvou informací se dá dále snížit počet segmentů spojením těch, které mají přibližně stejnou průměrnou křivost. Pro toto spojení jsem po několika experimentech zvolil rozdíl maximálně 0.01. Při nižších hodnotách rozdílu nedosáhneme požadovaného efektu, nejsou ovlivněny žádné segmenty. Naopak při vyšších číslech se začínají spojovat nežádoucí regiony.

Následující obrázek demonstruje model po těchto operacích s barevně vyznačenou průměrnou Gaussovou křivostí každého regionu. Je zcela zřejmé, že filtr by mohl odstranit nerovnosti na patře.



Obrázek 2.5.5 – Po aplikování vyhlazení

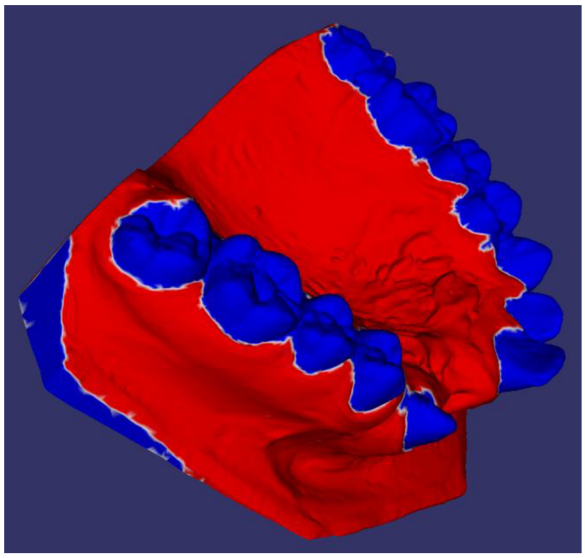
2.5.5 Další filtrace

Z obrázku 2.5.4.2 je patrné, že na zubech průměrná křivost výrazně převyšuje okolí. Pro toto srovnání jsem použil Gaussovu křivost, protože u té se projevují největší rozdíly mezi zuby a okolím (na špičkách zubů jsou vysoké kladné křivosti i po 20 iteracích vyhlazení).

Nyní pokud zvolím jen ty regiony, které mají vyšší křivost (na předchozím obrázku všechny, co jsou jiné než červené), a nepočítám do nich ty, které se výrazně odlišují velikostí (modře označená dáseň), oblouk již vzdáleně připomíná oblouk zubů. Okamžitě se však nabízí filtrace podle délky společné hranice. Pokud máme například region s velice nízkou křivostí, který však je obklopen z větší části potenciálními zuby, můžeme jej přiřadit též k potenciálním zubům. Stejně to platí naopak, viz například žlutý region na patře na obrázku 2.5.4.2.

Následně můžeme již bez problému spojit regiony potenciálních zubů a zbytku. Výsledek je zřejmý z obrázku 2.5.5.1. Bohužel i v této fázi zde zůstávají části detekované mimo zubní oblouk. To je způsobeno zmiňovaným vyhlazením, které model na hranách znatelně zakřiví, a z rovných ploch se tudíž stanou plochy s poměrně vysokou křivostí. Mohl by tedy následovat ještě poslední z filtrů, a to použití Gaussovy křivosti nevyhlazeného modelu a následné porovnání hodnot regionů zubního oblouku a jiných. Na všech testovaných modelech se tyto hodnoty pohybují v řádech kladných desetin na zubním oblouku a ne více než v řádech setin mimo ně. Proto jsem ještě provedl tuto filtraci.

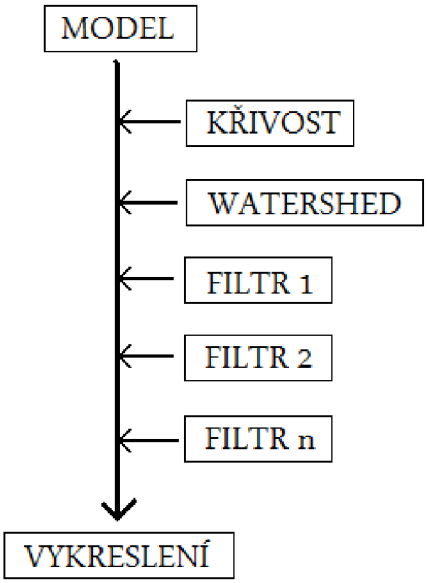
Ve všech zmiňovaných filtrech používám již křivost Gaussovu, která sice nemá tak dobré výsledky v metodě Watershed jako křivost střední, obsahuje ale největší rozdíly mezi hodnotami. Navíc dosahuje i záporných hodnot v místech, která mají záporné zakřivení. Proto vyhlazené zuby si zachovávají kladnou křivost po celém povrchu, kdežto patro má po většině povrchu křivost zápornou.



Obrázek 2.5.6 – Po sloučení regionů

2.5.6 Návrh struktury programu

Nyní přikročím k návrhu samotného programu. Dosud jsem se věnoval pouze návrhu jednotlivých algoritmů, nyní však rozeberu více strukturu vyhledávacího algoritmu jako celku. Kvůli experimentální povaze programu by bylo ideální, kdyby objektová struktura zaručovala nezávislost jednotlivých operací na sobě (použitá křivost, pořadí filtrů aj.).Operace s modelem by pak měly vypadat následovně:

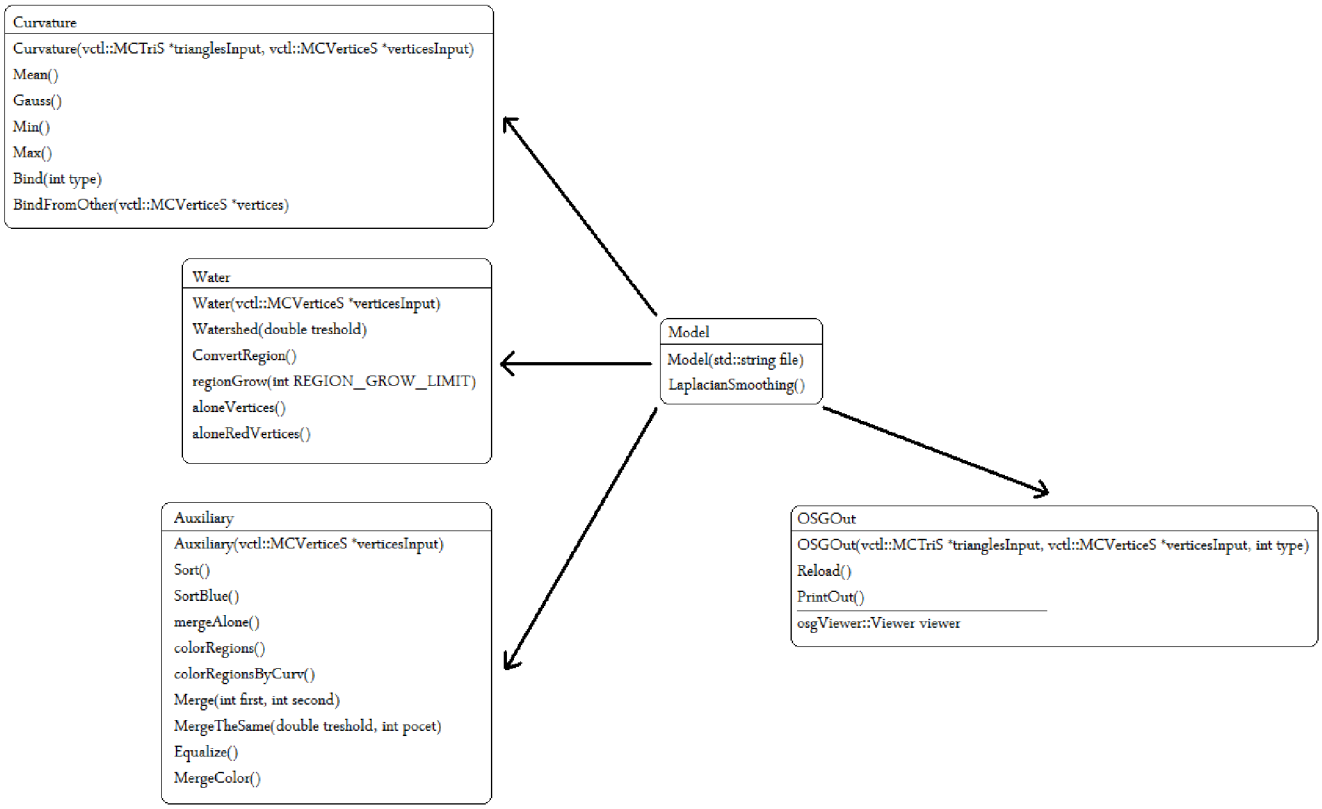


Obrázek 2.5.7 – Návrh aplikace

S ohledem na navržené schéma bude žádoucí vytvořit pro jednotlivé typy výpočtů různé třídy, které však budou pracovat nad jedním modelem. Samotný algoritmus bude poté kopírovat postup navržený v této kapitole. Pro jeho vykreslení, jak již bylo řečeno v kapitole 2.2, bude použit Open Scene Graph, respektive jeho **OSG Viewer.**

3 Implementace

S přihlédnutím na zmiňovaný návrh jsem implementoval aplikaci, která se shoduje s následujícím diagramem. Pro přehlednost jsou odebrány některé nepoužité, privátní či pomocné funkce. Pro úplný přehled viz programová dokumentace.



Obrázek 3.1.1 – Diagram tříd

V následujících kapitolách podrobně rozeberu jednotlivé funkce těchto tříd, pouze však ty, které byly použity ve výsledné aplikaci. Programový základ totiž obsahuje pro názornost i bývalé funkce testování a opuštěné přístupy k řešení.

3.1 Třída Model

Třída **Model** zajišťuje hlavní úložiště informací o modelu. Hlavními dvěma držiteli informace jsou proměnné **mesh** a **body**, které uchovávají seznamy bodů a trojúhelníků (vctl::MCVerticeS a vctl::MCTriS). Ihned při volání konstruktoru se třída pokouší otevřít vstupní soubor zadaný prvním parametrem. Pokud se toto nepodaří, program se ukončí s chybovým hlášením.

Samotná struktura jednotlivých bodů má velice výhodnou stavbu pro můj účel. Obsahuje atribut **Value** datového typu double, který je použitelný pro uživatelskou hodnotu, v mém případě křivost. Podle tohoto atributu se též implicitně body řadí do Vector Entity fronty. Druhým důležitým atributem je pak **ValuePtr**, což je obecný ukazatel. Zde jsem tudíž schopen uložit další důležitou proměnnou – barvu vrcholu.
Třída však kromě konstruktoru obsahuje ještě jednu důležitou funkci, a to je LaplacianSmoothing().

3.1.1 Funkce LaplacianSmoothing()

Tato funkce má za úkol vyhladit model uložený ve třídě, tudíž provede jednu iteraci Laplaceova vyhlazení. Metoda projde postupně celý model a pro každý vrchol vypočítá z jeho okolí nové souřadnice, které uloží do samostatného pole. Nové souřadnice se tak všechny počítají z původních. Teprve po spočítání všech souřadnic jsou přiřazeny svým vrcholům. Tímto způsobem se obecně získávají lepší výsledky Laplaceova vyhlazování než při přiřazování přímém, metoda je však náročnější na paměť.

3.2 Třída Curvature

Třída **Curvature** zajišťuje veškeré operace potřebné k výpočtu křivostí a jejich aplikací na model. Při vytvoření konstruktorem se do ní uloží odkaz na seznam bodů, nad kterými se budou operace provádět.

3.2.1 Funkce Mean(), Gauss(), Min() a Max()

Tyto metody zajišťují výpočet křivostí pro model. Jejich chování se liší pouze typem křivosti, který počítají, proto je zde popíši společně. Postupně se zde projdou všechny vrcholy na modelu a pro každý se z údajů o okolí vypočítá daná křivost podle vztahů z předchozích kapitol. K tomu pomáhá též funkce **double Angle(double x1, double y1, double z1, double x2, double y2, double z2)**, kterou zde neuvádím, ale která počítá úhly nutné k výpočtům. Jednotlivé výsledky se ukládají do externích souborů jako posloupnost čísel.

3.2.2 Funkce Bind(int type)

Po spočítání křivosti vzniknou soubory, které je nutné zkombinovat se samotným modelem. K tomu slouží tato metoda. Po vyhodnocení parametru (**enum** typ, možnosti **GAUSS, MEAN, MIN, MAX, NONE**), kterou z křivostí má sloučit s modelem, otevře soubor a pro každý bod načte dané číslo. To se ukládá do atributu **Value** každého vrcholu, který je určen pro uživatelský atribut ve

vctl::MCVertex. Metoda však očekává spočítané křivosti v souborech ve formátu a s názvy, které generují výše zmiňované funkce. Pokud tyto neexistují, program se ukončí s chybovým hlášením.

Funkce navíc ještě alokuje paměť pro uložení informací o barvě. Ta se ukládá do atributu **ValuePtr** jako ukazatel na typ integer. Poté jim přiřadí díky pomocným funkcím barevné hodnoty podle použité křivosti. Poslední možností **NONE** model zůstane bílý.

3.2.3 Funkce BindFromOther(vctl::MCVerticeS *vertices)

Tato funkce je speciálním případem metody **Bind**. Nepoužívá však jako vstupní informace soubory s křivostmi, nýbrž vstupní seznam vrcholů z jiného modelu. To se využívá pro spočítání křivostí na vyhlazeném modelu a jejich přiřazení k modelu původnímu. Funkce však, stejně jako předchozí, nekontroluje správné použití. Pokud je použita se vstupním seznamem jiného modelu (jiný počet vrcholů), skončí aplikace s největší pravděpodobností pádem.

3.3 Třída Water

Třída **Water** implementuje algoritmus Watershed spolu se základními filtry. Veškeré použití této třídy je podmíněno přiřazenou křivostí na model, není však ošetřená kontrola, tu je povinen programátor zajistit externě. Funkce taktéž pracuje nad konkrétním modelem, proto může být většina funkcí bez parametrů.

Důležité je však též zmínit, že funkce pracuje s hodnotami ve **ValuePtr** signalizujícími ID segmentů, ne jejich barvu (viz Curvature)! Proto po každém použití funkce je nutné před vykreslením konvertovat ID na barvy a naopak. K tomu slouží funkce **ConvertRegion**() a **ConvertRegionReverse**().

3.3.1 Funkce Watershed(double treshold)

Funkce **Watershed** je pravděpodobně nejdůležitější metoda třídy. Implementuje v návrhu zmiňovaný modifikovaný Watershed algoritmus. Nejprve vynuluje všechny ID regionů (**ID=0 znamená, že vrchol je hranice**), následně zaplní rekurzivně všechny vrcholy, které mají křivost menší než je parametr treshold. V programu se osvědčilo číslo 0.4, protože ještě nespojuje jednotlivé zuby, ale segmenty jsou poměrně velké.

Nakonec se berou vrcholy postupně od nejnižších křivostí a zjišťuje se, zda patří k nějakému regionu, nebo zůstávají hranicemi. Pokud však narazí na vrchol, který nesousedí s žádným regionem, nezakládá se nový, nýbrž se zvětší jeho křivost a posune se dále v seznamu. Tím se docílí generování menšího počtu regionů. Nový region se založí pouze tehdy, když už není jiná možnost.

Cílem této metody je model segmentovaný na části, přičemž hranice mají ID = 0, další regiony pak ID > 0.

3.3.2 Funkce regionGrow(int REGION_GROW_LIMIT)

Water třída však obsahuje ještě dva základní filtry pro její algoritmus. Prvním z nich je spojování malých regionů v jeden. Tento filtr je implementovaný tím nejjednodušším způsobem, spočítá velikosti všech segmentů, zjistí, zdali některé leží vedle sebe, a ty poté spojí. To se provede změnou hodnoty ID na nenulovou hodnotu. Není nutné měnit ID obou segmentů, o to se postará metoda **Auxiliary::Sort()** popsaná níže.

Experimenty bylo dokázáno, že pro automatickou detekci je nejlepší limit 45 bodů na segment, pro přesnější manuální výběr pak limit rovný 35.

3.3.3 Funkce aloneVertices()

Tato metoda též implementuje jeden z návrhů filtrů pro Watershed, konkrétně pak vymazání samotných hraničních vrcholů. Funguje též na jednoduchém principu procházení všech vrcholů, pokud nějaký z nich má za souseda maximálně jeden hraniční vrchol, bude smazán.

3.4 Třída Auxiliary

Poslední třídou využívanou pro samotnou detekci zubů je **Auxiliary**. Jak již sám název napovídá, jedná se o třídu pomocnou, doplňující program o zbytek filtrů zmíněných v předchozích kapitolách. V jednotlivých funkcích se též bere **ValuePtr** jako ID, jsou však i výjimky. Proto u každé funkce výslovně uvedu, který z přístupů používá.

3.4.1 Funkce Sort()

Funkce **Sort()** je jedna z nejdůležitějších funkcí třídy. Každá následující funkce totiž očekává seřazený seznam segmentů, obsahující seznam vrcholů segmentu, celkový počet vrcholů, jeho sousedy a průměrnou křivost. To má na starosti právě metoda **Sort**. Navíc ukazatel na jednotlivé segmenty uloží do pole indexované jejich ID, které automaticky přiřadí ke každému vrcholu (ID > 0).

Ze skutečnosti, že každý vrchol má přiřazené ID, vyplývá, že před zobrazením je nutné použít konvertovací funkci (**ConvertRegion()** a **ConvertRegionReverse()**).

3.4.2 Funkce MergeAlone()

Funkce předpokládá seřazené regiony do interního seznamu pomocí funkce **Sort()**. Projde celý seznam a pro všechny segmenty, které mají pouze jednoho souseda nebo příliš malou hranici

s druhým sousedem, tuto hranici smaže. Následně je segment ze seznamu vymazán. Pro optimální chod aplikace však důrazně doporučuji znovu provézt řazení.

Tato funkce využívá **ValuePtr** taktéž jako ID regionů.

3.4.3 Funkce MergeTheSame(double treshold, int pocet)

Tato metoda slouží pro porovnání, respektive spojení dvou segmentů v jeden s přihlédnutím na jejich křivost. První parametr udává maximální rozdíl v křivostech dvou sousedních regionů. Druhý parametr udává maximální velikost regionů, které tato metoda bude brát v úvahu. Například pokud vezmeme region o velikosti přibližně 4000 vrcholů, nemá význam ho spojovat (dokonce je to pro nás nežádoucí), protože tvoří 1/4 modelu. Po experimentování s touto metodou se vyplatilo použít parametry **(0.01, 1000)**. Metoda taktéž využívá **ValuePtr** jako ID.

3.4.4 Funkce ColorRegions() a ColorRegionsByCurv()s

Následující dvě funkce již pracují s barevnými údaji, ale vyžadují seřazené pole segmentů. Tyto pak pouze obarví pro uživatele snadno čitelnými barvami, buď podle počtu jejich sousedů, případně podle průměrných křivostí (nebo mediánů) segmentů.

3.4.5 Funkce Equalize()

Předposlední důležitá metoda, sloužící též k práci s barevnými údaji, očekává již striktně barevně rozdělený model. Bílá je vyhrazena pro hraniční body, červená pro dáseň a modrá pro zuby. Implementuje návrh filtru z kapitoly 2.5.5, tj. vyrovnání nerovností. Společně s funkcí **MergeBlue()** projde postupně všechny segmenty (pomocí pole) a pokud daný segment leží mezi většinou regionů s jinou barvou, obarví ho též na ni. Tím se vyrovnají velké výkyvy a díry v detekci zubů.

3.4.6 Funkce MergeColor()

Poslední z funkcí zde zmiňovaných pro práci s barevným modelem spojí regiony s jednou barvou do jednoho. Používá se pouze ke konci samotné detekce, je ale možné na takto spojené regiony znovu aplikovat předchozí metody.

3.5 Třída OSGOut

Zde jen v rychlosti zmíním základní informace o třídě **OSGOut**, která má za úkol vytisknout pomocí Open Scene Graph model na obrazovku. K tomu využívá dvě základní funkce.

Reload() se používá k načtení aktuálních informací do OSG struktury. Je proto nutné při každé změně, kterou by měl uživatel vidět na obrazovce, tuto proceduru volat. Druhá z důležitých metod je **PrintOut()**, měla by být v programu spuštěna pouze jednou, neboť způsobí spuštění OSG Vieweru

pro grafický výstup na obrazovku. Samozřejmě je zde ještě několik pomocných funkcí pro vytisknutí pouze detekovaných zubů aj., více však viz programová dokumentace.

3.6 Finální algoritmus

Existuje velké množství kombinací výše zmíněných metod k dosažení detekce zubního oblouku na čelisti. Z dlouhodobého experimentování a vývoje mé práce jsem však zvolil následující postup, který je obsažený ve výsledné aplikaci.

- Načíst střední křivost — **Curvature::Bind(MEAN)**
- Provést metodu Watershed — **Water::Watershed(0.4)**
- Spojit malé segmenty — **Water::regionGrow(45)**
- Vymazat samostatné hranice — **Water::aloneVertices()**

- Opakovat, dokud nebude žádná změna:
  - Rozdělit segmenty do seznamu — **Auxiliary::Sort()**
  - Smazat samotné segmenty — **Auxiliary::mergeAlone()**
  - Smazat samostatné hranice — **Water-::aloneVertices()**

- Připojit křivost z vyhlazeného — **Curvature::BindFromOther(Smooth**

- Spojit podobné křivosti — **Auxiliary::MergeTheSame(0.01, 1000)**
- Smazat samotné segmenty — **Auxiliary::mergeAlone()**
- Smazat samostatné hranice — **Water::aloneVertices()**
- Převést na barevné schéma — **Auxiliary::ConvertRegion()**

- Obarvit podle křivosti — **Auxiliary::colorRegionsByCurv()**

Zde v tomto místě je možné algoritmus zastavit pro manuální detekci.

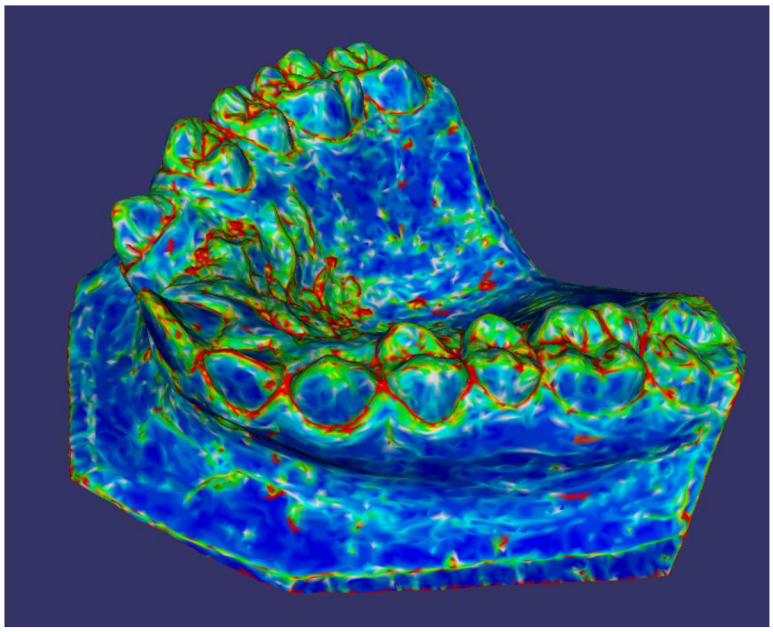
- Označit modré - potenciální zuby — **Auxiliary::BlueAllButBig()**
- Vyrovnat rozdíly — **Auxiliary::Equalize()**
  Auxiliary::MergeBlue()
- Spojit regiony stejné barvy — **Auxiliary::MergeColor()**
- Rozdělit segmenty do seznamu — **Auxiliary::SortBlue()**
- Připojit Gaussovu křivost — **Curvature::BindWithoutShow(GAUSS)**
- Nastavit informaci o mediánu — **Auxiliary::AssignMedian()**
- Filtrovat křivosti blízké 0 — **Auxiliary::CurvatureFilter(0.001)**
- Spojit stejné barvy — **Auxiliary::MergeColor()**

3.6.1 Možná optimalizace

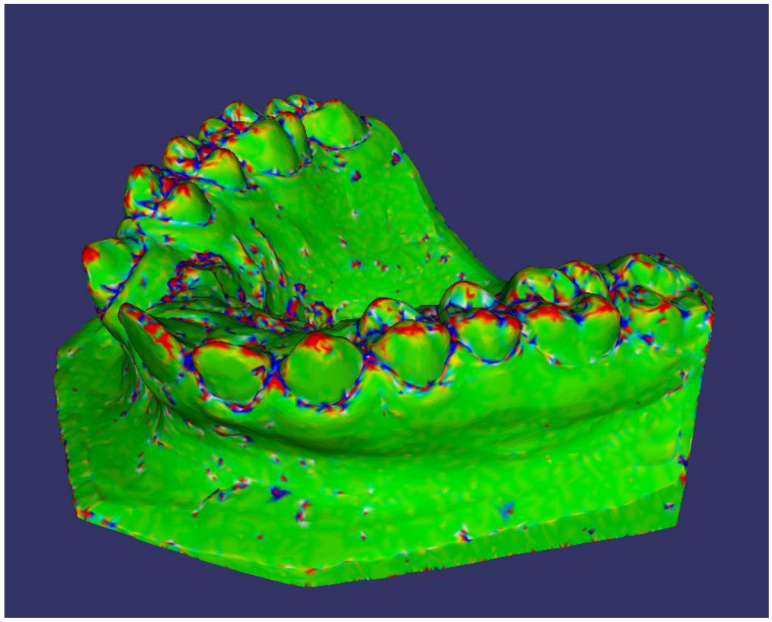
Z navrženého algoritmu je zřejmé, že program by šlo silně optimalizovat pro rychlost. Každá metoda totiž prochází model sekvenčně, v některých případech dochází např. k opakovanému volání řadící funkce, alokování nové paměti místo použití staré nebo zbytečného procházení modelem, a je zcela jasné, že tento přístup značně zpomaluje výpočet. Velké množství funkcí by bylo možné vložit do jedné, a ušetřit tak paměť i drahocenný čas procesoru. Je však důležité mít na mysli, že program byl navržen jako experimentální aplikace s možností nezávislého použití jednotlivých filtrů po sobě. Jak se lze přesvědčit v kapitole Návod k ovládání, program má i experimentální část, kde lze právě tuto vlastnost demonstrovat. Proto žádná optimalizace typu zmenšení počtu průchodů modelem nebyla použita. Je však vysoce pravděpodobné, že při teoretickém nasazení aplikace na trh by se tyto návrhy musely uskutečnit.

4 Dosažené výsledky

Použitelnost křivostí můžete porovnat na následujících obrázcích, demonstrujících graficky jednotlivé výpočty použité ve finální aplikaci.

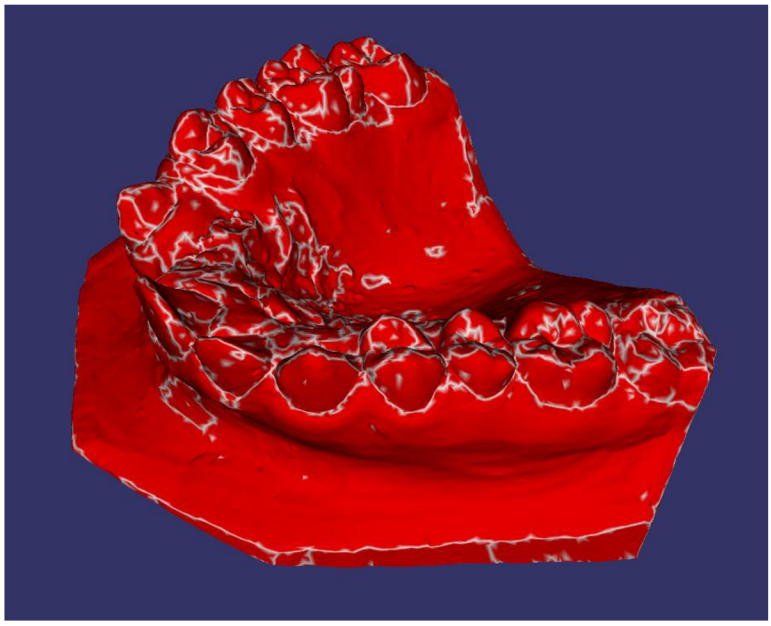


Obrázek 4.1.1 – Střední křivost

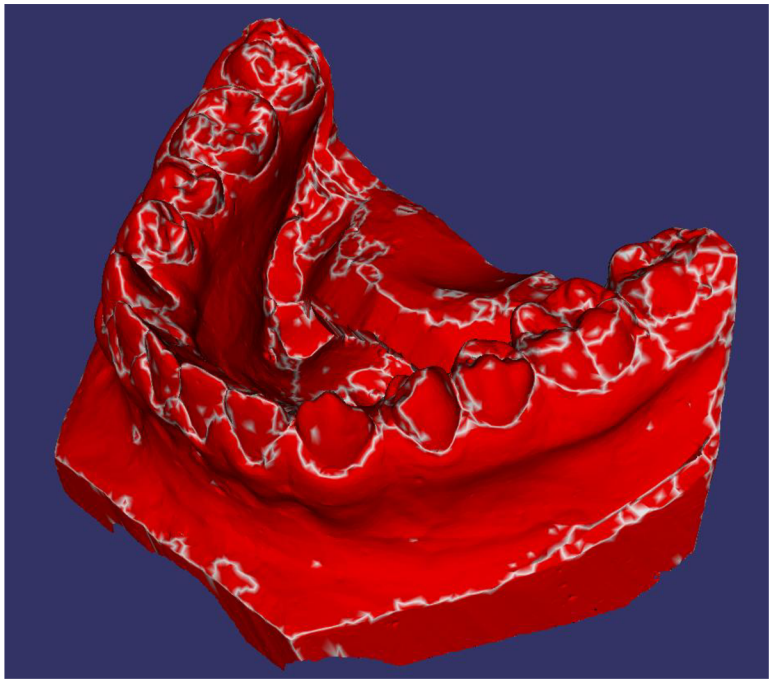


Obrázek 4.1.2 – Gaussova křivost

Na střední křivost je poté zavolána jednoduchá metoda Watershed:



Obrázek 4.1.3 – Model 1, Watershed



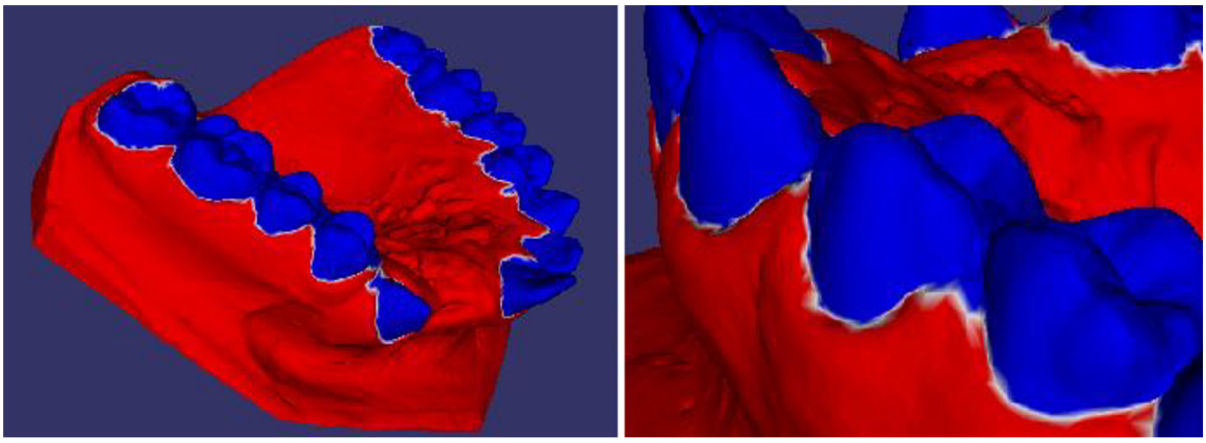
Obrázek 4.1.4 – Model 2, Watershed

Ve finálních výpočtech se na tento základ aplikují veškeré filtry popsané v kapitole 3. Dosažené výsledky detekce popisuji v následujících odstavcích.

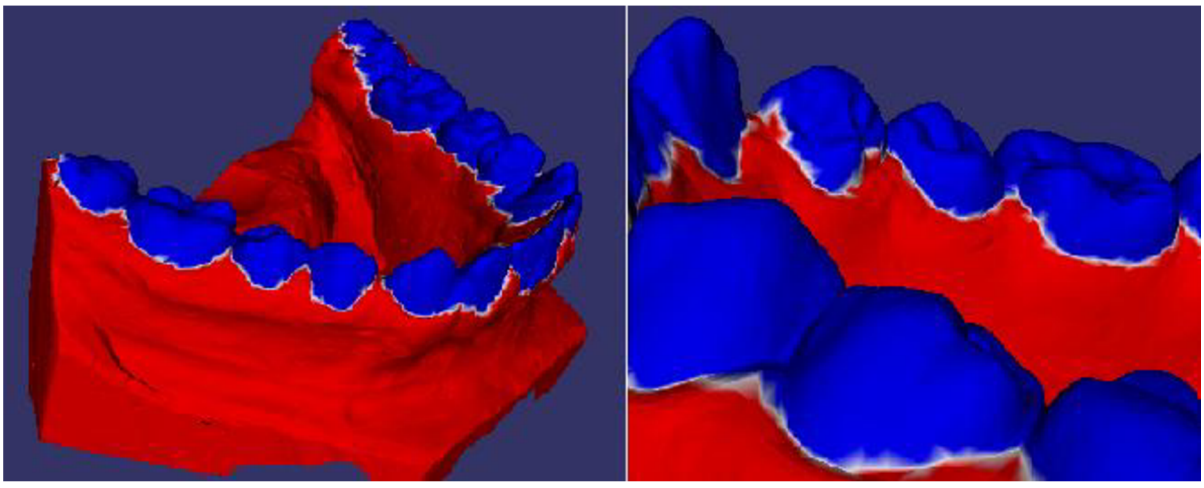
4.1 Finální aplikace

Po aplikování všech zmíněných filtrovacích algoritmů vznikl program ovládaný grafickým uživatelským rozhraním, který zahrnuje jak automatickou detekci zubního oblouku, tak manuální výběr zubů ze segmentovaného modelu.

Na následujících obrázcích demonstruji výsledky mé práce fotografiemi přímo z aplikace po provedené detekci automatickou metodou. Ta je schopna detekované zuby následně uložit do výstupního souboru. Všechny testované modely se svými výsledky podobaly, zajisté by však bylo pro další vývoj aplikace výhodné obdržet větší množství otisků zubních oblouků a aplikaci evaluovat. Zde však narážím na problém časové náročnosti takového úkolu a v této práci se tím již zabývat nebudu.

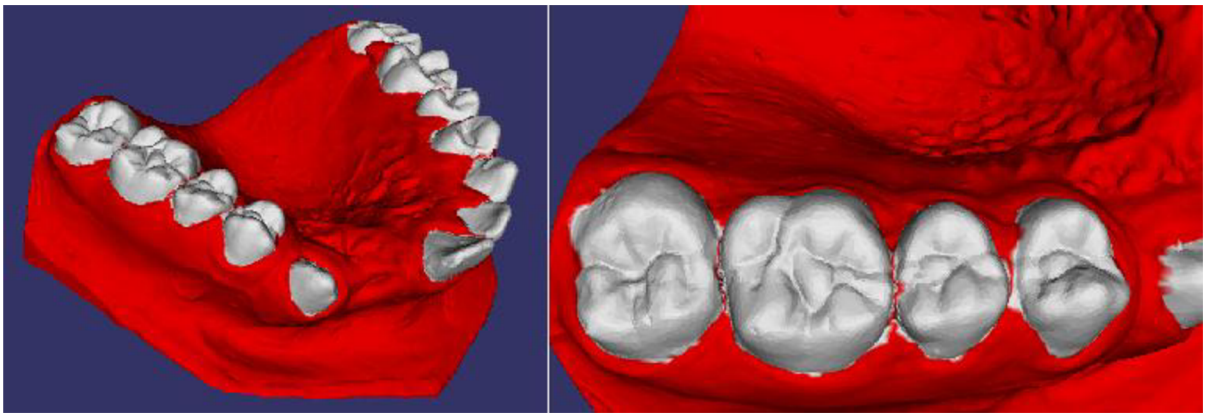


Obrázek 4.1.5 – Model 1, autodetekce

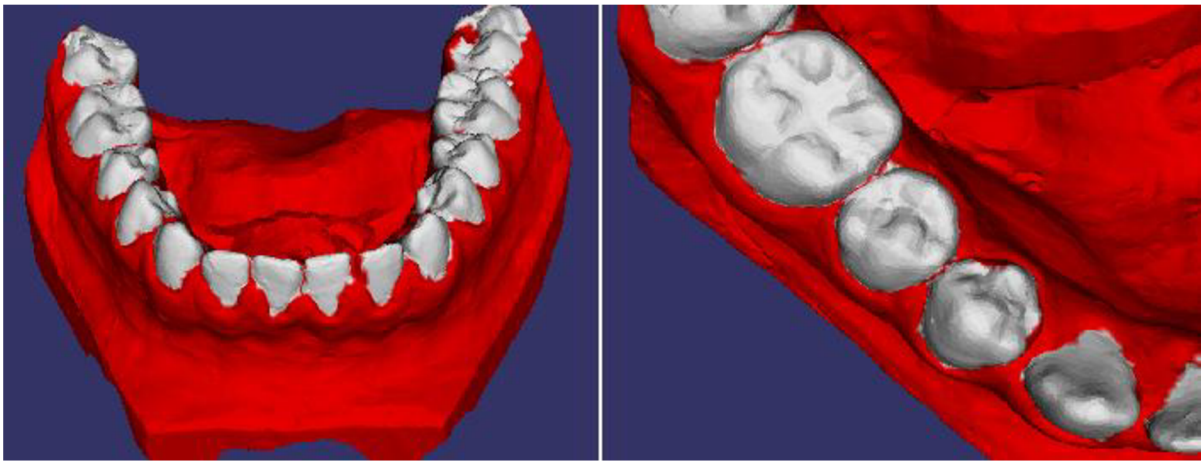


Obrázek 4.1.6 – Model 2, autodetekce

V případě, že se detekce nezdá dostatečně přesná, existuje zde možnost manuálního výběru z detekovaných regionů jednoduchým semínkovým vyplňováním uživatelem. Výsledky demonstrují následující obrázky.



Obrázek 4.1.7 – Model 1, manual



Obrázek 4.1.8 – Model 2, manual

Ze zde uvedených obrázků je patrné, že manuální mód je bezpochyby přesnější. Je však také pro uživatele náročnější, každý zub se skládá v průměru za 4.6 segmentů, které musí uživatel označit. Na druhou stranu jsou vyloučeny automatické chyby typu chybně označené dásně, které vznikají přílišnou snahou o zautomatizování algoritmu. Navíc jsou dostupné dva druhy manuálního režimu, uživatel může buď označit zuby, které chce uložit, nebo naopak zrušit části potenciálních zubů, které mu aplikace nabídne. Je jen na uživateli, která metoda je pro něho rychlejší.

Samozřejmě jsem si vědom, že je možné najít daleko přesnější algoritmus hledání zubů na čelisti jinou kombinací nabízených metod. Proto přikládám k funkční aplikaci testovací mód, ve kterém je uživatel schopen libovolně kombinovat tyto metody (důrazně však doporučuji přečíst si přílohu 1, Manuál programu, hlavně pak sekci ovládání testovacího režimu aplikace).

5 Závěr

V této práci jsem představil návrh a implementaci aplikace pro detekci zubů na polygonálním modelu čelisti. Postup jsem zvolil tak, aby byl co nejobecnější a nejjednodušší, tj. s minimem pevných parametrů. Všechny výpočty by se měly provádět se zřetelem na konkrétní model, ne na pevně daný parametr. Navržený algoritmus podle mého názoru splňuje požadavky na zadání práce, tj. v poloautomatickém režimu lze detekovat zub s chybou maximálně do jednoho milimetru. Samozřejmě pro demonstraci detekčního algoritmu je zde možné spustit i plně automatický mód, který se sice s menší přesností, zato však zcela sám, pokusí najít zubní oblouk.

Program jsem testoval na několika modelech čelistí, všude prokázal uspokojivé výsledky v automatickém hledání. Jsem si však vědom toho, že pro další vývoj by bylo nutné tento program řádně evaluovat, minimálně na několika desítkách modelů. Aplikaci jsem vytvořil v jazyce C++ s pomocí vývojového prostředí MS Visual Studio 2005 a pomocí toolkitů MDSTk a Open Scene Graph (viz kapitola 1).

Pro další vývoj aplikace bych v první řadě provedl zmíněnou evaluaci. Bez té totiž není možno zkontrolovat úspěšnost algoritmu, protože mnou testované množství je nedostačující. Pokud se toto potvrdí, zcela jistě by bylo možné program optimalizovat pro rychlost výpočtů. Z experimentálního důvodu je totiž program značně neefektivní. Jako poslední návrh na budoucí vývoj bych zmínil možnost porovnání detekovaných zubů s již uloženými modely. Poté by bylo zcela jistě možné zuby automaticky i přiřazovat k jednotlivým názvům.

Literatura

[1] Kršek Přemysl, Ing, Ph.D.: přednášky předmětu IZG: https://www.fit.vutbr.cz/study/courses/IZG/private/, 2. února 2008

[2] Adam Dawes: Tokamak Physics: http://www.adamdawes.com/programming/tokamak/04_TerrainMesh.html , 2. února 2008

[3] Wikibooks: Geometrické modelování: http://cs.wikibooks.org/wiki/Geometrick%C3%A9_modelov%C3%A1n%C3%AD, 2. února 2008

[4] Michiel Hazewinkel, Differential Geometry of curves and surfaces: http://eom.springer.de/D/d032170.htm, 2.února 2008

[5] Wikipedia: Křivost plochy: http://cs.wikipedia.org/wiki/K%C5%99ivost_plochy, 2.února 2008

[6] Image Processing and Analysis Group, Meusnier theorem seminar: http://noodle.med.yale.edu/seminar/shi/lecture4.pdf, 9.dubna 2008

[7] Wikipedia: Obrázek Minimal surfaces: http://commons.wikimedia.org/wiki/Image:Minimal_surface_curvature_planes-en.svg, 2. února 2008

[8] Igor V. Florinsky: Digital Terrain Modelling: http://iflorinsky.narod.ru/, 2. února 2008

[9] Nurhan Cetin, Laplacian Smoothing: http://www.inf.ethz.ch/personal/cetin/thesis/thesis/node14.html, 20.března 2008

[10] Gabriel Taubin: Geometric Signal Processing on Polygonal Meshes: http://mesh.caltech.edu/taubin/talks/gsp20001004.pdf , 5. dubna 2008

[11] Michal Španěl, Medical Data Segmentation Toolkit – MDSTk: http://www.fit.vutbr.cz/~spanel/mdstk/, 2. února 2008

[12] Open Scene Graph, http://www.openscenegraph.org/projects/osg, 2. února 2008

[13] Sun-Jeong Kim, Chang-Hun Kim, David Levin, Surface simplification using a discrete curvature norm: http://kucg.korea.ac.kr/~sjkim/paper/cag2002.pdf, 5.dubna 2008

[14] Watershed Segmentation, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/KIM1/Seminars/Watershed/index.htm 5.dubna 2008

Seznam příloh

Příloha 1. Manuál

Příloha 2. CD obsahující:

- dokumentace.pdf
- prog_dokumentace.chm (**doxygen**)
- zdrojové soubory spolu s projektem pro MS VS 2005
- přeložený spustitelný program (včetně ukázkových otisků).
- soubor README s instrukcemi pro překlad a instalaci

Příloha 1: Manuál

Aplikace je tvořena částí příkazové řádky, hlavní část ovládání je však součástí GUI přímo v aplikaci. Nejprve zde popíši argumenty příkazové řádky, poté podrobněji popíši ovládací menu aplikace.

Argumenty příkazové řádky

Aplikace se spouští ve formátu **název argument1 [argument2]**, přičemž jednotlivé části znamenají:

- **název**
  - název aplikace, při základním nastavení kompilace je to **detect.exe**
- **argument1**
  - zde se ve všech případech doplňuje jméno vstupního .stl souboru s modelem
- **argument2**
  - poslední argument je volitelný, tudíž je-li vynechán, spustí se program v uživatelském módu
  - pokud je zde doplněno jméno s informacemi o zubech (standardně $(název modelu).info), aplikace se spustí v uživatelském prostředí
  - pokud je poslední argument tvaru **–test**, spustí se program v testovacím módu

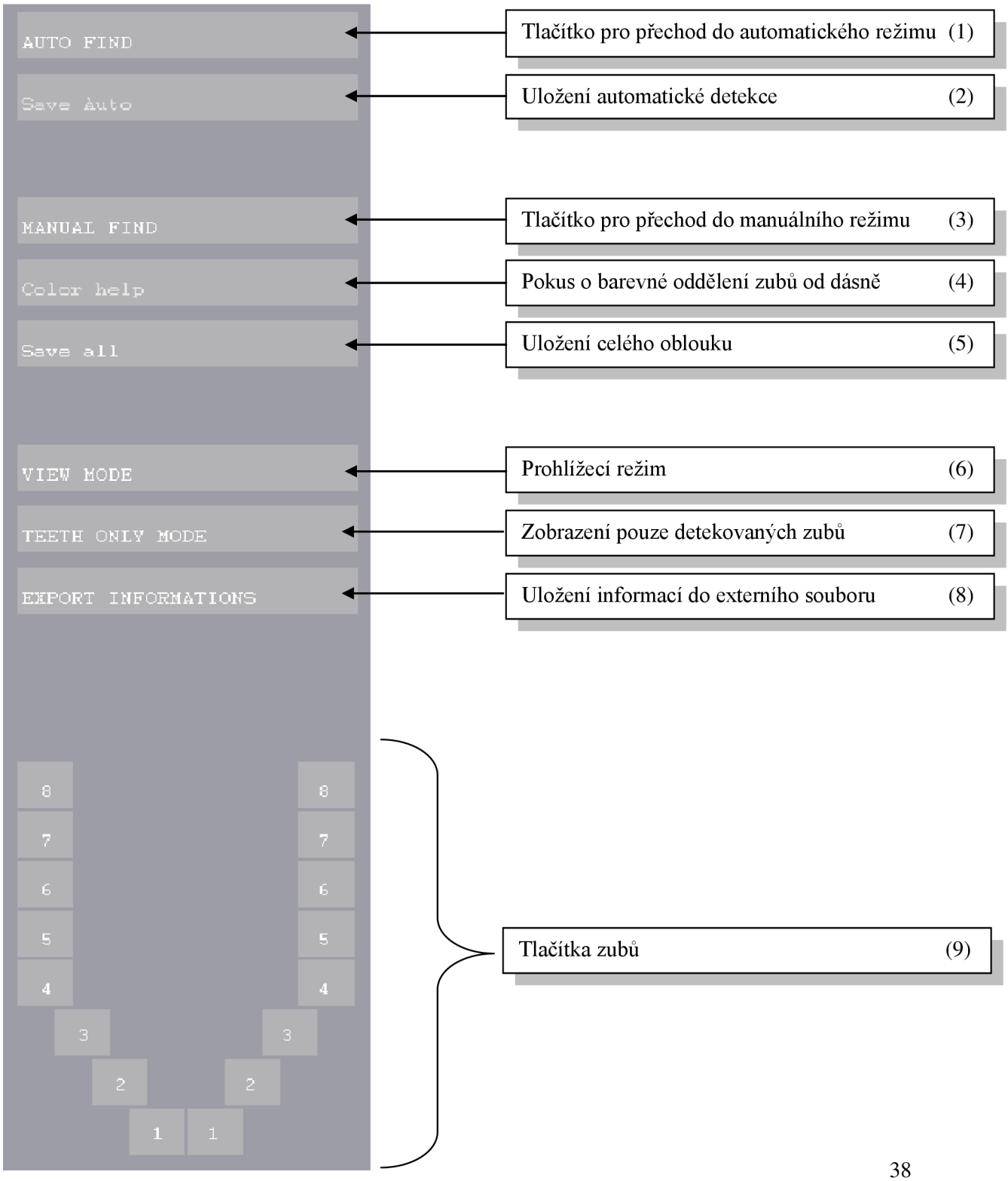
Spouštění programu tedy může vypadat následovně:

detect otisk_1.stl
detect otisk_2.stl otisk_2.stl.info
detect otisk_1.stl –test

Veškeré ostatní ovládání je pak součástí GUI aplikace, popsané níže.

Uživatelský mód

Uživatelský mód slouží pro demonstraci možného použití programu v praxi. Obsahuje nástroje pro automatickou detekci zubů, manuální detekci a různé prohlížecí techniky. Též je možno informace o jednotlivých zubech ukládat a poté exportovat.



Automatický režim

Pro automatickou detekci stiskněte tlačítko (1). Aplikace se pokusí aplikovat vyhledávací algoritmus a označí potenciální zuby modrou barvou. Zbytek čelisti bude označený červeně.

Při stisku uložení automatické detekce (2) se veškeré modré regiony uloží do všech tlačítek zubů (9)(každé tlačítko bude mít přiřazeno celý zubní oblouk, barva tlačítka bude žlutá).

Manuální režim

Zde existují dva možné přístupy. Pro detekci celého oblouku stiskněte tlačítko (3). Model se rozdělí do regionů, který je možno obarvit stiskem tlačítka (4), kde zuby budou označeny modře, zbytek červeně. Zde je možno dvojklikem pravým tlačítkem na určitý region měnit barvu modrá-červená. Po manuálních úpravách a stisknutí tlačítka pro uložení (6) se celá modrá část modelu uloží do všech tlačítek zubů (9) stejným způsobem jako v automatickém režimu.

Pokud je požadavek na uložení jednotlivých zubů do jednotlivých tlačítek(9), po stisknutí manuální detekce (3) levým dvojklikem lze vybrat regiony konkrétního zubu (barva zelená) a ten poté stiskem tlačítka konkrétního zubu (9) uložit. Tlačítko zezelená. **POZOR, po uložení zubu jeho označení zmizí a opětovným stiskem stejného tlačítka zubu uloženou informaci přepíšete (není označeno nic).**

Režim prohlížení

Pokud si přejete detekované zuby prohlížet, lze tak provézt stiskem tlačítka (6), případně tlačítka jakéhokoli zubu (9) v jiném než manuálním režimu (v manuálním se stiskem zubu ukládá). Barva tlačítka udává informaci, která je v něm uložená – bílá = k tlačítku nebylo nic přiřazeno, žlutá = v tlačítku je uložen celý zubní oblouk, zelená = tlačítko obsahuje manuálně vybraný zub.

Při stisku tlačítka (7) lze zobrazovat detekované zuby bez zbytku čelisti.Taktéž lze vybírat pomocí (9)

Uložení informací

Přejete-li si získané informace uložit do externího souboru, slouží k tomu tlačítko (8). Vše, co je uložené v tlačítkách zubů (9), se tak uloží do souboru tvaru **$(název modelu).info**.

Testovací mód

Tato možnost slouží pro pokusné zkoušení jednotlivých funkcí v programu. Obsahuje toto menu:

Poznámky k použití funkcí

- **!POZOR! Tlačítka nereagují graficky na stisk. Po stisknutí např. metody Watershed (06) je možné, že výsledek výpočtu se objeví za sekundu, možná i déle. Proto vyčkejte prosím výsledku výpočtu (znovustisknutí tlačítka během výpočtu může mít fatální následky).**
- Stisk některých tlačítek se nijak neprojeví na modelu, pouze se provede přiřazení např. údajů o křivosti. To je z důvodu zachování již proběhlých filtrů (např. stisk tlačítek 01 – 04 se poprvé vyznačí zobrazením grafické křivosti, po druhém stisku již však pouze obnovuje údaje bez změny vzhledu). Jedná se o tato tlačítka: 01, 02, 03, 04, 05, 16, 17.
- První činností musí být přiřazení křivosti a spočítání Watershed metody – tlačítka 01 – 06. Bez toho ostatní metody nereagují.
- Po stisku tlačítek Color (11, 12) lze používat tlačítka pro práci s barvami, tj. 13-19.
- Pro příklad uvádím sekvenci použitou při výpočtu v automatickém režimu:
    - Přiřadit modelu střední křivost (01)
    - Provést nad modelem modifikovaný Watershed (06)
    - Spojit malé regiony do jednoho (limit 45 vrcholů) (07)
    - Smazat samotné bílé vrcholy (08)
    - Spojit segmenty s jedním sousedem (09) (dokud není změna)
    - Smazat samotné bílé vrcholy (08)
    - Přiřadit modelu Gaussovu křivost vyhlazeného modelu (05)
    - Spojit segmenty s rozdílem křivosti do 0.01 (10)
    - Obarvit regiony podle křivosti (opt. Na Gaussovu) (11)
    - Obarvit modře všechny regiony, které nejsou červené nebo velké (13)
    - Vyrovnat rozdíly mezi modrou a červenou (14)
    - Spojit regiony se stejnou barvou (19)
    - Přiřadit modelu Gaussovu křivost (02)
    - Seřadit pouze modré segmenty do seznamu (15)
    - Přiřadit jednotlivým segmentům medián jejich křivostí (16)
    - Červenit všechny regiony s křivostí blízkou 0 (18)
    - Spojit regiony se stejnou barvou (19)
- Jak jsem se již zmínil v návrhu, tyto funkce jsou psány pro demonstraci jejich vlivu a nejsou optimalizovány pro rychlost. Omluvte prosím dobu výpočtu zejména v automatickém režimu.
- Omluvte též možný pád programu, testování bylo důkladné, ale není možné testovat všechny možné kombinace funkcí (program je ošetřen proti většině nelegálních operací, důrazně však doporučuji prostudovat programovou dokumentaci a zdrojový kód před použitím testovacího módu).