Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Projekce snímků výpočetní tomografie

Bc. Martin Lepeška

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Lepeška**
Osobní číslo:	**I17210**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Projekce snímků výpočetní tomografie**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem práce je navrhnout a vytvořit aplikaci vykreslující projekce medicínských volumetrických dat (CT, MR) načtených ze standardizovaného formátu DICOM. V teoretické části budou popsány základy DICOM standardu a princip náběru dat. Práce se dále bude zabývat významem, principem a optimalizací projekcí běžně používaných pro diagnostiku (MPR, MIP, MinP, SSD). Dále budou představeny vybrané techniky vizualizace objemových dat. V rámci práce bude provedena analýza, návrh a implementace aplikace, ve které budou objemová data a jejich projekce zobrazovány. Vykreslování projekcí bude realizováno metodou přímého vykreslování volumetrických dat s využitím raytracingu. Demonstrační aplikace praktické části bude implementována v jazyce Java s využitím OpenGL API prostřednictvím knihovny LWJGL a načítáním DICOM dat pomocí knihovny DCM4CHE.

Rozsah grafických prací:

Rozsah pracovní zprávy: **cca 60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

CLUNIE, David A. DICOM structured reporting Bangor, Pa.: PixelMed Pub., 2000, ISBN 0970136900
ENGEL, Klaus, Real-time volume graphics Wellesley, Mass.: A K Peters, 2006. ISBN 978-1-56881-266-3
WRIGHT, Richard S., Graham SELLERS a Nicholas HAEMEL OpenGL superbible: comprehensive tutorial and reference Sixth edition. Upper Saddle River, NJ: Addison-Wesley, 2014 ISBN 978-0-321-90294-8

Vedoucí diplomové práce: **Ing. Petr Veselý**
Katedra softwarových technologií

Datum zadání diplomové práce: **22. října 2018**
Termín odevzdání diplomové práce: **18. května 2019**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 17. listopadu 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10. 5. 2019

Martin Lepeška

PODĚKOVÁNÍ

Děkuji vedoucímu své diplomové práce, Ing. Petrovi Veselému, a Ing. Ivovi Skalickému za trpělivost, ochotu a zkušenosti, které mi předali i umožnili získat. Bez nich by tato práce nikdy nevznikla. Také děkuji své rodině za oporu.

ANOTACE

Práce se zabývá tvorbou aplikace pro vykreslování projekcí medicínských volumetrických dat. V první části jsou představeny formát DICOM a vybrané typy projekcí objemů. Druhá část se zabývá návrhem a implementací programu.

KLÍČOVÁ SLOVA

Java, OpenGL, MIP, MinIP, MPR, SSD, DICOM

TITLE

Projections of computer tomography images

ANNOTATION

This work is about developing application for rendering projections of medical volumetric data. First part introduces DICOM format and selected types of volume projections. Second part deals with design and implementation of program.

KEYWORDS

Java, OpenGL, MIP, MinIP, MPR, SSD, DICOM

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

AIP	Average Intensity Projection
API	Application Programming Interface
CT	Computer Tomography
DIMSE	DICOM Mesagge Service Element
DVR	Direct Volume Rendering
GUI	Graphical User Interface
GPU	Graphics Processing Unit
IOD	Information Object Definition
JOML	Java OpenGL Math Library
JSON	JavaScript Object Notation
LWJGL	LightWeight Java Game Library
MinIP	Minimal Intensity Projection
MIP	Maximal Intensity Projection
MPR	Multi Planar Reconstruction
OS	Operating System
PACS	Picture Archiving and Communication System
PDF	Portable Document Format
SCU	Service Class User
SCP	Service Class Provider
SOP	Service-Object Pair
SSD	Shaded Surface Display
XML	eXtensive Markup Language

ÚVOD

Cílem práce je vytvořit aplikaci předvádějící projekce a vizualizace volumetrických dat představené v nadcházejících kapitolách.

Po dokončení střední školy jsem v rámci praxe nastoupil do firmy vyvíjející diagnostický prohlížeč. Mým hlavním úkolem bylo zdokonalit funkce vykreslování medicínských snímků a projekcí. Získané zkušenosti se snažím zúročit a rozšířit v této práci, aby se pro čtenáře stala prvním krokem pro seznámení se s danou problematikou.

V první kapitole je čtenář seznámen s principy rentgenového záření, rentgenu a výpočetní tomografie. Také je zde popsáno, jak jsou snímky pořízené z těchto zařízení reprezentovány. V závěru kapitoly jsou porovnány dávky pohlceného záření z obou vyšetřovacích metod.

Ve druhé kapitole jsou vysvětleny základní pojmy a koncepty standardu DICOM, který byl navržen pro ukládání medicínských dat. Informace získané z této a předchozí části jsou klíčové pro porozumění ostatním částem práce.

Třetí kapitola se zabývá vizualizací volumetrických dat. Jsou zde popsány projekce běžně používané při diagnostice, ale i techniky sloužící k vizualizaci celého objemu. Představené výpočty jsou použity ve vytvořené aplikaci.

Čtvrtá kapitola pojednává o použitých technologiích. Pro knihovny je popsáno, jaký je jejich přínos, jakým způsobem a kde jsou ve zdrojovém kódu aplikace využity. V závěru je stručně popsán základní princip grafického API OpenGL.

Pátá kapitola je věnována krátké analýze a návrhu aplikace. Analýza stručně shrnuje představy o výsledné aplikaci. V návrhu jsou vysvětleny základní koncepty a myšlenky, jak vytvořený program funguje. Informace z této části umožní snadnější orientaci ve zdrojovém kódu.

Šestá kapitola rozvádí a popisuje další významné třídy, které vznikly při tvorbě programu. Jedná se především o nápady, jež urychlují proces vykreslování projekcí. Dále je zde věnována pozornost implementaci přenosové funkce zajišťující vykreslení celého objemu.

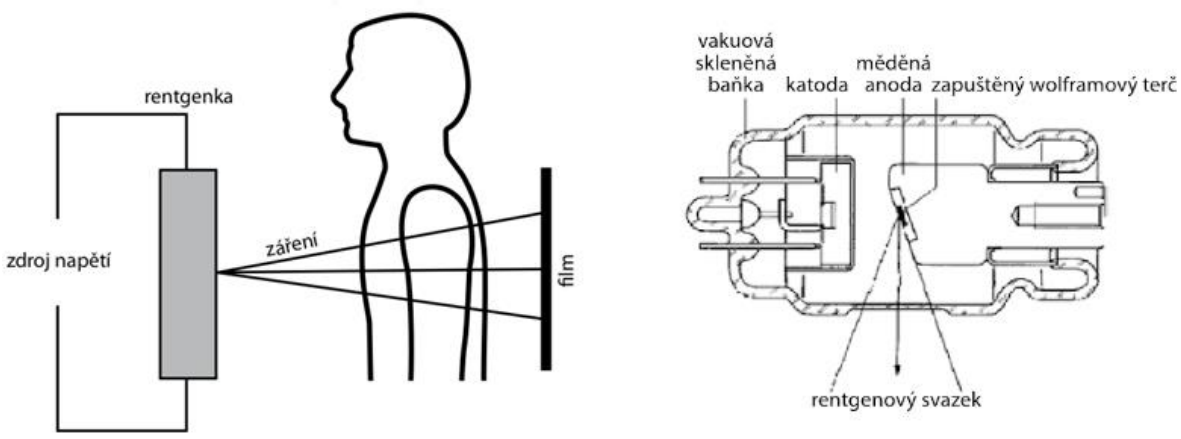
V závěrečné kapitole je zběžně popsáno základní ovládání programu. Pro bližší informace a instrukce jsou k práci přiloženy ukázková videa, testovací data a uživatelský manuál.

1 POŘÍZENÍ A REPREZENTACE SNÍMKU CT

Roku 1895 profesor Wilhelm Conrad Röntgen objevil Rentgenové záření. Jedná se o elektromagnetické vlnění s vlnovou délkou mezi 10^{-8} a 10^{-13}m. „*Prochází hmotou, v níž se částečně absorbuje a rozptyluje, přičemž množství absorbovaného a rozptýleného záření závisí na složení hmoty (jejím průměrném protonovém čísle, hustotě a tloušťce)“* [5]. Záření je používané jak rentgenem, tak i počítačovou tomografií. Základní princip obou zařízení je společný.

Na následujících schématech (Obr. 1) je předvedeno základní fungování RTG přístroje. Zdrojem záření je dioda, tzv. elektronka. Jako u běžné diody zde mezi žhavenou katodou a anodou proudí elektrony, z nichž je většina po dopadu na anodu přeměněna na teplo a malá část (1%) na rentgenové záření. Anoda bývá vyrobena z wolframu, který zlepšuje poměr přeměny a také má vyšší tepelnou odolnost. [6]

Rentgenové paprsky následně prochází tělem, kde je část pohlcena. Množství pohlcených paprsků závisí na materiálu. Například kosti pohltí více záření než měkké tkáně. Zbytek paprsků dopadá na film. [3]



Obr. 1: Schéma RTG přístroje a rentgenky

Zdroj: [3]

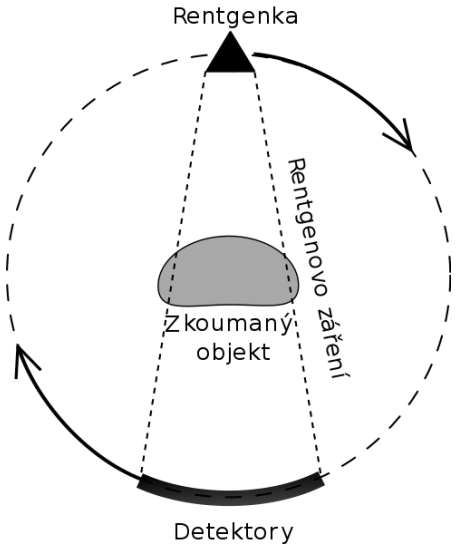
Následující ilustrace (Obr. 2) je příkladem rentgenového snímku břicha a pánve. Kosti jsou viditelné, avšak stav orgánů a měkkých tkání nelze ze snímku spolehlivě určit. *„Nedostatky rentgenových snímků však spočívají v tom, že jednotlivé orgány jsou zobrazeny sumárně, překrývají se. Nejsme tedy schopni vždy jednoznačně určit, kterými orgány rentgenový paprsek prošel, a touto metodou nelze vytvořit skutečný „anatomický“ řez těla“* [3].

Obr. 2: Ukázka rentgenového snímku

Zdroj: Vlastní tvorba

Počítačová tomografie (CT) řešící tento problém byla nezávisle na sobě objevena dvojicí vědců Godfreym Newbold Hounsfieldem a nezávisle na něm Allanem McLeod Cormackem v 80. letech minulého století. Vykreslení vrstvy, které tehdy zabralo až 20 minut, dnes díky pokrokům v oblasti IT trvá desítky vteřin. [3]

Zařízení nepořizuje pouze jeden snímek jako rentgen. Vyšetřovaná oblast je rovnoměrně rozdělena na vrstvy (řezy). Vzdálenost mezi vrstvami se nazývá tloušťka řezu. Čím je menší, tím více je zachyceno detailů, protože stoupá počet získaných snímků. Princip CT je představen na Obr. 3.



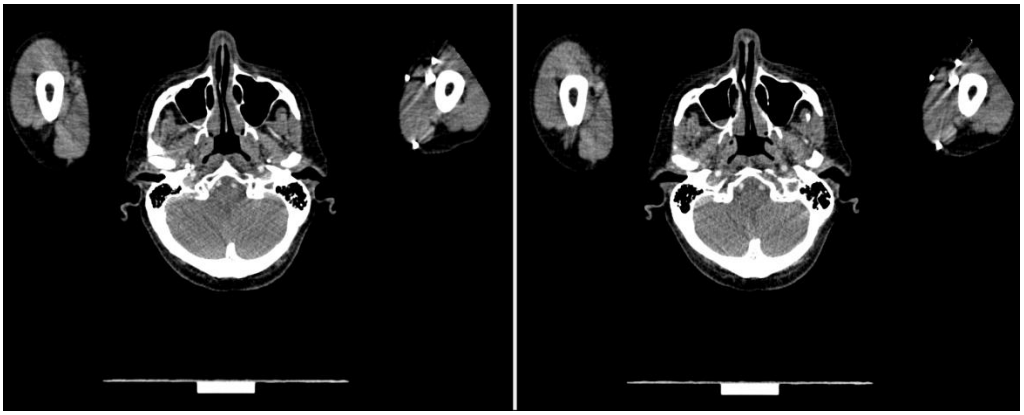
Obr. 3: Princip CT

Zdroj: Převzato z Wikipedie jako volné dílo

Kolem aktuálně snímané vrstvy krouží rentgenka vysílající rentgenové záření. Naproti rentgence jsou umístěny detektory. Ty zachycené záření transformují na signál, který je odeslán

do počítače. Ten výstupy zachycené z různých úhlů náběru při odlišné síle záření zkombinuje a pomocí inverzní Radonovy transformace[1] vypočítá snímek jedné vrstvy. [3] [6]

Na rozdíl od rentgenového snímku si diagnostik prohlíží na sebe navazující řezy (viz Obr. 4), což mu umožňuje zkoumat stav měkké tkáně i orgánů. Série řezů je možné pomocí algoritmů (představených v následující kapitole) promítat i v jiných rovinách, než je rovina náběru. Počítačová tomografie je využívána při vyšetření srdce, onemocnění v oblasti břišní dutin, osteoporózy končetin a mnohočetných poranění. [3]



Obr. 4: Dva po sobě jdoucí snímky CT

Zdroj: Vlastní tvorba

1.1.1 Denzita

Získané řezy jsou ukládány jako matice bodů. Pro každý bod je známa míra pohlceného záření. Ta se označuje jako tzv. denzita. Běžně se udává v uměle vytvořených Hounsfieldových jednotkách (viz vzorec 1). Přepočet využívá radiodenzitu vzduchu a vody. Radiodenzita udává relativní schopnost materiálu pohltit rentgenové záření oproti těmto látkám.

$$HU = 1000 \times \frac{\mu - \mu_{vody}}{\mu_{vody} - \mu_{vzduchu}} \quad (1)$$

Denzita bývá vykreslována ve stupních šedi, avšak na běžných monitorech není možné najednou zobrazit celou stupnici, protože nepodporují takovou hloubku šedé barvy. Řešením je používat speciální diagnostické černobílé monitory nebo zobrazovat pouze zvolené „okno“ z celého rozsahu. Všechny body s denzitou nižší nebo stejnou jako spodní hranice zvoleného intervalu jsou vykresleny černě. Naopak bodům s denzitou vyšší než horní hranice je přiděle-

[1] https://cs.wikipedia.org/wiki/Radonova_transformace

na bílá barva. Ostatním bodům je přidělena šedá barva v závislosti na relativní pozici v intervalu, viz vzorec 2).

$$R(HU) = \begin{cases} 0; HU \leq HU_{min} \\ HU \div (HU_{max} - HU_{min}), \\ 1; HU \geq HU_{max} \end{cases} \quad (2)$$

kde *R(HU)* je hodnota po provedení operace jasového okna nad hodnotou *HU*, HU_{min} a HU_{max} hranice zvoleného intervalu. Pro doplnění uvádí Tabulka 1 orientační hodnoty denzity vybraných tkání.

Tabulka 1: Orientační přehled denzity vybraných tkání

Zdroj: [5]

Materiál	Denzita (HU)
Vzduch	-1000
Plíce	⟨-800; -900⟩
Voda	0
Měkké tkáně	⟨25; 70⟩
Krev	40
Sražená krev	⟨65; 85⟩
Kosti, kalcifikace	>90
Ocel	20 000

1.1.2 Riziko z ozáření

Vystavení rentgenovému záření ve vyšších dávkách je pro lidské tělo nebezpečné. Tělo totiž absorbuje ionizující částice, jejichž energie stačí na vyražení elektronů z orbit atomů a následný vznik kationtů. *„Ionizované části molekul se stávají vysoce reaktivní a vedou k řadě chemických reakcí, které mohou způsobit smrt buňky, nebo změnit genetickou informaci"* [3].

Dávka pohlcené radiace je udávána v jednotkách Sievertů. Hodnota je určena součinem množství pohlcené energie na kilogram a tzv. jakostním koeficientem, který se liší pro různé typy materiálu a druhy ionizujícího záření. Průměrná dávka, kterou člověk absorbuje za rok, je přibližně 3,01 mSv. Následující Tabulka 2 obsahuje běžné dávky ozáření při vybraných

radiologických vyšetřeních. Z tabulky vyplývá, že při vyšetření CT pacient pohltí mnohem vyšší dávky záření. Riziko je vyvážené vyšší kvalitou pořízeného snímku. [3][10]

Tabulka 2: Běžné dávky záření

Zdroj: [3][23]

Vyšetření	Běžná efektivní dávka (mSv)	Srovnatelná doba pro stejnou dávku z přírodních zdrojů
Rentgen hrudníku	⟨0,01; 0,02⟩	10 dní
Rentgen zubu	0,005	1 den
Rentgen páteře	1.5	6 měsíců
Mamografie	0.4	7 týdnů
CT hlavy	⟨1,5; 2⟩	8 měsíců
CT páteře	6	2 roky
CT hrudníku	⟨5,8; 7⟩	2 roky
CT hrudníku, břicha a pánve	⟨9,9; 10⟩	3 roky
Srdeční CT angiogram	⟨6,7; 13⟩	4 roky

2 DICOM

„DICOM (Digital Imaging and Communications in Medicine) je všudypřítomný standard v radiologickém a kardiologickém odvětví pro výměnu obrazu a informací s obrazem spojených“ [1]. Standard je vzhledem ke své rozsáhlosti rozdělen na 18 částí, ve kterých jsou postupně popsány logické celky jako popis komunikace mezi dvěma zařízeními v nemocniční síti, vyměňované informace, způsob jejich uložení nebo například zabezpečení. Účelem této kapitoly je především seznámit čtenáře s částmi standardu nezbytnými pro pochopení zbytku práce. Názvy vysvětlovaných pojmů byly přeloženy do češtiny z anglického originálu.

2.1 Historie

Po objevení počítačové tomografie a dalších modalit vznikla vzhledem k vysokému množství různých výrobců diagnostických zařízení potřeba pro standardizování přenosu a ukládání vytvořených snímků. První verzi tohoto standardu s názvem ACR/NEMA 300 vytvořil v roce 1985 výbor zorganizovaný institucemi American College of Radiology a National Electrical Manufacturers Association. [4]

Mezi hlavní přínosy standardu patří:

- nezávislost vzniklých obrazových informací na výrobci zařízení;
- usnadnění vývoje systémů pro archivování snímků a komunikaci (PACS);
- umožnění tvorby databází s diagnostickými informacemi. [4]

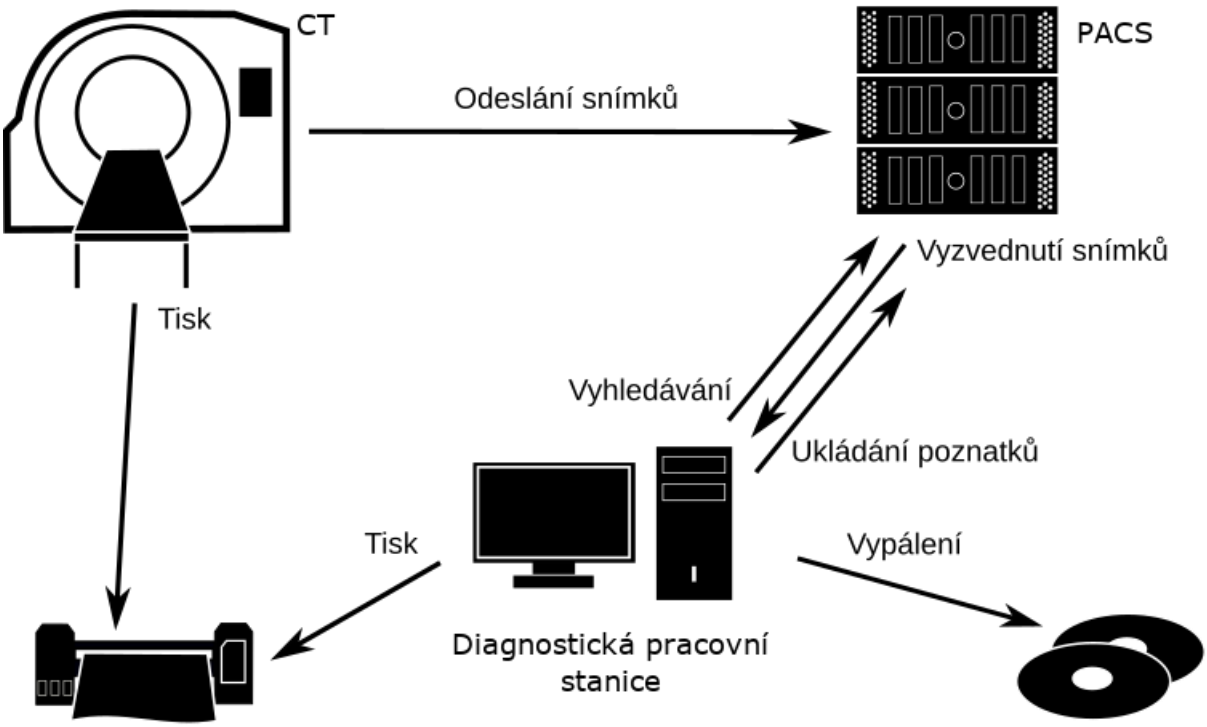
Standard se dále vyvíjel až do současné podoby. Třetí a zároveň aktuální verze byla přejmenována na DICOM a obohacena o další funkce, jako například:

- použitelnost v běžné síti, na rozdíl od předchozích verzí, které vyžadovaly přímé spojení;
- použitelnost v offline prostředí, díky specifikaci podporovaných standardů jako CD-R nebo FAT16;
- podrobnější specifikace chování zařízení při poskytování dané služby. [4]

2.2 Příklad komunikace zajištěné standardem DICOM

Na Obr. 5 je předveden příklad komunikace, kterou obstarává DICOM. Snímky nabrané z CT jsou odeslány na úložiště (PACS). Diagnostik je schopen si pomocí prohlížeče nainstalovaném na své pracovní stanici tyto snímky od úložiště vyžádat a v případě potřeby je

i vytisknout na speciální tiskárně. Měření a hlášení o nálezech, která následně vytvoří, jsou opět ukládány na PACS. DICOM také umožňuje načíst snímky a ostatní informace z tzv. offline média – CD-R. CD s uloženými snímky může pacient předat specializovanému lékaři na dodatečné zkoumání.



Obr. 5: Příklad komunikace mezi zařízeními pomocí DICOM standardu

Zdroj: Vlastní tvorba

Ve skutečnosti bývají snímky před odesláním na úložiště posílány navíc na tzv. PACS gateway. Jedná se o speciální stanici, kde obsluha zařízení kontroluje, zda nedošlo k žádným nedopatřením a náběr dat proběhl v pořádku.

2.3 Základní princip

Standard popisuje tzv. **Třídy Informačních Objektů** (Information Object Classes), které určují, co a jak bude ukládáno. S těmito informacemi manipulují služby popsané **Třídami Služeb** (Service Classes). **Třídy Párů Služeb a Objektů** (Service-Object Pair Classes) definují, za jakých podmínek a pro které objekty je služba použitelná. Typickým příkladem by mohla být **Třída Párů Služeb a Objektů** (SOP) Úložiště CT, kde úložiště je službou a CT informačním objektem. [4]

Koncovými body při komunikaci jsou tzv. aplikační entity, což je software nebo zařízení poskytující nebo používající služby. Aplikační entity poskytující službu jsou označeny jako Service Class Provider (SCP). Entitám používajícím službu je přiřazena role Service Class User (SCU). Zařízení může podporovat pro danou službu obě role a využívat je dle potřeby. [4]

Při vytváření spojení mezi aplikačními entitami dochází v první části k vyjednávání, kdy si zařízení vyměňují SOP třídy, aby zjistila, které služby mohou být použity a které naopak nejsou podporovány. [4][9]

Zjednodušený scénář komunikace v případě, že by zařízení potřebovalo uložit snímky CT na úložiště, by proběhl takto:

1) V první fázi dojde k tzv. navázání asociace. Žádající entita s rolí SCU odešle SOP třídy CT snímku včetně kódování, ve kterém je umí nabídnout.
2) SCP ze seznamu vybere dvojice, které podporuje a odešle je zpět. Tyto dvojice jsou nazývány tzv. Presentation Context a bývají označovány lichými čísly.
3) SCU nazpět odešle číslo dvojice s kombinací třídy a kódování, kterou použije, a následně odešle snímek jako proud bytů.
4) SCP nyní ví, co a jak SCU odesílá. Dokáže to tedy rozkódovat a odeslat potvrzení o správném přijetí.
5) SCU nyní odešle žádost o ukončení asociace.
6) SCP žádost potvrdí a spojení je ukončeno.

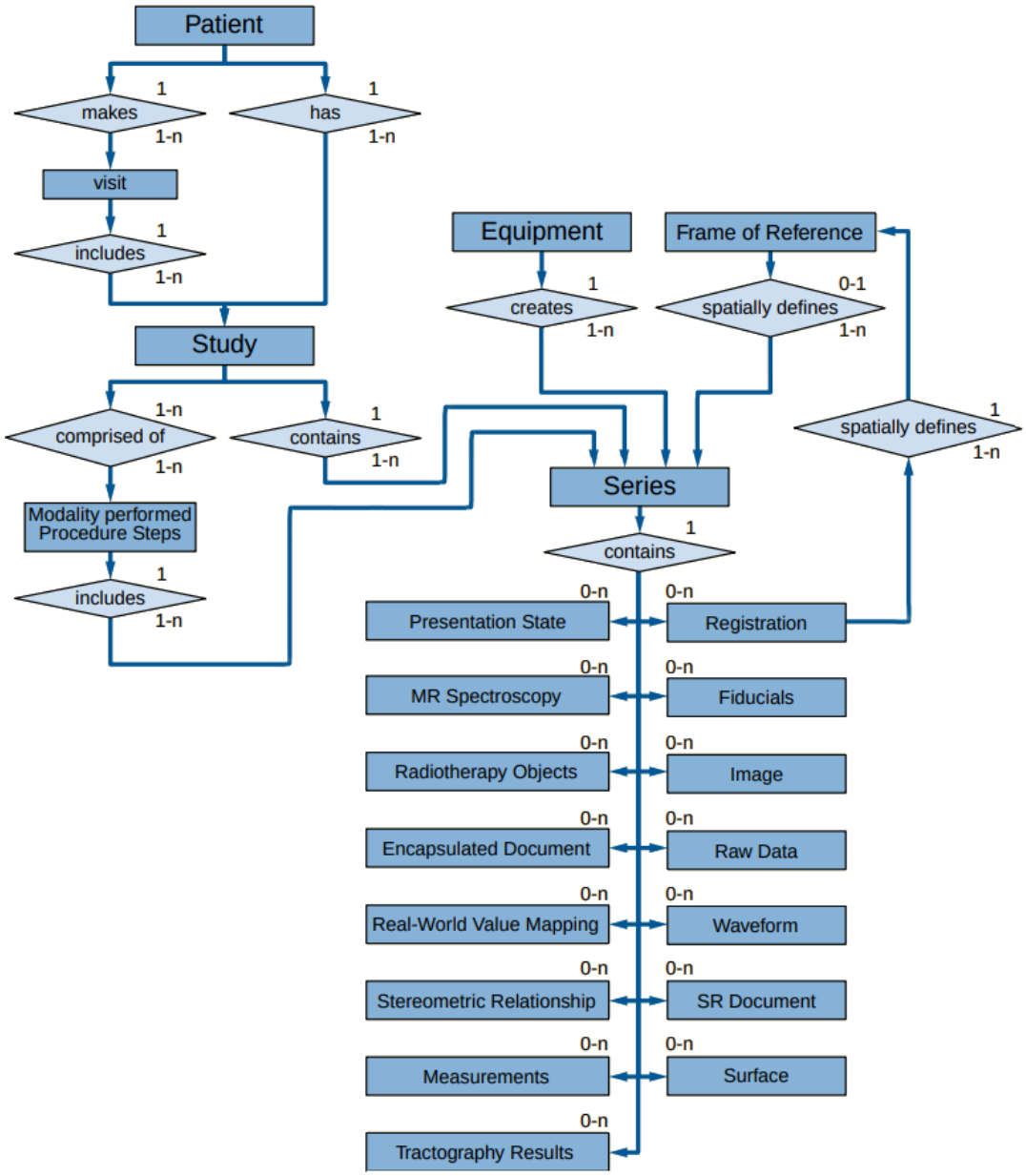
2.4 Třídy Informačních Objektů (IOD)

Standard v části PS 3.3 definuje tzv. **Třídy Informačních Objektů**. *Jedná se o abstrakci reálné informační entity (např. snímek z CT, strukturovaná zpráva apod.), která je zpracovávána příkazy DICOMu“* [4]. Každá z těchto tříd se skládá z atributů. Atributy jsou pojmenované vlastnosti nabývající hodnot nezávislých na kódování. [4]

Třídy se dělí na dva typy. Normalizované třídy obsahují pouze informace vlastní abstrahovanému objektu. Například třída reprezentující vyšetření by obsahovala informace o čase jeho výkonu, avšak jméno vyšetřovaného pacienta by patřilo přímo do třídy pacient. Složené třídy mohou navíc obsahovat informace, které se k nim vztahují. Třída CT snímku obsahuje atributy týkající se samotného obrazu, ale i údaje o pacientovi. [4]

Následující ilustrace (Obr. 6) reprezentuje model světa, kterým se DICOM zabývá. Bloky s tvarem kosočtverce představují vztahy mezi entitami, které jsou znázorněny jako obdélníky. Například pro pacienta může být vytvořena studie, která se skládá ze dvou sérií. První série obsahuje 120 snímků CT hrudníku s doprovodným měřením. Ve druhé je uložen průběh signálu z elektrokardiografu (waveform).

Entita *Frame of Reference* představuje místo v prostoru, vůči kterému jsou snímky v sérii relativně umístěné. Série může odkazovat pouze na jedno umístění, avšak jedno umístění může být použito více sériemi. [4]



Obr. 6: DICOM model reálného světa (upraveno)

Zdroj: [4]

2.5 Datové Sady

Vyměňované informace o objektech jsou uspořádány v tzv. **Datových Sadách** (Data Sets). Jedná se o množinu uspořádaných **Datových Elementů** (Data Elements), základních jednotek informace. Každý element se skládá z tagu, délky hodnoty a hodnoty samotné. Některé elementy také obsahují definovanou reprezentaci hodnoty, v případě, že je výběr ponechán na jejich tvůrci. Kompletní přehled elementů je definován v části standardu PS 3.6. [4][25]

Tag je unikátní identifikátor složený z čísla skupiny a čísla elementu v hexadecimálním formátu. Například element *Datum Narození Pacienta* má tag (0010, 0030). To znamená, že se nachází ve skupině *Pacient* 0010 a jeho číslo v rámci této skupiny je 0030. [4][25]

Délka hodnoty udává, kolik bajtů zabírá hodnota elementu. V závislosti na reprezentaci hodnoty se může jednat o 16 nebo 32 bitů. V případě, kdy délka nabývá hodnoty FFFFFFFF, není určena. Takový případ může nastat, pokud je ukládaná hodnota sekvence. [4]

Reprezentace hodnoty v elementu je určena dvěma bajty, z nichž každý znamená jeden znak, a dalšími dvěma rezervovanými pro budoucí použití. Tabulka 3 obsahuje výběr z možných hodnot pro demonstraci, jak mohou být ukládané informace odlišné. [4]

Tabulka 3: Výběr z možných Reprezentací hodnot Datového Elementu

Zdroj: [4]

Značení	Název	Popis	Povolené znaky	Délka
AE	Aplikační Entita	Unikátní řetězec reprezentující aplikaci	Běžné znaky	max. 16B
AS	Řetězec Věku	Třímístné číslo a znak reprezentující jednotku	0-9, D, W, M, Y	4B
DS	Desetinný řetězec	Číslo s plovoucí desetinnou tečkou	0-9, +, –, E, e, ‘.‘	max. 16B
PN	Jméno osoby	Pět volitelných částí oddělených ‘^‘	Běžné znaky	64 znaků za část
SQ	Sekvence	Sekvence vnořených datových setů	–	–
UC	Neomezené znaky	–	Běžné znaky, 0–9	max. 2^{32}-2B
UN	Neznámé	Binární data s neznámým kódováním	–	validní

Příkladem **Datového Elementu**[2] může být *Pacientovo Jméno* s tagem (0010, 0010). Označení reprezentace jeho hodnoty je PN. Jedná se tedy o jméno osoby, které se musí skládat pouze z běžných znaků. Jména jsou oddělena znakem '^' a nemohou být delší než 64 znaků.

2.6 Služby

Služby se skládají z tzv. **Elementů Služeb Zpráv standardu DICOM** (Dicom Message Service Element). Jedná se o základní operace, které je možné se zpracovávaným objektem provést. Elementy se dělí na dvě skupiny podle toho, se kterými typy objektů pracují. Do skupiny kompozitních příkazů patří:

- *C_Store,* operace pro odeslání objektu (např. CT snímku) na úložiště,
- *C_Find* představující vyhledávací dotaz na úložiště,
- *C_Get* pro odeslání informací uživatelské aplikaci na základě atributů odeslaných uživatelem,
- *C_Move* umožňující přesunovat i kopírovat kompozitní objekty v rámci aplikací,
- *C_Echo,* kontrola stavu aplikační entity. [4][25]

Vzhledem k tomu, že se kompozitní **Třídy Informačních Objektů** skládají z více tříd, lze je smazat pouze odstraněním všech normalizovaných tříd, které jsou jejich částí. Mezi příkazy pro práci s normalizovanými objekty patří:

- *N_Event_Report*, nahlášení události, například dokončení ukládání,
- *N_Get*, vyžádání informací o objektu s daným identifikátorem,
- *N_Set,* aktualizace informací o objektu s daným identifikátorem,
- *N_Action*, žádost pro vykonání určené akce na serveru,
- *N_Create*, vytvoření objektu pro budoucí použití,
- *N_Delete,* smazání objektu. [4]

Třídy párů objektů a služeb (SOP), které již byly zmíněny v kapitole 2.3, definují propojení IOD a služeb. Jedná se o sémantiku a pravidla, jenž mohou omezit použití některých DIMSE pro specifické informační objekty nebo jejich atributy. SOP třídy tedy určují, za jakých podmínek mohou být dané příkazy aplikovány a co bude jejich výstupem. [4]

[2] Kompletní přehled Datových Elementů: https://dicom.innolitics.com/ciods

Konkrétní službu popisuje tzv. **Služební Třída** (Service Class), která se skládá ze SOP tříd. Mezi služby patří například služba **Úložiště** (Storage) nebo **Dotaž/Vyzvedni** (Query/Retrieve). Služebním třídám se věnuje část standardu PS 3.4. [4]

2.7 Existující prohlížeče

Pro prohlížení souborů uložených ve standardu DICOM existuje celá řada open-source i proprietárních prohlížečů s různými funkcemi. Příkladem bohatě vybaveného diagnostického prohlížeče je OsiriX[3], který se kromě běžného prohlížení snímků a projekcí specializuje na vizualizaci objemů ve vysoké kvalitě pro použití v neurochirurgii. [35]

Opačným typem prohlížeče je open source projekt DICOM Viewer and Editor[4]. Program je určen především pro prohlížení a editaci obsahu **Datové Sady**. Aplikace neumožňuje zobrazovat vizualizace či rekonstrukce, ale jen náhledy snímků série.

Dalším open source projektem je 3D slicer[5]. Jedná se o program a zároveň i platformu určené pro analýzu a vizualizaci objemových dat. Vzhledem k rozsáhlosti platforma využívá modulární architekturu, aby uživatel spouštěl pouze části, které potřebuje. Aplikace není určená pro klinické použití, ale spíše pro výzkum.

Aplikace DroidRender[6] se zaměřuje pouze na operační systém Android. Prohlížeč umožňuje zobrazit většinu projekcí a vizualizací představených v této práci přímo na chytrém telefonu. Program není určen pro klinické použití.

Jedním z diagnostických prohlížečů, které pochází z České republiky, je program Dicompass[7]. Program podporuje většinu rekonstrukcí popsaných v této práci. Kromě operačního systému Windows prohlížeč podporuje i linuxové distribuce a MacOS.

---

[3] https://www.osirix-viewer.com

[4] https://sourceforge.net/projects/dicomeditorbybenp/

[5] https://www.slicer.org/

[6] https://play.google.com/store/apps/details?id=com.luolai.droidrender

[7] http://www.dicompass.cz

3 VIZUALIZACE VOLUMETRICKÝCH DAT

V této kapitole jsou popsány techniky pro zobrazení volumetrických dat v jiných rovinách, než jen rovině pořízení. „*Tyto techniky umožňují zkoumání jemných anatomických detailů, které by bylo obtížné vyhodnotit pouze pomocí osových rekonstrukcí“* [8]. V této práci jsou za volumetrická data, dále v této práci označovaná jako objemy, považovány třídimenzionální matice bodů s informacemi o obraze. Jsou zde uvedeny jak matematické vztahy pro výpočet tak i ukázky kódu s využitím shaderů OpenGL. Funkce shaderů je vysvětlena v kapitole 4.

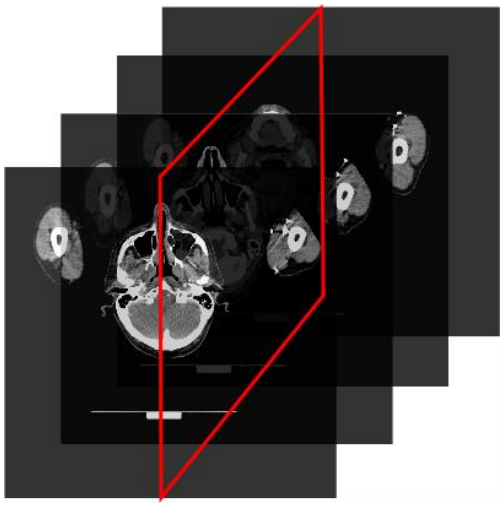
3.1 Multiplanární rekonstrukce (MPR)

Multiplanární rekonstrukce je technika pro zobrazení série CT snímků v libovolné rovině. *“Tato technika je zvláště užitečná při hodnocení kosterních struktur, protože některé fraktury a naklonění kloubu nemusí být očividné na axiálních řezech”* [7]. Na následujícím Obr. 7 jsou předvedeny nejběžněji používané roviny. Axiální rovina bývá rovinou náběru dat. Zbylé dvě roviny jsou k ní kolmé a umožňují nám tak zobrazit pomyslný řez napříč pacientem.

Obr. 7: Ukázka MPR z axiální, koronální a sagitální roviny

Zdroj: Vlastní tvorba

Pro lepší představu následuje ilustrace (Obr. 8) zobrazující červeně zvýrazněný obdélník, v němž se nachází poslední MPR z předchozího obrázku. Rovina nemusí být umístěna pouze ve středu objemu, ale i v jakémkoliv jiném bodě.



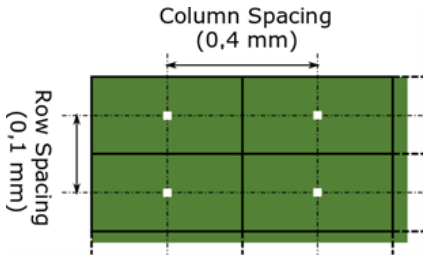
Obr. 8: Rovina sagitálního MPR umístěná v půlce objemu

Zdroj: Vlastní tvorba

3.1.1 Objem ze snímků v OpenGL

Sérii snímku lze považovat za objem. Jedná se o trojrozměrné pole, jehož buňky, tzv. voxely, mají předem zvolenou výšku, šířku i hloubku. Všechny voxely v rámci jednoho objemu mají tyto rozměry stejné. Voxely tedy mívají tvar kvádru. Na rozdíl od běžných případů lze mezi voxely objemů popsaných v této práci interpolovat.

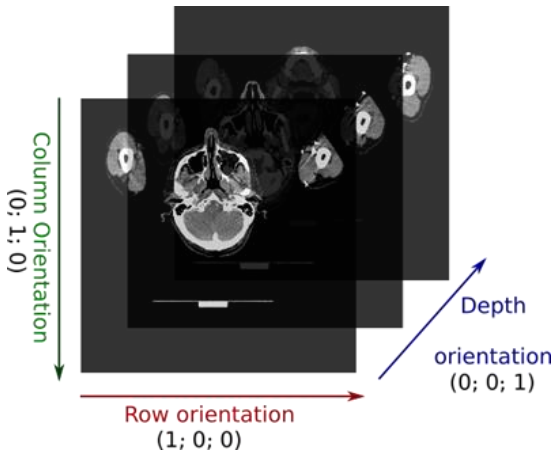
Snímky série bývají uloženy ve formátu DICOM. Tento typ souboru na začátku obsahuje tzv. atributy, jejichž přítomnost, obsah a reprezentace je dána standardem. Atributy použité v této kapitole jsou definovány jako součást IOD (viz kapitola 2.4) snímku CT. Informaci o počtu voxelů v řádcích a sloupcích v sobě nesou atributy *Columns* a *Rows*. Pro zjištění rozměrů voxelu lze použít atribut *PixelSpacing*, což jsou dvě čísla reprezentující vzdálenosti středů voxelů mezi řádky a sloupci. Obr. 9 obsahuje příklad demonstrující, co tyto atributy reprezentují. Za hloubku voxelu považujeme atribut *SliceThickness*, mezeru mezi řezy při náběru.



Obr. 9: Atribut PixelSpacing

Zdroj: Vlastní tvorba

Poslední potřebnou informací, kterou potřebujeme o objemu vědět, je orientace řezů při náběru. Jedná se o dva jednotkové vektory uložené v atributu *ImageOrientation* udávající informace o směru řádků a sloupců objemu. Z těchto dvou vektorů lze pomocí vektorového součinu vypočítat směr hloubky, tzv. Depth orientation, který je na ně kolmý (viz. Obr. 10).



Obr. 10: Vektory orientace objemu

Zdroj: Vlastní tvorba

Při vykreslování pomocí OpenGL lze využít principu vykreslování *3D textury*. Jedná se o strukturu obsahující obrazová data, která je uložena v paměti grafické karty. Výhodou využití *3D textury* je, že pokud jsou požadovány obrazové informace o bodu, který leží mezi snímky, grafická karta automaticky vrátí interpolovanou hodnotu mezi nejbližšími hodnotami v textuře.

Pro nahrání snímků do *3D textury* postačí informace o počtu snímků, délce řádků a sloupců. Pro zobrazení snímku tak, aby odpovídal realitě, je třeba při vykreslování počítat i s rozměry voxelů, jinak by se výsledný snímek mohl zobrazit deformovaně. Ostatní atributy, které byly představeny v předchozí části, jsou použity při výpočtu texturovacích souřadnic.

Texturovací souřadnice udávají, z jaké části textury má grafická karta čerpat při přiřazování barvy fragmentu tvaru, jenž bude vykreslen. V tomto případě se jedná o soustavu čtyř bodů reprezentujících obdélník v prostoru *3D textury* objemu.

3.1.2 Výpočet texturovacích souřadnic projekce

Je známa velikost voxelu $\boldsymbol{s_v}$ a textury $\boldsymbol{s_t}$. Velikost objemu $\boldsymbol{s}$ lze určit jako

$$\vec{\boldsymbol{s}} = (s_{vx} \cdot s_{tx};\quad s_{vy} \cdot s_{ty};\quad s_{vz} \cdot s_{tz}). \tag{3}$$

Dále jsou známy sloupcové vektory orientace objemu *rowOr*, *colOr* a *depthOr*, z nichž lze sestavit matici orientace

$$\boldsymbol{O} = \left[\overrightarrow{rowOr} \quad \overrightarrow{colOr} \quad \overrightarrow{depthOr}\right]. \tag{4}$$

Pro zobrazení objemu z jiné, než původní roviny, je třeba znát její orientaci, která je podobně složena z nových vektorů orientace

$$\boldsymbol{R} = \left[\overrightarrow{rRowOr} \quad \overrightarrow{rColOr} \quad \overrightarrow{rDepthOr}\right]. \tag{5}$$

Dále potřebujeme znát relativní hloubku projektované roviny *d*. Ta popisuje, jak hluboko v objemu se rovina s danou orientací nachází. Pokud je hodnota rovna 0 nebo 1, rovina se dotýká rohů objemu. V případě, že je hodnota rovna $\frac{1}{2}$, rovina prochází středem.

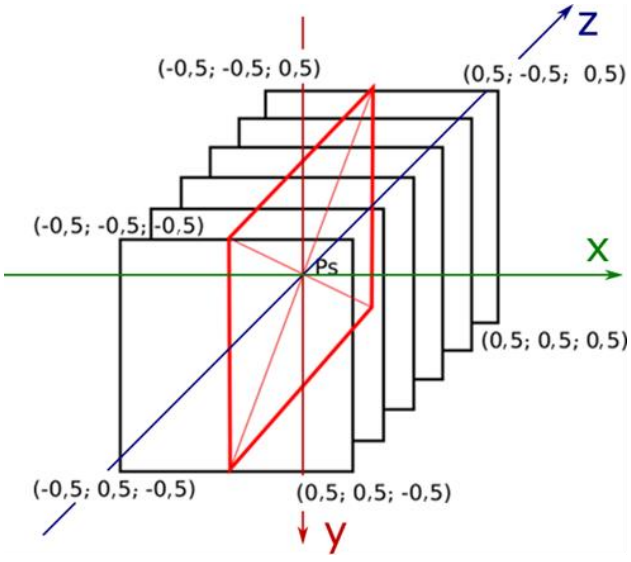
Bod ležící ve středu objemu, kolem něhož je prováděna rotace, je dán jako

$$P_S = \frac{\boldsymbol{O} \times \vec{\boldsymbol{s}}}{\boldsymbol{2}}. \tag{6}$$

Pro zjištění odsazení rohu objemu vůči středu (viz. Obr. 11) je vypočítána původní pozice daného rohu, která je následně pomocí nové orientace $\boldsymbol{R}$ natočena

$$P_C = \left(\left(\overrightarrow{rowOr} \times (\boldsymbol{s}_x \cdot \boldsymbol{c}_x)\right) + \left(\overrightarrow{colOr} \times (\boldsymbol{s}_y \cdot \boldsymbol{c}_y)\right) + \left(\overrightarrow{depthOr} \times (\boldsymbol{s}_z \cdot \boldsymbol{c}_z)\right)\right) \times \boldsymbol{R}, \tag{7}$$

kde $\boldsymbol{c}$ je relativní pozice rohu vůči středu v rámci objemu.



Obr. 11: Relativní pozice rohů vůči středu objemu

Zdroj: Vlastní tvorba

Dále se určí vektor $\boldsymbol{v}$, jehož složkami jsou rozdíly mezi minimálními a maximálními pozicemi rohů v jednotlivých osách, pak lze zjistit pozici rohu zobrazované projekce v rámci rotovaného objemu pomocí přičítání délky $\boldsymbol{v}$ k neměnící se pozici středu

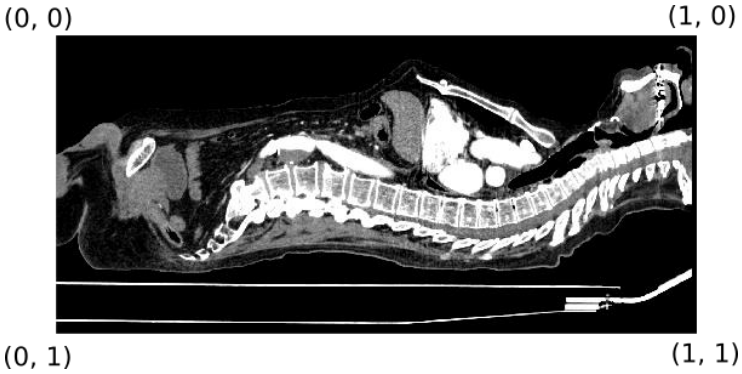
$$p_i = P_S + \left(\left(\overrightarrow{rRowOr} \times \left((u_{ix} - 0{,}5) \cdot \boldsymbol{v}_x\right)\right)\right) + \left(\left(\overrightarrow{rColOr} \times \left((u_{iy} - 0{,}5) \cdot \boldsymbol{v}_y\right)\right)\right) + \left(\left(\overrightarrow{rDepthOr} \times \left((d - 0{,}5) \cdot \boldsymbol{v}_z\right)\right)\right), \quad (8)$$

kde u_i je relativní pozice rohu zobrazované projekce.

Pro převedení na texturovací souřadnice, jejichž příklad je uveden na Obr. 12, stačí úprava

$$t_i = \frac{\boldsymbol{O}^{-1} \times p_i}{\boldsymbol{s}}, \quad (9)$$

kde za výsledek dělení vektorů považujeme vektor sestavený z podílů odpovídajících složek.



Obr. 12: Texturovací souřadnice

Zdroj: Vlastní tvorba

Pro vykreslení pomocí OpenGL jsou tak zjištěny všechny potřebné informace. V OpenGL je pouze vykreslen obdélník s rozměry vektoru $\boldsymbol{v}$, který je texturován objemem s vypočítanými souřadnicemi. V případě, že je výsledek vykreslen uživateli, je vhodné transformovat jasové hodnoty jednotlivých fragmentů (pixelů) pomocí jasového okna, které bylo popsáno v minulé části v podkapitole o denzitě.

3.1.3 Příklad výpočtu

Pokud by měl být vytvořen objem ze tří po sobě jdoucích snímků, pro které jsou známy atributy *PixelSpacing* a *SliceThickness*, pak platí, že velikost voxelu je dána jako:

$$\overrightarrow{\boldsymbol{s_v}} = (PixelSpacing_x;\quad PixelSpacing_y;\quad SliceThickness) = (5;\quad 5;\quad 5). \quad (10)$$

Pomocí hodnot atributů *Cols*, *Rows* a počtu řezů *n* lze zjistit velikost textury:

$$\overrightarrow{\boldsymbol{s_t}} = (Cols; \quad Rows; \quad n) = (3; \quad 3; \quad 3). \tag{11}$$

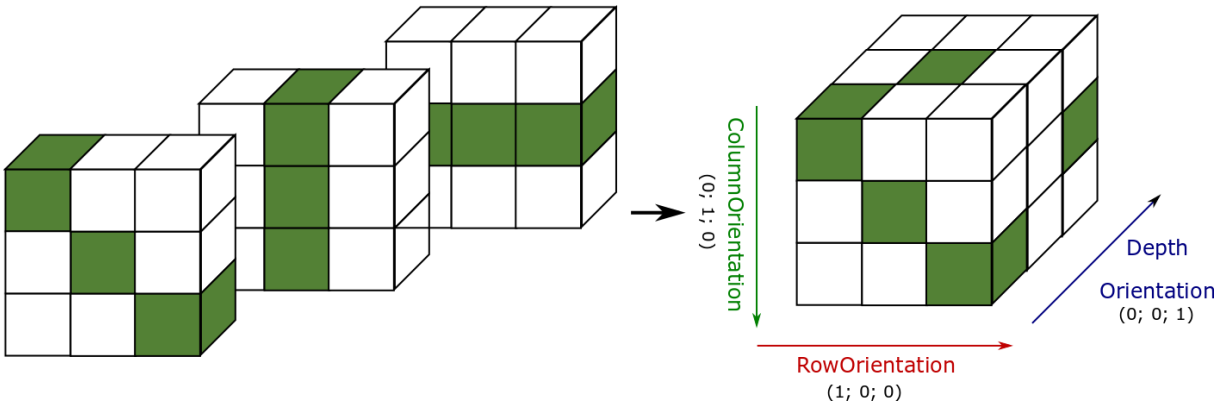
Použitím vztahu 3 je určena celková velikost objemu:

$$\vec{\boldsymbol{s}} = (s_{vx} \cdot s_{tx}; \quad s_{vy} \cdot s_{ty}; \quad s_{vz} \cdot s_{tz}) = (15; \quad 15; \quad 15). \tag{12}$$

Dále je třeba určit matici orientace objemu $\boldsymbol{O}$. Směr náběru dat, vektor *depthOr*, je dán vektorovým součinem vektorů, které jsou hodnotami atributu *ImageOrientation*:

$$\boldsymbol{O} = \left[\overrightarrow{rowOr} \quad \overrightarrow{colOr} \quad \overrightarrow{depthOr}\right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{13}$$

Na Obr. 13 je předveden objem použitý v tomto příkladě. Malé krychle reprezentují voxely načtené ze snímků.



Obr. 13: Ukázkový objem

Zdroj: Vlastní tvorba

Nyní je třeba určit směr, ze kterého bude planární rekonstrukce provedena, a hloubku v rámci objemu *d*, v jaké se bude rekonstrukce nacházet. Směr je dán jednotkovými vektory *rRowOr*, *rColOr* a výsledkem jejich vektorového součinu *rDepthOr*. Z těchto vektorů je vytvořena matice $\boldsymbol{R}$:

$$\boldsymbol{R} = \left[\overrightarrow{rRowOr} \quad \overrightarrow{rColOr} \quad \overrightarrow{rDepthOr}\right] = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}. \tag{14}$$

Aby rovina projekce procházela objemem v jeho polovině, je zvolena hodnota *d* 0.5. Pozice středu objemu je vypočítána vztahem:

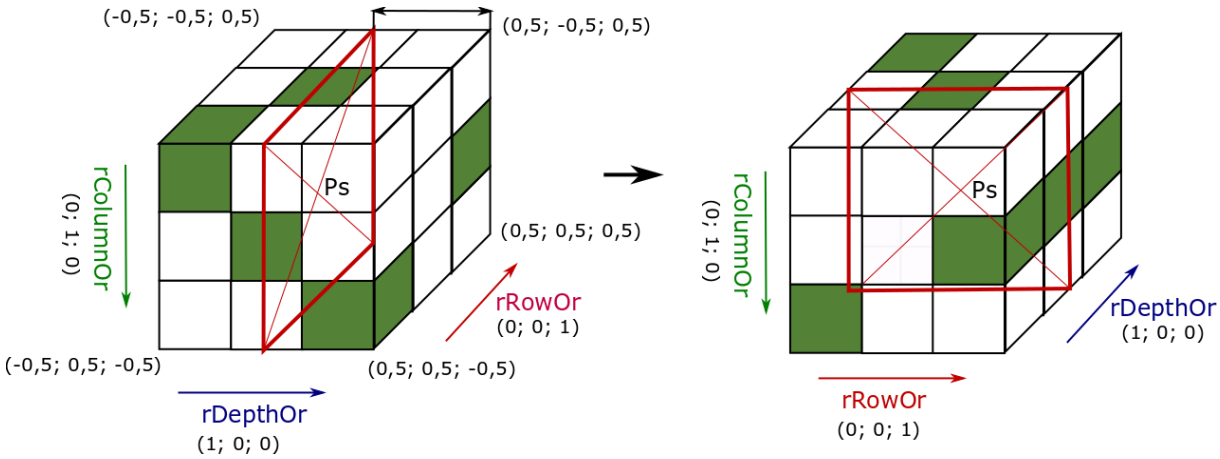
$$P_s = \frac{\boldsymbol{o}\times\vec{\boldsymbol{s}}}{\boldsymbol{2}} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \times \frac{(15;\ \ 15;\ \ 15)}{2} = (7{,}5;\ \ \ 7{,}5;\ \ \ 7{,}5). \tag{15}$$

Pozici pravého dolního rohu objemu z pohledu roviny rekonstrukce lze zjistit:

$$P_{(0{,}5;\ \ 0{,}5;\ \ -0{,}5)} = \begin{pmatrix} \overrightarrow{rowOr} \times (\boldsymbol{s}_x \cdot \boldsymbol{c}_x) \\ +\overrightarrow{colOr} \times (\boldsymbol{s}_y \cdot \boldsymbol{c}_y) \\ +\overrightarrow{depthOr} \times (\boldsymbol{s}_z \cdot \boldsymbol{c}_z) \end{pmatrix} \times \boldsymbol{R} = \begin{pmatrix} (1;0;0) \times (15 \cdot 0{,}5) \\ +(0;1;0) \times (15 \cdot 0{,}5) \\ +(0;0;1) \times \big(15 \cdot (-0{,}5)\big) \end{pmatrix} \times \boldsymbol{R}$$

$$= (\boldsymbol{7,5};\ \boldsymbol{7,5};\ \boldsymbol{-7,5}) \times \begin{bmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix} = (-7{,}5; 7{,}5; 7{,}5). \tag{16}$$

Tímto postupem jsou zjištěny pozice všech rohů. Na Obr. 14 je předvedeny objemy před a po natočení do směru rekonstrukce.



Obr. 14: Rotace ukázkového objemu

Zdroj: Vlastní tvorba

Po nalezení pozic všech rohů je z nich určen vektor ***v***, jenž udává rozdíly mezi minimálními a maximálními pozicemi rohů v jednotlivých osách. Hodnotu vektoru ***v*** lze tedy považovat za velikost projekce pro dané rozměry. Vzhledem k tomu, že v tomto případě je velikost objemu ve všech rozměrech stejná, se hodnota vektoru ***v*** neliší od hodnoty vektoru ***s***.

Jsou-li známy rozměry výsledné projekce, pak je relativní umístění pravého spodního rohu projekce vůči středu objemu vyjádřeno vztahem:

$$p_4 = P_S + \left(\left(\overrightarrow{rRowOr} \times \left((u_{4x} - 0.5) \cdot \boldsymbol{v}_x\right)\right)\right) + \left(\left(\overrightarrow{rColOr} \times \left((u_{4y} - 0.5) \cdot \boldsymbol{v}_y\right)\right)\right) +$$

$$\left(\left(\overrightarrow{rDepthOr} \times \left((d - 0.5) \cdot \boldsymbol{v}_z\right)\right)\right) = (7{,}5;\ 7{,}5;\ 7{,}5) + \left((0; 0; 1) \times \left((1 - 0{,}5) \cdot 15\right)\right) +$$

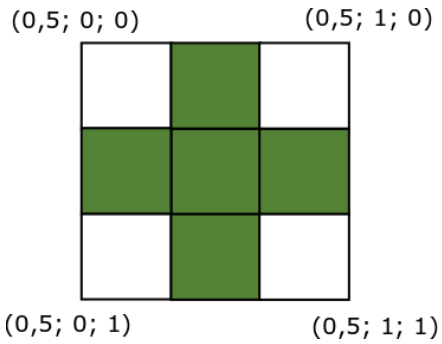
$$\left((0; 1; 0) \times \left((1 - 0{,}5) \cdot 15\right)\right) + \left((1; 0; 0) \times \left((0{,}5 - 0{,}5) \cdot 15\right)\right) = (7{,}5;\ 15;\ 15), \quad (17)$$

Kde u_i je pozice rohu v rámci textury (viz Obr. 12).

Texturovací souřadnice jsou získány převedením pozic rohů p_i za pomoci vzorce 9. Texturovací souřadnice pravého spodního rohu tedy je:

$$t_4 = \frac{\boldsymbol{o}^{-1} \times p_4}{\boldsymbol{s}} = \frac{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times (7{,}5\ ;15\ ;15)}{(15;15;15)} = (0.5\ ;\ 1;\ 1). \quad (18)$$

Na Obr. 15 je zobrazena výsledná projekce se všemi vypočítanými texturovacími souřadnicemi.



Obr. 15: Výsledná projekce ukázkového objemu

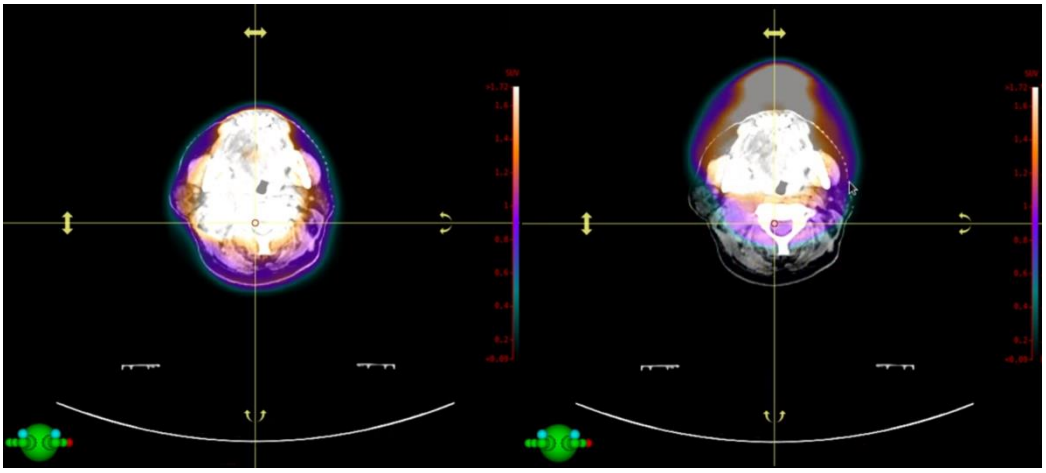
Zdroj: Vlastní tvorba

3.1.4 Možná rozšíření a optimalizace

MPR je základem pro ostatní projekce, kde jsou využity stejné výpočty. Samotné se dá ale využít pro další zobrazení. Jedním z těchto rozšíření jsou fůze, které přes sebe vykresluje MPR různých objemů jako vrstvy. Vrstvám jsou přiřazeny palety transformující denzitu na barvu a průhlednost.

Na Obr. 16 je předvedena ukázka fůze využívající dvě vrstvy. V pravé části je jedna z vrstev posunuta pro lepší rozeznatelnost jednotlivých vrstev. Spodní vrstva je planární rekonstrukcí objemu složeného z CT snímků. Vrchní obarvená vrstva je vytvořena z objemu složeného ze snímků pozitronové emisní tomografie (PET), což je zobrazovací metoda z oboru nukleární medicíny. „*Jaderná medicína je odvětví lékařského zobrazování, které používá malé množ-*

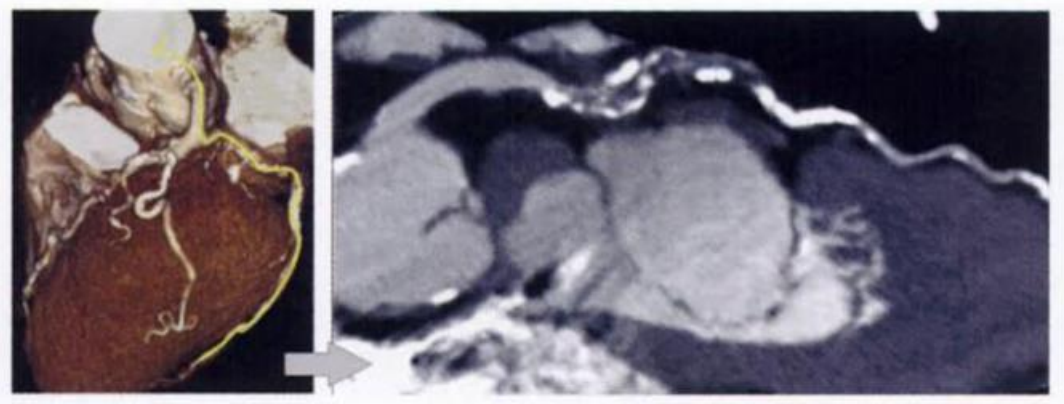
ství radioaktivního materiálu k diagnóze a stanovení závažnosti nebo léčbě různých nemocí, včetně mnoha typů rakoviny, srdečních onemocnění, gastrointestinálních, endokrinních, neurologických poruch a dalších abnormalit v těle" [11].



Obr. 16: Fůze CT a PET v programu DicompassW

Zdroj: Vlastní tvorba

Zakřivené MPR je variantou, kdy je promítaná rovina zakřivená. *„Tento typ vykreslovací techniky je aplikován na objemy obsahující cévní struktury a narovnává jakoukoliv křivku skrze pacienta"* [12]. Na Obr. 17 je ukázka MPR zobrazující rovinu danou žlutě zvýrazněnou křivkou. [7]



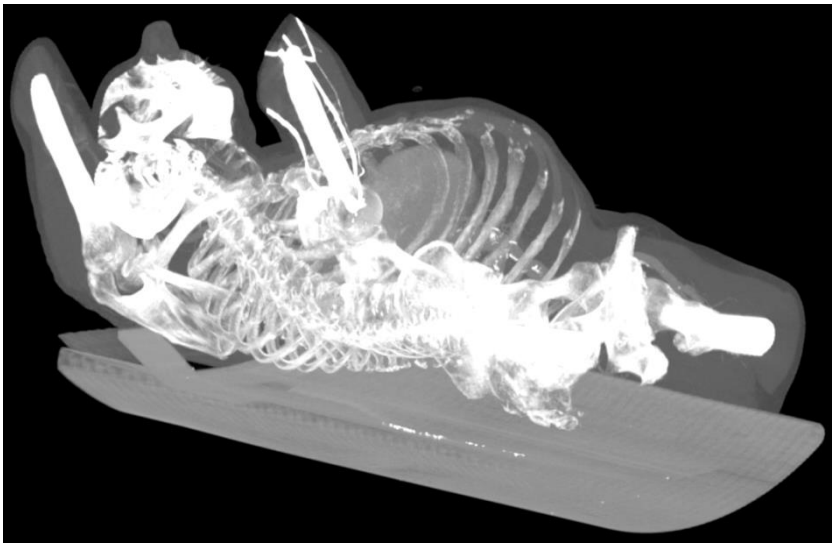
Obr. 17: Demonstrace zakřiveného MPR koronární tepny

Zdroj: [12]

3.2 Projekce maximální intensity (MIP)

MIP na rozdíl od MPR nezobrazuje obsah pouze jedné roviny, ale zvoleného úseku objemu. Pro určení hodnoty pixelu na souřadnicích XY je vždy vybrán voxel s maximální hodnotou

denzity z voxelů ležících ve směru paprsku (viz. Obr. 18). *„Detekce voxelů s vyšší denzitou umožňuje radiologovi lépe porozumět rozpětí a morfologii těles jako jsou cévy, uzliny, kalcifikace, chirurgické klipy a cizí tělesa“* [8].



Obr. 18: Ukázka MIP

Zdroj: Vlastní tvorba

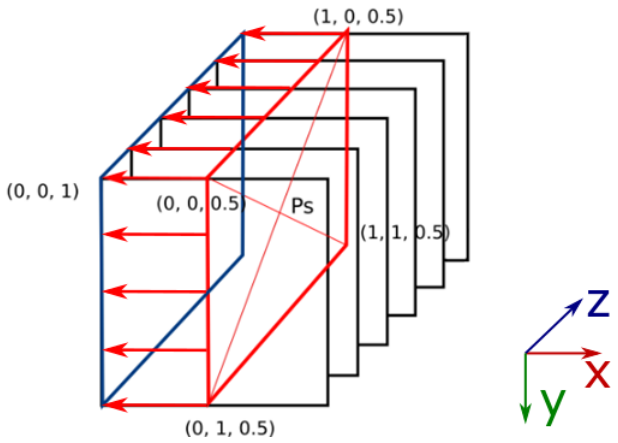
Pro snazší pochopení je funkce paprsku předvedena na Obr. 19. Na rozdíl od běžných metod sledování paprsku je paprsek v této práci používán pro průchod všech voxelů v jeho směru a určení výsledné barvy.

Obr. 19: Funkce paprsku v MIP

Zdroj: Vlastní tvorba

3.3 Výpočet texturovacích souřadnic projekce

Pro tuto techniku lze využít využít vzorce z předchozí podkapitoly o MPR. Pro každý fragment zpracovaný v shaderu je totiž vyslán paprsek hledající maximální hodnotu v textuře objemu. Na Obr. 20 je předvedena tato myšlenka při vykreslování MIP ze sagitální roviny. Paprsky jsou reprezentovány červenými šipkami.



Obr. 20: Souřadnice při výpočtu MIP

Zdroj: Vlastní tvorba

Pro zjištění směru, kterým se paprsek ubírá, lze použít vzorec

$$\vec{\boldsymbol{d}} = t_{(0,\ 0,\ 1)} - t_{(0,\ 0,\ 0.5)}, \tag{19}$$

kde $t_{(0,\ 0,\ 1)}$ a $t_{(0,\ 0,\ 0.5)}$ jsou texturovací souřadnice pro daný bod objemu.

Dále je třeba znát délku vektoru, což je dvojnásobná délka vektoru paprsku, protože při jeho výpočtu byl výchozí bod umístěn ve středu objemu. Jeho hodnota je tedy dána jako

$$|\vec{d}| = \sqrt{\vec{d}_x^2 + \vec{d}_y^2 + \vec{d}_z^2} \cdot 2. \tag{20}$$

Délka jednoho kroku na cestě paprsku je nejmenší z rozměrů voxelu

$$\Delta_{\vec{d}} = min(s_{vx}, s_{vy}, s_{vz}). \tag{21}$$

Jak demonstruje Zdroj. kód 1, v shaderu je z původní pozice fragmentu v textuře objemu (UV) vysílán paprsek hledající nejvyšší hodnotu voxelu. Při průchodu také dochází ke kontrole, že bod na paprsku neleží mimo texturu.

Zdroj. kód 1: Algoritmus MIP v GLSL

```
vec3 krok = paprsek*delkaKroku;
int pocetKroku = int(delkaPaprsku/delkaKroku)+1;
vec3 pozicePaprsku = UV + float(pocetKroku/2.0)*krok;
float hodnota = -1;
for(int i=0; i< pocetKroku; i++){
     if(paprsekSeNachaziVObjemu(pozicePaprsku)) {
        hodnota = max(hodnota, texture(texturaObjemu, pozicePaprsku).r);
      }
     pozicePaprsku -= krok;
}
vyslednaBarva = vec4(hodnota, hodnota, hodnota, 1.f);
```

3.3.1 Možná rozšíření a optimalizace

Nevýhodou MIP je, že některé struktury, například kosti, mohou ve výsledné projekci překrýt objekty zájmu. Tomu lze částečně zamezit pomocí určení tzv. desky. Místo celého objemu je zobrazena pouze část. Paprsek projde pouze snímky nacházejícími se v desce. Této techniky lze snadno docílit tím, že při výpočtu paprsku jsou použity body na krajích desky a v shaderu dojde k posunutí na vhodnou počáteční pozici. [13] [7]

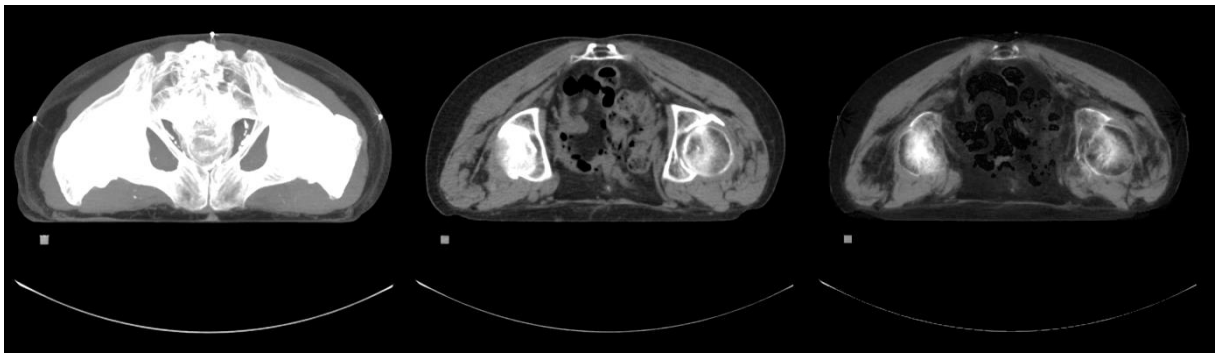
Další možností je definovat tzv. oblast zájmu. Jedná se o část objemu, které paprsek nebude při průchodu objemem ignorovat. Ostatní části nejsou na výsledné projekci zobrazeny. Jednoduše implementovatelnou oblastí zájmu může být například krychle uvnitř objemu. Moderní nástroje umožňují uživateli si oblast nakreslit v interaktivním grafickém režimu. [7]

3.4 Projekce průměrné intensity (AIP)

Technika využívá stejný algoritmus jako MIP s deskou, ale místo nejvyšší nalezené hodnoty je vrácena hodnota průměrná. Deska obvykle obsahuje pouze několik snímků. Výsledek se podobá „silnějšímu“ MPR. „*To může být užitečné pro charakterizaci vnitřních struktur pevného orgánu nebo stěn dutých struktur, jako jsou krevní cévy nebo střeva“* [7].

3.5 Projekce minimální intensity (MinIP)

Podobně jako AIP MinIP počítá pouze s tenkou deskou místo celého objemu. Na rozdíl od průměrné hodnoty voxelů na cestě paprsku ale vrací nejnižší nalezenou hodnotu. Vrácená hodnota ve skutečnosti nemusí být skutečně nejnižší, protože by algoritmus v některých částech objemu našel pouze vzduch. Proto je vhodné definovat i krajní hodnotu, kterou voxel musí mít, aby jej paprsek neignoroval. *„MinIP není běžně používán, ale může být využit při generování obrazů centrálních dýchacích cest nebo oblastí zachyceného vzduchu v plicích“* [7]. Na následujícím Obr. 21 jsou předvedeny rozdíly mezi typy projekcí pánve. Levá projekce typu MIP zobrazuje především kosti s vysokou denzitou. Oproti tomu pravá projekce typu MinIP zobrazuje měkké tkáně, v tomto případě střeva a jejich obsah. Prostřední projekce je MPR ze středu objemu pro porovnání.

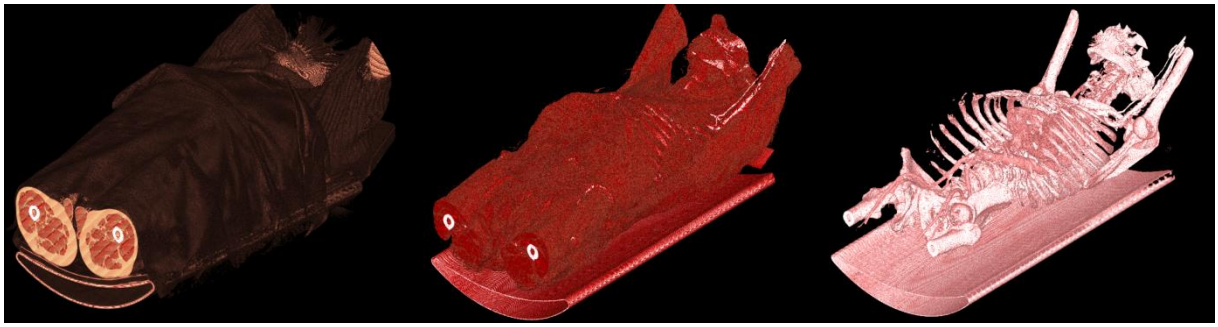


Obr. 21: Ukázka MIP, MPR a MinIP

Zdroj: Vlastní tvorba

3.6 Zobrazení stínovaného povrchu (SSD)

SSD poskytuje 3D pohled na povrch objemu. Na rozdíl od ostatních projekcí se paprsek zastaví na prvním nalezeném voxelu, jehož hodnota přesahuje minimální hodnotu jasového okna. Na Obr. 22 jsou předvedeny zobrazení při různých hraničních hodnotách.



Obr. 22: Ukázka SSD při různých minimálních hodnotách voxelu

Zdroj: Vlastní tvorba

Hodnota je dále převedena na Hounsfieldovy jednotky, kterým je pomocí palety přiřazena barva. *„SSD se používalo k demonstraci nálezů, jako jsou zlomeniny poté, co byly diagnostikovány na dvourozměrných obrazech“* [7].

3.6.1 Výpočty v shaderu

Paletu, která transformuje denzitu na barvu lze reprezentovat pomocí jednorozměrné textury, kde souřadnice x bude hodnota HU. Před použitím palety je třeba nejprve převést hodnoty voxelů, která nabývají hodnot $\langle 0; 1 \rangle$, na Hounsfieldovy jednotky.

Hodnoty z textury pořízené zařízením totiž z technických důvodů nemusí vždy odpovídat přímo jednotkám denzity. Pro převod jsou použity DICOM atributy *Rescale Slope* a *Rescale Intercept*

$$HU = d * RescaleSlope + RescaleIntercept\ , \tag{22}$$

kde *d* je hodnotou voxelu.

Pro to, aby výsledný obraz vypadal plasticky, je třeba počítat také s osvětlením. Gradient okolí voxelu lze považovat jako normálu osvětleného povrchu, jak je převedeno ve

Zdroj. kód 2. Gradient je zde počítán z hodnot voxelů.

Zdroj. kód 2: Hledání normály voxelu

```
vec3 spoctiNormaluVoxelu(vec3 pozVoxelu, vec3 velikostVoxelu)
   vec3 n = vec3(
         texture(objem, pozVoxelu - vec3(velikostVoxelu.x, 0.0, 0.0)).r
         - texture(objem, pozVoxelu + vec3(velikostVoxelu.x, 0.0, 0.0)).r,
        texture(objem, pozVoxelu - vec3(0.0, velikostVoxelu.y, 0.0)).r
         - texture(objem, pozVoxelu + vec3(0.0, velikostVoxelu.y, 0.0)).r,
        texture(objem, pozVoxelu - vec3(0.0, 0.0, velikostVoxelu.z)).r
         - texture(objem, pozVoxelu + vec3(0.0, 0.0, velikostVoxelu.z)).r);
   return normalize(n);
}
```

Následující příklad (Zdroj. kód 3) počítá pouze s ambientní a difůzní[8] složkou světla. Také pracuje s předpokladem, že všechny složky těla mají stejné světelné vlastnosti materiálu. Předvedený algoritmus je tedy možné dále vylepšit dodáním informací o vlastnostech zobrazovaného materiálu, například v podobě další jednorozměrné textury.

Zdroj. kód 3: Intenzita světla

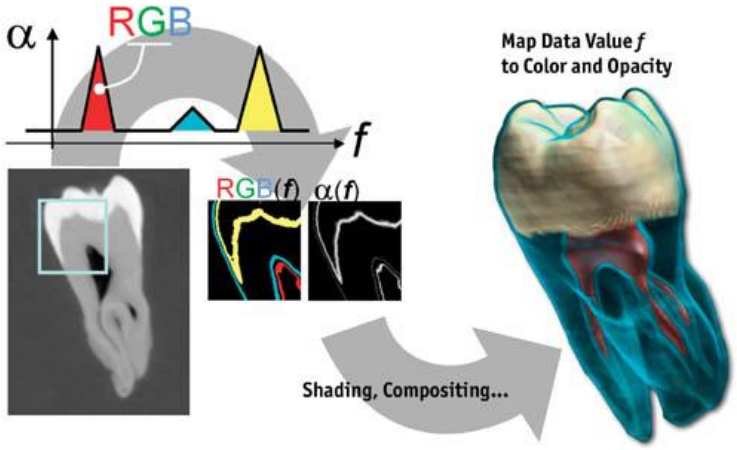
```
float spoctiIntenzituSvetla(vec3 normalaVoxelu, vec3 pozicePaprsku, vec3
poziceSvetla) {
  vec3 smerSvetla = normalize(poziceSvetla - pozicePaprsku);
  float difuzniPodil = max(dot(smerSvetla, normalaVoxelu), 0.0);
  return AmbientniSlozkaMaterialu + DifuzniSlozkaMaterialu*difuzniPodil;
}
```

Výsledná barva je součinem hodnoty přiřazené paletou a takto vypočtené intenzity.

[8] Vychází z Phongova osvětlovacího modelu: https://learnopengl.com/Lighting/Basic-Lighting

3.7 Metoda přímého vykreslování volumetrických dat (DVR)

Tato technika na rozdíl od SSD prochází celý objem. O výsledné barvě fragmentu rozhoduje každý voxel, kterým paprsek projde. Hodnoty voxelů jsou totiž transformovány pomocí tzv. přenosové funkce a kombinovány pomocí tzv. alpha míchání[9]. (viz. Obr. 23). *„Úlohou přenosové funkce je zdůraznění vlastností dat pomocí mapování hodnot a dalších datových měření na optické vlastnosti. Nejjednodušší a nejpoužívanější přenosové funkce jsou jednorozměrné a mapují rozsah hodnot dat na barvu a průhlednost“* [14]. [15]



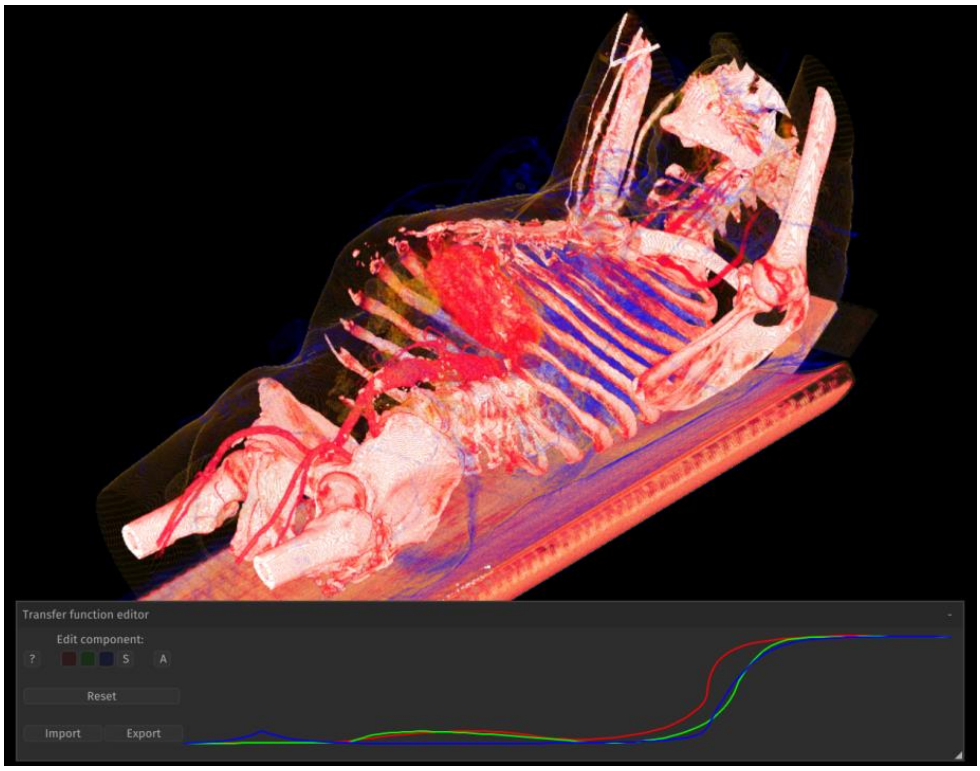
Obr. 23: Využití přenosové funkce

Zdroj: [14]

Jednoduchou verzí algoritmu pro DVR je vyslat paprsek ze zadní části objemu směrem k pozorovateli. Pro každý voxel je nalezena barva a průhlednost pomocí přenosové funkce. Barva je upravena pomocí výpočtu osvětlení, který se neliší od výpočtu představeného v předchozí podkapitole. Následně je voxel přidán k předchozím voxelům pomocí alpha míchání. Výstup této verze algoritmu je předveden na Obr. 24.

Podobně jako u SSD lze takto použitý algoritmus vylepšit dodáním informací o světelných vlastnostech materiálu nebo také akumulováním světla pohlceného před tím, než paprsek dorazil do voxelu. Algoritmus je relativně pomalý oproti ostatním představeným technikám, protože prochází celý objem. Částečným řešením je neprocházet prázdnými částmi objemu. [15]

[9] https://www.sciencedirect.com/topics/computer-science/alpha-blending



Obr. 24: Ukázka přímého vykreslování a přenosové funkce v programu CTSeer

Zdroj: Vlastní tvorba

Další možností, jak vylepšit kvalitu zobrazení, je vytvořit lepší přenosovou funkci. Funkce mohou brát v potaz další vlastnosti obrazu, než pouze denzitu. Je možné je tvořit metodou pokusu a omylu, generovat na základě analýzy dat objemu nebo například pomocí evolučních algoritmů. [15]

4 POUŽITÉ TECHNOLOGIE

V následující části jsou představeny knihovny použité v praktické části. Program je napsán v programovacím jazyce Java, který byl zvolen pro svůj poměr čitelnosti, funkcionality a rozšíření. Jako grafické API je využito OpenGL ve verzi 3.3. Tato verze podporuje všechny potřebné formáty textur pro práci se snímky CT a zároveň se způsob používání API neliší od nejnovějších verzí. Projekt se zdrojovými kódy byl vytvořen ve vývojovém prostředí IntelliJIdea. Stáhnutí použitých knihoven je automaticky vyřešeno nástrojem Gradle. [16]

4.1 DCM4CHE

Dcm4che je sada open source aplikací a knihoven pro zdravotnictví dostupná s trojitou licencí MPL/GLP/LGPL (uživatel si může zvolit licenci, která mu nejvíce vyhovuje). Základem projektu je implementace standardu DICOM. Součástí projektu je i aplikace s rozhraními pro uchovávání a správu snímků DCM4CHEE založená na platformě JBoss. Její architektura se skládá z modulů, z nichž každý poskytuje vlastní službu. Tento přístup usnadňuje správu kódu a jeho případné úpravy. [17] [32]

Ostatní poskytované aplikace jsou většinou programy s jediným účelem, jako je konvertování souboru typu DICOM do XML, převedení souboru formátu PDF do DICOM nebo hromadné odeslání zvolených snímků na server s úložištěm. [17]

V praktické části je využita pouze část DCM4CHE pro načítání snímků sérií, které jsou uloženy ve formátu DICOM. Zdroj. kód 4 demonstruje, jak pomocí knihovny načíst dataset s informacemi o snímku reprezentovaný třídou *Attributes* s daty obrazu v běžných případech, kdy nejsou data zakomprimována.

Zdroj. kód 4: Načtení datasetu a obrazu pomocí DCM4CHEE

```
try(DicomInputStream din = new DicomInputStream(sourceFile)) {
     din.setIncludeBulkData(DicomInputStream.IncludeBulkData.URI);

     Attributes attrs = din.readDataset(-1,-1);
     Object pixelData = attrs.getValue(Tag.PixelData);

     if(pixelData instanceof BulkData) {
          InputStream dataStream = ((BulkData) pixelData).openStream();
          …
     } else {
          throw new IOException("CTSeer does not support compressed DICOM
               files!");
     }
}
```

4.2 LightWeight Java Game Library

„LWJGL je knihovna pro multiplatformní přístup k populárním nativním API pro vývoj grafických (OpenGL, Vulkan), zvukových (OpenAL) a paralelně počítajících (OpenCL) aplikací“ [18]. Metody zpřístupněné v Javě jsou mapovány na funkce knihoven API. Díky tomu programátor zároveň získává výhody vyššího programovacího jazyka i funkce a možnosti nižších úrovní. Kromě zmíněných API nabízí LWJGL mapování na další knihovny vhodné pro tvorbu her, mezi které patří například OpenVR, CUDA nebo napojení na herní platformu Steam. Knihovna je volně dostupná s licencí BSD. [18]

4.2.1 GLFW

GLFW je jedna z knihoven zpřístupněných skrze LWJGL. Knihovna umožňuje vytváření oken na desktopových operačních systémech, zpracování uživatelského vstupu z klávesnice, myši a herních periferií. Obsah oken lze vykreslovat pomocí grafických API OpenGL a Vulkan. Knihovna je napsána v jazyce C a je šiřitelná s licencí zlib/libpng. [19]

V programu jsou volání statických funkcí GLFW sjednocena do tříd *GLFWWindow*, která reprezentuje okno zobrazené uživateli, a *GLFWInputCallback*, jenž zpracovává hlášení o změnách stavu kurzoru. Zdroj. kód 5 je ukázkou vytvoření jednoduchého okna s proměnlivou velikostí o šířce 800px a výšce 600px.

Zdroj. kód 5: Vytvoření okna v GLFW

```
glfwInit(); // inicializace GLFW
// Parametry pro vytvorene okno
glfwDefaultWindowHints();
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3); //OpenGL 3.3
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
// lze menit velikost okna
glfwWindowHint(GLFW_RESIZABLE, GL_TRUE);
// vytvoreni a zobrazeni
int windowId = glfwCreateWindow(800, 600, „Titulek“, NULL, NULL);
glfwShowWindow(windowId);
```

4.2.2 STB

STB je další z knihoven zpřístupněných v Javě pomocí LWJGL. Jedná se o sadu knihoven s rozličnými funkcionalitami pro vývoj her dostupnou s licencí Public domain. V praktické části byly využity části knihovny pro dekódování obrázků ve formátu PNG a načítání fontů. STB nabízí i knihovny pro detekci neuvolněné paměti, zápis obrazu ve formátu PNG na disk nebo generátor herních dlaždicových map. Následuje příklad (Zdroj. kód 6), jak je pomocí

knihovny LWJGL načten obraz ve formátu PNG do textury OpenGL. Smysl třídy *Texture2D* je popsán v poslední části této kapitoly. [24]

Zdroj. kód 6: Načtení textury pomocí knihovny STB

```
private static ATexture decodeAndLoadTexture(ByteBuffer imageBuffer,
     boolean is2D) throws IOException {
     try (MemoryStack stack = stackPush()) {
          IntBuffer w = stack.mallocInt(1);
          IntBuffer h = stack.mallocInt(1);
          IntBuffer comp = stack.mallocInt(1);
          ByteBuffer decodedImage = stbi_load_from_memory(imageBuffer, w, h,
          comp, 4);

          Texture2D texture = new Texture2D(w.get(0),
          h.get(0), TextureFormat.RGBA8_BYTE_UNSIGNED);
          texture.load(decodedImage, true);

          return texture;
     }
}
```

4.2.3 Nuklear

Jedná se o nástroj pro tvorbu GUI s tzv. okamžitým režimem. Knihovna je volně šiřitelná při splnění podmínek licence MIT. Zatímco tradiční knihovny pro tvorbu GUI reprezentují komponenty jako objekty a uchovávají si jejich stav, okamžitý režim je založen na proceduralním programování s ukládáním pouze nezbytných informací. K zpracování uživatelského vstupu dochází při plánování, co bude vykresleno, jak demonstruje Zdroj. kód 7. Vykreslovací příkazy pro grafickou kartu bývají obvykle vygenerovány jako dávka, kterou může programátor zavolat ve vhodném okamžiku. [20][21]

Zdroj. kód 7: Vykreslení panelu pomocí knihovny Nuklear

```
NkRect rect = NkRect.mallocStack(stack);
if (nk_begin(context, "Memory", nk_rect(x, y, WIDTH, HEIGHT, rect),
     PANE_FLAGS)) {

      if (nk_tree_state_push(context, NK_TREE_TAB,
          "Runtime",runtimeTabState)) {
            nk_layout_row_dynamic(context, HEADING_HEIGHT, 1);
            nk_label(context, "Runtime", NK_TEXT_CENTERED);

            nk_layout_row_dynamic(context, ROW_HEIGHT, 2);
            nk_label(context, "Available: ", NK_TEXT_ALIGN_LEFT);
            nk_label(context, toMegaBytes(runtime.maxMemory()),
            NK_TEXT_ALIGN_RIGHT);
            …
            nk_tree_pop(context);
      }
      …
}
nk_end(context);
```

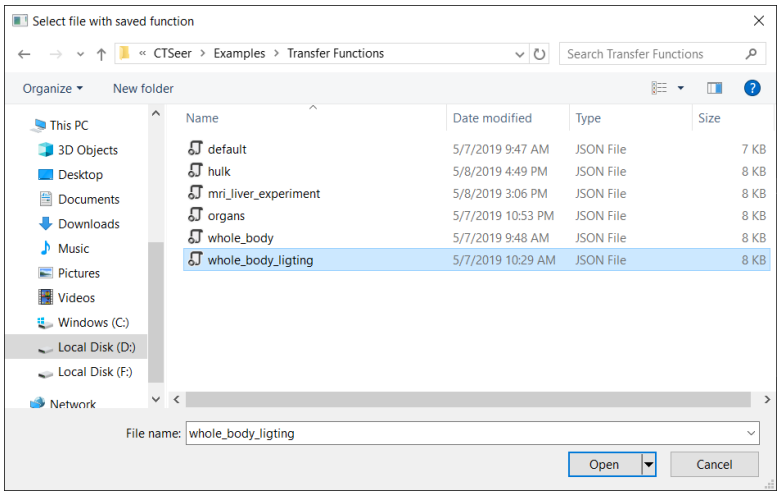
Výhodou tohoto přístupu jsou snazší tvorba vlastních ovládacích prvků a minimální nárůst kódu k údržbě. Oproti tomu je třeba brát v úvahu, že při každém vykreslování uživatelského prostředí musí být opět vykresleny i nepozměněné části. Další nevýhodou je ztráta typických funkcí tradičních GUI jako je například serializace do strukturovaného textového souboru nebo datové vazby. Obr. 25 je ukázkou grafického prvku vykresleného pomocí knihovny.

Obr. 25: Panel vykreslený pomocí knihovny Nuklear

Zdroj: Vlastní tvorba

4.2.4 Tiny File Dialogs

Knihovna je podobně jako Nuklear tzv. *Single header library*. Tyto knihovny bývají psané v jazyce C a skládají se pouze z jednoho hlavičkového souboru. Poskytují programátorovi specifickou množinu funkcí, aniž by mu příliš komplikovaly proces kompilace výsledné aplikace. Tiny FD se skládá ze 7 funkcí pro zobrazení různých typů dialogů. Tyto dialogy lze zobrazit na systémech Windows, MacOS a linuxových distribucích. Ve speciálních případech je možné použít i konzolovou verzi dialogů. Na Obr. 26 je předveden dialog pro výběr souboru, který je využit v programu CTSeer, jenž byl vytvořen v rámci této práce. Knihovna je dostupná s licencí zlib/libpng.



Obr. 26: Ukázka dialogu poskytovaném knihovnou na OS Windows 10

Zdroj: Vlastní tvorba

4.3 Java OpenGL Math Library

Knihovna poskytuje třídy pro matice a vektory i s metodami provádějící operace lineární algebry. *„Cílem JOML je poskytovat jednoduše použitelné, bohaté a účinné operace lineární algebry, které jsou potřeba v jakékoliv 3D aplikaci“* [28]. Knihovna je v práci použita pro reprezentaci vektorů a matic. Také je využita ve výpočtech rozměrů a dodatečných parametrů projekcí. Zdroj. kód 8 je ukázkou práce s knihovnou, kde je vytvořena matice pro zvětšení a následné posunutí vektoru. Knihovnu lze používat při splnění podmínek licence MIT.

Zdroj. kód 8: Práce s knihovnou JOML

```
Vector4f vector = new Vector4f(1, 0, 0, 0);
Matrix4f transformMatrix = new Matrix4f()
                            .translate(x, y, 0)
                            .scale(scaleX, scaleY, scaleZ);

transformMatrix.transform(vector);
```

Třídy jsou v knihovně navržené tak, aby většina operací, jako například vynásobení dvou matic, měnila stav objektu. Tato vlastnost je pro výpočty prováděné v představené aplikaci nevhodná, protože některé vektory bývají použity v dalších dodatečných výpočtech. Z tohoto důvodu byl vytvořen singleton *MathUtil* poskytující metody, které nejprve vytvoří kopii objektu, jenž by byl jinak změněn. Ve Zdroj. kód 9 je proveden výpočet pozice jednoho z rohů objemu po natočení dle vzorce Multiplanární rekonstrukce (MPR). V kódu je použita metoda *multiply* ze třídy *MathUtil*.

Zdroj. kód 9: Výpočet vzorce 7 s použitím metody multiply třídy MathUtil

```
Vector3d halfDiagonal = multiply(volume.rowOrientation,
     volumeSize.x  *corner.x)
     .add(multiply(volume.colOrientation, volumeSize.y*corner.y))
     .add(multiply(volume.depthOrientation, volumeSize.z*corner.z));
Vector3d vertexProjection = multiply(halfDiagonal, rotationMat);
```

4.4 Gson

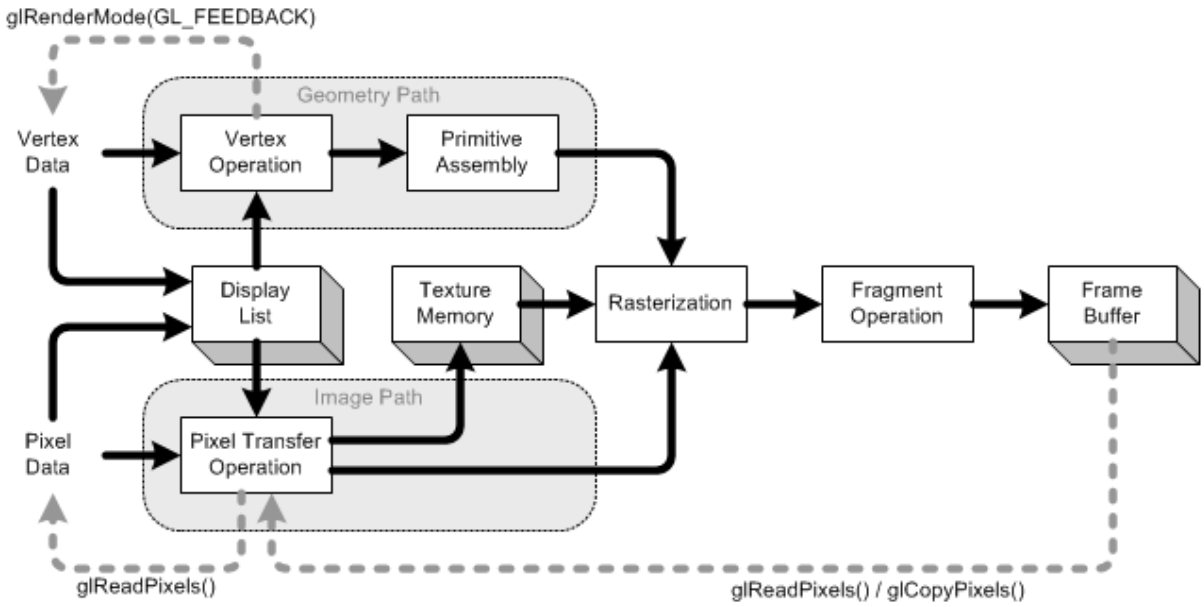
Gson je knihovna pro serializaci Java objektů do JavaScriptového objektového zápisu (JSON) a zpětnou deserializaci. Na rozdíl od ostatních knihoven dokáže Gson konvertovat objekty bez jakékoliv anotace a podporuje genericitu. Serializace je v programu využívána při ukládání přenosových funkcí vytvořených uživatelem. Následuje Zdroj. kód 10 s ukázkou, jak je v programu provedeno pomocí knihovny ukládání přenosové funkce. Knihovna používá licenci Apache 2.0. [22]

Zdroj. kód 10: Serializace pomocí knihovny Gson

```
public static void save(File file, TransferFunction function) throws
Exception {
     Gson gson = new GsonBuilder().setPrettyPrinting().create();
     FileWriter writer = (new FileWriter(file));
     gson.toJson(function, writer);
     writer.close();
}
```

4.5 OpenGL

OpenGL je standard specifikující grafické aplikační rozhraní. *„Grafické API (Application Programming Interface) je standardizovaná sada funkcí, tříd a struktur, kterou může programátor použít. Implementace částí API závisí na operačním systému nebo výrobci HW, pro který je rozhraní určené“* [30]. Níže (Obr. 27) je předvedena tzv. OpenGL pipeline, model ilustrující, jak funguje proces vykreslování.



Obr. 27: OpenGL "pipeline"

Zdroj: [34]

Pro vykreslení je třeba nejprve dodat OpenGL souřadnice vrcholů trojúhelníků, z nichž se skládá vykreslovaný model, a texturovací souřadnice určující, jak bude model potažen texturou. Tyto souřadnice jsou označovány jako tzv. vertexy. [31]

GPU dodané souřadnice převezme a umožní je dále ovlivnit pomocí tzv. vertex shaderu. Jedná se o program, který je spuštěn při zpracovávání každého vrcholu. Tvorba shaderu je úlohou programátora, který tak dostává další zodpovědnost, ale i možnosti. Například může dodané souřadnice posunout v prostoru nebo otočit. K tomu shader potřebuje obdržet transformační

matice a další informace. Předávání těchto informací je realizováno pomocí proměnných označovaných jako tzv. *uniform* proměnné. [31]

Transformované trojúhelníky jsou pomocí tzv. rasterizace[10] rozděleny na fragmenty o velikosti pixelu. Každému fragmentu je v tzv. fragment shaderu přiřazena barva. Tento proces opět ovlivňuje programátor, který může určit, z jaké části textury bude barva přejata nebo jak bude fragment osvětlen. [31]

Fragmenty ze všech modelů jsou na sebe naskládány v závislosti na jejich umístění. Fragmenty skryté za jinými nejsou vykresleny nebo dojde k prolnutí barev, pokud jsou fragmenty umístěné nad nimi částečně průhledné. Vykreslený obraz je uložen ve speciálním bufferu, který je nazýván *frame buffer*. Jeho obsah může být vykreslen přímo na obrazovku nebo použit jako textura při dalším vykreslování. [31]

V praktické části je OpenGL využito pro vykreslení projekcí a snímků série. Pro jednodušší práci byly funkce API sjednoceny do tříd umístěných v balíčku *gl*. Zdroj. kód 11 obsahuje ukázku použití třídy *Texture2D* z tohoto balíčku a následně funkce, které by jinak programátor musel zavolat.

Zdroj. kód 11: Porovnání vytvoření textury pomocí objektu oproti funkcím OpenGL

```
// pouziti tridy gl.Texture2D
Texture2D texture = new Texture2D(width, height,
              TextureFormat.RGBA8_BYTE_UNSIGNED);
texture.load(imageData, true); //true pro linearni filtrovani

// volani funkci OpenGL
int texId = glGenTextures();
glBindTexture(GL_TEXTURE_2D, texId);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA8,
GL_UNSIGNED_BYTE, imageData);
```

[10] https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

5 ANALÝZA A NÁVRH APLIKACE

V této kapitole jsou představeny základní stavební myšlenky aplikace ve formě diagramů návrhových tříd. Aplikace je navržena s ohledem na možné a snadné rozšíření o nové funkcionality, jako je vykreslování dalších projekcí nebo nové nástroje. V návrhu byl kladen důraz na oddělení částí programu sloužících k vykreslení projekce od částí pro její zobrazení a nastavení parametrů vykreslování.

5.1 Požadavky

Hlavní požadavky na aplikaci byly uvedeny již v zadání práce. Vyžadovány jsou hlavně funkce pro zobrazení různých typů projekcí. Program by měl nabízet možnosti úpravy jasového okna a zobrazení desky u projekcí, kde je to vhodné. Kompletní přehled požadavků je uveden v Příloha D.

Nefunkční požadavky kladou důraz především na spustitelnost aplikace ve více operačních systémech a rychlost programu, které je docíleno vykreslováním projekcí na GPU za pomoci API OpenGL. Pro ověření splnění těchto požadavků je k aplikaci dodána série obsahující 1067 řezů.

5.2 Analytické třídy

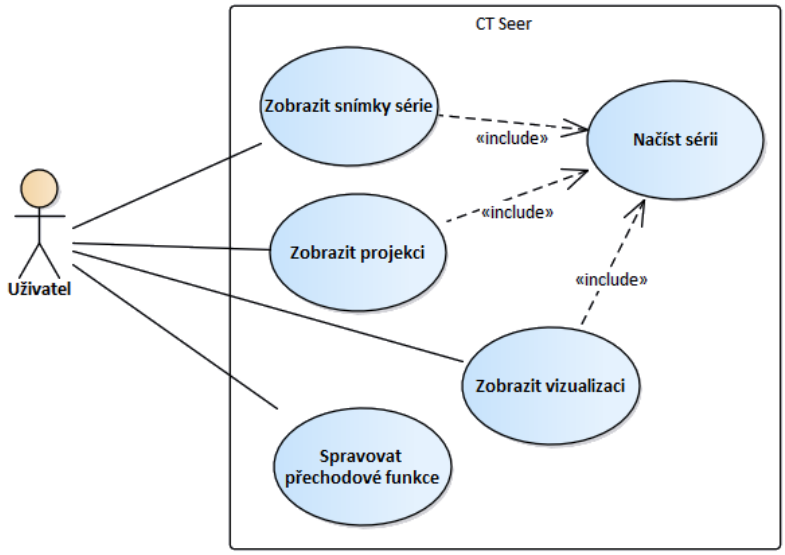
Analytické třídy jsou předvedeny v Příloha E. Návrhové třídy jsou z tohoto diagramu odvozeny, avšak některé ze zodpovědností původních tříd jsou dále delegovány. Největší změny se projevují ve třídě *GUI*, která je v pozdější fázi rozdělena na *GUIOverlay*, *ControlOverlay* a *RenderRequestFactory*. Z těchto důvodů jsou třídy popsány až v návrhové části.

5.3 Diagram případů užití

Na Obr. 28 je předveden diagram případů použití aplikace CTSeer. V podstatě jsou zde shrnuty požadavky ze zadání. Všichni uživatelé programu mají stejnou roli a cíl – zobrazit obrazová data v podobě snímku ze série, projekce či vizualizace.

MPR, MIP, MinP a AIP jsou považovány za projekce, zatímco SSD a VRT vizualizace. Zobrazení zahrnují případ *Načíst sérii*, jenž obsahuje scénář popisující, jak uživatel dodá data pro vykreslení.

Přenosové funkce mohou být spravovány i v případě, kdy nebyla dodána obrazová data, aby si uživatel mohl funkce nejprve prohlédnout a teprve poté vybrat objem k zobrazení.

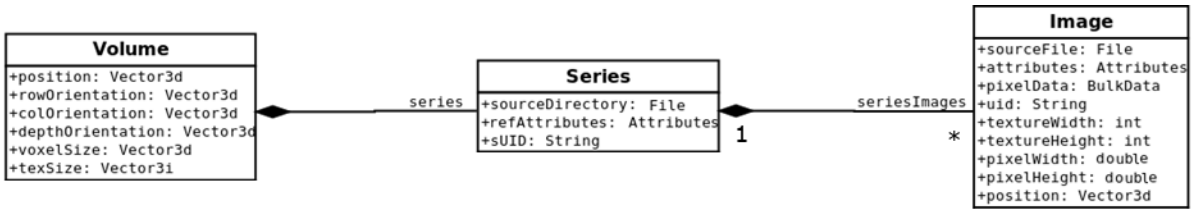


Obr. 28: Use case diagram programu CT Seer

Zdroj: Vlastní tvorba

5.4 Abstrakce DICOM entit

Na Obr. 29 jsou zobrazeny vztahy mezi třídami z hlediska použití aplikace. Uživatel nejprve načte sérii snímků v podobě adresáře s jednotlivými snímky. V průběhu zobrazování série se může rozhodnout sestavit ze série objem, jehož projekci si následně nechá vykreslit.



Obr. 29: Diagram tříd pro reprezentaci DICOM entit

Zdroj: Vlastní tvorba

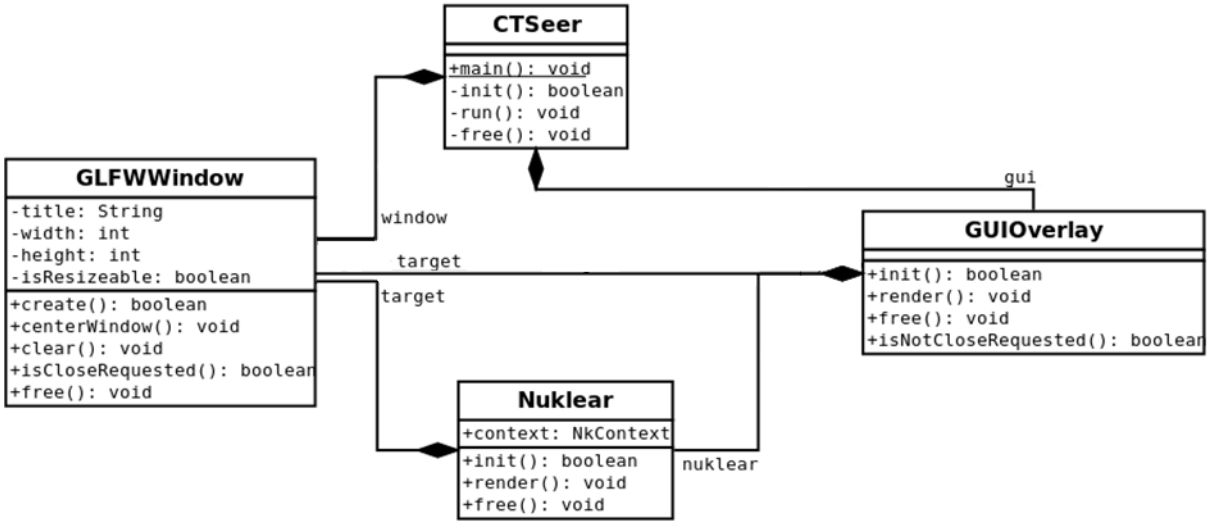
Ve třídě *Image*, jenž reprezentuje snímek, se nachází atributy typu *Attributes* a *BulkData*. Jedná se o třídy z knihovny Dcm4chee. První z nich reprezentuje datovou sadu s informacemi o vlastnostech obrázku. Bližší informace o složení datové sady byly představeny v první kapitole. Třída *BulkData* obsahuje nezkomprimovaná data obrazu. Ostatní atributy jsou načteny z datové sady a použity při výpočtech rozměrů vykreslovaného obrazu. Atribut `uid` slouží pro jednoznačnou identifikaci obrazu.

Třída *Series* reprezentuje množinu snímků, které společně tvoří sérii po sobě následujících snímků. Hlavním atributem je zde adresář, ve kterém jsou snímky uloženy. Podobně jako snímky i série obsahuje atribut pro jednoznačnou identifikaci.

Třída *Volume* je vytvořena ze třídy *Series*. Z datové sady používá další informace nezbytné pro výpočet texturovacích souřadnic projekce. Jedná se o rozměry voxelu, orientaci snímku, velikost matice s hodnotami voxelů a pozici objemu v prostoru.

5.5 Hlavní třídy

Na následujícím diagramu Obr. 30 jsou předvedeny hlavní třídy logických celků aplikace. Většina z nich obsahuje operace `init()` a `free()` pro alokaci a dealokaci prostředků napojených nativních knihoven. Operace `main()` třídy *CTSeer* je zamýšlena jako vstupní metoda programu.



Obr. 30: Hlavní třídy programu

Zdroj: Vlastní tvorba

Třída *GLFWWindow* je navržena jako využití návrhového vzoru fasáda[11] pro jednodušší práci s knihovnou GLFW a jí poskytnutým oknem. Obsahuje operace pro pozicování okna na obrazovce, vyčištění obsahu před jeho novým vykreslením a zjištění, zda uživatel nezažádal o ukončení programu. Zpracování uživatelského vstupu je záležitostí implementace, kde

[11] https://sourcemaking.com/design_patterns

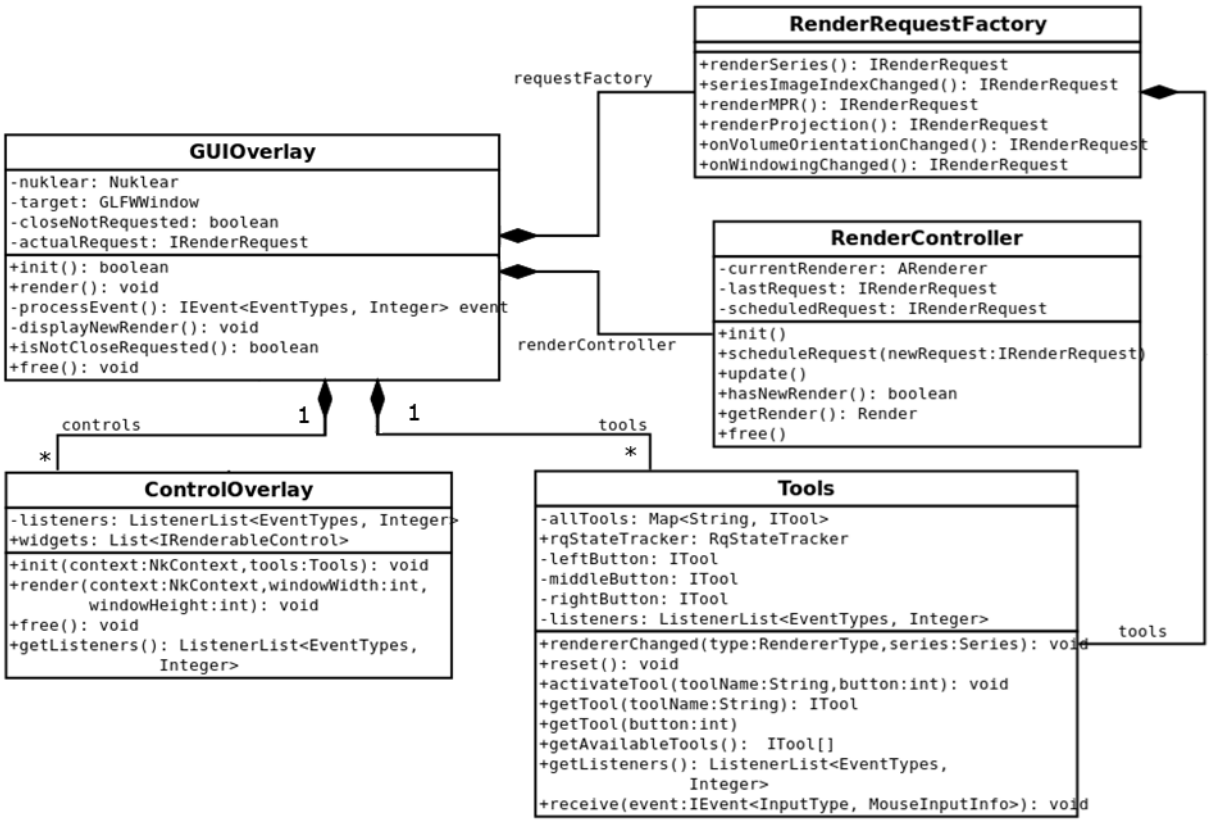
jsou informace o aktivitě myši a klávesnice, jenž knihovna GLFW také poskytuje, předány knihovně Nuklear.

Nuklear je podobně jako *GLFWWindow* fasádou pro práci s knihovnou pro tvorbu grafického uživatelského rozhraní. Statická volání knihovny jsou používána při tvorbě GUI ve třídách, jenž mají přístup k instanci třídy *NkContext*. Operace `render()` z těchto volání vygeneruje příkazy pro GPU, které jsou následně vykresleny.

GUIOverlay pomocí zmíněných fasád řídí proces vykreslování. Tato třída je podrobněji popsána v následující kapitole.

5.6 Třídy grafického rozhraní

Na Obr. 31 je představen diagram s dalšími třídami navázanými na třídu *GUIOverlay*. Podobně jako v předchozí kapitole jsou u tříd, které pracují s nativními knihovnami nebo grafickou kartou, navrženy operace `init()` a `free()`. Způsob komunikace mezi prvky diagramu je realizován pomocí třídy *ListenerList* a je blíže popsán v následující kapitole.



Obr. 31: Třídy grafického rozhraní

Zdroj: Vlastní tvorba

ControlOverlay je určena jako kontejner pro grafické nástroje a nabídky, se kterými uživatel pracuje. Nástroje implementují rozhraní *IRenderableControl*, které pouze vyžaduje, aby tvůrce nástroje určil, kde se nástroj bude zobrazovat a implementoval operaci pro vykreslení.

Zatímco *ControlOverlay* je určeno pro zobrazování nástrojů, třída *Tools* je navržena pro uchování jejich stavu. Dále realizuje předání uživatelského vstupu při práci s nástrojem do třídy reprezentující příslušný nástroj. Dojde-li ke změně zobrazované projekce, snímku nebo celé série, stav nástrojů může být změněn. Toho je docíleno pomocí operací `rendererChanged()` a `reset()`.

RenderRequestFactory je inspirována návrhovým vzorem továrna[11]. Nabízí operace pro vytváření instancí tříd implementujících rozhraní *IRenderRequest*, což jsou požadavky s parametry na vykreslení projekce či snímku pro třídu *RenderController*. *GUIOverlay* tyto požadavky generuje při reakci na uživatelovu práci s nástroji. Parametry jsou čerpány ze stavu třídy *Tools*.

RenderController pomocí operace `scheduleRequest()` přijímá požadavky pro vykreslení obrazu. Třída je navržena tak, aby zde docházelo k rozhodování, jak a kdy bude projekce vykreslena, nezávisle na zbytku aplikace. Pro pravidelnou aktualizaci vnitřního stavu je určena operace `update()`. Podrobnější popis chování operace obsahuje Příloha A. *GUIOverlay* se při vykreslování pomocí operace `hasNewRender()` ptá, zda *RenderController* nepřipravil nový obraz a případně ho pomocí `getRender()` použije.

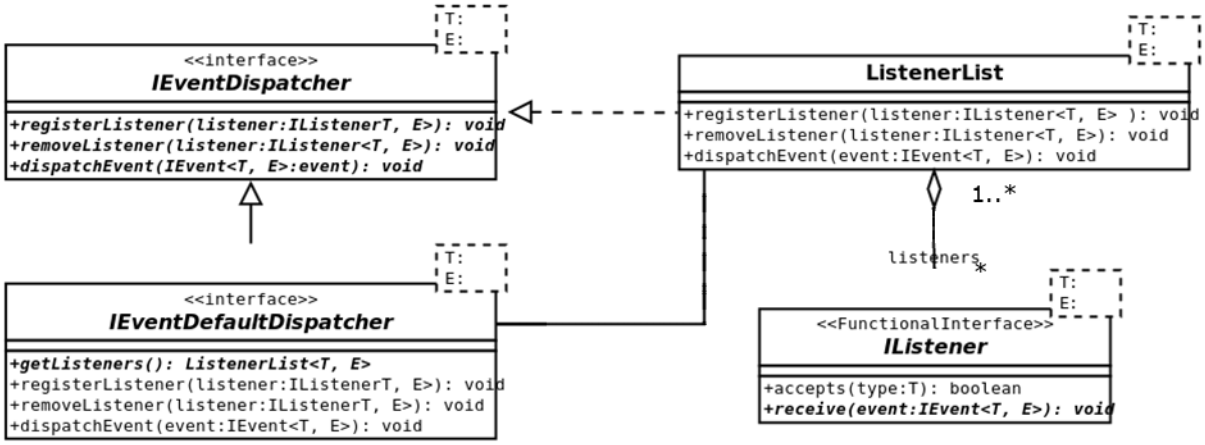
5.7 Komunikace mezi třídami

Komunikace mezi třídami, kde dochází k mnoha různým typům změn stavu, na němž jsou závislé ostatní třídy, je realizována pomocí návrhového vzoru posluchač[11]. Pro zjednodušení práce při implementaci tohoto vzoru byly navrženy dodatečné třídy a rozhraní, které jsou popsány na Obr. 32.

Třída implementující rozhraní *IEvent* pouze musí implementovat operace pro zjištění, jaké typy událostí podporuje, a zpřístupnění hodnoty události. Pro rozšíření možných případů použití jsou rozhraní generická, aby bylo možné určit, jaké třídy budou reprezentovat typ události a hodnotu.

Třída s rozhraním *IListener* představuje posluchače, který očekává událost od poslouchaného objektu. Před odesláním události objektu se odesílatel může přesvědčit, že posluchač tento typ

události očekává pomocí operace `accepts()`. Tato operace je defaultně implementována, aby bylo možné rozhraní použít jako funkcionální.



Obr. 32: Rozšíření návrhového vzoru posluchač

Zdroj: Vlastní tvorba

Rozhraní *IEventDispatcher* představuje odesílatele události. Odesílatel musí být schopen přijmout či odebrat zájemce o události a rozeslat událost.

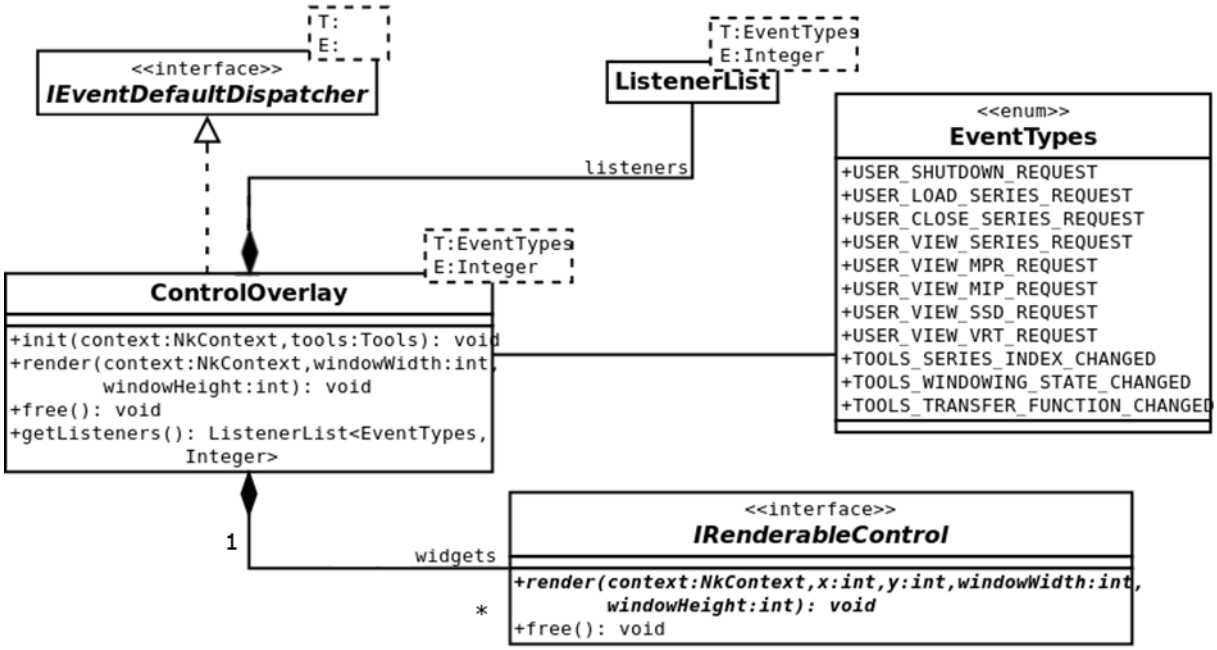
Třída *ListenerList* pouze zapouzdřuje kolekci posluchačů, kterým umožňuje odeslat hromadnou zprávu. Zprávu obdrží pouze posluchači, kterým operace `accepts()` pro daný typ zprávy vrátí `true`.

Rozhraní *IEventDefaultDispatcher* využívá třídy *ListenerList* a s pomocí například defaultních implementací představených v Javě 8 za uživatele implementuje operace rozhraní *IEventDispatcher*. V implementující třídě tedy pro zprovoznění stačí vytvořit instanci *ListenerList* a zpřístupnit ji rozhraní skrze operaci `getListenerList()`.

5.8 Události ze třídy ControlOverlay

Ve třídě *ControlOverlay* jsou využity rozhraní představená v předchozí kapitole. Obr. 33 demonstruje jak. Třída může při interakci uživatele s některým z nástrojů, které obsahuje, odeslat událost.

Možné hodnoty typů události jsou uloženy ve výčtu *EventTypes*. Zde jsou využity především typy událostí pro zvolení položky v nabídce dostupných projekcí, jako je například `USER_VIEW_MPR_REQUEST`.



Obr. 33: Události ze třídy *ControlOverlay*

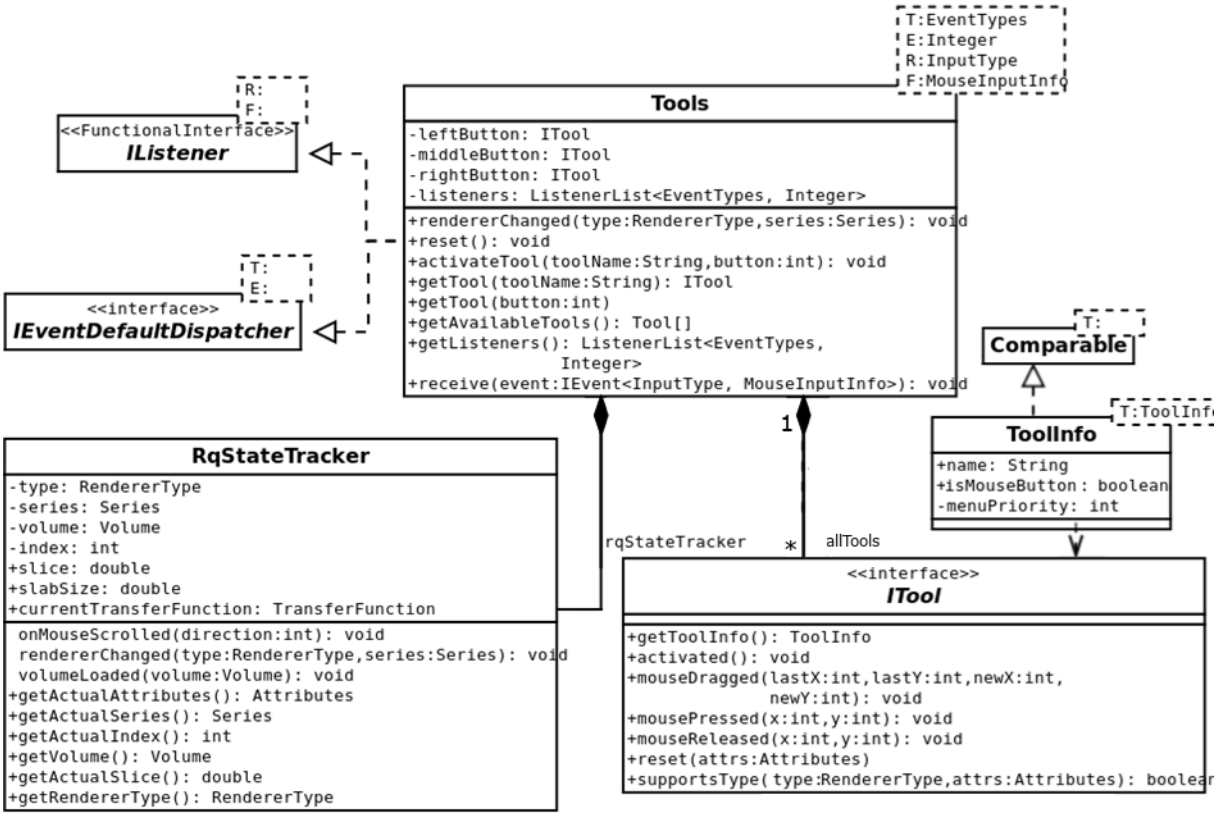
Zdroj: Vlastní tvorba

5.9 Stav nástrojů

Stav nástrojů není uchován ve třídách, které se starají o jejich vykreslování, protože je často využíván z ostatních částí aplikace. Také může dojít ke změně hodnoty nástroje nepřímým vlivem uživatele. Například může dojít k nastavení výchozích hodnot jasového okna poté, co uživatel zažádá o zobrazení nového snímku. Třída *Tools* je schopná posílat i přijímat události. Posílané jsou události o změně stavu nástrojů, přijímané jsou události představující chování myši. Podrobný přehled navržených tříd je zobrazen na Obr. 34.

Nástroje musí implementovat rozhraní *ITool*, které požaduje, aby každý nástroj poskytoval strukturu s popisem vlastností (*ToolInfo*). Při táhnutí a interakci s tlačítkem myši, na kterém je nástroj navázaný, jsou skrze *Tools* volány příslušné metody. Při změně zobrazované projekce je z *Tools* volána operace `supportsType`(), aby bylo ověřeno, že nástroj je na projekci aplikovatelný.

Informace o nástroji jsou uchovány ve třídě *ToolInfo*, která implementuje rozhraní *Comparable*. Položky v panelu nástrojů tak mohou být prioritně řazeny.



Obr. 34: Stav nástrojů

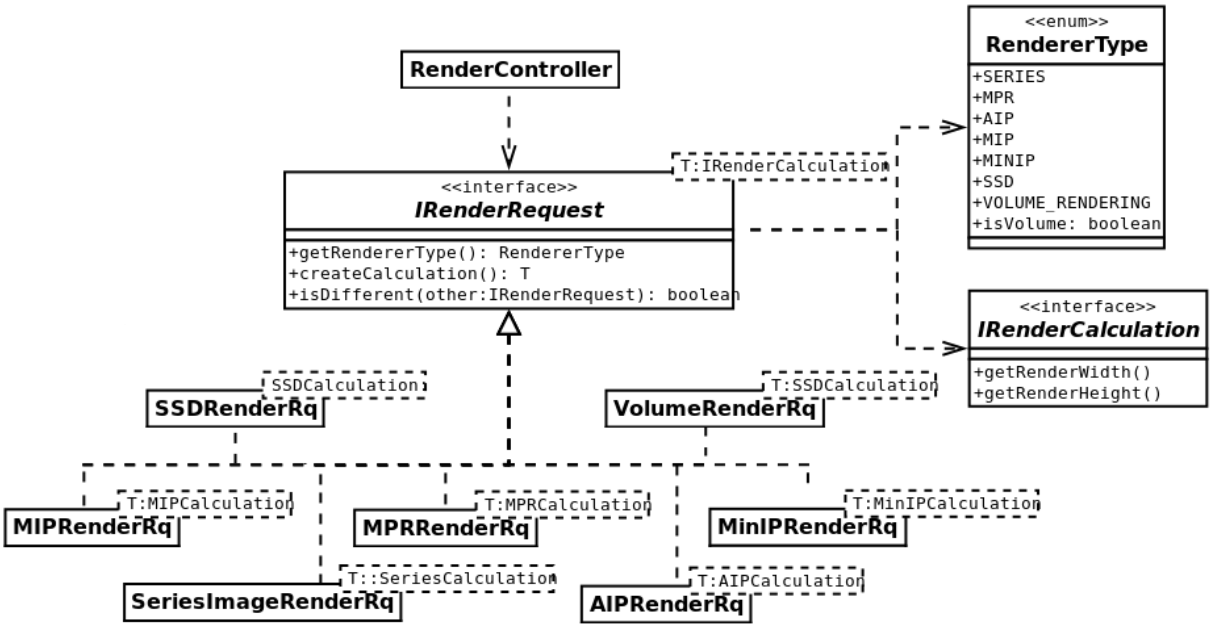
Zdroj: Vlastní tvorba

Pro schraňování aktuálně použitých parametrů pro zobrazovanou projekci či snímek slouží třída *RqStateTracker*. Z *Tools* je skrze balíčkově chráněné operace aktualizován stav po každé, kdy je obdržena událost se zprávou o změně. Z ostatních částí programu je tedy možné zjistit, jaké jsou aktuální parametry, ale nelze je přímo měnit, aby nedošlo ke změně stavu bez informování všech závislých objektů.

5.10 Parametry pro vykreslení nové projekce

Parametry vytvořené pomocí *RenderRequestFactory* jsou zapouzdřeny do vlastních tříd, které implementují rozhraní *IRenderRequest*. Obr. 35 obsahuje přehled vztahů mezi parametry a zbytkem aplikace.

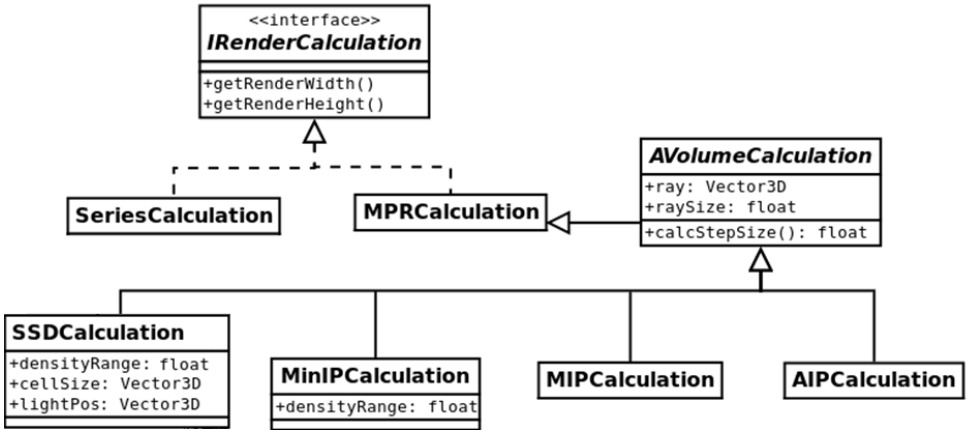
Rozhraní požaduje, aby bylo z parametrů možné zjistit, který typ projekce by měl být vykreslen. Typy projekcí jsou v celé aplikaci reprezentovány pomocí výčtu *RendererType*. Třídy jsou mezi sebou porovnatelné, aby bylo možné předejít vícenásobnému vykreslování stejných parametrů.



Obr. 35: Třídy požadavků

Zdroj: Vlastní tvorba

Posledním důležitým požadavkem rozhraní je možnost z parametrů vytvořit třídy s výpočty, v nichž jsou podle postupů z předchozích kapitol zjištěny rozměr vykreslovaného obrazu a dodatečné parametry pro shader. Diagram tříd těchto výpočtů je zobrazen na Obr. 36.



Obr. 36: Třídy výpočtů pro projekce

Zdroj: Vlastní tvorba

Protože všechny představené výpočty projekcí přímo vychází z MPR, je na jeho výpočtech založen zbytek výpočtů pro zobrazení objemu. Dodatečné atributy jsou předány z parametrů požadavku pro vykreslení při vytváření. Výpočty mohou být předány přímo objektu, který má na starosti vykreslování.

6 IMPLEMENTACE

Zde jsou popsány vybrané koncepty a třídy, které vznikly při implementaci aplikace. Každou část lze považovat za doprovodné komentáře ke zdrojovému kódu programu. Většina kapitoly se věnuje samostatným částem vykreslovacího procesu. Závěrečná podkapitola vysvětluje princip editoru přenosové funkce.

6.1 Třída *RenderView*

Jak již bylo zmíněno v předchozí kapitole, *RenderController*, třída zodpovědná za vykreslování projekcí, vykreslí obraz pouze pro takové požadavky, které se liší od svých předchůdců. Aby tato podmínka měla smysl, je třeba zabránit přísunu požadavků od nástrojů, které neovlivňují parametry projekce, ale pouze její výsledek. Jedná se o nástroje na posun, přiblížení výsledného obrazu a invertování barev, částečně i nástroj jasového okna.

V první fázi je projekce vykreslena do *frame bufferu* s připojenou *texturou*. V druhé fázi je *textura* skrze *RenderController* předána zbytku aplikace na vykreslení uživateli. Teprve při vykreslování pro uživatele jsou provedeny operace zmíněných nástrojů. Pokud tedy uživatel při projekci SSD natočí objem a následně si jej přiblíží, první fáze bude provedena jen po rotaci.

6.1.1 Nástroj jasového okna

Funkce jasového okna neovlivňuje vykreslování projekce v případech, kdy není třeba stanovit hraniční hodnotu voxelu. Všechny projekce, které vrací obraz ve stupních šedi, totiž vykreslují do *textury*, jejíž formát `R16` dokáže pojmout celý barevný rozsah obrazu. V případě projekcí MPR, MIP a AIP tedy není nutné transformovat hodnoty voxelů v první fázi. Je-li pro vykreslení použita textura s pouze jedinou barevnou složkou, bude transformace provedena v druhé fázi.

Pro ostatní projekce a vizualizace je třeba provézt i překreslení textury z první fáze. MinIP vyžaduje nejnižší povolenou hodnotu, aby jeho výstupem nebyly pouze hodnoty voxelů s denzitou vzduchu. Při vykreslování SSD a VRT je použita textura s formátem `RGB`. Údaje o denzitě se zde transformují na barvu a provedení operace jasového okna by tak bylo velmi obtížné.

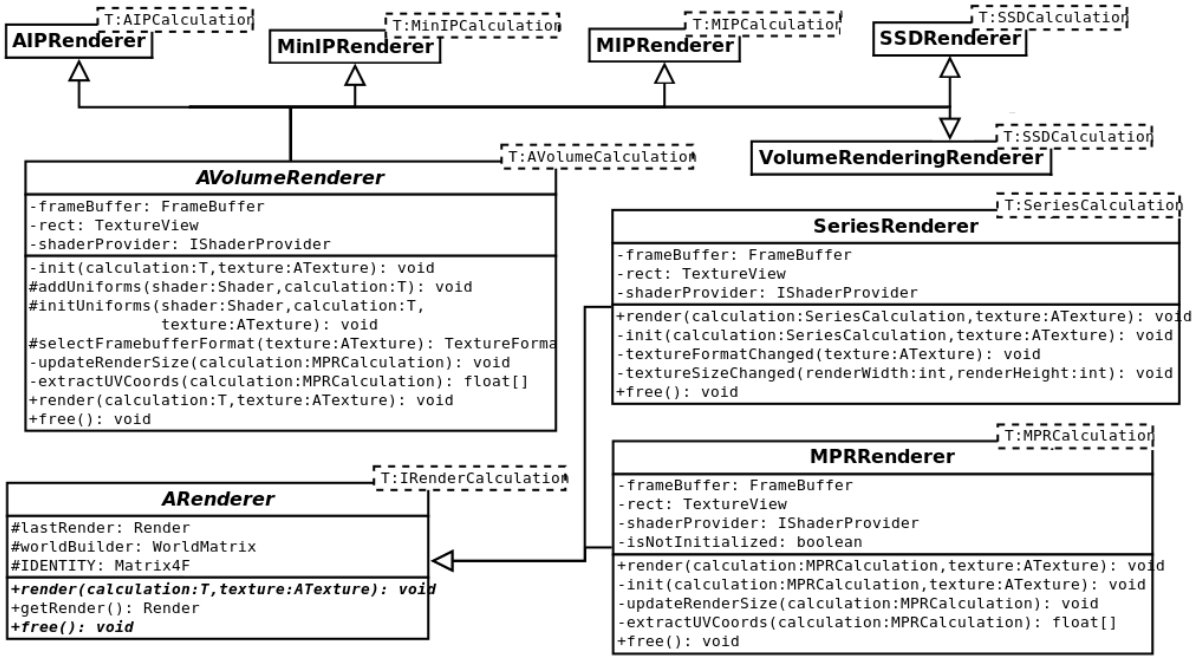
6.1.2 Třída *TextureView*

Jednou z částí, ze kterých se třída *RenderView* skládá, je *TextureView*. Tato třída pochází z balíčku *gl*. Obsahuje další třídy balíčku, které abstrahují komponenty a principy OpenGL. Programátorovi pouze stačí dodat texturu obrázku, shadery, transformační matice a pravidelně volat metodu pro vykreslení. O alokaci, připojení bufferů s vertexy a jejich změnu při obměně textury se postará *TextureView*.

6.2 Třída *ARenderer* a její potomci

Mezi třídami odpovědnými za vykreslování platí podobné vztahy jako mezi třídami výpočtů projekcí. *RenderController* při obdržení požadavku na vykreslení projekce navíc očekává, že budoucí požadavky budou na stejný typ projekce. Proto je instance třídy pro vykreslení dané projekce vytvořena pouze při obdržení prvního požadavku svého typu. Ve chvíli, kdy *RenderController* obdrží požadavek jiného typu, jsou prostředky třídy uvolněny a dojde k vytvoření nové instance.

V diagramu na Obr. 37 jsou zobrazeny všechny třídy, které mají vykreslování projekcí a vizualizací na starosti.



Obr. 37: Třídy pro vykreslování projekcí a vizualizací

Zdroj: Vlastní tvorba

RenderController očekává, že každá vykreslovací třída implementuje funkce pro vykreslení daného výpočtu projekce a uvolnění prostředků. Proces je podrobněji určen ve třídě *AVolumeRenderer*, která je předkem většiny ostatních tříd. Proto následuje bližší popis metod této třídy. Přesné pořadí a podmínky pro vyvolání metod jsou předvedeny na diagramu v Příloha B.

6.2.1 Třída *AVolumeRenderer*

Vykreslování začne voláním metody `render()` poté, co *RenderController* načte texturu objemu a vypočte parametry projekce z požadavku. Odsud je volána metoda `init()`, kde dojde k alokaci potřebných objektů OpenGL, pokud již nebyly vytvořeny nebo se jejich nastavení liší od potřebného. Jedná se zejména o *frame buffer*, u něhož se může lišit typ textury nebo její velikost. Typ textury si mohou potomci třídy samostatně vybrat přetížením metody `selectFrameBufferFormat()`.

Podobným způsobem si potomci také určí, jaké se mají použít *shadery*, pokud již nebyly načteny. Po načtení shaderů je zavolána metoda `addUniforms()`, kde dojde k registraci *uniformů*. Nezávisle na tom, zda se jedná o první inicializaci, je následně volána metoda `initUniforms()`. Zde jsou *uniformům* přiřazeny hodnoty získané z výpočtů.

Potomkům třídy *AVolumeRenderer* stačí pouze přetížit metody `addUniforms()` a `initUniforms()`, které definují *uniformy* pro předání hodnot do shaderu, a případně `selectFrameBufferFormat()` pro vlastní výběr typu navrácené textury s vykresleným obrazem.

6.3 Nástroj pro natočení objemu

V podkapitole o MPR byly představeny výpočty pro zobrazení projekce ze směru, který je dán vektory *rowOrientation* a *columnOrientation*. Aby uživatel nemusel tyto vektory manuálně zadávat, byl navrhnut nástroj *VolumeRotateTool*. Jedná se o běžnou implementaci rozhraní *ITool*, jenž poskytuje metody volané při tahání kurzoru po obrazovce. Táhne- li uživatel vertikálně, dojde k otočení vektoru *rowOrientation* kolem *columnOrientation*. V případě, že je myš tažena horizontálně, jsou vektory prohozeny.

Natočení vektoru *v* okolo jednotkového vektoru *k* o úhel θ je vypočítáno pomocí tzv. Rodriguesovy[12] rotační formule:

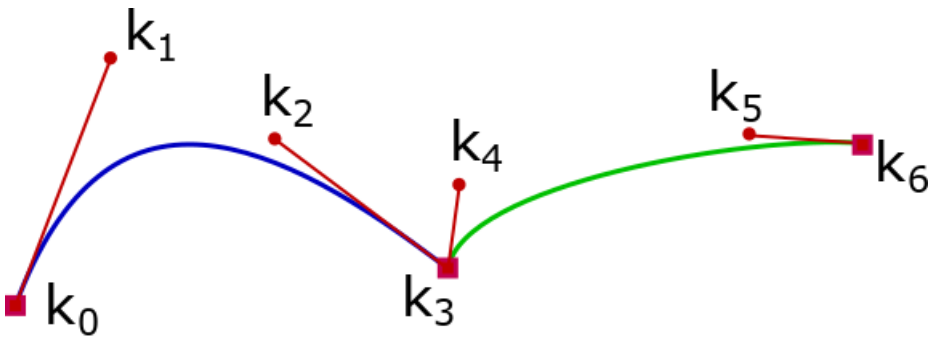
$$\vec{v}_{rot} = \vec{v} \cdot \cos\theta + (\vec{k} \times \vec{v}) \cdot sin\theta + k(\vec{k} \cdot \vec{v})(1 - cos\theta). \quad (13)$$

6.4 Přenosová funkce

Místo hledání univerzální přenosové funkce na základě obrazových dat byl zvolen přístup, který uživateli poskytuje větší svobodu – vytvoření vlastní. Barvové složky funkce jsou reprezentovány křivkami, jejichž body může uživatel posunovat. Osa X takové funkce udává denzitu, Osa Y neprůhlednost barevné složky voxelu s danou hodnotou denzity. Hodnoty na osách se pohybují v intervalu ⟨0; 1⟩. Typ použité křivky musí umožňovat snadnou editaci a obsahovat dostatečný počet řídících bodů, aby bylo možné snadno modelovat přenosovou funkci.

6.4.1 Kompozitní Bézierova křivka

Každa barevná složka přenosové funkce (viz kapitola 3.7) je v programu reprezentována vlastní kompozitní kubickou Bézierovou křivkou. Kompozitní Bézierova křivka je seřazená skupina křivek, pro které platí, že koncový bod dané křivky je zároveň startovním bodem následující. Toto pravidlo je předvedeno na Obr. 38.



Obr. 38: Složení kompozitní kubické Bézierovy křivky

Zdroj: Vlastní tvorba

Modře zvýrazněná křivka je dána body k_0 až k_3. Zelená křivka v bodě k_3 navazuje na předchozí. Díky tomu je dosáhnuto základní spojitosti C^0. Posune-li uživatel jedním z bodů k_0 až k_2,

[12] https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

ovlivní pouze modrou část. Díky této vlastnosti si může uživatel křivku rozdělit na intervaly denzity, z nichž každý bude mít rozdílnou barvu.

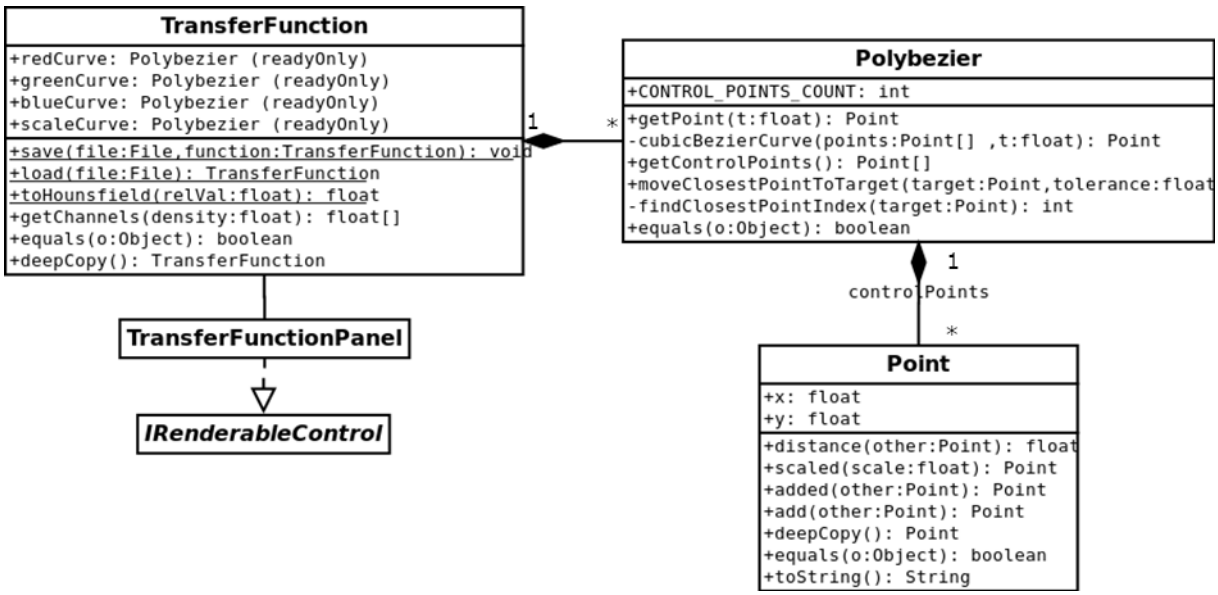
Pozice bodu v rámci kubické křivky je dána:

$$p = p_0 \cdot (1-t)^3 + p_1 \cdot 3 \cdot (1-t)^2 \cdot t + p_2 \cdot 3 \cdot (1-t) \cdot t^2 + p_3 \cdot t^3, \qquad (23)$$

kde *t* je parametr v Bézierově intervalu ⟨0;1⟩ a reprezentuje relativní umístění bodu v rámci křivky. [33]

6.4.2 Třída *Polybezier*

Přenosová funkce je v kódu reprezentována třídou *TransferFunction*. Ta se skládá z instancí tříd *Polybezier*, jenž představují kompozitní Bézierovy křivky. Vztahy mezi třídami jsou blíže popsány na Obr. 39.



Obr. 39: Vztahy třídy Polybezier

Zdroj: Vlastní tvorba

TransferFunction obsahuje metody `equals()` a `deepCopy()` pro předávání v rámci požadavků na vykreslení. Přenosové funkce musí být porovnatelné, aby bylo možné zjistit, zda se požadavek liší od předchozího. Metoda `getChannels()` vrací pole s hodnotami křivek na ose Y pro relativní hodnotu density, která je dále považována za parametr *t*.

Aby křivky pokryly celou osu X, nelze první a poslední řídící body křivek posunovat horizontálně. Díky tomu lze hodnoty z osy X také považovat za parametr *t*.

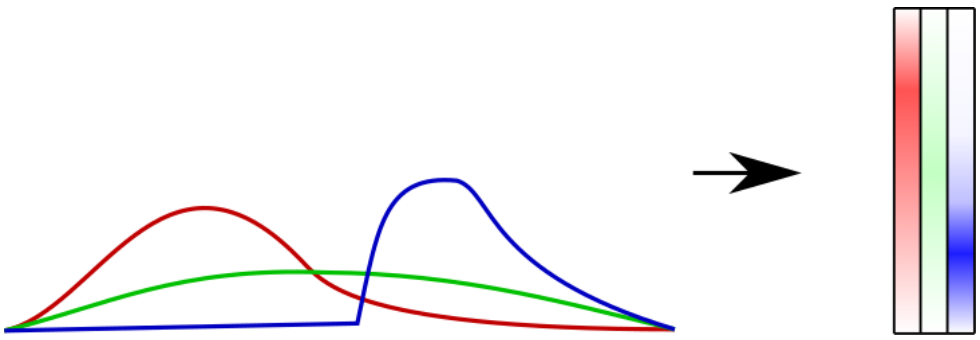
Instance *Polybezier* jsou přímo přístupné pro potřeby vykreslování. Metoda `moveClosestPointToTarget()` zajišťuje změnu pozice bodu, pokud za něj uživatel táhne kurzorem. Pro zjištění hodnoty na ose Y slouží `getPoint()`. Zde je třeba nejprve určit segment, pro něhož je daný parametr *t* skutečně určen, protože *Polybezier* je z pohledu *TransferFunction* jednotná křivka. Parametr je následně přepočítán tak, aby se pohyboval v intervalu <0;1> pro daný segment. Nalezené řídící body jsou společně s parametrem předány `cubicBezierCurve()`, kde je spočítána pozice bodu na základě vzorce 14. Kód funkce `getPoint()` je předveden ve Zdroj. kód 12.

Zdroj. kód 12: Metoda getPoint()

```
public Point getPoint(float t) {
     if(t <= 0) return controlPoints[0];
     for(int i =3; i < controlPoints.length; i += 3) {
          if(t <= controlPoints[i].x) { // found ending knot
               float localT = (t - controlPoints[i-3].x) /
                    (controlPoints[i].x  - controlPoints[i-3].x);
               return cubicBezierCurve(Arrays.copyOfRange(controlPoints,
                    i-3, i+1), localT);
          }
     }
     return controlPoints[controlPoints.length -1];
}
```

6.4.3 Použití funkce v shaderu

Vytvořenou funkci je potřeba předat do shaderu, aby paprsek věděl, jak přiřadit barvy voxelům. Proto tento účel je z funkce vytvořena textura. Pro každý řádek textury je vypočítán parametr *t*, který je dosazen do metody `getChannels()`. Textura má tolik sloupců, kolik přenosová funkce kompozitních křivek. Funkce je tedy vzorkována. Pro lepší pochopení je proces předveden na Obr. 40.



Obr. 40: Tvorba textury přenosové funkce

Zdroj: Vlastní tvorba

Na Zdroj. kód 13 je ukázán způsob, kterým je textura použita v shaderu. Funkce očekává jako vstupní argument denzitu v intervalu ⟨0; 1⟩, která je použita jako souřadnice pro zjištění přiřazené barvy.

Zdroj. kód 13: Použití textury transferFunction v shaderu

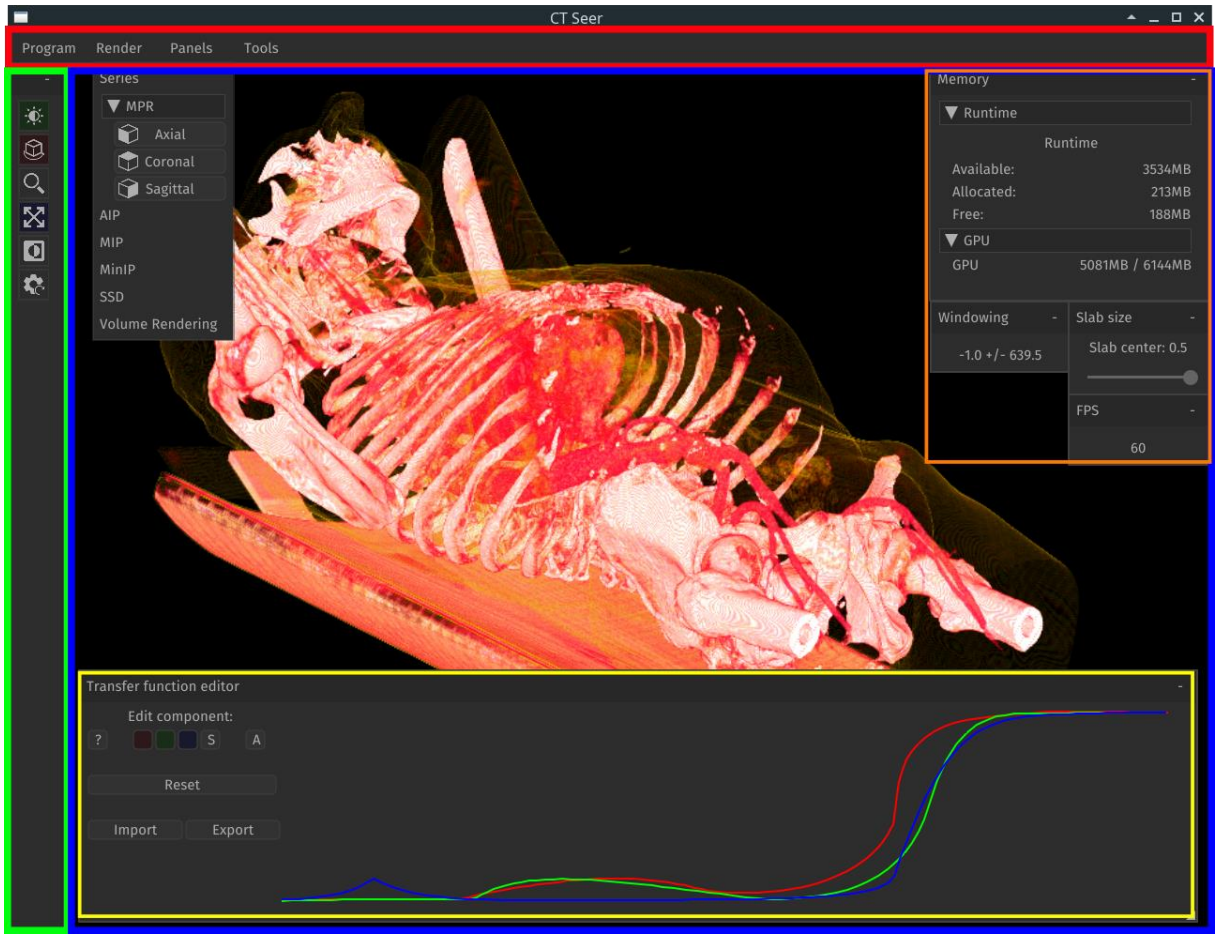
```
vec4 convertToRGBA(float relativeDensity) {
    vec4 color = vec4(0, 0, 0, 0);
    color.r = texture(transferFunction, vec2(0.15, relativeDensity)).r;
    color.g = texture(transferFunction, vec2(0.50, relativeDensity)).r;
    color.b = texture(transferFunction, vec2(0.85, relativeDensity)).r;
    color.a = max(color.r, max(color.g, color.b));

    return color;
}
```

7 OVLÁDÁNÍ APLIKACE

Aplikace je dodávána jako spustitelný archiv JAR. Aplikace byla otestována na operačních systémech Windows 10 a linuxové distribuci Manjaro. Pro spuštění aplikace je vyžadováno, aby byl na systému nainstalován Java Runtime Environment 8 a vyšší. Další podmínkou je podpora API OpenGL ve verzi 3.3.

Grafické prostředí aplikace se skládá především z panelů. Díky tomu je uživatel schopen si složit prostředí tak, aby vyhovovalo jeho stylu práce. Hlavní částí, ze kterých se aplikace skládá, jsou předvedeny na Obr. 41.



Obr. 41: Části aplikace CT Seer

Zdroj: Vlastní tvorba

7.1 Vykreslovací plátno a panely

Modře zvýrazněnou částí je vykreslovací plátno. Na plátně je vždy zobrazen aktuálně vykreslený obraz projekce. Na něm jsou zobrazeny panely, které uživatel aktivoval v hlavním menu. Panely je možné stiskem myši na záhlaví a následným tažením přesunovat. Většinu jich lze

také pomocí ikonky v pravém horním rohu srolovat. Opětovným zvolením jejich položky v hlavním menu je lze skrýt.

7.2 Hlavní menu

Červeně zvýrazněná část je hlavní roletové menu. Je rozdělená na čtyři kategorie. Kategorie *Program* obsahuje volbu pro načtení série, kdy se objeví dialog pro vybrání složky obsahující DICOM soubory se snímky. Po zvolení je série okamžitě načtena a program zobrazí první snímek. Ostatní volby v této sekci jsou pro ukončení programu nebo zobrazení panelu s informacemi o programu.

Kategorie *Render* slouží k výběru projekce, kterou chce uživatel zobrazit. Podporované projekce jsou MPR, AIP, MIP, MinIP, SSD a VRT. Po zvolení program převede zobrazenou sérii na objem a projekci okamžitě zobrazí.

Panels slouží k zapínání a vypínání panelů. Nejdůležitějším panelem je *Viewer tools*, zeleně zvýrazněný panel s ikonami nástrojů. Další panely se nachází v oranžově zvýrazněné části. Panel *Memory* slouží především pro vývojové účely, zobrazuje aktuální využití operační a případně i grafické paměti. Panel *FPS* ukazuje, kolikrát za sekundu je okno překresleno. Panel *Windowing* udává aktuální rozsah jasového okna.

Tools umožňuje přepínat panely s nástroji, které se nevešly do *Viewer tools*. První z nich je panel *Slab size* sloužící k nastavení rozměrů desky při vykreslování pouze částí objemů. Nástroji *Transfer function editor* bude věnována podkapitola 7.4.

7.3 Panel nástrojů

Zeleně zvýrazněný panel obsahuje ikony nástrojů. Kliknutím na ikonku daným tlačítkem myši se nástroj naváže na zvolené tlačítko. Ikonka nástroje navázaného na levé tlačítko myši je zvýrazněna červeně, prostřední modře a střední zeleně. Stisknutím navázaného tlačítka a tažením myši na vykreslovacím plátně dochází k interakci s nástrojem. Například při práci s nástrojem pro úpravu jasového okna je při tažení směrem nahoru zvýšena pozice středu okna a při tažení směrem doleva zvětšena jeho velikost.

Ikony nástrojů se zobrazují v závislosti na zobrazené projekci a obrazu. Při zobrazení snímku série nemá smysl natáčet objem. Pro pohyb v rámci objemu nebo série je použito kolečko myši.

Po sobě jdoucí nástroje na obrázku jsou:

1) volba jasového okna;
2) natočení zobrazeného objemu;
3) přiblížení či oddálení vykresleného obrazu;
4) posun vykresleného obrazu;
5) invertování zobrazených barev;
6) návrat všech parametrů na původní hodnoty.

7.4 Editor Přenosové funkce

Pro upravování přenosové funkce při vykreslování celého objemu byl navržen nástroj, který je na Obr. 41 žlutě zvýrazněn. Panel je rozdělen na ovládací část s tlačítky a vykreslovací s křivkami reprezentujícími přenosovou funkci.

Tlačítko „*?*“ aktivuje nápovědu, kterou je možné opětovným klepnutím na tlačítko schovat. Obarvená tlačítka aktivují režim úprav pro příslušnou barevnou křivku. Osa X této křivky reprezentuje hodnotu denzity v Hounsfieldových jednotkách. Konkrétní hodnotu si lze zobrazit podržením pravého tlačítka myši na zvoleném místě. Osa Y určuje přiřazenou hodnotu neprůhlednosti dané barvy. Osy nejsou ve vykreslovací části zobrazeny z důvodu šetření místem. Pro úpravu všech hodnot je možné použít křivku zobrazenou po kliknutí na tlačítko *S*. Tato křivka udává koeficient, kterým je výsledná barva pro danou denzitu vynásobena. Pro návrat do režimu zobrazení slouží tlačítko *A*.

Vytvořenou přenosovou funkci je možné ukládat do souborů formátu JSON a opět načítat pomocí tlačítek *Export* a *Import*. Tlačítko *Reset* vrátí křivky do stavu, kdy se objem vykresluje pouze ve stupních šedi.

ZÁVĚR

Všechny cíle práce byly splněny. V rámci práce byly představeny základní pojmy standardu DICOM, princip výpočetní tomografie a projekce volumetrických dat, které jsou využívané v radiodiagnostice. Cílem vytvořené aplikace je demonstrovat projekce a vizualizace představené v této práci. Program není určen pro klinické použití.

Program využívá pro tvorbu grafického prostředí pouze jednoduchou knihovnu. Vykreslování objemů realizuje pomocí grafického API OpenGL. Implementace na takto nízké úrovni se pozitivně projevuje na rychlosti programu, která je při vizualizaci objemů kritická. Program tak splnil všechny požadavky zadané v analýze. Testování aplikace proběhlo na vybrané množině anonymizovaných sérií. Testovací série jsou přiloženy k této práci.

Pro vykreslení celého objemu byl také vyvinut jednoduchý editor usnadňující tvorbu přenosové funkce. Změny funkce provedené uživatelem se okamžitě projevují na vizualizaci, což umožňuje uživateli interaktivně prohledávat celý objem a zobrazit pouze části těla, o které se zajímá.

Projekt se skládá ze 130 tříd s 11 000 řádky kódu. Počet tříd a řádků není vhodná metrika pro posouzení celkové kvality, ale doufám, že jejich poměr alespoň částečně poukazuje na čistotu a dostatečnou přehlednost kódu, aby se v něm čtenář mohl snadno zorientovat a případně jej i rozšířit.

Při psaní jsem se pohyboval mezi obory radiologie, reprezentace dat, počítačové grafiky a vývoje aplikací. Význam a smysl této práce vnímám zejména v tom, že se věnuje sloučení těchto světů dohromady. Každý obor je složitou problematikou. Jen samotný stručný úvod do kombinace těchto oborů poskytl více než dostatek materiálu pro tuto práci.

POUŽITÁ LITERATURA

[1] CLUNIE, David A. *DICOM structured reporting*. Bangor: PixelMed Pub., 2000. ISBN 978–1–56881–266–3.

[2] ENGEL, Klaus. *Real–time volume graphics*. Wellesley: A K Peters, 2006. ISBN 978–1–56881–266–3.

[3] SEIDL, Zdeněk a kol. *Radiologie pro studium i praxi*. Praha: Grada Publishing, a.s, 2012. ISBN 978–80–247–4108–6.

[4] *Digital Imaging and Communications in Medicine (DICOM)* [online]. Arlington: National Electrical Manufacturers Association, 2011 [cit. 2019–04–09]. Dostupné z: ftp://medical.nema.org/medical/dicom/2011/

[5] HEŘMAN, Miroslav a kol**.** *Základy Radiologie*. Olomouc: Univerzita Palackého v Olomouci, 2014. ISBN 978–80–244–2901–4.

[6] BUSHBERG Jerrold, SEIBERT Anthony, LEIDHOLDT Edwin, BOONE John. *The essential physics of medical imaging*. 3. Philadelphia : Lippincott Williams & Wilkins, 2012. ISBN 978–1451118100.

[7] DALRYMPLE, Neal, Srinivasa PRASAD, Michael FRECKLETON a Kedar CHINTAPALLI. Introduction to the Language of Three–dimensional Imaging with Multidetector CT. *RadioGraphics* [online]. 2005 [cit. 2019–04–10]. DOI: 10.1148/rg.255055044. Dostupné z: https://pubs.rsna.org/doi/10.1148/rg.255055044

[8] PERANDINI, Simone, N FACCIOLI, A ZACCARELLA, TJ RE a R Pozzi MUCELLI. The diagnostic contribution of CT volumetric rendering techniques in routine practice. *Indian Journal of Radiology and Imaging* [online]. 2010 [cit. 2019–04–11]. DOI: 10.4103/0971–3026.63043. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2890933/

[9] BIDGOOD, W. Dean, Steven C. HORII, Fred W. PRIOR a Donald E. VAN SYCKLE. Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging. *National Center for Biotechnology Information* [online]. 1997 [cit. 2019–04–17].

PMID: 9147339, PMCID: PMC61235. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC61235/

[10] *Sources and effects of ionizing radiation* [online]. New York: United Nations Scientific Committee on the Effects of Atomic Radiation, 2010, s. 4 [cit. 2019–04–22]. ISBN 978–92–1–142274–0. Dostupné z: http://www.unscear.org/unscear/en/publications/2008_1.html

[11] Positron Emission Tomography – Computed Tomography (PET/CT). *RadiologyInfo.org* [online]. 2017 [cit. 2019–04–22]. Dostupné z: https://www.radiologyinfo.org/en/info.cfm?pg=pet

[12] OPPELT, Arnulf, ed. Imaging Systems for Medical Diagnostics: Fundamentals, Technical Solutions and Applications for Systems Applying Ionizing Radiation, Nuclear Magnetic Resonance and Ultrasound. 2. Erlangen: Publicis Corporate Publishing, 2006. ISBN 3–89578–226–2.

[13] OUDKERK, Matthijs, ed a kol. Coronary Radiology. Berlin: Springer, 2004. ISBN 3–540–43640–5.

[14] FERNANDO, Randima, ed a kol. *GPU Gems: Programming Techniques, Tips and Tricks for Real–Time Graphics*. Boston: Addison–Wesley Professional, 2004. ISBN 978–0321228321.

[15] ZHANG, Qi, Roy EAGLESON a Terry M. PETERS. Volume Visualization: A Technical Overview with a Focus on Medical Applications. *J Digit Imaging* [online]. PubMed Central, 2010, 2011 [cit. 2019–04–23]. DOI: 10.1007/s10278–010–9321–6. PMID: 20714917, PMCID: PMC3138940. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3138940/

[16] *OpenGL: The Industry's Foundation for High Performance Graphics* [online]. Beaverton (Oregon): Khronos Group, 2019 [cit. 2019–04–24]. Dostupné z: https://www.opengl.org

[17] *Dcm4che.org: Open Source Clinical Image and Object Management* [online]. [cit. 2019–04–23]. Dostupné z: https://www.dcm4che.org

[18] *Lightweight Java Game Library* [online]. [cit. 2019–04–23]. Dostupné z: https://www.lwjgl.org

[19] *GLFW* [online]. 2019 [cit. 2019–04–23]. Dostupné z: https://www.glfw.org

[20] *Nuklear: A single-header ANSI C gui library* [online]. Micha Mettke [cit. 2019-04-23]. Dostupné z: https://github.com/vurtun/nuklear

[21] *Dear ImGui: Bloat-free Immediate Mode Graphical User interface for C++ with minimal dependencies* [online]. ocornut [cit. 2019-04-23]. Dostupné z: https://github.com/ocornut/imgui

[22] *Gson: A Java serialization/deserialization library to convert Java Objects into JSON and back* [online]. Mountain View: Google [cit. 2019-04-23]. Dostupné z: https://github.com/google/gson

[23] Radiation Dose in X-Ray and CT Exams. *RadiologyInfo.org* [online]. Reston: American College of Radiology, 20. 3. 2019 [cit. 2019-04-29]. Dostupné z: https://www.radiologyinfo.org/en/info.cfm?pg=safety-xray

[24] *STB libraries* [online]. Sean Barett, 2016 [cit. 2019-04-29]. Dostupné z: https://stb.handmade.network

[25] NOUMEIR, Rita a Jean-François PAMBRUN. Teaching DICOM by Problem Solving. *J Digit Imaging* [online]. PubMed Central, 2012 [cit. 2019-05-04]. DOI: 10.1007/s10278-012-9471-9. PMID: 22476384, PMCID: PMC3447100. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3447100/

[26] MARTIN, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. New Jersey: Prentice Hall, 2008. ISBN 978-0132350884.

[27] WRIGHT, Richard S., Graham SELLERS a Nicholas HAEMEL. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. 6. New Jersey: Addison-Wesley, 2014. ISBN 978-0-321-90294-8.

[28] *JOML – Java OpenGL Math Library* [online]. Kai Burjack, 2019 [cit. 2019-05-10]. Dostupné z: https://github.com/JOML-CI/JOML

[29] *Tinyfiledialogs* [online]. Joshua Granick, 2019 [cit. 2019-05-10]. Dostupné z: https://sourceforge.net/projects/tinyfiledialogs/

[30] LEPEŠKA, Martin. *Herní 2D engine v Javě*. Pardubice, 2017. Bakalářská práce. Univerzita Pardubice.

[31] *An intro to modern OpenGL. Chapter 1: The Graphics Pipeline* [online]. San Jose: Joe Groff, 2010 [cit. 2019-05-10]. Dostupné z: http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Chapter-1:-The-Graphics-Pipeline.html

[32] WARNOCK, Max J a kol. Benefits of Using the DCM4CHE DICOM Archive. *J Digit Imaging* [online]. PubMed Central, 2007, [cit. 2019-05-11]. DOI: 10.1007/s10278-007-9064-1. PMCID: PMC2039778, PMID: 17917780. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2039778/

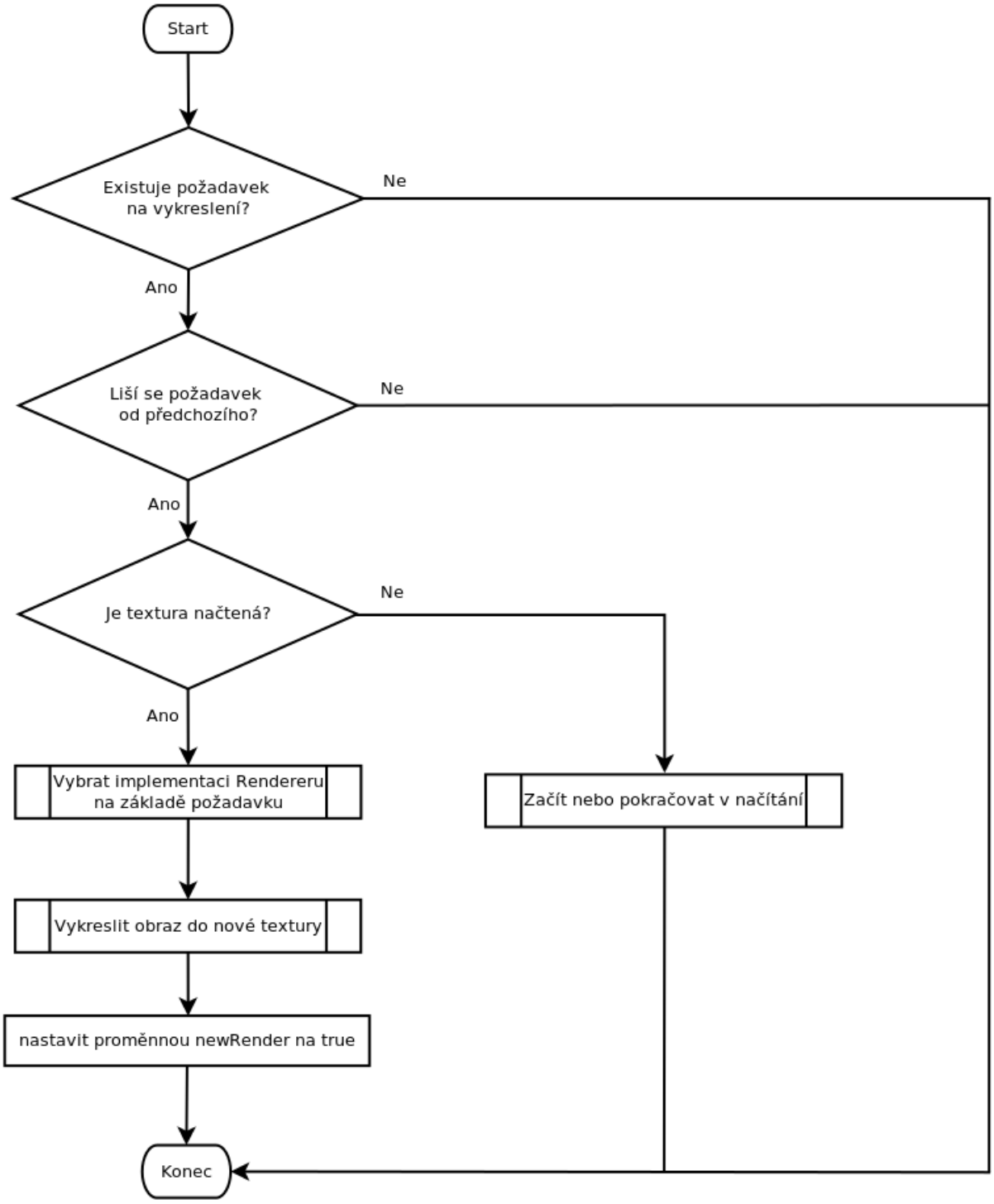
[33] *A Primer on Bézier Curves* [online]. Pomax, 2019 [cit. 2019-05-12]. Dostupné z: https://pomax.github.io/bezierinfo/

[34] *OpenGL Rendering Pipeline* [online]. Song Ho Ahn, 2005 [cit. 2019-05-14]. Dostupné z: http://www.songho.ca/

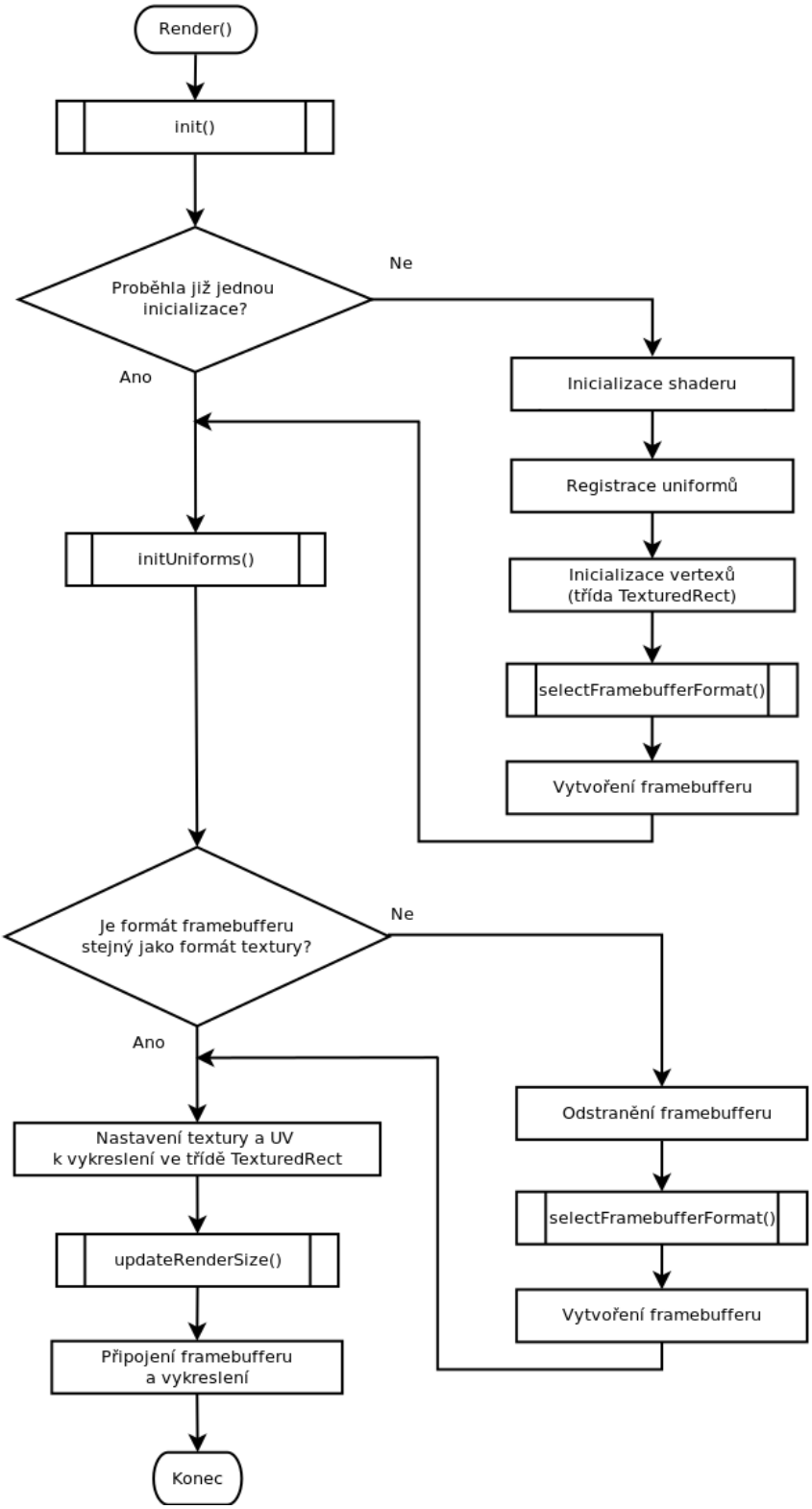
[35] SPIRIEV, Toma, Vladimir NAKOV, Lili LALEVA a Christo TZEKOV. OsiriX software as a preoperative planning tool in cranial neurosurgery: A step-by-step guide for neurosurgical residents. *Surgical Neurology International* [online]. 2017 [cit. 2019-05-15]. DOI: 10.4103/sni.sni_419_16. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5655755/

PŘÍLOHY

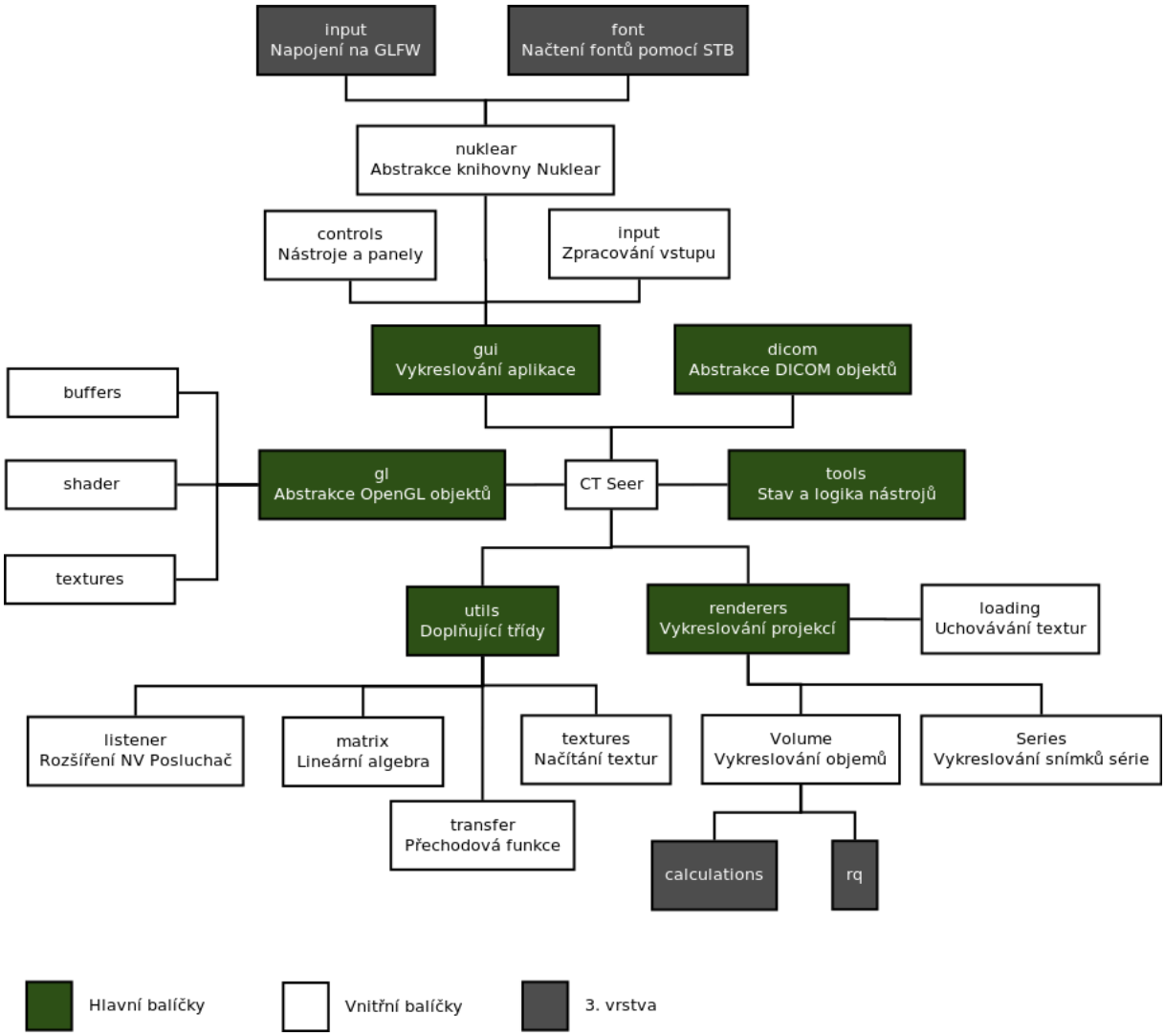
PŘÍLOHA A – ZJEDNODUŠENÝ PROCES PŘIPRAVENÍ NOVÉ PROJEKCE



PŘÍLOHA B – POŘADÍ VOLANÝCH FUNKCÍ VE TŘÍDĚ AVOLUMERENDERER

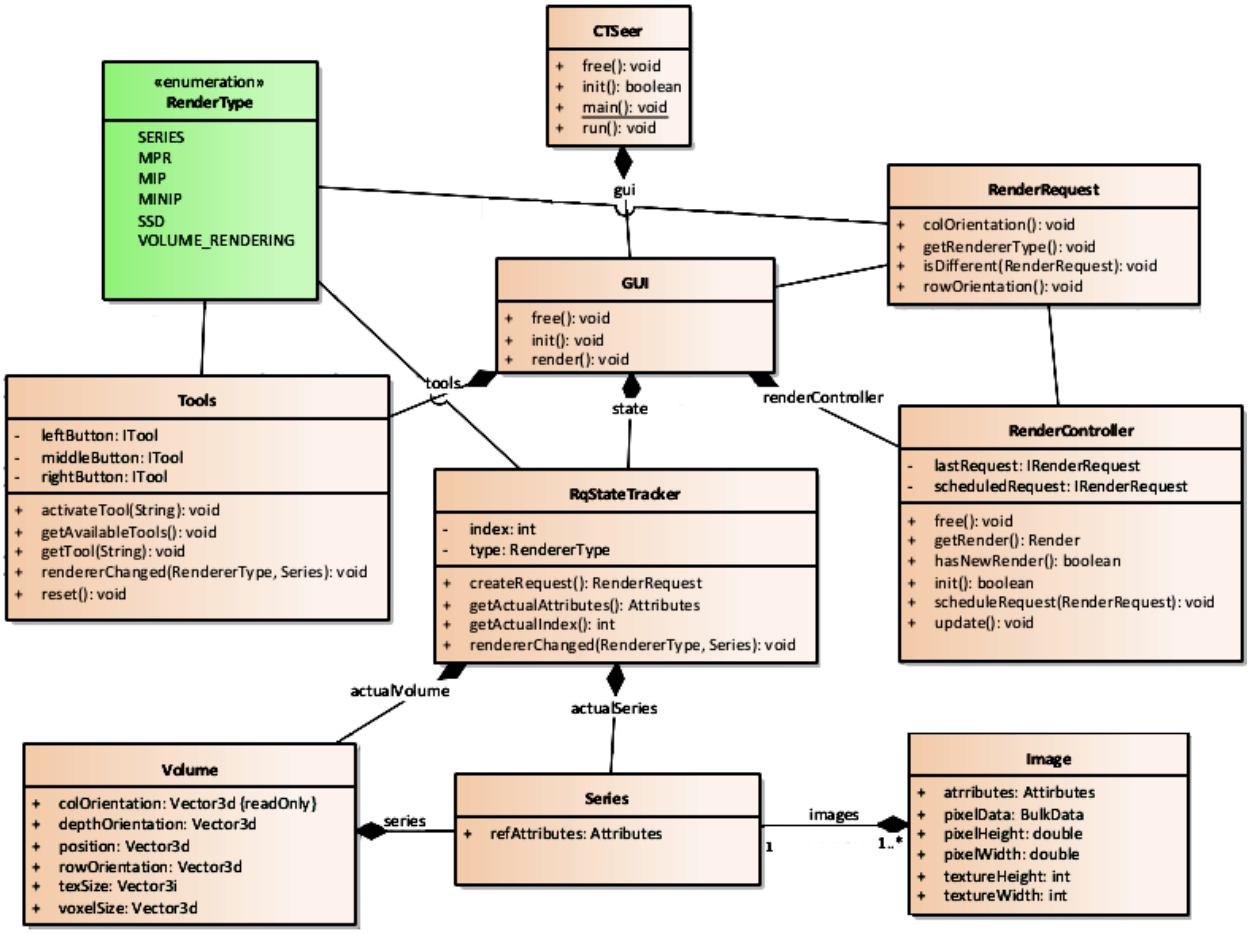


PŘÍLOHA C – BALÍČKY PROJEKTU CTSEER



PŘÍLOHA D – POŽADAVKY NA PROGRAM CTSEER

PŘÍLOHA E – DIAGRAM ANALYTICKÝCH TŘÍD



PŘÍLOHA F – OBSAH PŘILOŽENÉHO DVD

Položka	Umístění
Spustitelný JAR programu CTSeer	`CTSeer/CTSeer.jar`
Série DICOM snímků pro testování	`CTSeer/Examples/Series/`
Příklady přenosových funkcí	`CTSeer/Examples/Transfer Functions/`
InteliJ IDEA projekt programu	`Projekt/CTSeer`
Programátorská dokumentace	`Dokumentace/javadoc/`
Enterprise Architect projekt s diagramy	`Dokumentace/CTSeer.eapx`
Uživatelská dokumentace	`Dokumentace/Manual.pdf`
Video ukázky funkcí programu	`Dokumentace/Videa`
Diplomová práce ve formátu PDF	`LepeskaM_ProjekceSnimku_PV_2019.pdf`