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

JAVA KLIENT PRO 3D ZOBRAZENÍ MEDICÍNSKÝCH OBRAZOVÝCH DAT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ BIREŠ
AUTHOR

BRNO 2009

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

JAVA KLIENT PRO 3D ZOBRAZENÍ MEDICÍNSKÝCH OBRAZOVÝCH DAT

JAVA CLIENT FOR 3D DISPLAING OF MEDICAL IMAGE DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ BIREŠ
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. PŘEMYSL KRŠEK, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá interaktivním zobrazováním 3D medicínských obrazových dat v jazyce Java. Cílem práce je zjistit přínosy využití tohoto jazyka v medicínských vizualizacích a najít vhodné prostředky pro jeho další využití. Součástí práce je vývoj ukázkové aplikace vytvořené v prostředí tohoto jazyka. V závěru práce jsou zhodnoceny přínosy a nevýhody použitých technologií a je navržen další možný rozvoj testovací aplikace.

Abstract

The thesis considers use of Java language in interactive 3D displaying of medical image data. Main point is to find out advantages of using this language in medical visualisations and find suitable tools for its future usage. At the conclusion it evaluates advantages and disadvantages of used technologies and suggests possible development of th appliacation

Klíčová slova

Java, Java 3D, CT, MR, MRI, volumetrická data, objemová data, zobrazení volumetrických dat, zobrazení objemových dat, zobrazení medicínských dat

Keywords

Java, Java 3D, CT, MR, MRI, volumetric data, volumetric data rendering, medical data rendering

Citace

Jiří Bireš: Java klient pro 3D zobrazení medicínských obrazových dat, bakalářská práce, Brno, FIT VUT v Brně, 2009

Java klient pro 3D zobrazení medicínských obrazových dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Přemysla Krška Ph.D. a k jejímu vypracování jsem využil zdroje uvedené v seznamu použité literatury.

.....
Jiří Bireš
20. května 2009

Poděkování

Rád bych poěkoval vedoucímu me práce panu doc. Ing. Přemyslu Krškškovi Ph.D. za odborné konzultace a poskytnuté rady. Dále bych rád poděkoval všem, kteří mi pomohli ať již radou nebo poskytnutou podporou.

Obsah

1 Úvod **3**

2 Rozbor problematiky **4**
2.1 Objemová data . . . 4
2.2 Objemová data v lékařství . . . 5
2.3 Získávání objemových dat . . . 5
2.3.1 Magnetická rezonance . . . 5
2.3.2 Výpočetní tomografie . . . 7
2.4 Uchovávání medicínských dat . . . 8
2.5 Vizualizace objemových dat . . . 8
2.6 Práce s medicínskými daty . . . 9

3 Návrh aplikace **10**
3.1 Úvod . . . 10
3.2 Požadavky na aplikaci . . . 10
3.3 Struktura aplikace . . . 10
3.4 Návrh grafického uživatelského rozhraní . . . 11
3.4.1 Hlavní okno aplikace . . . 12
3.4.2 Volba vhodné 3D grafické knihovny . . . 12
3.5 Návrh programu . . . 13
3.5.1 Třídy objemu a řezů . . . 13
3.5.2 Vytvoření textury z řezu . . . 17
3.5.3 Třídy pro vizualizaci řezů . . . 18

4 Implementace **21**
4.1 Implementační nástroje . . . 21
4.1.1 Jazyk Java . . . 21
4.1.2 Swing . . . 22
4.1.3 Java 3D . . . 22
4.1.4 Vývojové prostředí . . . 24
4.1.5 Systém pro správu verzí . . . 25
4.2 Implementace návrhu . . . 25
4.2.1 Implementace třídy objemu . . . 25
4.2.2 Implementace řezů . . . 26
4.2.3 Okno Hustoty . . . 26
4.2.4 Grafické rozhraní . . . 27
4.2.5 Scéna . . . 27
4.2.6 Generování textury . . . 29

4.2.7 Výkonnostní testy 30
4.2.8 Postranní panel 31
4.2.9 Načítání objemových dat 31
4.2.10 Nápověda programu 31
4.2.11 Testování aplikace 31
4.3 Nedostatky implementace 31

5 Výsledky **33**
5.1 Funkcionalita aplikace 33
5.2 Výkon aplikace 33
5.3 Použité vývojové prostředí 34
5.3.1 Jazyk Java 34
5.4 Java 3D 34
5.5 Vývojové prostředí Netbeans 35

6 Závěr **36**
6.1 Možnosti dalšího rozvoje aplikace 36

A Obsah CD **38**

Kapitola 1

Úvod

Tato bakalářská práce se zabývá vytvořením testovací aplikace pro interaktivní zobrazování prostorových medicínských dat v jazyce Java a volbou vhodných vývojových nástrojů.

Druhá kapitola je stručným úvodem do problematiky objemových dat, možností jejich popisu a reprezentace. Dále se zaměřuje na objemová data v medicíně a věnuje se jejich získávání pomocí moderních vyšetřovacích metod, jako je například výpočetní tomografie nebo magnetická rezonance. Věnuje se také využití těchto dat a jejich dalšímu zpracování, především však jejich vizualizaci, která je pro medicínské aplikace klíčová. Motivací pro vytváření vizualizací objemových dat může být zpřesňování diagnózy, plánování zákroků nebo například předoperační příprava lékařského týmu.

Další kapitola se zabývá návrhem výsledné aplikace, tak aby splňovala požadavky kladené na její funkcionalitu, výkon a stabilitu. Návrh aplikace nelze zanedbat, čas do něj investovaný se odrazí v náročnosti implementace aplikace. Protože Java je objektově orientovaný jazyk, je i výsledný návrh představený ve třetí kapitole objektový. V této kapitole je také představen návrh vzhledu uživatelského rozhraní a nastíněn problém výběru vhodné grafické knihovny. Volba správné grafické knihovny je klíčová pro výkon a použitelnost výsledné aplikace. Jazyk Java neposkytuje ve svém API žádnou možnost vykreslování trojrozměrné grafiky, proto je nutné zvolit některou z externích knihoven.

Čtvrtá kapitola se věnuje popisu jazyka Java, zvolené grafické knihovny a vývojového prostředí použitého pro tvorbu aplikace. Kapitola popisuje chronologický postup implementace všech částí programu a rozebírá problémy, se kterými jsem se setkal při vývoji, a jejich řešení implementovaná ve výsledné aplikaci. Také jsou zde diskutovány výsledky prvních výkonnostních testů aplikace a opatření, která bylo nutné podniknout pro snížení náročnosti aplikace na výpočetní zdroje systému.

Před závěrem práce hodnotím použité nástroje a vývojové prostředí, jakožto vhodnost jazyka Java pro medicínské aplikace na základě údajů získaných při tvorbě a testování mnou vytvořeného programu.

Závěrem je vyhodnocen celkový přínos implementace programu a tohoto textu. V této části je také navrženo několik možných směrů dalšího rozvoje aplikace, které by mohly přinést další informace o možnostech využití jazyka Java v medicínských aplikacích.

Kapitola 2

Rozbor problematiky

V této kapitole je vysvětlen pojem objemová data, rozebráno jejich využití pro lékařské účely a vhodné způsoby jejich získávání. Dále kapitola uvádí možnosti reprezentace nasnímaných dat a jejich další analýzy. Obecné informace o objemových datech a jejich vizualizaci pochází především z [15], části o využití objemových dat v medicínských aplikacích pak převážně ze zdrojů [7], [2] a [8].

2.1 Objemová data

Objemová data jsou forma popisu prostorových dat. V současné počítačové grafice se pro reprezentaci objemových dat nejčastěji využívají dva přístupy, hraniční a objemová.

Hraniční reprezentace dat využívá k popisu dat plošné útvary, tzv. polygony, kterými reprezentuje hranice mezi jednotlivými prostředími. Objekt je tak popsán množinou ploch kopírujících povrch objektu. Tento popis prostorových dat je známý především z počítačových her a 3D animací vytvářených pomocí animačních programů jako je např. 3D Studio Max nebo Cinema 4D. V současné době už i do těchto odvětví proniká druhý způsob reprezentace prostorových dat, data objemová.

Objemová data se soustřeďují nikoliv na popis hranic objektů, ale na reprezentaci objemu těles. Mají široké využití ve vědeckých simulacích, medicínských aplikacích, při vizualizaci naměřených dat v meteorologii, chemii, seizmologii a dalších vědních oborech, využití nachází i v průmyslu, kde je díky nim možné odhalovat defekty výsledných výrobků.

Objem lze vyjádřit různými technikami, uveďme několik příkladů:

- CSG (Constructive Solid Geometry) je způsob popisu těles pomocí primitivních těles, jako jsou například koule, kvádry, jehlany, a logických operací mezi nimi. Tato technika je vhodná pro popis těles například ve strojírenství. Ukázka objektu reprezentovaného pomocí CSG je na obrázku 2.1.

- Implicitní plochy popisují tělesa funkcemi, ze kterých lze vypočíst hustotu materiálu tělesa v každém bodu v prostoru. Implicitní plochy jsou vhodné např. pro reprezentaci tekutin, elektrostatických nebo magnetických polí kolem těles. Výhodou této formy popisu je především paměťová nenáročnost metody a možnost popsat i některá velmi složitá tělesa pomocí relativně jednoduchých rovnic. Vizualizace pomocí implicitních ploch je uvedena v příkladu na obrázku 2.2.

- Popis tělesa objemovými elementy, ty si lze představit jako rozšíření dvojrozměrné reprezentace objektů pomocí mřížky do trojdimenzionálního prostoru. Data v mřížce

mohou být popsána hodnotami vrcholů mřížky (buněk) nebo voxelů. Ukázka těchto dvou přístupů zjednodušená do dvojrozměrné mřížky je na obrázku 2.3.

Pro medicínské aplikace se využívá poslední zmíněná metoda popisu těles, tato práce se tedy věnuje pouze této metodě. Voxely slouží k popisu hustoty tkáně. V uniformní mřížce mají všechny voxely stejnou velikost. Data se po nasnímání mohou různě zpracovávat, lze je vyhlazovat, upravovat jejich kontrast, převzorkovávat, také je možné je dále segmentovat, analyzovat, nebo z nich vytvářet modely.

2.2 Objemová data v lékařství

V lékařství mohou mít objemová data mnohé využití ať již ke studiu, diagnostice, nebo předoperační přípravě. Pro lékaře může být neocenitelná možnost připravit se na vykonání zákroku předem nebo naplánovat náročnou operaci, pro takovou přípravu je kritické udělat si představu o příčinách stavu pacienta. K tomu lze využít řadu neinvazivních metod.

Nejznámějším a nejstarším způsobem radiologického vyšetření je dozajista rentgen, který je hojně využíván v diagnostice. Tato metoda poskytuje plošný snímek tkáně, která utlumuje intenzitu záření.

Prudký pokrok v oblasti výpočetní techniky v uplynulých letech způsobil rozvoj i dalších neinvazivních vyšetřovacích metod, které jsou s ní pevně svázané. Jako příklad můžeme uvést výpočetní tomografii, neboli CT (z angl. computer tomoghraphy) a magnetickou rezonanci, neboli MR (z angl. magnetic resonance). Výhodou těchto metod oproti již zmíněnému rentgenu je především to, že poskytují nikoliv plošný snímek ale prostorový obraz tkáně, který může být dále různě zpracováván a reprezentován.

2.3 Získávání objemových dat

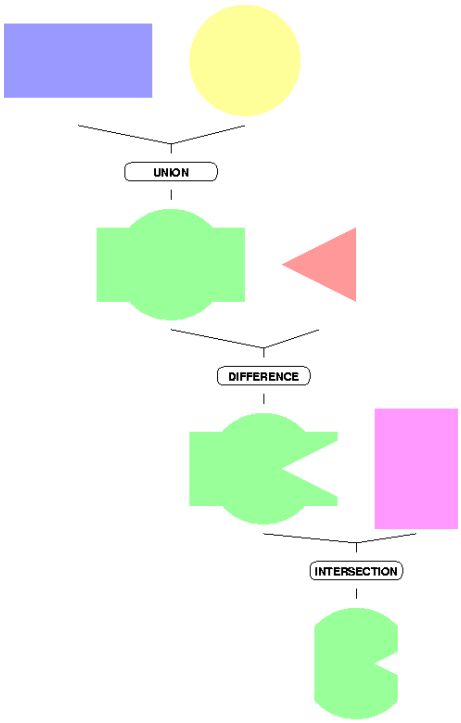
Vhodná objemová data je možné získávat celou řadou metod, v následující části popisuji dvě technologie často používané v moderní medicíně.

2.3.1 Magnetická rezonance

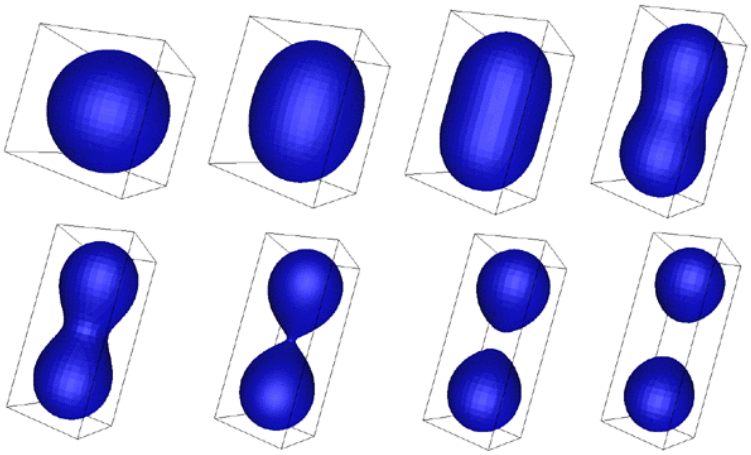
Magnetická rezonance (zkráceně MR) využívá principu fyzikální spektroskopické metody měření nukleární magnetické rezonance (NMR), která ve frekvenčním magnetickém poli detekuje atomová jádra s nenulovým jaderným spinem. Jednoduše řečeno jádra s lichým počtem nukleonů, kterými jsou protony a neutrony.

Atom vodíku, který je hlavní složkou organických molekul spolu s uhlíkem, má jádro tvořeno pouze jedním protonem. Díky tomu MR detekuje veškeré živé tkáně, což je hlavní výhoda této metody oproti rentgenovým metodám, které ve většině případů vyžadují kontrastních látek, protože většina tkání je pro rentgenové záření neviditelná. Navíc magnetické pole ve srovnání s rentgenovým zářením, alespoň podle dosavadních studií, nepůsobí rizikově na organizmus. Nevýhodou MR jsou vyšší pořizovací náklady na vybavení laboratoře.

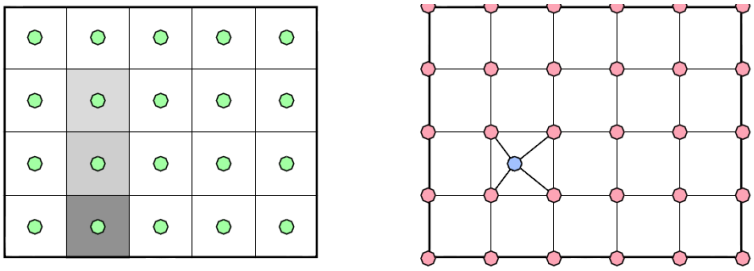
Vyšetření metodou MR probíhá ve vyšetřovacím tunelu, kde je pacient vystaven působení silného magnetického pole. Toto pole uvádí jádra atomu vodíku, vázaných v tkáních, do stavu s vysokou energií (jaderného excitovaného stavu). Po ustání působení magnetického pole se začnou jádra vracet různou rychlostí zpět do původního stavu, přičemž vyzařují energii. Ta je měřena cívkami, ve kterých se indukuje elektrický proud. Informace o jeho velikosti je dále zpracována a lze z ní vytvořit obraz snímaných tkání, jako je např. na



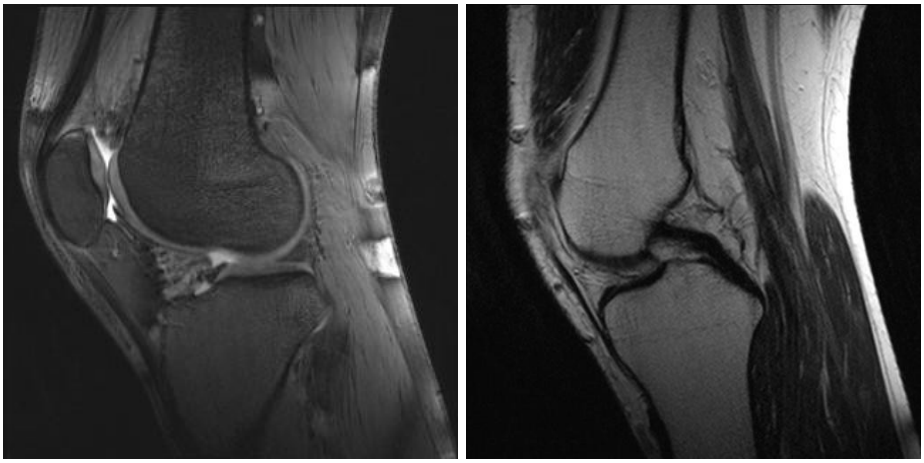
Obrázek 2.1: Ukázka jednoduchého objektu vytvořeného metodou CSG. [6]



Obrázek 2.2: Obrázek znázorňuje oddalování dvou kontrolních bodů a výpočet jimi vytvořeného povrchu metodou implicitních ploch. [1]



Obrázek 2.3: Rozdíl v reprezentaci objemu pomocí voxelů (vlevo) a buněk (vpravo). [9]



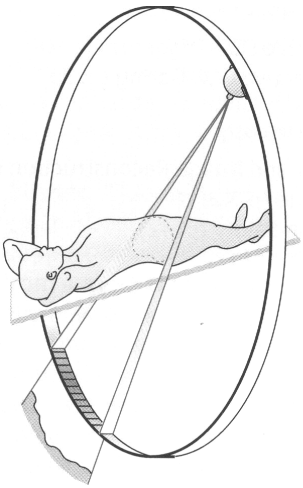
Obrázek 2.4: MR snímky kolena. [7]

obrázku 2.4. Výsledná data nasnímaná pomocí CT poskytují vyšší tkáňový kontrast oproti MR.

2.3.2 Výpočetní tomografie

Výpočetní tomografie vznikla v 70. letech. Je to metoda využívající pro snímání pacientova těla rentgenové záření. Výhodou CT oproti rentgenu je schopnost snímat konkrétní část těla, aniž by se výsledný obraz překrýval s dalšími tkáněmi, skrze které prošel rentgenový paprsek, což zlepšuje čitelnost výsledných dat a umožňuje nám to složit z nasnímaných hodnot prostorová data.

Výpočetní tomografii lze využít i pro angiografické vyšetření. Při CT angiografii je pacientovi vstříknuta do krevního řečiště kontrastní chemická látka, která zvyšuje viditelnost cév na výsledných snímcích.



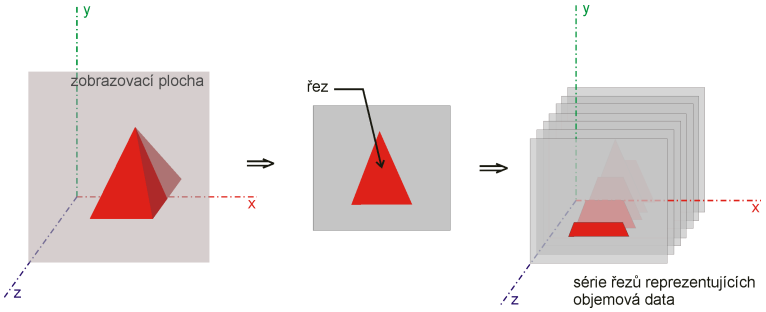
Obrázek 2.5: Princip CT: pacient je prozařován otáčející se rentgenkou. [7]

Přístroj pro výpočetní tomografii má tvar prstence, kde po kruhové trajektorii obíhá rentgenový zářič (tzv. rentgenka) a prozařuje snímaný objekt. Při průchodu rentgenového záření tkání objektu se jeho intenzita snižuje v závislosti na schopnosti tkáně toto záření

tlumit, k tomu dochází nejvíce v kostech, zatímco v měkkých tkáních, jako jsou vrstvy tuku nejméně. Záření dopadá na snímače, kde je vyhodnocována jeho intenzita. Objekt je takto nasnímaný z mnoha úhlů a z výsledných intenzit jsou vypočítány hustoty tkání v každé části snímaného objektu, které jsou uloženy v prostorové matici. Z této matice lze vygenerovat sadu řezů tkáněmi snímaného objektu.

2.4 Uchovávání medicínských dat

Objemová data lze ukládat jako jediný soubor obsahující celý objem dat nebo jako kolekci dvojrozměrných snímků, z nichž každý zobrazuje jeden řez objemem.



Obrázek 2.6: Uložení objemových dat jako série snímků.

Pro uchovávání objemových medicínských dat jsou běžné formáty pro ukládání obrazových dat nevhodné, protože nemusí být schopné uchovávat všechny potřebné informace, proto se k těmto účelům používají specializované formáty.

Jedním z těchto formátů je standard DICOM (Digital Imaging and Communications in Medicine). Specifikace tohoto formátu udává nejen formát ukládání souborů, ale i protokoly pro výměnu informací po síti. Formát souborů DICOM umožňuje ukládat kromě obrazových dat v různé kvalitě i textová metadata, např. jméno pacienta, informace o uložených snímcích, nastavení přístroje atd. Aktuální verze formátu DICOM je 3.0.

2.5 Vizualizace objemových dat

> „Vizualizace volumetrických dat je zpusob, respektive skupina technik, jak získat podstatné informace z bloku volumetrických dat, které bývají pri zobrazení standardními metodami skryty.“ [8]

Objemová data lze vizualizovat různými technikami, volba způsobu vizualizaci závisí na účelu vizualizace a na prostředcích, které máme k dispozici.

Základní metodou zobrazování je vykreslování snímků získaných při vyšetření. Tato metoda je nejjednodušším a nejméně náročným způsobem zobrazování objemu. Nevýhodou této metody je, že lze zobrazit pouze plošné řezy a je tedy pro uživatele obtížné udělat si přehled o objemu jako celku.

Další možností je složit z nasnímaných řezů objem, tímto objemem vést libovolné roviny a vizualizovat data, která se na ně promítnou. Tyto zobrazovací plochy je možné umístit do prostoru, aby uživatel získal lepší představu o vizualizovaných objemech. Tento způsob zobrazení používá mnou navržená aplikace.

Další skupinou metod pro zobrazování objemů jsou metody hledající povrch. Tyto metody hledají v objemu prahy, kde hodnota hustoty dosahuje určité úrovně a z nich pak vytvářejí plochy. Tyto plochy jsou následně pokryty sítí polygonů, která může být vykreslena pomocí běžných technik používaných v 3D grafice. Nevýhodou metod je nemožnost zobrazovat objekty, které nejsou pevné, například tekutiny či plynné útvary, tato nevýhoda je však vykoupena jejich efektivností, jakmile je jednou vytvořen polygonální model probíhá vykreslování velmi rychle, v mnoha odvětvích není potřeba vykreslovat amorfní objekty a tato metoda pro ně může být vhodná. U těchto metod však může docházet k chybám ve vizualizaci dat, drobné objekty nemusí být zobrazeny a mohou se objevovat artefakty v polygonální síti modelu.

Do další skupiny metod patří tzv. přímé zobrazovací techniky. Ty se vyhýbají konstrukci těles z objemových dat a snaží se objem vykreslovat přímo. Tyto metody se mají za cíl odstranit artefakty, které mohou vznikat v metodách využívajících hledání povrchu. Přímé vykreslování objemu umožňuje zobrazovat i amorfní objekty. Metody většinou pracují na základě vrhání paprsku a sumarizace hustoty tkáně protnuté tímto paprskem. Nevýhodou těchto metod je především jejich vysoká výpočetní náročnost. Více o těchto metodách lze nalézt v [8].

Existují i grafické akcelerátory pro zobrazování objemových dat, jedná se však o velmi specializovaný hardware, tomu odpovídá i cena a zastoupení těchto akcelerátorů na trhu. Příkladem takového akcelerátoru může být karta VolumePro 500, viz [14].

Při vizualizaci je také důležité správná reprezentace hustoty tkáně. Často se pro zobrazování dat využívá stupnice šedé barvy, ta poskytuje 256 odstínů včetně černé a bílé. Data z CT nebo MR snímků však mohou využívat palety s vyšším počtem hodnot, obvykle se pro jejich uložení využívá 12 nebo 16 bitových hodnot. Lineární převod hustoty tkáně na barevnou informaci by způsobil významnou redukci detailů a znemožnil by odlišit od sebe některé tkáně s podobnou úrovní hustotu. Pro převod z hustoty tkáně na barevnou informaci je možné využít tzv. *Windowing*. Při využití této techniky jsou definovány parametry pro tabulku převodu hustoty tkáně na barevnou informaci a při zobrazování tkání dochází k převodu podle této tabulky. Česky se tato tabulka nazývá okno hustoty. Okno hustoty může definováno svým středem a šířkou, tento přístup je velmi intuitivní a často využívaný. Změnou parametrů okna hustoty můžeme dosáhnout generování různých snímků, můžeme vizualizovat celý rozsah nebo se zaměřit pouze na jeho část a získat tak detailnější informace pouze o ní. Tento přístup ke generování barevné reprezentace tkání bude použit i v mnou navržené aplikaci.

2.6 Práce s medicínskými daty

Důležitou součástí práce s medicínskými daty je jejich zpracovávání a segmentace. Zpracováním dat mám na mysli především jejich obrazové úpravy, například vyhlazování šumu, nebo detekci hran objektů. Tyto operace se provádí s daty pro vizualizaci.

Kromě dat vizualizačních, lze na základě hustoty tkáně z medicínských dat vygenerovat objem segmentační. Ten slouží k přiřazení voxelů k jednotlivým typům tkáně, lze tak oddělit například kůži, svalovou hmotu a kosti. Segmentace je netriviální proces, který může být vykonáván jak ručně tak automaticky s pomocí vhodných algoritmů, tak i jako kombinace těchto přístupů. Detailní popis segmentace je nad rámec této práce, více informací lze nalézt například v [4].

Kapitola 3

Návrh aplikace

3.1 Úvod

Tato kapitola uvádí požadavky na aplikaci stanovené na začátku vývoje. I přes to, že aplikace není určena pro praktické použití, je důležité tyto požadavky implementovat tak, aby bylo možné vyzkoušet vhodnost jazyka Java pro vývoj reálných aplikací pro práci s objemovými daty. Dále se kapitola věnuje návrhu grafického rozhraní a aplikační logiky pro práci s objemovými daty a generování řezů. Kvalitní návrh může v budoucnu ušetřit problémy při rozšiřování aplikace, proto je důležité věnovat mu dostatek úsilí a času, především návrhu zobrazovací části, která je závislá na knihovnách hostitelského systému, a proto musí být navržena nezávisle na použité grafické knihovně.

3.2 Požadavky na aplikaci

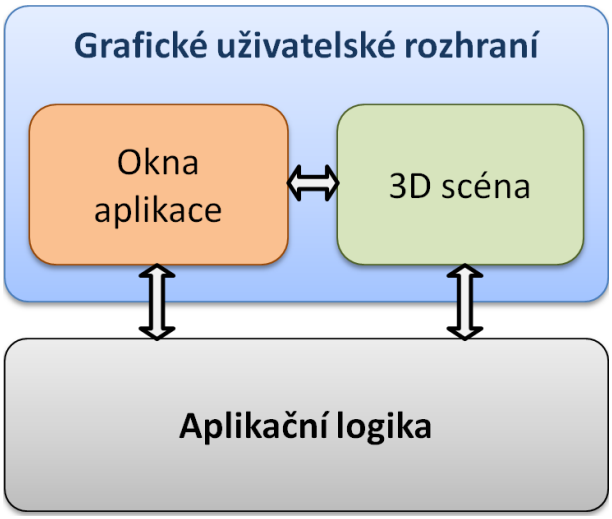
Následující seznam shrnuje základní požadavky, které by měla výsledná aplikace splňovat. V průběhu vývoje vyvstávaly další nároky, ať již funkční nebo výkonnostní, které zde nejsou uvedené a bude jim věnován prostor až v kapitole věnující implementaci aplikace.

- Aplikace pro zobrazení medicínských dat ve pomocí zobrazovacích ploch.
- Načítání reálných medicínských dat.
- Práce se zobrazovacími plochami - pohyb, přidávání, odebírání.
- Práce se scénou - pohyb kamery, otáčení scénou, zoom.
- Implementace densitního okna.
- Implementace v jazyce Java.
- Využití vhodné knihovny pro práci s 3D grafikou.
- Výsledný program musí být multiplatformní.

3.3 Struktura aplikace

Pro budoucí vývoj aplikace je nezbytné udělat si představu o její struktuře a rozvržení funkcionality do oddělených vzájemně komunikujících celků. Z obrázku 3.1 je vidět, že

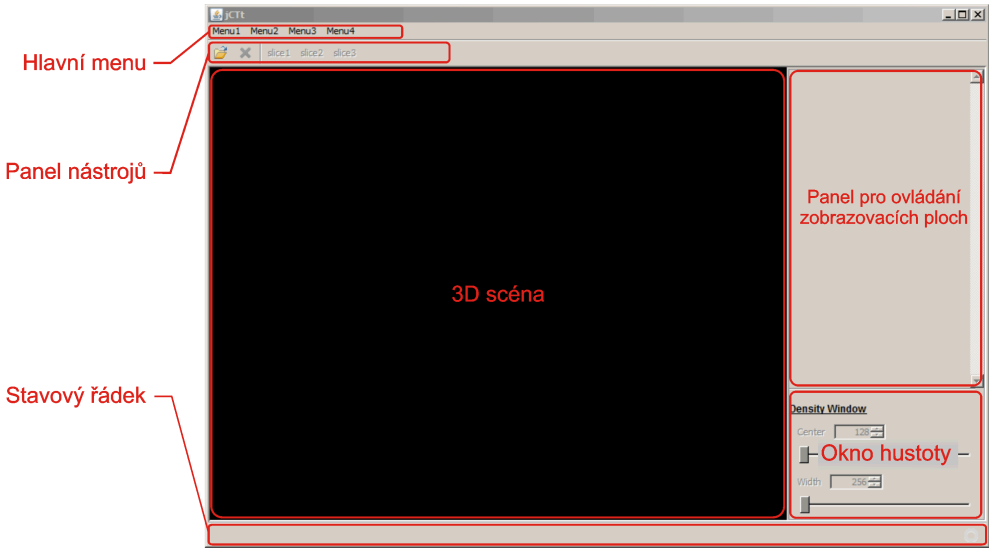
aplikace je rozdělena do dvou vrstev - grafického rozhraní a aplikační logiky. Aplikační logika bude obsahovat části zapouzdřující práci s objemovými daty nezávisle na jejich grafické reprezentaci. Vrstva grafického rozhraní se skládá z dvou částí, první z nich tvoří hlavní okno aplikace, dialogy a další prvky kromě 3D scény, která je druhou částí grafického rozhraní. Důvody a přínosy tohoto řešení budou uvedeny ve zvláštní kapitole.



Obrázek 3.1: Struktura aplikace.

3.4 Návrh grafického uživatelského rozhraní

Z pohledu uživatele je jedním z nejdůležitějších požadavků na aplikaci pohodlné intuitivní grafické rozhraní, proto je při návrhu potřeba zaměřit se i na tento aspekt vývoje. Jak již bylo uvedeno v kapitole věnující se struktuře aplikace, grafické rozhraní se skládá ze dvou částí, okna aplikace a pohledu na 3D scénu, oběma z nich se budeme věnovat v této kapitole.



Obrázek 3.2: Uživatelské rozhraní.

Prvotní návrh grafického rozhraní je patrný z obrázku 3.2. Výsledné vzhled nebo funkčnost grafického rozhraní se může v průběhu práce ještě změnit v závislosti na další funkcionalitě programu.

3.4.1 Hlavní okno aplikace

Hlavní menu aplikace bude poskytovat funkcionalitu pro práci s daty, především pro načítání, popřípadě generování náhodných objemů pro testovací účely, dále přístup k nastavení programu a nápovědu k aplikaci.

Nejčastěji používané funkce programu je vhodné umístit do panelu nástrojů, kde je bude mít uživatel rychle k dispozici. Mezi ně patří především funkce pro práci se scénou, přidávání a odebírání zobrazovacích ploch.

Boční menu umístěné vpravo obsahuje panel pro práci s oknem hustoty a kontejner pro ovládací panely ploch. Tento kontejner umožňuje posouvání pro případ, že by těchto panelů bylo víc, než je schopen zobrazit najednou. Panely ploch slouží pro nastavování parametrů souvisejících s konkrétním zobrazovací rovinou, jako je například její pozice. Panel okna hustoty slouží k nastavování parametrů funkce pro přepočet hustoty tkáně na barevnou informaci. Funkce vyžaduje nastavení středu okna a jeho šířky. V případě potřeby dalších ovládacích prvků lze boční panel rozšířit o záložky a rozdělit ovládací prvky do záložek.

Stavový řádek bude využit především pro tisk zpráv o průběhu déle trvajících operací a dalších informací pro uživatele.

Největší část grafického rozhraní bude věnována panelu se zobrazením pohledu na vizualizaci řezů pomocí zobrazovacích ploch. Pro implementaci tohoto panelu je potřeba zvolit vhodnou grafickou knihovnu, tomu se bude věnovat následující část textu. Z hlediska uživatelského rozhraní je důležité ovládání scény, to můžeme rozdělit na 2 části - ovládání kamery a ploch. Pohybem kurzoru a držením levého tlačítka myši bude možné otáčet kamerou kolem pevného středu. Pozici tohoto středu lze ovlivňovat pohybem myši při přidržení pravého tlačítka. Přibližování a oddalování kamery lze nastavovat buď kolečkem myši, nebo pohybem myši zároveň s držením prostředního tlačítka. Zobrazovacími plochami lze pohybovat přidržením klávesy *ctrl*, výběrem plochy levým tlačítkem a pohybem myši.

3.4.2 Volba vhodné 3D grafické knihovny

Výběr správné knihovny pro vykreslování 3D grafiky je klíčový pro dodržení požadavků kladených na aplikaci, především na platformovou nezávislost programu, i pro správný návrh aplikace zohledňující všechny možnosti, které nám použitá knihovna nabízí. Na rozličných platformách a operačních systémech mohou být k dispozici různá rozhraní pro vykreslování akcelerované grafiky, na systému Windows je to například DirectX nebo OpenGL, na Solarisu a systémech založených na Linuxu je to jen OpenGL. Naše požadavky by tak lépe splňovalo OpenGL, avšak firma Sun zveřejnila knihovnu Java 3D, která umožňuje programovat aplikace využívající akcelerovanou 3D grafiku v Javě při zachování výhod Javy, tj. objektově orientovaného přístupu a široké podpory platforem.

Knihovna Java 3D je dostupná pro operační systémy Windows, Mac OS, Solaris a Linux, přičemž na každém z nich využívá nejvhodnější dostupné rozhraní pro vykreslování, například na systému Windows bude ve většině případů využívat DirectX, zatímco na Linuxu OpenGL.

Pro implementaci aplikace tedy použijeme knihovnu Java 3D. Při návrhu je i přes to vhodné zachovat odstup od přílišné závislosti na této knihovně, protože se může stát, že

v průběhu implementace se knihovna ukáže jako nevhodná a bude potřeba využít místo ní jinou, vhodnější.

3.5 Návrh programu

3.5.1 Třídy objemu a řezů

Při návrhu práce s objemovými daty se nejprve musíme zaměřit na reprezentaci voxelu. Voxel je analogií pixelu v třídimenzionálním prostoru, můžeme si ho představit jako krychli nebo kvádr. Pro reprezentaci medicínských dat v této aplikaci potřebujeme uchovávat ve voxelu informaci o hustotě tkáně a o jeho velikosti. Velikost voxelu můžeme definovat jeho šířkou, výškou a délkou, přičemž tyto parametry mohou být různé, avšak musí být stejné pro všechny voxely v objemu. Informaci o velikosti voxelu je tedy zbytečné ukládat pro každý voxel zvlášť a může být uchovávána v objemu.

Pokud by aplikace pracovala i se segmentačním objemem, bylo by vhodné zvážit výhody uložení voxelu v objektovém typu, který by uchovával jak informaci o hustotě, tak i segmentační data. Tato aplikace však používá pouze data hustoty tkáně. Reprezentace voxelu objektem je pro nás nevýhodná, protože všechny objekty v Javě dědí od základního objektu *Object*, což by zvětšilo velikost paměti nutné pro uložení voxelu, snížilo rychlost přístupu k voxelu a výsledný přínos by byl diskutabilní. Výhodnější pro naše použití je reprezentace voxelu primitivním bezznaménkovým datovým typem s velikostí alespoň dvanáct bitů, což je dostatečné pro uložení hodnot v rozsahu 0 až 4096.

Pro reprezentaci samotného objemu můžeme využít některou z následujících tří možností:

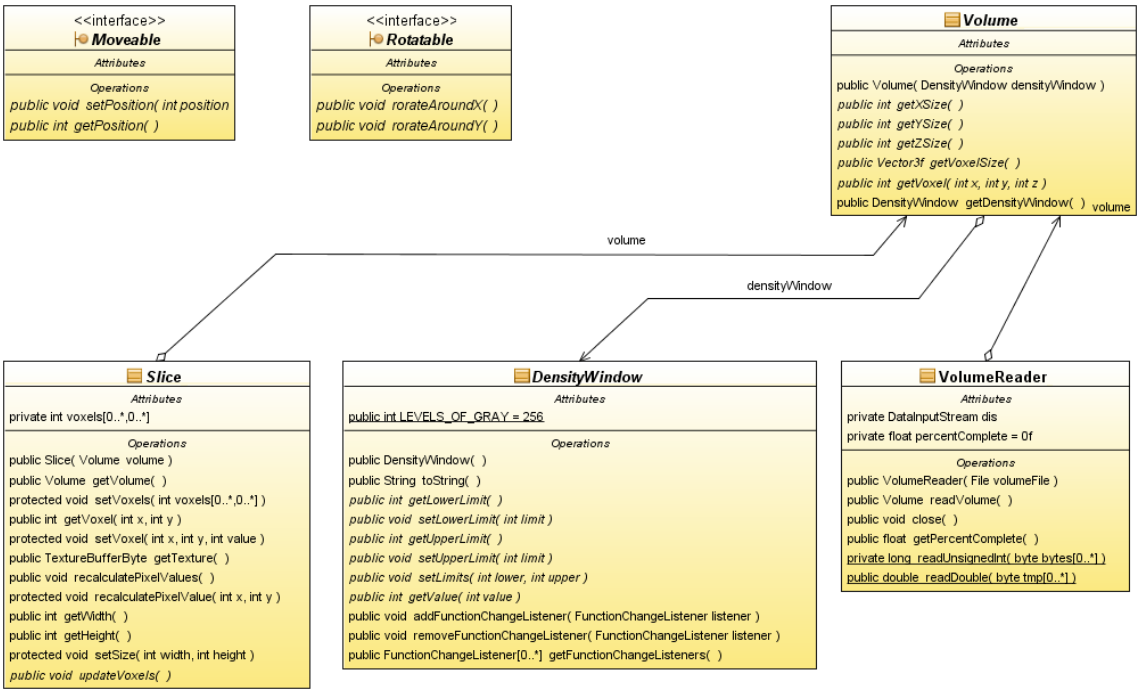
- Trojrozměrné pole primitivních datových typů - výhodou je intuitivní přístup k voxelům v objemu a snadné kopírování pole pro řezy, které je realizovatelné zachováním jednoho indexu třírozměrného pole a měněním zbylých dvou. Nevýhodou je pomalejší přístup k prvkům pole, než by tomu bylo u ostatních navržených řešení.

- Jednorozměrné pole primitivních datových typů s mapovací funkcí - výhodou je rychlejší přístup k prvkům než v případě trojrozměrného pole, ostatní výhody předchozího návrhu zůstanou zachovány díky mapovací funkci. Tu lze využít pro přepočet souřadnic v trojrozměrném prostoru na indexy prvků jednorozměrného pole.

- Nativní knihovna pro práci s objemovými daty - výhodou tohoto řešení je jeho vysoká rychlost a přímá kontrola nad přidělovanou pamětí a její správa programátorem, nikoliv běhovým prostředím Javy. Největší nevýhodou tohoto řešení je právě použití nativního kódu, protože bychom tím ztratili výhody přenositelnosti programu. Další nevýhodou je zároveň výhoda tohoto řešení, a to nutnost spravovat paměť ručně, mohou tak vzniknout další problémy s přístupem do již uvolněné paměti nebo naopak neuvolňováním paměti již nepotřebné. Pro zachování výhod nativní knihovny je potřeba naprogramovat ji v některém z jazyků nižší úrovně, například C, C++ nebo assembleru. S takovou knihovnou lze v Javě pracovat s využitím JNI (Java Native Interface) nebo JNA (Java Native Access).

Jako vhodný kompromis mezi výkonem a náročností implementace se jeví uchovávání objemových dat jako jednorozměrného pole s mapovací funkcí. V případě, že by toto řešení

nebylo po výkonové stránce dostačující, je při správném návrhu možné uchýlit se k třetímu popsanému řešení navrženém výše, tj. nahradit tuto část aplikace nativní knihovnou napsanou v nižším programovacím jazyce.



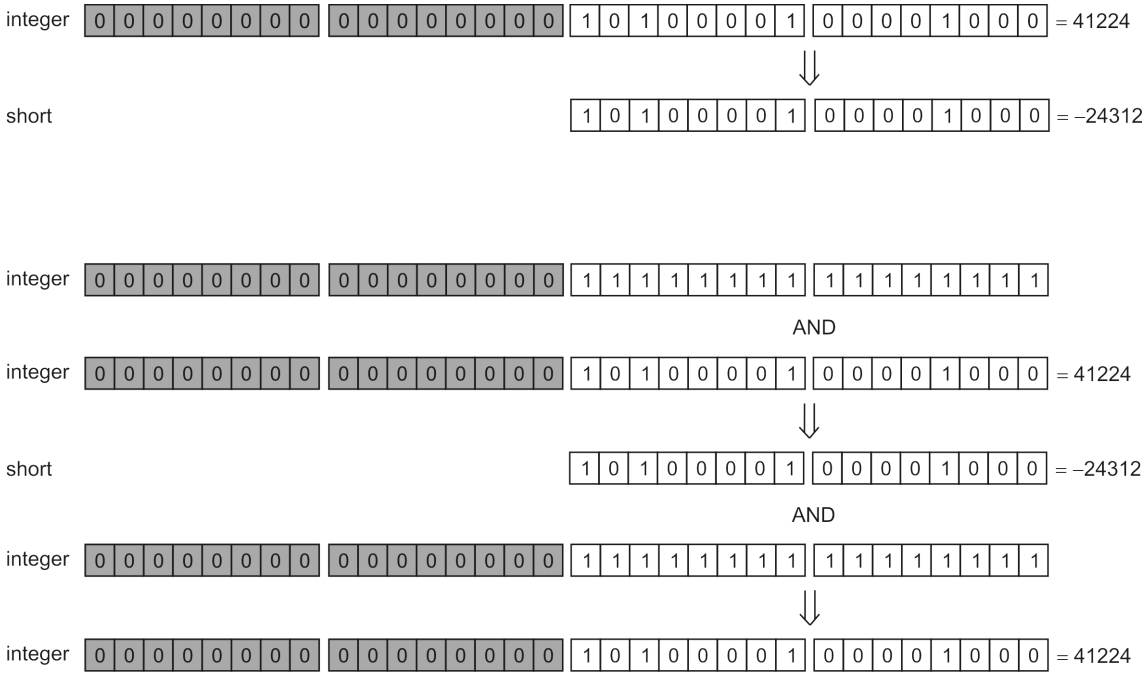
Obrázek 3.3: Diagram tříd pro práci s objemovými daty.

Počáteční návrh diagramu tříd a rozhraní pro práci s objemovými daty je na obrázku 3.3. Základem pro uložení objemových dat je abstraktní třída *Volume*, ta však nedefinuje datový typ použitý pro uložení jednotlivých voxelů ani jakým způsobem budou ukládány, pouze operace nutné pro práci s objemem. Mezi tyto operace patří vrácení hodnoty voxelu na požadovaných souřadnicích metodou *getVolume*, metody pro zjištění velikosti objemu a velikosti voxelů objemu.

Instancím potomků třídy *Volume* musí být již při jejich vytvoření předáno okno hustoty pro převod hustoty tkáně na barevnou informaci. To je zajištěné existencí jediného konstruktoru, který přebírá jako parametr instanci třídy *DensityWindow*.

Pro potřeby aplikace je dostačující velikost voxelu 12 bitů, můžeme tedy použít datový typ *short*, který má velikost 16 bitů. Navrhneme tedy třídu *VolumeShort*, která bude potomkem třídy *Volume* a bude používat pro uložení voxelu tento datový typ. Typ *short* je, jako všechny číselné datové typy v Javě, znaménkový, jeden bit je tedy použit pro uložení znaménka. Pro zvětšení rozsahu hodnot hustoty tkáně na 16 bitů můžeme využít i znaménkový bit. Vytvoříme metody *intToVoxelValue* a *voxelToIntValue*. Postup uložení 16 bitového bezznaménkového čísla do datového typu *short* je na obrázku 3.4. K uložení a načtení čísla je využita operace logického součinu.

Abstraktní třída *DensityWindow* deklaruje funkcionalitu požadovanou od implementací okna hustoty, jsou to především metody pro nastavování mezí okna (*setLowerLimit* a *setUpperLimit*), mezi kterými se provádí přepočet. Dále je zde metoda *getValue* pro přepočet hustoty tkáně na barevnou informaci. Třída implementuje metody pro přidání listenerů, které budou vyvolány, pokud se jakkoliv změní parametry ovlivňující výslednou barevnou informaci, tedy změna dolního nebo horního limitu.



Obrázek 3.4: Uložení 16 bitové hodnoty do proměnné typu *short*.

Třída *LinearDensityWindow* je implementací abstraktní třídy okna hustoty, která lineárně přepočítává hodnoty v nastavených mezích.

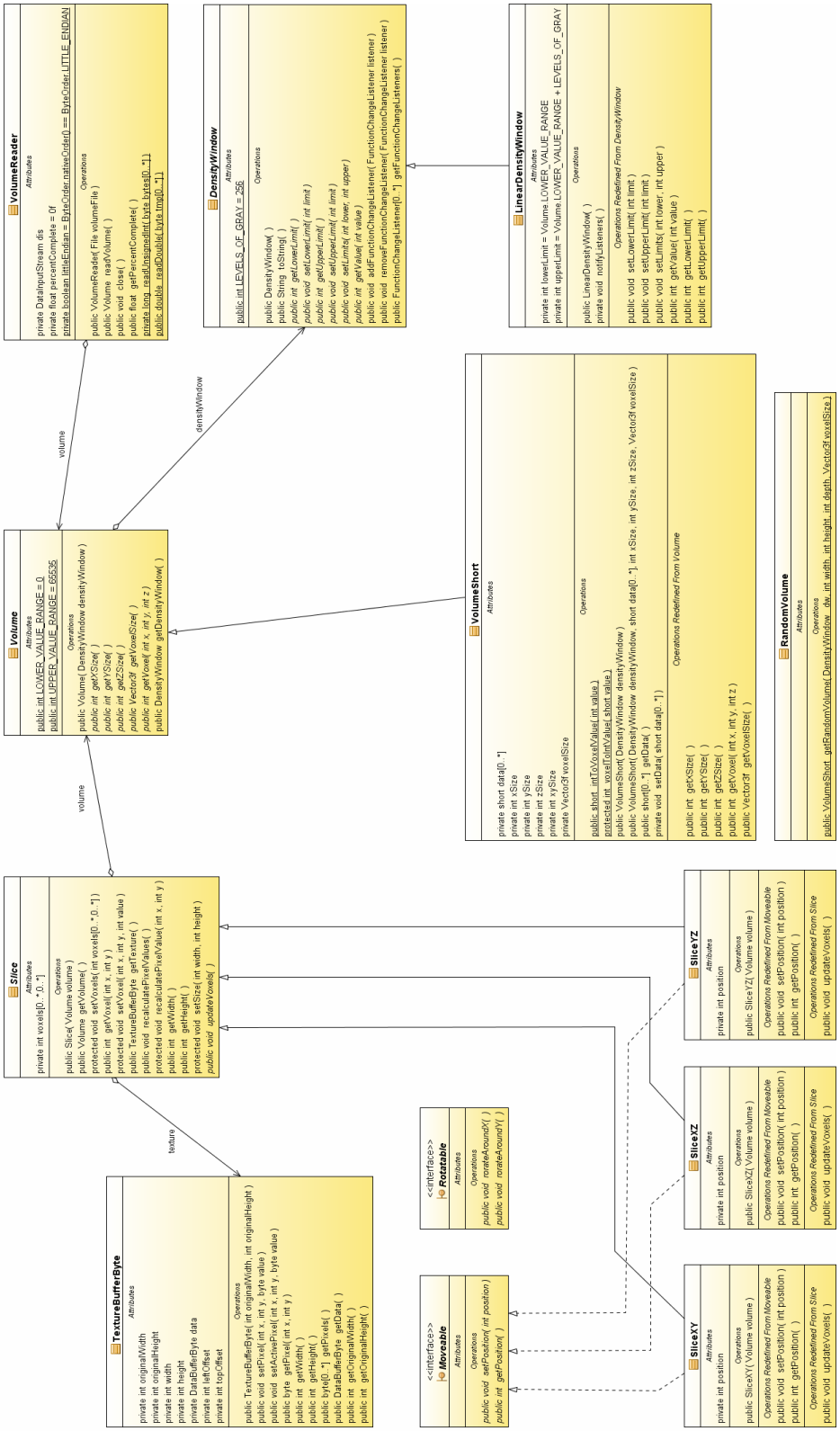
Třídy řezů objemem vychází z abstraktní třídy *Slice*. Řez objemem je možné vytvořit pouze nad již existujícím objemem. Pro ostatní třídy ponechává *Slice* viditelné pouze metody pro zjištění velikosti řezu a hodnoty jednotlivých voxelů, ze kterých se skládá, a textury vygenerované oknem hustoty náležícím k objemu, ke kterému patří tento řez. Výslednou texturu lze získat pomocí metody *getTexture*, která vrací objekt typu *TextureBufferByte*. Procesu generování textury se budu podrobněji věnovat dále.

Třída *Slice* neimplementuje výběr voxelů z objemu, obecný algoritmus pro výběr by byl komplikovaný a jeho efektivní implementace náročná. Jako lepší řešení se jeví deklarování metody *updateVoxels* jako abstraktní a ponechání její implementace na potomcích třídy. Vytvoření specializovaných tříd pro každý z rovinných řezů umožní napsat efektivnější algoritmus pro výběr správných voxelů, např. pokud budeme vybírat voxely na rovině xy stačí pro vytvoření řezu ponechat souřadnici z konstantní a zkopírovat do řezu všechny voxely s touto souřadnicí. Takto můžeme vytvořit řezy pro roviny dané všemi třemi osami souřadného systému a v případě, že budeme chtít vytvořit i jiné řezy, se můžeme uchýlit k obecnějšímu algoritmu, který ale bude pravděpodobně méně výkonný.

Třída *Slice* a její potomci neumožňují pohyb ani rotaci zobrazovacích ploch. Aby bylo možné se všemi plochami pracovat stejně nehledě na to jak možnosti pohybu či rotace implementují, je potřeba vytvořit jednotné rozhraní s těmito akcemi. Za tímto účelem vznikla rozhraní *Moveable* a *Rotateable*. Tato rozhraní určují metody, které musí plochy implementovat, aby s nimi bylo možno manipulovat, ale samotnou implementaci této akce ponechávají na nich samotných. Pro využití v aplikaci bylo nutné vytvořit tři plochy rovnoběžné s rovinami xy, xz a yz.

Na obrázku 3.5 je diagram tříd pro práci s objemem obsahující i konkrétní implementace tříd použitých v aplikaci. V Diagramu přibyla třída *RandomVolume*, která slouží pro

generování náhodných objemových dat pro účely testování.



Obrázek 3.5: Úplný diagram tříd pro práci s objemem.

3.5.2 Vytvoření textury z řezu

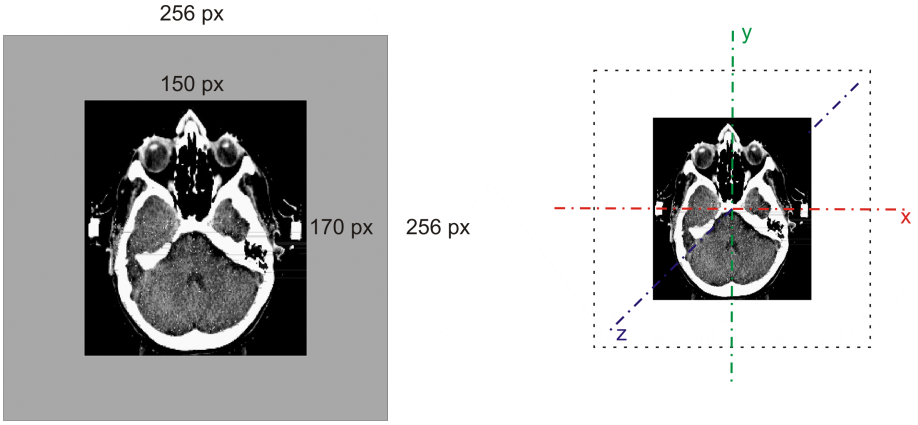
Výpočetně nejnáročnější částí je vytvoření bitmapové textury z voxelů řezu. Novou texturu je nutné vytvářet vždy, když uživatel změní meze okna hustoty nebo pohne řezem, proto je rychlé generování textury klíčové pro pohodlnou práci s programem.

Pohyb zobrazovací plochou vyvolá následující sled akcí:

1. Vypočtení souřadnic voxelu, který bude načten z objemu
2. Načtení voxelu a jeho uložení do řezu
3. Výpočet barevné hodnoty pixelu na základě nastavení okna hustoty.

Změna parametrů okna hustoty pouze vyvolá pouze nový výpočet výsledné barvy pixelu, není tedy potřeba znovu načítat voxel z objemu do řezu.

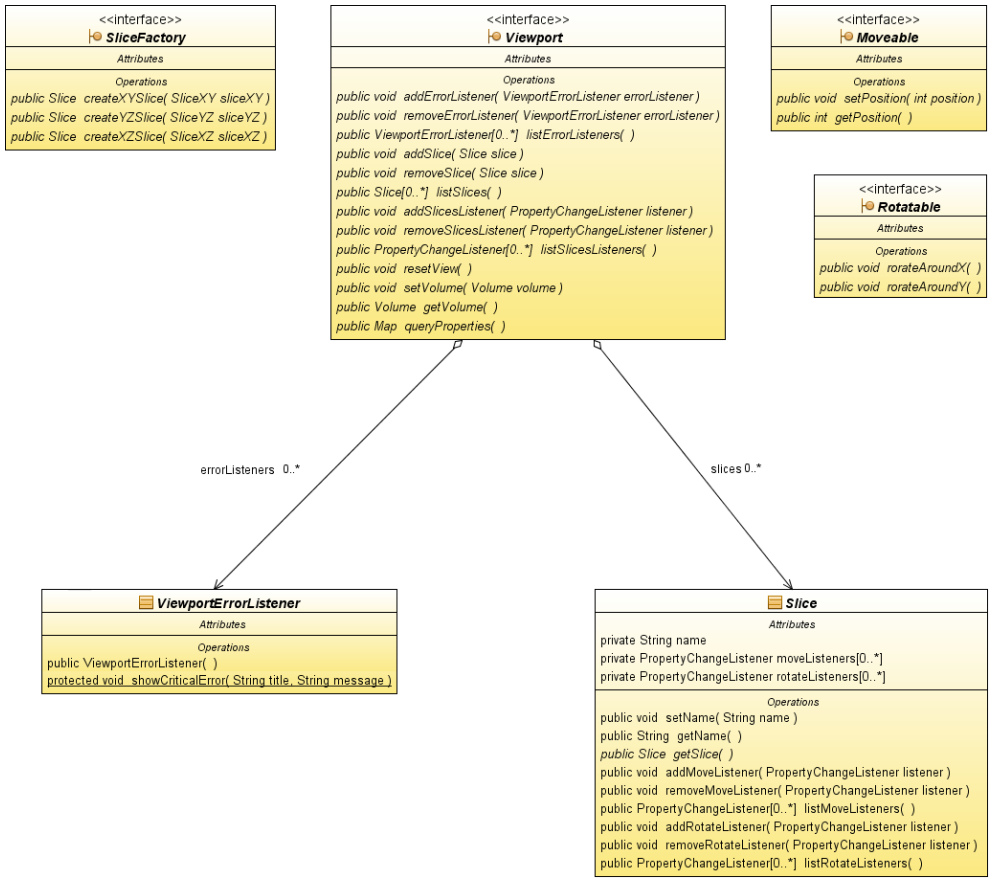
Tyto akce je ovšem potřeba provést pro každý voxel řezu, např. pro řez o velikosti stran 512 voxelů je potřeba tento sled akcí vykonat 262 144krát. Následně je potřeba vygenerovat z barevných informací voxelu texturu a tu aplikovat na vizualizaci řezu. Tyto výpočetně náročné operace musí být prováděny i při rychlém posouvání velkých zobrazovacích ploch tak, aby nezpomalovaly odezvu aplikace. Základní knihovna jazyka Java umožňuje velmi rychle generovat obrázky z dat připravených ve stejném formátu, jako je výstupní formát obrázku, proto je vhodné tomu přizpůsobit práci s barevnou informací generovanou řezy. Výsledné textury budou zbarveny ve stupních šedi interně reprezentovaných jako 8-bitová hodnota. Pro tyto účely byla navržena třída *TextureBufferByte*, která takto uchovává informace o barvách jednotlivých voxelů. Z této třídy je možné vytvořit texturu vhodnou pro vizualizaci. Tato třída také umožňuje generovat textury, jejichž strany mají rozměry o mocninách dvou i v případě, že řez těmto rozměrům neodpovídá. Ukázka mapování takových textur je na obrázku 3.6 Pomocí zvoleného rozhraní Java 3D je tedy možné zobrazit tyto textury i na grafických kartách, které to přímo nepodporují, bez ztráty jakýchkoliv informací.



Obrázek 3.6: Mapování textur nepodporovaných rozměrů na geometrické reprezentace zobrazovacích ploch. Doplněný okraj je oříznut zobrazí se pouze textura vygenerovaná z objemu.

3.5.3 Třídy pro vizualizaci řezů

Jak již bylo zmíněno v části 3.3 věnující se struktuře aplikace je vizualizace objemových dat oddělená od ostatních částí aplikace a je ji třeba navrhnout do jisté míry nezávisle na použité grafické knihovně. Diagram tříd pro vizualizaci řezů je na obrázku 3.7.



Obrázek 3.7: Diagram tříd pro vizualizaci řezů.

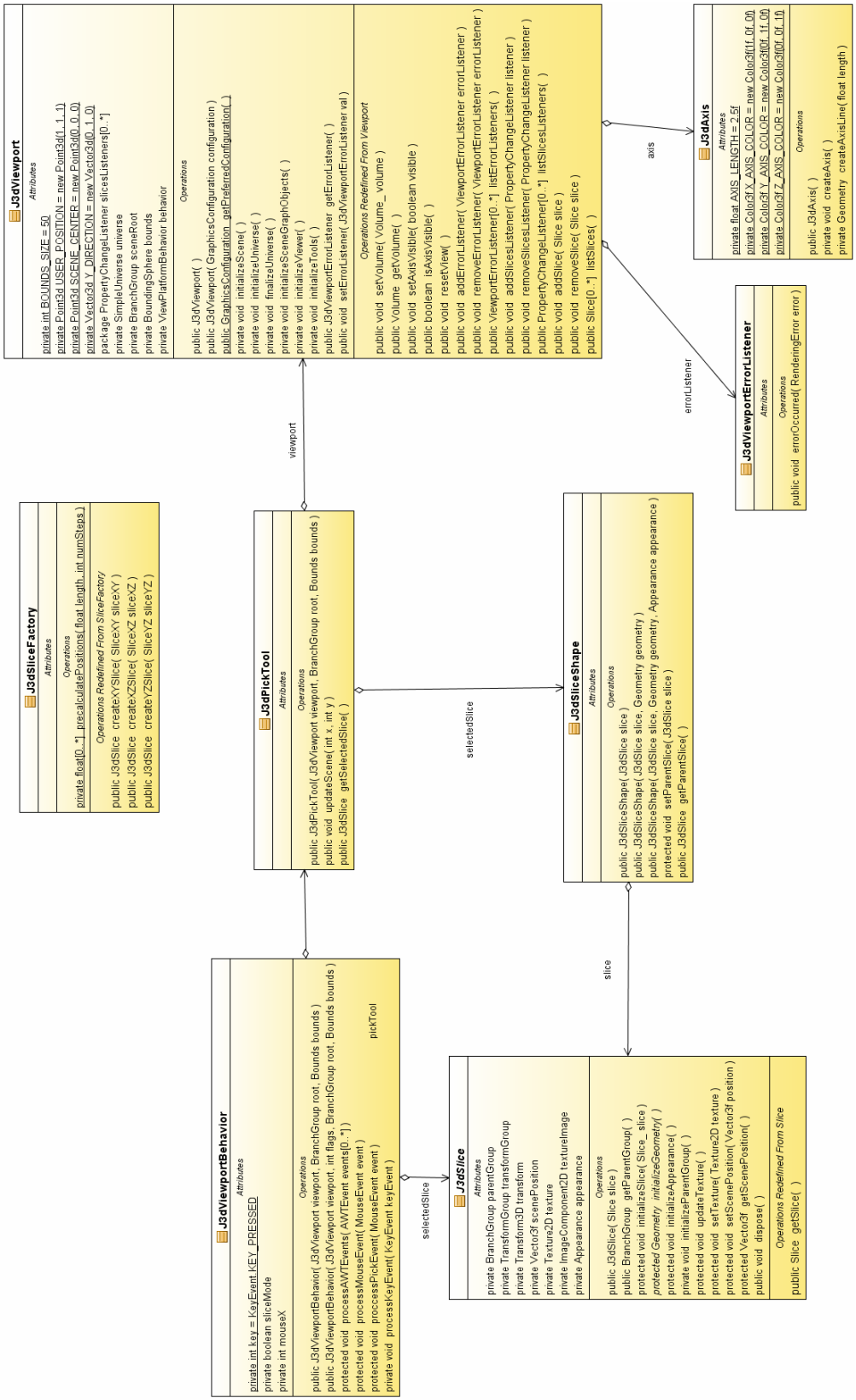
Hlavním prvkem vizualizace je rozhranní *Viewport*, které deklaruje, jakou funkcionalitu musí poskytovat komponenta pro vykreslování scény. Samotnou implementaci metod ponechává na třídách odvozených, které již budou vytvořeny pomocí konkrétní knihovny. Třída deklaruje metody pro přidávání a odebírání zobrazovacích ploch ze scény, *addSlice* a *removeSliceListener*, a umožňuje na tyto akce navázat libovolnou jinou akci pomocí tzv. listenerů, tyto listenery lze přidávat metodou *addSlicesListener* a odebírat pomocí *removeSlicesListener*. Vzhledem k nezávislosti na použité knihovně je také potřeba unifikovat způsob hlášení chyb vykreslovacího kontextu a způsob zjišťování podporovaných funkcí použitého rozhraní. Pro získání seznamu podporovaných funkcí byla navržena metoda *queryProperties*.

Chyby grafického rozhraní jsou ošetřovány pomocí třídy *ViewportErrorListener* a jejích potomků. Tyto třídy v sobě nesou akce potřebné pro obsluhu chyb a k *Viewport*u jsou přiřazovány a odebírány od něj metodami *addErrorListener* a *removeErrorListener*.

Třída *Slice* je předkem všech tříd pro vizualizaci řezů, představuje zobrazovací rovinu, implementuje metody *addMoveListener* a *addRotateListener* pro navázání událostí na pohyb plochy a může rozšiřovat rozhraní *Moveable* a *Rotateable* pro doplnění implementace

pohybu a rotace, podobně jako třída pro generování řezu z objemu. Třída opět nedefinuje reprezentaci plochy ve scéně, ta musí být implementována v některém z jejích potomků určeném pro konkrétní grafické rozhraní.

UML diagram tříd pro mnou použitou knihovnu Java 3D je na obrázku 3.8.



Obrázek 3.8: Diagram tříd pro vizualizaci řezů v knihovně Java 3D.

Kapitola 4

Implementace

Tato kapitola popisuje implementaci navrženého programu, problémy, které se vyskytly v průběhu práce a použitá řešení. Také představuje nástroje a některé knihovny použité při tvorbě programu.

Pochopení kapitoly nevyžaduje znalost jazyka Java, naopak se snažím vyhýbat se detailům závislým na použitém jazyku a soustředit se na obecná řešení a použité algoritmy.

4.1 Implementační nástroje

Cílem této práce je zjištění, zda je platforma Java vhodná pro vytváření aplikací podobného typu, proto je potřeba popsat a zhodnotit vlastnosti použitých nástrojů a knihoven. Tato kapitola se zaměřuje především na jejich obecný popis. O praktických zkušenostech z tvorby programu pojednává kapitola 5.

4.1.1 Jazyk Java

Jazyk Java je produktem společnosti Sun Microsystems, pod její záštitou je rozšiřována jeho specifikace a jazyk je dále rozvíjen. Java byla představena veřejnosti v roce 1995 a od té doby prošla mnoha vylepšeními, jak výkonnostními, tak i přidáváním nové funkcionality do základní knihovny tříd, tzv. Java Core API.

Java v současnosti obsahuje širokou škálu knihoven, počínaje prostředím pro programování nenáročných aplikací určených pro mobilní telefony, přes frameworky pro vytváření grafických uživatelských rozhraní, zpracování XML dokumentů, regulárních výrazů, komunikaci pomocí socketů až po technologii Java EE pro programování výkonných serverových aplikací určených pro použití v podnikové sféře.

Mezi významné výhody jazyka Java, které vývojářům usnadňují tvorbu aplikací, patří například:

- Nezávislost na architektuře počítače.
- Automatická správa paměti.
- Podpora objektově orientovaného programování.
- Bezpečné běhové prostředí - programu může být, nastavením běhového prostředí Javy, odepřena třeba možnost přístupu k síti nebo souborovému systému.

Programy napsané v jazyce Java vyžadují pro svůj běh virtuální stroj, tzv. Java Virtual Machine. JVM je kolekcí programů, které vytváří abstraktní stroj a zajišťuje tak stejnou interpretaci aplikací i na různých platformách. Program napsaný v Javě je nejprve přeložen do mezikódu, tzv. bytekódu. Bytekód je poté interpretován a optimalizován pro danou platformu při běhu aplikace v JVM. Virtuální stroj shromažďuje při běhu aplikace statistiky kódu a nejpoužívanější části následně překládá do nativního kódu platformy, to umožňuje soustředit optimalizace pouze na ty části aplikace, kde to bude nejvíce užitečné.

I přes řadu optimalizací, které Java provádí, se k některým účelům nehodí. Značnou nevýhodou je nutnost nahrát do paměti celé JVM, což znamená větší paměťové nároky u malých aplikací. Další nevýhodou je pomalejší start programů, který je způsobený již zmíněným spouštěním JVM. U některých aplikací také může být problematická absence bezznaménkových datových typů nebo preprocesoru.

Pro mnoho aplikací, může být nezbytné využít některou nativní knihovnu, ať již pro komunikaci pomocí některého nestandardního rozhraní nebo pro hardwarovou akceleraci 3D grafiky. Nativní knihovny lze využít pomocí rozhraní JNI nebo JNA. Pro vykreslování 3D grafiky bude potřeba využít jedno z těchto rozhraní. Jejich využitím ovšem přicházíme o řadu výhod Javy, nativní kód je závislý na platformě, pro kterou je přeložený, a nelze v něm použít automatickou správu paměti, veškerá alokace a uvolňování paměti musí být řízené programátorem.

Pro usnadnění používání knihoven třetích stran podporuje Java automatické generování dokumentace z komentářů tříd a metod zapsaných v kódu. Takto vygenerovaná dokumentace je dostupná ve formě WWW stránek a je často distribuována výrobci přímo s knihovnami. Další informace o jazyce Java jsou dostupné v [13], [5] a na [11].

4.1.2 Swing

Swing je knihovna pro tvorbu grafických uživatelských rozhraní v jazyce Java vyvíjená firmou Sun a od Javy 1.2 je upřednostňovaná před starší knihovnou AWT. Knihovna Swing nabízí mnoho předdefinovaných komponent pro tvorbu formulářových aplikací, jako jsou například vykreslování 2D grafiky, tabulek, formátovaného textu a další možnosti. Knihovna je navržena pro důsledné oddělení grafického rozhraní od aplikační logiky použitím architektury model-view-controller. Podporuje také podmínky pro splnění pravidel přístupnosti programů, jako je přizpůsobení pro hlasové čtečky nebo ovládání GUI pomocí klávesnice.

Více o knihovně Swing se lze dozvědět v [13], citeJava3D a [11].

4.1.3 Java 3D

Pro implementaci aplikace jsem zvolil knihovnu Java 3D, což je vysokoúrovňová knihovna pro tvorbu, vykreslování a manipulaci s grafem 3D scény. Java 3D byla původně také vytvářena firmou Sun, ale v roce 2004 byla uvolněna jako komunitní projekt. Informace o knihovně Java 3D jsem čerpal z [3] a [10].

Na rozdíl od jiných knihoven pro práci s 3D grafikou v Javě není Java 3D pouze portem těchto knihoven pro Javu, ale plně objektově orientovanou knihovnou. Pro vykreslování grafiky knihovna využívá buď rozhraní DirectX nebo OpenGL podle toho, které je na systému dostupné. Tato rozhraní jsou dále využívána Javou 3D.

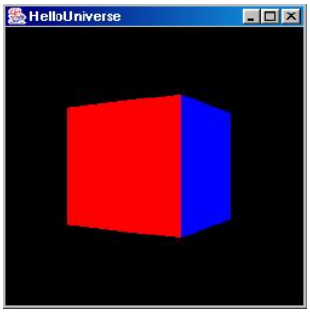
Hlavní výhodou, která rozhodla pro volbu knihovny Java 3D, je podpora mnoha platforem. V seznamu podporovaných operačních systémů jsou:

- Windows - podporovány jsou jak OpenGL tak DirectX
- Linux - OpenGL
- Mac OS - OpenGL
- Solaris - OpenGL

Jedním z požadavků na výslednou aplikaci byla její multiplatformnost a tu tato knihovna zajišťuje v dostatečné míře.

Java 3D skrývá nízkoúrovňové prvky grafické knihovny a tvorbu scény nahrazuje vytvářením grafu, do kterého se seskupují tělesa, světla, transformace a další objekty. Graf scény podporuje například i detekci kolizí a umísťování zdrojů zvuku, tyto možnosti zde zmiňuji pouze na okraj, neboť jsou pro účely aplikace irelevantní.

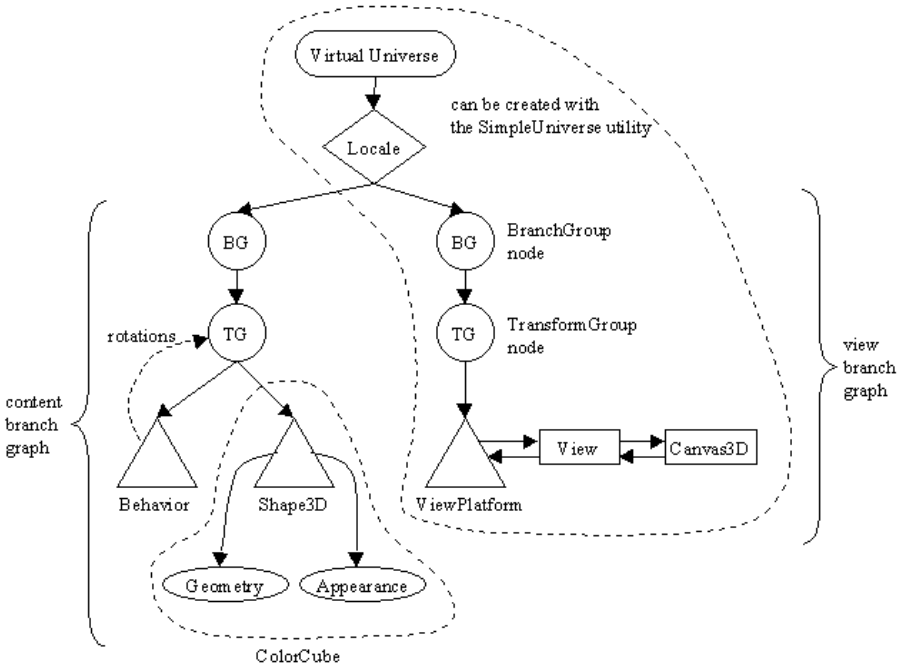
Další významnou výhodou je dobrý výkon knihovny, ta nabízí jak nízkoúrovňové tak vysokoúrovňové optimalizace. Na nízké úrovni využívá vlastností akcelerovaného grafického rozhraní. Optimalizace na vysoké úrovni zahrnují volby, zda se určité operace mají provádět, nebo ne. Jak již bylo zmíněno, knihovna Java 3D může obsluhovat i detekci kolizí, výběr objektů ze scény nebo prostorový zvuk. Pokud některou z těchto možností nevyužijeme, je možné ji vyřadit ze seznamu prováděných akcí.



Obrázek 4.1: Ukázka jednoduché aplikace v Javě 3D. Obrázek je převzatý z [3].

Na obrázku 4.1 je ukázka jednoduchého programu napsaného pomocí Java 3D, který vykreslí rotující krychli. Obrázek 4.2 pak ukazuje graf scény tohoto programu. Uzel grafu *VirtualUniverse* je rodičovským uzlem scény. *Locale* určuje umístění grafu scény v prostoru. Větev grafu označená jako *view brach group* obsahuje kameru pro snímání scény a transformace určující její pozici. Druhá větev, označená *content brach group*, obsahuje objekty 3D scény (uzly *Shape3D*), jejich transformace (uzly typu *TG - TransformGroup*), kontejnerové uzly (*BG - BranchGroup*), chování (uzly *Behavior*), jim přiřazený vzhled (*Appearance*) a geometrii (*Geometry*). Barevná krychle na obrázku 4.1 je tvořena uzly zahrnutými do skupiny *ColorCube*. Uzly typu *Behavior* obsahují výkonný kód, který se provádí při nějaké události, například po uplynutí nějaké doby, nebo při překreslení scény. Uzel tohoto typu v grafu scény zařizuje rotaci krychle vždy po uplynutí stanovené doby. Uzel typu *Shape3D* obsahuje geometrickou reprezentaci krychle a její vzhled - barvy jednotlivých polygonů.

Vytváření scén v Javě 3D spočívá především ve vytváření grafů scény a pomocných tříd.



Obrázek 4.2: Graf scény pro aplikaci na obrázku. Převzato z 4.1 [3].

J3dTree

Struktura grafu scény může být velmi složitá a může obsahovat mnoho uzlů. U rozsáhlých grafů vzniká problém udržet si o nich správný přehled a vyvarovat se vytváření chyb v organizaci grafu.

Java 3D umožňuje rekurzivně procházet uzly a zjišťovat o nich různé informace. Je samozřejmě možné vytvořit si knihovnu, která by umožňovala zobrazit a procházet strukturu grafu, již existuje taková knihovna, nazývá se J3dTree a lze ji snadno vložit do již existujícího projektu. Podrobný přehled o scéně, který nám tato knihovna poskytuje, lze využít i pro optimalizaci, například eliminaci nepotřebných uzlů nebo sloučení společných transformací nad objekty.

Více informací o této knihovně lze nalézt například v [3].

4.1.4 Vývojové prostředí

Při výběru vývojového prostředí jsem volil ze dvou nejznámějších prostředí pro jazyk Java, a to Eclipse a Netbeans. Pro Netbeans rozhodly především mé dlouhodobé zkušenosti z firemního prostředí.

Netbeans je multiplatformní otevřené vývojové prostředí, které podporuje kromě Javy mnoho jazyků a jeho funkcionalitu lze rozšířit pomocí pluginů. Prostředí nabízí propracovanou správu projektů a knihoven, podporu pro práci s databázemi a aplikačními servery. Pro usnadnění vývoje aplikací nabízí inteligentní doplňování kódu, tzv. code completion, které podporuje mnoho jazyků včetně Javy, C/C++, PHP a Javascriptu. Další užitečné nástroje jsou k dispozici pro automatické refaktorování kódu, tyto nástroje usnadňují restrukturalizaci již napsaného kódu a umožňují například automatické vytváření konstruktorů nebo obalování členských proměnných tříd do přístupových metod.

Pomocí pluginu UML pro Netbeans jsou vytvořeny i všechny UML diagramy v této práci.

Další informace o vývojovém prostředí Netbeans lze nalézt na stránkách projektu [12].

4.1.5 Systém pro správu verzí

Pro správu verzí aplikace jsem využil systém SVN hostovaný serverem Assembla. Implementace klienta SVN v prostředí Netbeans mi nevyhovovala a rozhodl jsem se proto využít volně šiřitelný klient Tortoise SVN, který nabízí stejnou funkcionalitu a integruje se do kontextového menu systému Windows.

4.2 Implementace návrhu

Na následujících řádcích jsou rozepsané kroky implementace programu v pořadí, v jakém probíhaly.

4.2.1 Implementace třídy objemu

Na počátku bylo nutné vytvořit rozhraní *Volume* popsané v kapitole o návrhu programu v sekci 3.5.1. Dalším krokem byla implementace třídy *VolumeShort*, která je potomkem tohoto rozhraní.

Třída *VolumeShort* pracuje s datovým typem *short*. Data hustoty tkáně jsou popsána bezznaménkovými hodnotami, postup jakým lze ukládat do znaménkového datového typu, který používá Java, bezznaménkové hodnoty, byl naznačen v taktéž v sekci 3.5.1. Metody používající tyto algoritmy jsou využívány vždy při ukládání nebo načítání hodnoty voxelu z objemu.

Mimo objem je vhodné pracovat s hodnotami hustoty tkáně reprezentovanými 32-bitovým datovým typem *integer*. Tento typ je výsledkem všech celočíselných matematických a logických operátorů, vyhneme se tedy neustálému přetypovávání výsledků, nehledě na to, že práce s tímto datovým typem je nepatrně rychlejší. Použití 32-bitového datového typu pro reprezentaci voxelů řezu paměťové nároky aplikace výrazně neovlivní v porovnání s použitím 16 bitového typu pro reprezentaci voxelů v objemu.

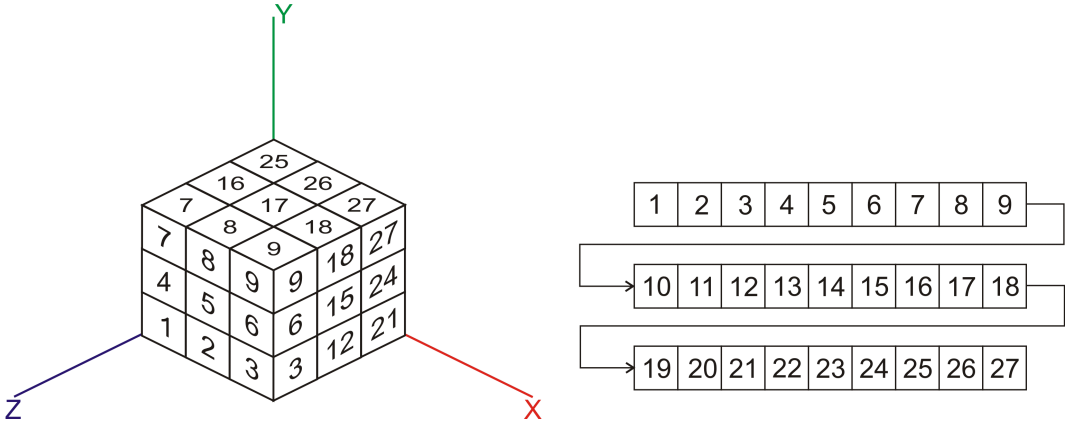
Velikost datového typu	Velikost objemu 512^3 voxelů	Velikost řezu 512^2 voxelů
16 bitů	262 144 kB	512 kB
32 bitů	524 288 kB	1024 kB

Tabulka 4.1: Porovnání paměťové náročnosti objemu a řezů při použití 32 a 16bitového datového typu.

Dále bylo nutné implementovat generátor náhodných objemů libovolné velikosti, aby bylo možné ověřit správnost třídy pro uchování objemu. Tento generátor byl nakonec napojen i na grafické rozhraní aplikace aby umožnil generování libovolných objemů i uživatelům.

V části věnující se návrhu jsem se rozhodl použít pro uložení voxelů v objemu jednorozměrné pole s mapovací funkcí. Tato funkce bude zpřístupňovat prvky pole pomocí souřadnic v trojrozměrném prostoru. Rovnice pro mapovací funkci vypadá následovně:

$$i = x + (y * S_X) + (z * S_X * S_Y)$$



Obrázek 4.3: Ilustrace mapovací funkce a skutečného uložení dat v jednorozměrném poli.

Kde proměnná i je index prvku v jednorozměrném poli, x souřadnice voxelu na ose x, y souřadnice voxelu na ose y, z souřadnice na ose z, S_X je velikost objemu podél osy x a S_Y velikost podél osy y.

Pro přístup k jednotlivým voxelům vznikla metoda *getVoxel*, která využívá výše uvedenou mapovací funkci.

Později bude také nutné implementovat načítání skutečných medicínských dat ze souborů.

4.2.2 Implementace řezů

Třídy řezů zajišťují výběr správných dat z objemu a generování barevné informace jednotlivých voxelů řezu na základě nastavení okna hustoty. Výběr dat z objemu je potřeba provést vždy, když dojde k pohybu zobrazovací plochou.

1	2	3
4	5	6
7	8	9

7	8	9
16	17	18
25	26	27

3	12	21
6	15	24
9	18	27

Obrázek 4.4: Řezy pro roviny xy, xz a yz vytvořené z objemu na obrázku 4.3.

Výběr voxelů je časově náročná činnost, jejíž délka je dána jak velikostí řezu, tak i jeho typem. Práce s jednorozměrným polem je nejefektivnější přistupujeme-li k jeho prvků sekvenčně. Jak je vidět z ukázek řezů na obrázku 4.4, řez xy lze z objemu zkopírovat jako jednu posloupnost hodnot, řez xz jako sled několika posloupností, při kopírování řezu yz je potřeba kopírovat voxely jednotlivě, nejefektivnější je tedy kopírování řezu xy, kdežto kopírování řezu yz zabere nejvíce času.

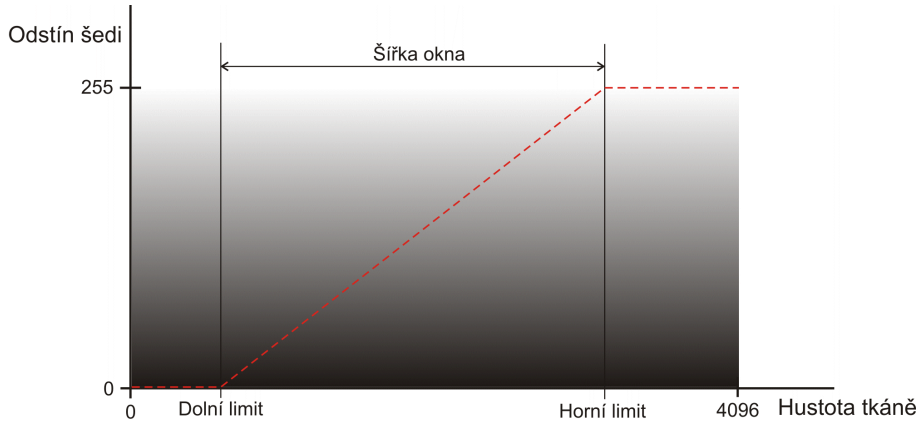
4.2.3 Okno Hustoty

Následně bylo možné implementovat okno hustoty, jeho princip byl podrobněji vysvětlen v kapitole 2.5. Jen ve zkratce připomeňme, že jeho funkcí je, podle požadavků uživatele, převést hodnotu hustoty tkáně jednotlivých voxelů na barevnou informaci. Toho je dosaženo lineární převodní funkcí a vyfiltrováním hodnot mimo požadované parametry.

Okno hustoty má parametr specifikující interval, ve kterém dochází k převodu. Tento interval je dán svým středem a šířkou. Oba tyto parametry nastavuje při práci uživatel aplikace. Interně jsou tyto hodnoty přepočteny na dolní a horní mez intervalu. Tkáň s hustotou pod specifikovaným intervalem je reprezentována jako černá, nad intervalem pak jako bílá. Hodnoty v mezích intervalu jsou přepočítávány podle následující funkce

$$c = \frac{255}{L_H - L_D} * (i - L_D)$$

L_H je dolní mez intervalu okna hustoty, L_D mez horní, i intenzita hustoty tkáně a c výsledná úroveň šedé barvy.



Obrázek 4.5: Ukázka lineární převodní funkce hustoty tkáně na stupně šedi.

Po vytvoření okna hustoty bylo možné otestovat správnou funkčnost tříd objemu a generování řezů. Již v této části se objevil problém s kopírováním dat pro řez *yz*, které je několikanásobně pomalejší než generování řezu *xy*.

4.2.4 Grafické rozhraní

Grafické rozhraní využívá knihovnu Swing, která je součástí API Javy. Rozhraní bylo vytvořeno pomocí návrháře integrovaného v Netbeans.

Samotný postup implementace grafického rozhraní není zajímavý, ovšem oproti prvotnímu návrhu došlo v průběhu vývoje programu k významné změně. Původní implementace uživatelského rozhraní byla přepracována s využitím Swing Application Framework (nezaměňovat s knihovnou Swing). Tato knihovna umožňuje snadné vytváření vícevláknových uživatelských rozhraní a snadno lokalizovatelných aplikací a přispívá tak jak k snadnější tvorbě programu, tak k jeho komfortnějšímu užívání. Pomocí vláken můžeme přesunout dlouhotrvající operace do pozadí, aniž by došlo k „zamrzání“ programu.

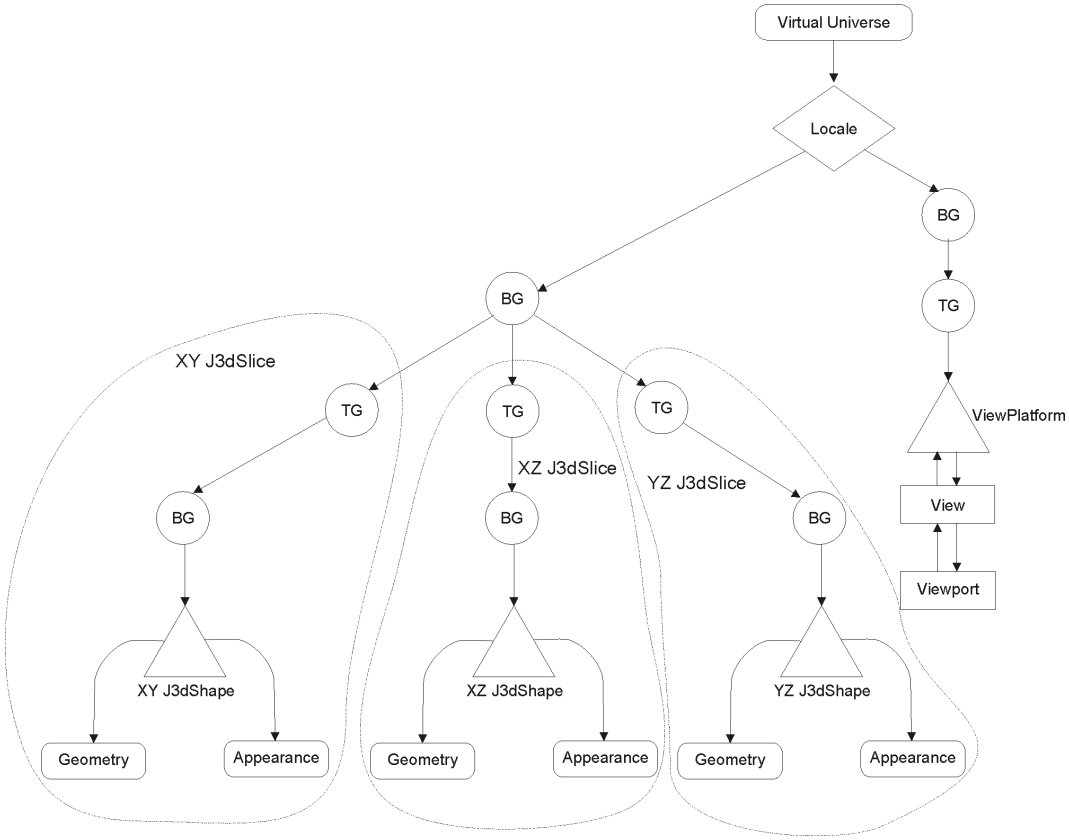
4.2.5 Scéna

Po vytvoření uživatelského rozhraní bylo možné přistoupit k implementaci scény.

Implementace scény pro knihovnu Java3D je tvořena jednou instancí třídy *J3dViewport*, tato třída představuje komponentu do které je vykreslován pohled na scénu. Rozšiřuje *J3dPanel* knihovny Java 3D a implementuje metody nezbytné pro inicializaci grafu scény, nastavení světel, pozice pozorovatele a dalších parametrů.

Abstraktní třída zobrazovací plochy *J3dSlice* zahrnuje funkcionalitu společnou pro všechny plochy, tedy převážně inicializační metody. Konkrétní implementace ploch připravených pro vložení do scény poskytuje třída *J3dSliceFactory*. Všechny plochy podporují metody pro aktualizaci textur v závislosti na změně své pozice, tyto pozice jsou vypočítány při vytváření plochy, aby nebylo nutné je počítat vždy znovu.

Graf scény je na obrázku 4.6. Všechny objekty scény jsou umístěny do společného uzlu scény. Jednotlivé zobrazovací plochy jsou představovány hierarchickou strukturou uzlů typu *TransformGroup*, *BranchGroup* a *J3dShape*. Uzly typu *TransformGroup* obsahují transformace aplikované na podřízené uzly, tedy posun a rotaci ploch. Uzly *BranchGroup* slouží pouze jako kontejnery pro objekty *J3dShape*. Tyto objekty jsou vlastní polygonální reprezentací ploch, obsahují geometrii a vlastnosti vzhledu, např. texturu a její nastavení.



Obrázek 4.6: Graf scény pro vizualizaci.

Pohyb pozorovatele a zobrazovacích ploch

Pohyb ploch a ovládání kamery je kontrolováno v první řadě myší, umístění ploch bude možné kontrolovat i z bočního panelu v grafickém rozhraní, tato funkce bude ale implementována až dodatečně.

Panel použitý pro implementaci třídy *J3dViewport* narušuje událostní model knihovny Swing a zavádí vlastní ošetřování vstupních událostí pomocí tzv. chování. Scéně zobrazené v panelu je možné přiřadit specializovaný typ chování, tzv. chování pozorovatele. Události klávesnice a myši nezpracované výše v aplikaci jsou předávány tomuto chování a mohou

v něm být ošetřeny. Jako reakci na tyto události lze modifikovat transformace objektu *ViewPlatform*, který určuje pozici pozorovatele.

Součástí knihovny Java 3D je třída *OrbitBehavior*, ta slouží pro ovládání pozice pozorovatele pomocí klávesnice a myši.

Tuto třídu nelze upravit, ovšem pro vybírání a pohyb zobrazovacích ploch pomocí myši to bylo nutné. Implementoval jsem proto vlastní třídu pozorovatele *J3dViewportBehavior*, která mění své chování v závislosti na tom, zda pracuje v režimu práce s plochami, nebo režimu pohybu kamery. Práci s plochami se budu věnovat dále v této sekci v části 4.2.5. Při pohybu pozorovatele se řízení scény předá třídě *orbitBehavior*.

Výběr a pohyb ploch

Uzly *J3dShape* slouží nejen pro zapouzdření geometrie a vzhledu, ale podporují i výběr jednotlivých ploch. Pro výběr objektu ve scéně nabízí Java mechanizmus pojmenovaný *picking*, jeho chování lze definovat na základě mnoha různých parametrů. Pro nás je dostačující implementovat výběr plochy kliknutím myši. Mechanizmus výběru je následující:

1. Uživatel klikne myší
2. Souřadnice myši se přepočítají na souřadnice v trojrozměrném prostoru a z těchto souřadnic je vyslán paprsek paralelní ke směru pohledu kamery.
3. Pokud paprsek protne objekt typu *J3dSlice* dojde k jeho výběru. Pokud protíná více objektů tohoto typu je vzat v úvahu pouze první průsečík paprsku s objektem. Výběr jiného typu objektu nevyvolá žádnou akci.

Vybraný objekt typu *J3dSlice* obsahuje odkaz na řez, který představuje. Tento řez je uložen jako aktuálně vybraný a následný pohyb myši je překládán na změnu jeho pozice. Zároveň jsou při změnách pozice řezu vyvolávány události registrované v seznamu posluchačů (tzv. listenerů) pro pohyb řezu, tím je možné reagovat na pohyby, vyvolané myší ve scéně, reagovat i jinde v aplikaci.

4.2.6 Generování textury

Generování nové textury je potřeba provést vždy při pohybu zobrazovací plochy nebo změně parametrů okna hustoty.

Proces vytvoření nové textury je následující:

1. Výpočet barevné informace pro každý pixel řezu a jejich uložení do instance třídy *TextureBufferbyte* náležící řezu
2. Vytvoření nového obrázku z instance třídy *TextureBufferByte*
3. Nahrazení textury zobrazovací plochy nově vygenerovaným obrázkem

Třída *textureBufferByte* byla implementována pro usnadnění generování obrázku z hodnot šedi. Ukládá jednotlivé pixely v bufferu, ze kterého Java umožňuje efektivně vytvářet rastrové obrázky. Dalším důvodem pro vznik této třídy byl požadavek na práci s texturami libovolných rozměrů, i tento požadavek třída splňuje, více o způsobu jakým jsou ukládány takovéto textury lze nalézt v 3.5.2.

Problémem, který je na první pohled patrný z postupu generování nové textury, je neustálé vytváření nových obrázků. Nahrazené textury musí být včas uvolňovány, aby nedocházelo k plýtvání pamětí. Uvolňování paměti garbage collectorem ovšem vyžaduje procesorový čas a může se stát, že dojde ke zpomalení generování nových textur v důsledku uvolňování starých.

4.2.7 Výkonnostní testy

Po implementaci dosud zmíněných částí programu bylo potřeba provést subjektivní testy použitelnosti aplikace na několika různě výkonných sestavách. Z těchto testů a následného profilování aplikace vyplynulo několik závěrů:

- Zvolená grafická knihovna není slabým článkem programu, byla výkonná i na integrovaných grafických kartách starých několik generací. Na tom má také zásluhu velmi jednoduchý graf scény s minimem objektů, není tedy potřeba vykreslovat mnoho textur a polygonů.
- Výběr voxelů pro řezy z objemu je dostatečně efektivní i na slabších sestavách. Generování řezu yz je sice výrazně pomalejší než u řezu xy, ale stále nezpůsobuje žádné znatelné zpomalení aplikace.
- Generování textury je nejnáročnější částí procesu aktualizace scény. Při posunu zobrazovací plochy o jeden voxel proběhne vytvoření nové textury dostatečně rychle a opět má jen minimální vliv na výkon aplikace. K drastickému snížení výkonu aplikace však dochází zejména při rychlejším posuvu ploch o více voxelů. Zatímco při posuvu plochy xy byl tento úbytek výkonu v mezích únosnosti, při manipulaci s plochami xz a zvlášť yz dochází neustálému „zasekávání“ aplikace.

Z výsledků vyplývá, že je potřeba soustředit se na zefektivnění výběru voxelů a generování textur.

Snížení výkonu aplikace pozorované při rychlých posuvech zobrazovacích ploch je způsobené zasláním mnoha požadavků na generování řezů. Pokud uživatel rychle přesune plochou o 100 voxelů aplikace se pokusí vygenerovat všech 100 řezů. Generování všech řezů je zbytečné, protože uživatel nestihne všechny změny textury zaregistrovat.

Řešení výkonnostních problémů

Nejvhodnějším řešením problémů s výkonem generování řezů se ukázalo být přesunutí generování textury do samostatného vlákna. Od této techniky jsem očekával přínos především na vícejádrových procesorech. Ukázalo se však, že toto řešení dramaticky zvýšilo výkon aplikace i na starších jednojádrových počítačích.

Nový postup generování textur využívá vlákno *TextureUpdater*, které se vytvoří vždy s novou zobrazovací plochou. Vlákno vyčkává na zaslání požadavku na vygenerování nové textury plochy. Ten je odeslán, když s ní uživatel pohne. Vlákno vygeneruje novou texturu podle pozice plochy. Při dalším pohybu o několik voxelů jsou odeslány nové požadavky na vygenerování textury. Ty jsou však ignorovány dokud vlákno nedokončí vytváření prvního řezu. Po vygenerování řezu vlákno zjistí, že přišly další požadavky pro nové textury. Zjistí aktuální pozici plochy, která je nastavená posledním požadavkem a je její tedy aktuální pozicí, a vygeneruje novou texturu, která je následně přiřazena vybrané ploše.

Po otestování této úpravy byl již chod aplikace naprosto plynulý a textury ploch jsou také vytvářeny správně, nic tedy nebrání využití tohoto postupu.

4.2.8 Postranní panel

Po vyřešení výkonnostních problémů aplikace jsem přistoupil k tvorbě postranního panelu, ten je opět vytvořen pomocí modulu pro návrh GUI v Netbeans. Z panelu je možné měnit pozici zobrazovacích ploch zadáním hodnoty nebo pomocí posuvníku. Pokud uživatel pohne plochou v pohledu na scénu, pak musí být tyto hodnoty také aktualizovány v panelu. Toho lze dosáhnout přidáním posluchače (listeneru) pro pohyb ke všem řezům ve scéně, tyto listenery budou pouze aktualizovat pozice ploch v postraním panelu při jejich pohybu pomocí myši.

4.2.9 Načítání objemových dat

Objemová data lze načítat z některého ze specializovaných formátů, např. DICOM, nebo z tzv. „raw“ dat. Soubory s těmito daty obsahují pouze rozměry uloženého objemu, hodnoty a rozměry voxelů, nejsou v nich uložena žádná metadata popisující snímky.

Pro načítání „raw“ dat jsem vytvořil třídu *VolumeReader* a přidal do menu a toolbaru aplikace možnost otevření datového souboru.

4.2.10 Nápověda programu

Nápověda programu ve formě WWW stránek je přístupná z menu aplikace.

4.2.11 Testování aplikace

Po dokončení vývoje byl program otestován na několika různě výkonných sestavách. Problémy s rychlostí aplikace se nevyskytli na žádné z nich. Objevily se však chyby související se správou paměti, po zavření souboru nebyla správně uvolněna paměť obsazená objemem. Tato chyba byla způsobena vlákny pro generování textur, která byla stále aktivní a udržovala odkazy na zobrazovací plochy. Problém byl vyřešen ukončením činnosti vláken při odebrání ploch ze scény. Poté se již žádné problémy s pamětí nevyskytly.

4.3 Nedostatky implementace

Zde bych rád nastínil některé chyby a neduhy aplikace, kterými trpí i výsledná verze, a které by bylo ještě potřeba vyřešit.

- Pohyb zobrazovacích ploch je vždy ovládán pohybem myši zprava doleva a obráceně nehledě na to, jak jsou plochy natočené ke kameře. Spíše než závada ve funkčnosti je to chyba uživatelského rozhraní, kterou by bylo potřeba vyřešit.

- Při nastavování parametrů okna hustoty pomocí posuvníků v bočním panelu nedochází k zastavení jezdců při překročení limitu šířky nebo pozice středu okna. I přes pokusy o řešení této chyby se mi ji nepodařilo odstranit. Špatně zvolené hodnoty na posuvnících neovlivňují skutečné nastavení okna hustoty a při zadání manipulací s nastavením okna pomocí vstupních textových polí jsou hodnoty na posuvnících opraveny na korektní.

- Občas se projeví chyba v překreslování grafického rozhranní, kdy se, při minimalizaci nebo přesunu okna do pozadí při načítání souboru s objemovými daty, nepřekreslí

tlačítka v toolbaru a menu, případně i části bočního panelu. Jedná se pouze o kosmetickou chybu, při změně velikosti okna aplikace již k překreslení dojde. Tuto chybu jsem bohužel neobjevil včas, abych ji stihl opravit.

- Nefunkční Java Web Start, aplikaci nelze spustit pomocí technologie Web Start z internetu. Na vině jsou použité nativní knihovny Java 3D. Opět se nejedná o významnou chybu, protože aplikace je určena pouze pro testovací účely. V případě skutečné implementace by bylo nutné zvážit, zda je problém potřeba řešit, vzhledem k účelu aplikace se domnívám, že to není prioritní.

Kapitola 5

Výsledky

Úkolem práce bylo ověřit praktickou použitelnost jazyka Java pro vizualizaci medicínských dat. Důležitou součástí práce tedy musí být i zhodnocení výsledků a prostředků využitých k jejich dosažení.

5.1 Funkcionalita aplikace

Z hlediska výsledné funkcionality můžeme prohlásit, že aplikace splňuje požadavky specifikované zadáním. Největší problémy zůstávají v grafickém rozhraní. Ty by však bylo možné opravit například implementováním vlastních komponent pro nastavení parametrů okna hustoty a vytvořením vhodné funkce pro výpočet pohybu zobrazovacích ploch na základě posunu myši.

Díky možnosti otevření souborů s nasnímanými daty bylo možné ověřit funkčnost aplikace na skutečných medicínských datech. K dispozici jsem měl dva vzorky dat. V obou případech byla programem správně interpretována a práce s nimi byla bezproblémová.

5.2 Výkon aplikace

Pro praktické využití aplikace je nutné, aby splňovala požadavky na pohodlnou práci i s velkými objemy dat při zachování rychlé odezvy uživatelského rozhraní.

Bohužel nemám k dispozici žádnou jinou aplikaci navrženou ke stejnému účelu a nemohu tak porovnat její výkon se mnou vytvořenou aplikací. I v případě, že bych takovou aplikaci k dispozici měl, jednalo by se pouze o subjektivní srovnání rychlosti aplikace. K hlubšímu testování výkonu by bylo nutné implementovat dvě aplikace, jednu v C/C++ a druhou v jazyce Java, podle stejného návrhu.

Porovnání výkonu jazyků Java a C/C++ komplikuje i nedostatek aktuálních věrohodných testů. Výsledky, které jsem měl k dispozici, pocházely z testů, při nichž byla většinou využita o několik verzí starší implementace jazyka Java a JVM.

Testování aplikace probíhalo jak na systémech MS Windows, tak na Linuxu, konkrétně na distribuci Ubuntu. Pro testování bylo využito několik různě výkonných sestav od netbooku s procesorem Intel Atom až po čtyřjádrovou pracovní stanici, na všech počítačích byl program použitelný s dostatečnou výkonnostní rezervou.

Pro další výkonnostní optimalizace tedy neexistuje žádný důvod, možná by bylo nutné k nim přikročit, pokud by se implementovala některá rozšíření zmíněná v 6.

5.3 Použité vývojové prostředí

V této části jsou shrnuty vlastnosti použitých vývojových nástrojů a knihoven a jejich vhodnost pro využití v praxi.

5.3.1 Jazyk Java

Vývoj programů v jazyce Java je velmi pohodlný a intuitivní. Díky syntaxi, která vychází z jazyka C je kód dobře čitelný i pro programátory, kteří s používáním tohoto jazyka nemají zkušenosti, i proto se Java často využívá pro ukázky algoritmů.

Jednoznačným přínosem je zjednodušení tvorby programů, vývoj v jazyce Java je rychlejší než v jazyce C, přispívá k tomu i automatická správa paměti kontrolovaná JVM a existence velkého množství knihoven, ať již dodávaných přímo s jazykem, nebo poskytovaných třetími stranami. Správa paměti ve výsledné aplikaci funguje správně a nedochází ke zbytečným alokacím nadměrného množství systémových prostředků.

Jako významný přínos pro vývoj aplikací lze hodnotit i mechanizmus HotSwap poskytovaný virtuálním strojem. Tento mechanizmus umožňuje při běhu programu přímo upravovat kód bez nutnosti aplikaci znovu kompilovat a spouštět, usnadňuje tak vývoj především velkých aplikací, užitečný je však téměř vždy když je potřeba rychle vyzkoušet změnu v aplikaci.

Jako významnou nevýhodu při tvorbě jakýchkoliv aplikací v jazyce Java je nutnost mít pro jejich spuštění v paměti načtený virtuální stroj JVM, ten sám o sobě zabírá několik desítek MB operační paměti. Tato vlastnost je nevýhodná zejména při tvorbě menších aplikací. Při práci s velkými objemy dat se stává spotřeba paměti JVM méně významnou.

Další nevýhodou je nemožnost aplikovat šablony na primitivní datové typy, Java umožňuje jejich použití pouze pro objekty. Tato možnost by usnadnila především tvorbu tříd pro práci s objemem a výběrem dat pro zobrazovací plochy.

Pro tvorbu zadaného programu tedy převažují výhody nad nevýhodami, kvůli relativně malému rozsahu programu jsem využil pouze zlomek vlastností jazyka. Není vyloučeno, že při tvorbě většího projektu by byl poměr pro a proti rozdílný. Vždy záleží na konkrétních požadavcích na aplikaci.

5.4 Java 3D

Knihovna Java 3D nabízí mocné prostředí především pro komplikovanější programy určenými pro práci s většími scénami s více objekty. Na malých scénách, jako je ta v mém programu, se nemohou projevit všechny optimalizace, které provádí nad objekty v grafu scény.

Výhodou je použití nativního vykreslovacího rozhraní na hostitelském systému, za další výhodu může být považována i plně objektová orientace knihovny. Pro práci s komplexními scénami přináší Java 3D další výhody jako je například snižování úrovně geometrie nebo ořezávání objektů mimo záběr pozorovatele, tyto vlastnosti jsou ale pro aplikace tohoto typu zbytečné.

Pro využití v podobných vizualizacích je tedy přínos použité knihovny přinejmenším sporný. Pokud přímo nevyžadujeme některou z možností této knihovny, může být výhodnější využití některého nízkoúrovňového grafického rozhraní. Jako příklad lze uvést dva porty knihovny OpenGL pro jazyk Java - LWJGL a JOGL.

5.5 Vývojové prostředí Netbeans

Prostředí Netbeans výrazně pomáhá k usnadnění vývoje aplikací. Praktickou vlastností je především inteligentní nabízení kódu pro doplnění a mnoho předdefinovaných maker urychlujících psaní některých zdlouhavých konstrukcí jazyka. Další výhodou je zobrazování nápovědy k funkcím přímo v prostředí editoru kódu.

Další výhodou je analýza napsaného kódu, která upozorňuje na vznik potencionálních chyb při překladu, lze se tak vyhnout mnoha zbytečným překladům programu.

Netbeans lze rozšířit řadou pluginů, jako příklad lze uvést nástroj pro tvorbu UML diagramů, který umožňuje navrhnout program a následně z diagramů vygenerovat jeho kostru.

Netbeans je moderní a přizpůsobitelné vývojové prostředí s širokou podporou jazyků a pro vývoj aplikací ho lze jen doporučit.

Kapitola 6

Závěr

Předmětem práce bylo vyzkoušení vhodnosti jazyka Java pro praktické využití ve vizualizacích medicínských dat. Výsledkem je testovací aplikace, která slouží pro interaktivní zobrazování reálných medicínských dat. I přestože aplikace není určená pro praktické využití, význam její existence to nesnižuje, protože z ní lze vycházet při návrhu dalších projektů podobného typu, které by již mohly najít uplatnění v praxi.

Z implementace a zkušeností z používání testovací aplikace shrnutých v kapitole 5 lze dospět k názoru, že Java je vhodná i pro tento typ aplikací. Samozřejmě vždy záleží na konkrétní aplikaci a při volbě prostředí je třeba řádně zvážit výhody a nevýhody použití daných technologií.

Závěrem lze tedy prostředí jazyka Java pro tvorbu podobných typů aplikací doporučit díky výhodám, které nám přináší.

6.1 Možnosti dalšího rozvoje aplikace

Pro podrobnější rozbor problematiky by bylo vhodné implementovat další funkcionalitu, která by vhodnost jazyka Java dále prověřila. Pro nejbližší rozvoj aplikace lze navrhnout některé z těchto možností:

- Implementace práce se segmentačními daty, přidání možnosti tato data jak načítat, tak i editovat.
- Rozšíření o některou z metod pro vykreslování objemu, ať již založenou na metodách hledajících povrch, nebo na přímém zobrazování objemu.

Literatura

[1] Bourke, P.: Implicit surfaces.
tt http://local.wasp.uwa.edu.au/ pbourke/miscellaneous/implicitsurf/, 1997, [cit. 2009-05-11].

[2] Campr, P.: *Získávání 3D modelu lidských tkání z obrazových dat CT*. Diplomová práce, Západočeská univerzita v Plzni, 2005.

[3] Davison, A.: *Programování dokonalých her v Javě*. Computer Press, 2006, ISBN 80-7226-944-5.

[4] Felkel, P.: VolumePro 500 - Grafický akcelerátor pro zobrazování objemových dat [online]. `http://www.elektrorevue.cz/clanky/00030/index.html`, 2000, [cit. 2009-05-11].

[5] Hagar, P.: *Practical Java*. Addison-Wesley, 2000, ISBN 0-201-61646-7.

[6] Hattne, J.; Fange, D.; Elf, J.: *MesoRD User's Guide [online]*. 2006, [cit. 2009-05-14].

[7] Horák, M.: CT a MR zobrazování, 2008, seminář Počítačová grafika v lékařství. MFF UK.

[8] Kubíček, R.: *Vizualizace značených buněk modelového organismu*. Diplomová práce, Vysoké učení technické v Brně, 2007.

[9] Pelikán, J.: Visualizace objemových dat [online], 2009, seminář Počítačová grafika v lékařství. MFF UK.

[10] WWW stránky: Java 3D Parent Project [online]. `https://java3d.dev.java.net/`, [cit. 2009-04-11].

[11] WWW stránky: Java SE Desktop Overview [online].
`http://java.sun.com/javase/technologies/desktop/`, [cit. 2009-05-11].

[12] WWW stránky: NetBeans IDE 6.5 Features.
tt http://www.netbeans.org/features/index.html, [cit. 2009-05-11].

[13] Zakhour, S.; aj: *Java 6: Výukový kurz*. Brno, Česká republika: Computer Press, 2007, ISBN 978-80-251-1575-6.

[14] Španěl, M.; Beran, V.: Obrazové segmentační techniky: Přehled existujících metod [online]. `http://www.fit.vutbr.cz/ spanel/segmentace/`, 2005, [cit. 2009-05-11].

[15] Žára, J.; aj: *Moderní počítačová grafika. 2. vyd.* Brno, Česká republika: Computer Press, 2004, ISBN 80-251-0454-0.

Dodatek A

Obsah CD

- Zdrojové soubory programu
- Knihovny potřebné pro přeložení programu
- Přeložená aplikace pro různé platformy
- Nápověda k programu
- Testovací data pro aplikaci
- Zdrojové soubory textové zprávy
- Výsledná textová zpráva