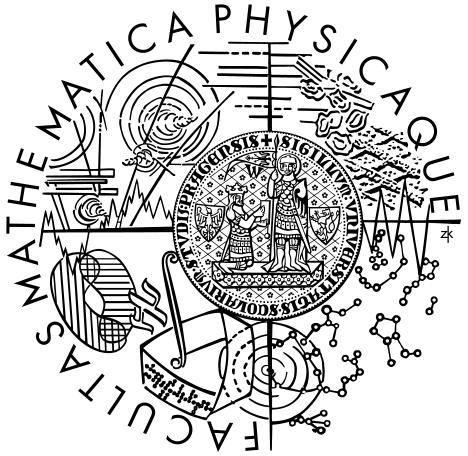
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Miroslav Baron

Systém pro tvorbu editorů přechodových funkcí

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Václav Krajíček
Studijní program: Informatika
Studijní obor: Programování

Praha 2011

Na tomto místě bych rád poděkoval RNDr. Krajíčkovi za odborné vedení, Mgr. Kolomazníkovi za poskytnutí prohlížeče objemových dat a za rychlou pomoc při řešení problémů a také rodině za podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Systém pro tvorbu editorů přechodových funkcí

Autor: Miroslav Baron

Katedra: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Václav Krajíček

Abstrakt: Existuje mnoho způsobů navrhování přechodových funkcí pro DVR. Tato práce představuje obecný systém editorů těchto funkcí, který je současně frameworkem pro tvorbu nových typů editorů. Jeho jednotné rozhraní a datová struktura navržená pro přechodovou funkci zajišťuje nezávislost na zobrazovacím enginu a umožňuje současné používání různých editorů. Implementace nového způsobu návrhu je tedy oproštěna o nutnost řešit komunikaci se zobrazovacím enginem a začlenění editoru do systému je realizováno několika dobře definovanými kroky. V systému je již implementováno několik zajímavých způsobů navrhování přechodových funkcí a v této práci je zkoumána jejich užitečnost.

Klíčová slova: Přechodová funkce, Framework, Editace, Uživatelské Rozhraní

Title: Framework for designing transfer function editors

Author: Miroslav Baron

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Václav Krajíček

Abstract: There are many ways to define a transfer function for direct volume rendering (DVR). This thesis presents an universal system of transfer function editors which works also as framework for creating new types of these editors. Unified interface with proposed transfer function data structure gives us independency on used graphic engine and the possibility to work with more different editors at the same time. We can implement new technique for designing transfer function without the need of dealing with the graphic engine and we can add our new editor to this existing system in few well defined steps. We have also designed some interesting ways to define transfer function and we discuss their usefulness here.

Keywords: Transfer Function, Framework, Editing, User Interface

Obsah

Seznam obrázků

Úvod

Motivace

Existují různé metody pro získávání objemových dat, jako příklad můžeme uvést výpočetní tomografii (CT – Computed Tomography). V závislosti na použité metodě se mohou poskytovaná data lišit a to zejména rozsahem hodnot. U CT se jedná typicky o 12-ti bitové hodnoty [6]. Většinou tento rozsah (jako u CT) nelze celý zobrazit na monitoru počítače. Proto vznikla potřeba přechodových funkcí. Jedná se o funkce, které mapují rozsah hodnot těchto dat do oblasti zobrazitelných hodnot. Výsledkem aplikace přechodové funkce na hodnotu obsaženou v objemových datech je informace pro zobrazovací engine, jež může popisovat obecně jakékoli optické vlastnosti zobrazitelné v počítačové grafice [5].

Cílem používání přechodových funkcí je poskytnout člověku náhled na tato data. Z toho plyne, že návrh „dobré“ přechodové funkce je závislý na posouzení náhledu člověkem a také na tom, co přesně je třeba zobrazit. Například v lékařství je několik možných způsobů zkoumání dat, můžeme chtít vidět pouze kosti, nebo pouze arterie, či jiný druh tkáně. Z těchto důvodů nelze návrh přechodových funkcí plně automatizovat a je vždy nutné alespoň „doladit“ některé parametry či poskytnout hodnocení automaticky generované funkce [8]. To dalo vzniknout mnoha více či méně automatizovaným postupům návrhu přechodových funkcí jako například Style Transfer Functions [9], návrh pomocí logaritmické stupnice [7], nebo výběr z generovaných návrhů [8].

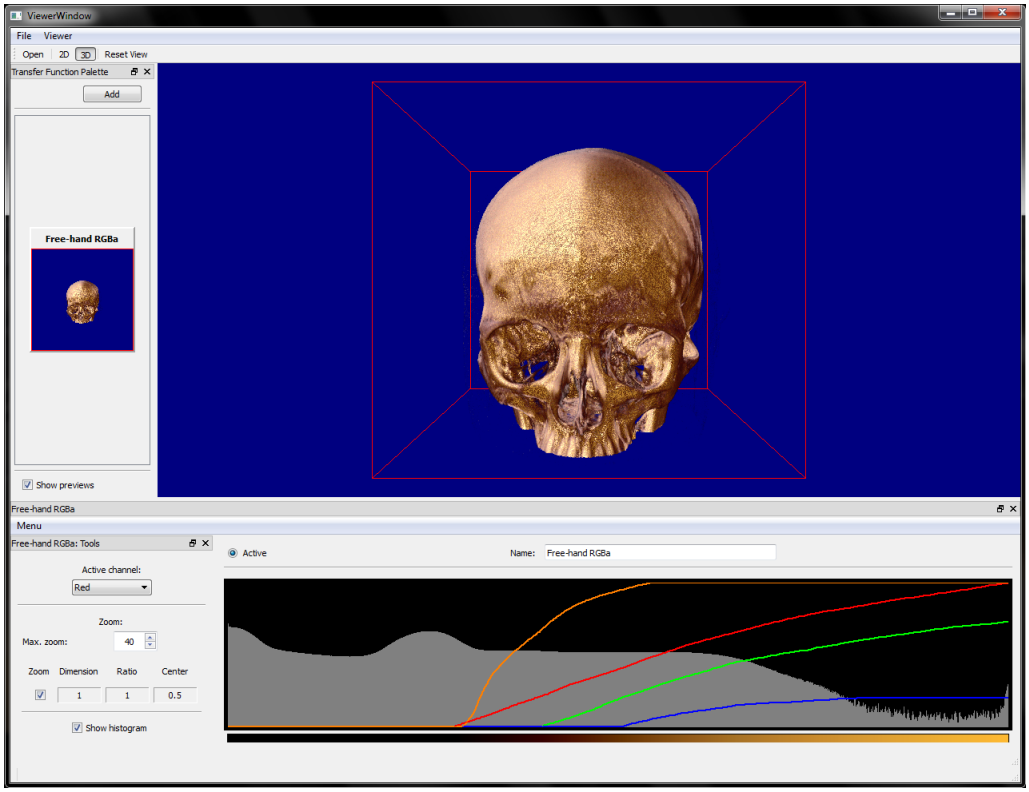
Cíle práce

Našim cílem bylo vytvořit obecný systém editorů přechodových funkcí, který by byl současně frameworkem pro tvorbu nových typů editorů. Základním požadavkem bylo jednotné rozhraní a datová struktura obsahující přechodovou funkci a s tím související požadavek na nezávislost na zobrazovacím enginu. Dále, aby bylo možné mít současně otevřených více editorů, nezávisle na jejich typu. Typem editoru rozumějme metodu návrhu přechodové funkce. Důležitým požadavkem na tento systém jako na framework byla možnost snadného rozšíření o nový typ editoru. Jak bylo již zmíněno, existuje mnoho způsobů návrhu přechodové funkce a úkolem tohoto systému je poskytnout možnost jednoduchého rozšíření o nový způsob návrhu a zajistit jednotné rozhraní pro zobrazovací engine, který použije výslednou přechodovou funkci pro zobrazení objemových dat, jako například na obrázku 1.

Aby bylo možné demonstrovat požadované vlastnosti, obsahuje výsledná aplikace implementaci několika typů námi navržených editorů a používá pro zobrazení výsledků zobrazovací engine vyvíjený pro projekt MedV4D [3].

Projekt MedV4D

Cílem projektu MedV4D [3] je vývoj komplexního software pro zpracování a analýzu medicínských obrazových dat. Medicínská obrazová data (například ra-



Obrázek 1: Ilustrační příklad

diogramy, CT, MRI – Magnetic Resonance Imaging, ultrazvuk) se v dnešní době stala základním nástrojem pro lékařské diagnózy. Jsou úspěšně používána ve vývoji nových lékařských postupů, vzdělávání lékařského personálu a dokonce při chirurgických zákrocích.

Potřeba tohoto projektu používat přechodové funkce pro zobrazování lékařských dat byla podnětem pro vznik této práce.

Struktura textu

V kapitole 1 si povíme něco o přechodových funkcích obecně a o tom, jak je můžeme dělit. Dále si v této kapitole řekneme o možných způsobech navrhování přechodových funkcí.

Kapitola 2 popisuje implementaci celého systému. Je zde popsána struktura, jednotlivé části systému, použité datové struktury a také v této kapitole najdeme popis implementace námi vytvořených editorů.

Dále si uvedeme několik experimentů 3, kde si ověříme některé vlastnosti přechodových funkcí a ukážeme si použití implementovaných editorů v praxi.

V závěru pak uvedeme možnosti dalšího rozšiřování a shrneme si, čeho jsme v této práci dosáhli.

Uživatelský manuál A popisuje grafické uživatelské rozhraní systému a také implementovaných editorů. Najdeme zde také návody a postupy, které nám osvětlí používání a způsob práce jak se systémem, tak s jednotlivými editory.

V příloze B můžeme najít popis obsahu přiloženého DVD.

Příloha C stručně popisuje překlad projektu pomocí programu CMake.

1. Přechodové funkce

Existuje více typů přechodových funkcí, ale také existuje mnoho způsobů jejich návrhu . V této kapitole si nejprve obecně nadefinujeme přechodové funkce a řekneme si podle čeho je můžeme dělit. Dále uvedeme několik způsobů dělení jejich návrhu a nakonec si představíme pár zajímavých postupů pro nalezení vhodné přechodové funkce.

1.1 Definice

Zde uvedeme jednu možnou definici přechodové funkce [8]:

Definice. *Obecně je přechodová funkce τ funkce mapující kartézský součin skalárních hodnot F na kartézský součin optických vlastností O:*
$\tau : F_1 \times F_2 \times ... \times F_n \longrightarrow O_1 \times O_2 \times ... \times O_m$,
kde n určuje dimenzi mapovaných hodnot a m určuje počet optických vlastností, které funkce umožňuje ovlivňovat. Velikost n je běžně označována jako dimenze přechodové funkce.

1.2 Dělení přechodových funkcí

Přechodové funkce lze dělit [5] podle vstupní a výstupní dimenze.

1.2.1 Dělení dle vstupní dimenze

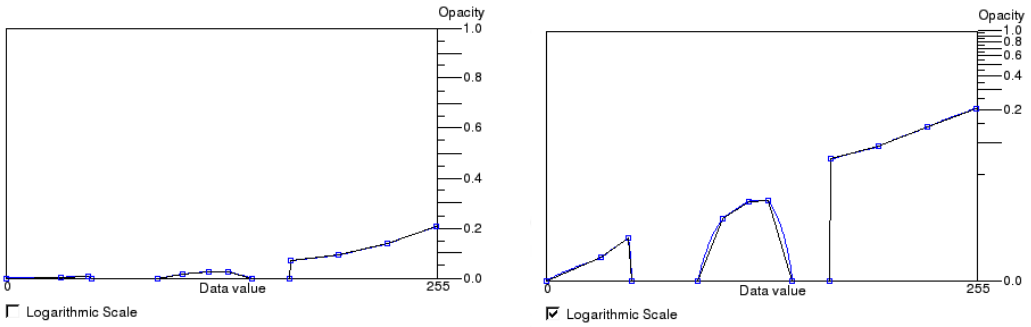
Vstupní dimenze neboli také dimenze přechodové funkce je hlavním parametrem. Určuje nám kolik informací máme pro návrh. Možnost použití další dimenze se nabízí, chceme-li pro návrh použít například gradient, či druhou derivaci první dimenze. Velikost gradientu udává rychlost změny vstupních dat.nám umožňuje nalézt hranice mezi homogenními oblastmi v objemových datech a tím odlišit jinak neodlišitelné oblasti se stejnými hodnotami. Využití gradientu je užitečné například v lékařství, kde často potřebujeme rozlišit mezi různými druhy tkání. Druhá derivace vstupních hodnot umožňuje detekci hran v obrazu.

Některé metody získávání objemových dat však poskytují přirozené použití více-dimenzionálních přechodových funkcí, jedná se například o metodu magnetické rezonance (MRI). Na rozdíl od CT, kdy zkoumáme jen jeden parametr vyšetřované tkáně, při magnetické rezonanci získáváme informaci téměř o deseti parametrech [6].

Dimenze přechodové funkce je obvykle 1 nebo 2, jelikož návrh funkcí pro vyšší dimenze bývá dosti náročný.

1.2.2 Dělení dle výstupní dimenze

Výstupní dimenze nám určuje počet optických vlastností, které funkce umožňuje ovlivňovat. Zde můžeme zahrnout všechny optické vlastnosti, které lze zobrazit prostředky počítačové grafiky, respektive použitým zobrazovacím enginem. Základní přechodové funkce umožňují ovlivňovat pouze zobrazovanou průhlednost



Obrázek 1.1: Manuální návrh přechodové funkce [7]

a barvu, ty nejjednodušší pouze průhlednost (příklady základních přechodových funkcí a jejich vliv na zobrazovaná data můžeme vidět v kapitole 3). Například pro potřeby lékařství jsou tyto funkce dostačující. Jsou však oblasti, jako například tvorba ilustrativních obrázků pro lékařské publikace popsaná v [9], kde je žádoucí ovlivňovat také například odraz světla či index lomu světla.

1.3 Způsoby návrhu přechodových funkcí

1.3.1 Manuální návrh

Návrh přechodové funkce může záviset pouze na vstupu navrhovatele. Jedná se o postup, kdy člověk jistým způsobem (závislým na konkrétní implementaci) navrhuje průběh funkce a sleduje výsledek její aplikace na objemová data v prohlížeči. Návrh přechodové funkce může například spočívat v kreslení křivek udávajících průhlednost a barvu. Manuální návrh může být složitý a obyčejně vyžaduje alespoň minimální znalost problému. Na druhou stranu poskytuje navrhovateli nad funkcí plnou kontrolu, poskytuje volnost při navrhování a pro přechodové funkce s dimenzí nejvýše 2 bývá přijatelný.

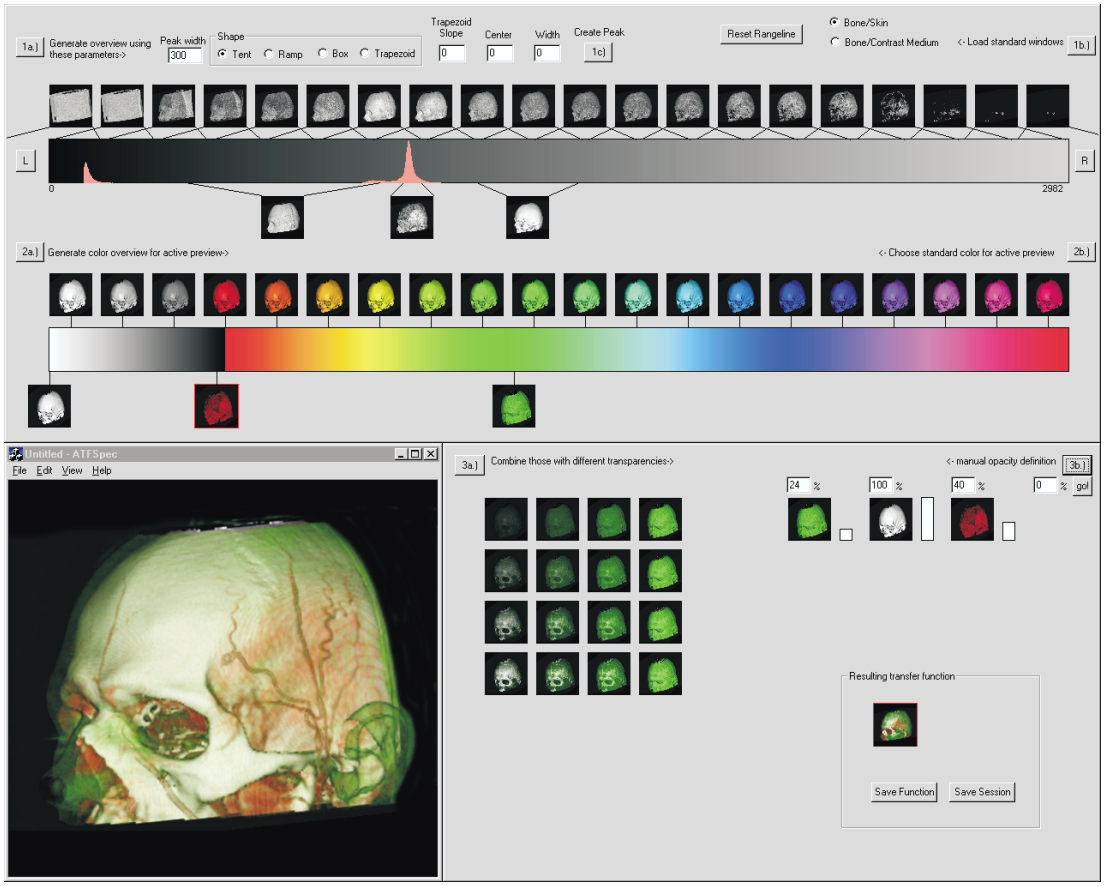
V této práci je implementováno několik editorů s tímto způsobem návrhu pro ukázku použití systému a pro ověření požadovaných vlastností. Popis těchto editorů a jejich grafických rozhraní je zahrnut v uživatelském manuálu A.

Na obrázku 1.1 můžeme vidět návrh přechodové funkce z publikace [7], vlevo je stupnice osy y lineární a vpravo je logaritmická. Takováto editace přechodové funkce je jedním z příkladů manuální editace.

1.3.2 Poloautomatizovaný návrh

Pro usnadnění návrhu přechodové funkce, například pro více dimenzí, se používá postup, kdy aplikace s použitím nějakého algoritmu sama ze vstupních dat vygeneruje návrh přechodové funkce a navrhovateli poskytne například možnost ji upravit nebo pouze „doladit“ úpravou několika parametrů ovlivňujících algoritmus.

Tento způsob více či méně snižuje nutnost znalosti problému navrhovatelem, výsledky však mohou být silně závislé na použitém algoritmu.



Obrázek 1.2: Výběrový návrh přechodové funkce [8]

1.3.3 Výběrový návrh

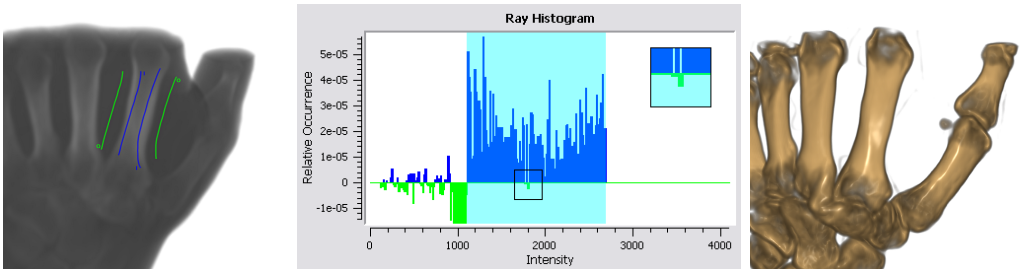
Výběrový návrh by se dal téměř označit jako automatický. Návrh přechodové funkce se řídí jistým algoritmem podobně jako u poloautomatizovaného návrhu, avšak aplikace vygeneruje přechodové funkce pokrývající co nejvíce doménu parametrů ovlivňujících algoritmus. Aplikace poté již jen nabídne kolekci těchto funkcí a navrhovatel pouze vybere přechodovou funkci, která mu nejvíce vyhovuje.

Tímto způsobem lze navrhovatele plně oprostit od znalosti problému, stačí pouze, aby si vybral přechodovou funkci, která mu nejvíce vyhovuje. Úspěšnost tohoto postupu však plně závisí na zvoleném algoritmu a schopnosti dostatečně pokrýt celou doménu jeho parametrů.

Příklad takového návrhu můžeme vidět na obrázku 1.2 z publikace [8]. Autoři v tomto případě navíc oddělili návrh jednotlivých výstupních dimenzí přechodové funkce a poskytují kolekci vygenerovaných funkcí pro každou dimenzi zvlášť. Navrhovatel si zvolí z každé kolekce a výsledná přechodová funkce vznikne složením všech výstupních dimenzí.

1.3.4 Lokalizační návrh

Jedná se o postup, kdy navrhovatel přechodové funkce určitým způsobem označí oblast přímo na zobrazených objemových datech a tím určí hodnoty, které chce zobrazit/obarvit. Poté se pomocí jistého algoritmu vygeneruje přechodová funkce znázorňující danou oblast hodnot.



Obrázek 1.3: Lokalizační návrh přechodové funkce [10]

Takovýto návrh přechodové funkce ovšem vyžaduje přímou spolupráci se zobrazovacím enginem a je tedy nevhodný pro náš systém, jelikož jeden z požadavků byl nezávislost na použitém zobrazovacím enginu.

V publikaci [10] najdeme popis postupu odpovídající tomuto druhu návrhu. Autoři zvolili postup, kdy navrhovatel do již zobrazených dat načrtne několik křivek. Podle jejich umístění se s použitím autory popsaného algoritmu navrhne přechodová funkce. Příklad můžeme vidět na obrázku 1.3 z této publikace.

1.4 Shrnutí

Popsané druhy přechodových funkcí a způsoby jejich editace nám dávají širokou škálu možných editorů. Když si ještě uvědomíme, že každý druh návrhu může mít také mnoho způsobů ovládání a různá grafická rozhraní (například u manuálního návrhu máme možnost kreslit křivky od ruky či definovat pouze body, lze použít logaritmickou stupnici popsanou v [7] a podobně), tak dostáváme velké množství editorů přechodových funkcí pro implementaci. Kdybychom jich implementovali jen několik, nejspíš bychom pro každý také znovu řešili komunikaci se zobrazovacím enginem a — pokud bychom nevytvářeli pro každý novou aplikaci — doplňovali aplikaci o nová tlačítka, měnili běh aplikace pro konkrétní editor a podobně. Z výše zmíněného vyplývá potřeba vytvořit systém, který reflektuje tyto nedostatky. Jediné, co by měl tvůrce nového editoru udělat, je implementovat pouze vlastní způsob editace. Navíc by bylo vhodné, aby byl editor složen z několika základních částí. To by umožňovalo vytvořit nový editor pouze změnou jedné či více částí.

2. Implementace

2.1 Úvod

V této kapitole popíšeme implementaci našeho systému pro tvorbu editorů přechodových funkcí . Následující kapitoly popisují jednotlivé části systému a to jednak datové struktury, použité algoritmy a abstraktní předky tříd určené pro rozšiřovaní, tak implementaci konkrétních typů námi navržených editorů. Nejprve si ale uvedeme několik základních informací o prostředcích použitých pro vývoj.

2.1.1 Zobrazování objemových dat

Aby jsme měli představu, jak použití navržených přechodových funkcí vypadá, je součástí aplikace prohlížeč, který umožňuje zobrazení objemových dat. Data lze zobrazit dvojrozměrně, kdy si prohlížíme vždy jeden konkrétní průřez tojrozměrným snímkem [6], nebo trojrozměrně, kde můžeme vidět celý trojrozměrný snímek najednou. Námi používaný prohlížeč je vyvíjen pro projekt MedV4D Mgr. Kolomazníkem. Pro naše potřeby jsme jej upravili, aby poskytoval pouze funkce spojené s tímto systémem.

2.1.2 Grafické rozhraní

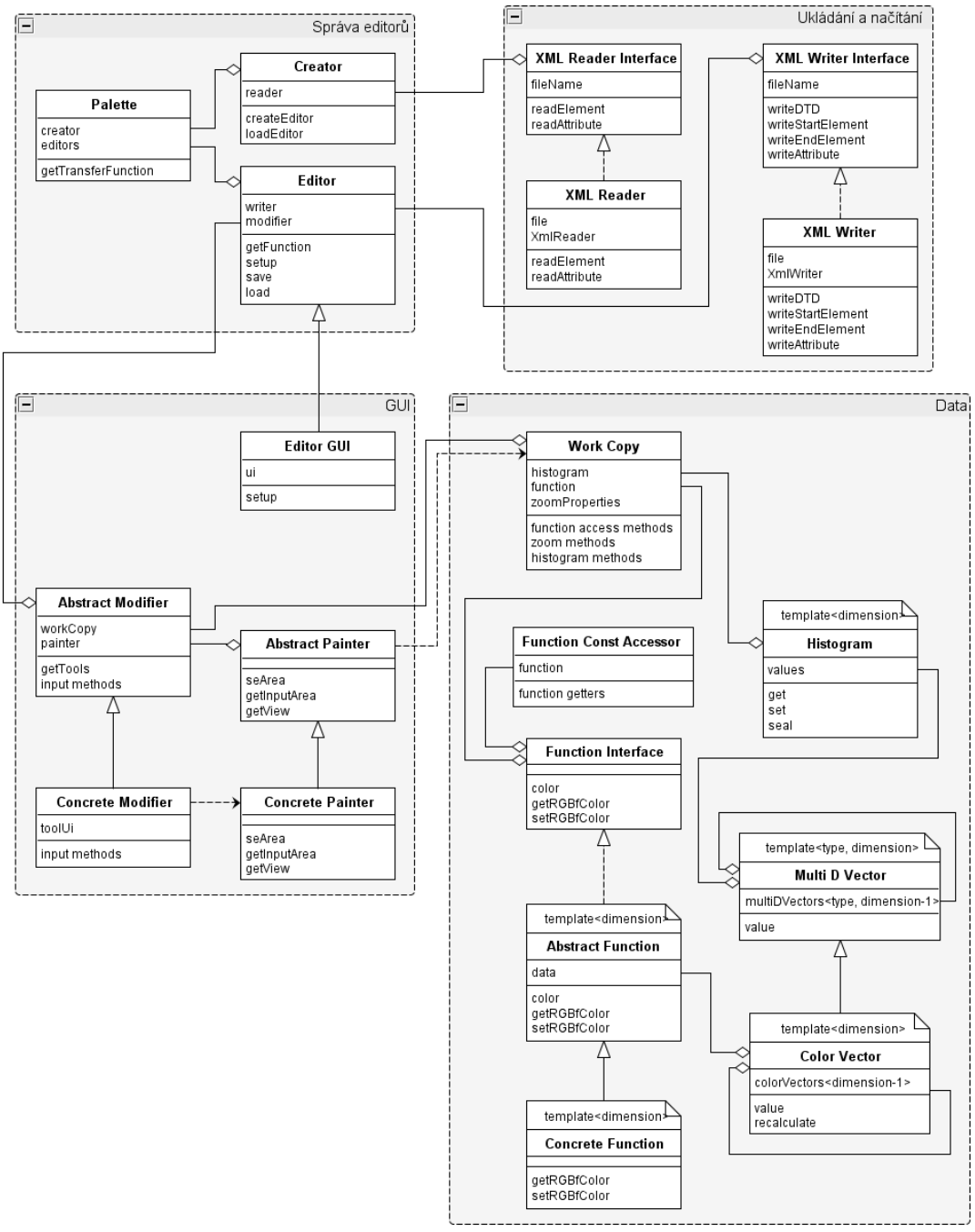
Tento systém obsahuje také grafické rozhraní (GUI – Graphical User Interface). Navíc, abychom mohli dosáhnout našich cílů, musí všechny typy editorů z principu používat stejný framework pro tvorbu grafického rozhraní a to ten samý, který je používán tímto systémem. Grafické rozhraní by mělo být jednoduché a intuitivní a pro jeho implementaci je používána multiplatformní knihovna Qt. Popis všech v této práci zmíněných Qt tříd můžeme najít v dokumentaci [1].

2.1.3 Programovací jazyk

Veškeré zdrojové kódy jsou psány v programovacím jazyce C++. Použitý jazyk je rozšířen o knihovnu Boost [2], která je rovněž využívána v projektu MedV4D. Naše implementace využívá mimo jiné jazykových konstrukcí jazyka C++ známých jako šablony. Popis datových struktur vyžaduje od čtenáře znalost jazyka C++ a také těchto šablon. Pro neznalé čtenáře doporučujeme studijní materiál [4].

2.2 Struktura systému

Na obrázku 2.1 je znázorněn UML diagram struktury celého systému. Jsou zde všechny důležité části systému a u každé části jou uvedeny metody a členy, které jsou pro danou část význačné. Množina potomků, která může být obsáhlá a také může mít složitější hierarchii, je na obrázku 2.1 pro zjednodušení označena jako Concrete a příslušná část. Jednotlivé části budou popsány v následujících kapitolách. Pro přehlednost jsou názvy na obrázku 2.1 pouze orientační (například třída TFXmlWriterInterface je na obrázku nazvána XML Writer Interface a podobně).



Obrázek 2.1: Struktura systému

2.3 Pojmy

Nyní si vysvětlíme význam několika pojmů používaných v celé kapitole 2 pro zjednodušení textu.

- přechodová funkce – instance třídy implementující rozhraní Function Interface (potomek třídy Abstract Function)
- konstantní funkce – instance třídy Function Const Accessor
- data – objemová data načtená a zobrazovaná prohlížečem aplikace MedV4D, která jsou modifikována podle dat uložených v přechodové funkci
- paleta – instance třídy Palette
- registrace – posloupnost kroků potřebná pro korektní začlenění části editoru do systému
- editor – instance potomka třídy Editor
- přípustný editor - editor, jehož přechodová funkce může být použita na otevřená data (všechny editory jsou přípustné, pokud nejsou otevřená žádná data)
- aktivní editor – přípustný editor, jehož přechodová funkce je právě používána na otevřená data, pokud nejsou žádná data otevřena, pak je to ten, jehož přechodová funkce by byla použita, pokud by data byla otevřena
- modifier – instance potomka třídy Abstract Modifier
- tools – grafická komponenta obsahující prvky grafického rozhraní sloužící jako rozšíření základního grafického rozhraní pro potřebu modifieru
- free-hand – způsob kreslení, kreslení volnou rukou (tažení kurzorem po obrazovce)
- painter – instance třídy potomka třídy Abstract Painter
- pracovní kopie – instance třídy Work Copy
- creator – instance třídy Creator
- reader – instance třídy implementující rozhraní XML Reader Interface
- writer – instance třídy implementující rozhraní XML Writer Interface
- aplikace – aplikace, která používá paletu (prohlížeč prjektu MedV4D)

Pokud budeme hovořit o částech editoru, jedná se o modifier, painter a přechodovou funkci. Typem některé části editoru (například typ painteru) rozumíme druh potomka abstraktního předka příslušné části.

2.4 Přechodová Funkce

Abstraktní třída *Abstract Function* je společným předkem všech přechodových funkcí, jedná se o šablonovanou třídu , kde dimenzi přechodové funkce určuje parametr šablony. Tento abstraktní předek obsahuje čistě virtuální metody *getRGBfColor* a *setRGBfColor*, které zpřístupňují danou barvu v barevném modelu RGB (Red, Green, Blue) nezávisle na typu přechodové funkce.Tyto metody musí být implementovány konkrétním potomkem.

Abstract Function řeší ukládání, načítání funkce a přímý přístup k barvám přechodové funkce. Při ukládání a načítání jsou volány také virtuální metody umožňující uložení a načtení dodatečných dat konkrétního potomka. Tyto metody mají implicitní prázdnou implementaci, implicitně se tedy neukládají žádná další data kromě samotné přechodové funkce.

2.4.1 Jednotné rozhraní

Jelikož systém umožňuje mít zároveň otevřeny editory různých dimenzí, pracuje systém pouze s nešablonovaným rozhraním *Function Interface*, který obsahuje všechny veřejné metody šablonované třídy Abstract Function. Ta je potomkem tohoto rozhraní. Function Interface tedy slouží pouze pro definici virtuálních metod. Neobsahuje data, která vyžadují šablonový parametr, ani neimplementuje žádnou metodu pracující s těmito daty. Proto nevyžaduje šablonový parametr a zároveň umožňuje používat šablonovanou třídu implementující toto rozhraní.

2.4.2 Konstantní funkce

Aby při vyžádání přechodové funkce pro modifikaci dat nedocházelo ke zbytečnému kopírování přechodové funkce (ta může být velmi rozsáhlá při více dimenzích), a také aby systém neposkytoval pouze ukazatel na přechodovou funkci, což by umožňovalo modifikovat přechodovou funkci mimo systém a to je nepřípustné, obsahuje systém třídu *Function Const Accessor*.

Function Const Accessor obaluje přechodovou funkci, resp. její nešablonované rozhraní a zpřístupňuje pouze nemodifikující metody přechodové funkce. Díky tomu, že si konstantní funkce drží pouze ukazatel na přechodovou funkci, nedochází ke zbytečnému kopírování (kopíruje se pouze konstantní funkce, tzn. pouze ukazatel) a zároveň je zaručeno, že nepůjde přechodovou funkci modifikovat mimo systém.

2.4.3 Datová struktura

Přechodová funkce může být více-dimenzionální, to znamená, že data uložena v ní musí mít podobu vícerozměrného pole, kde počet jeho rozměrů je zadán parametrem šablony. Aby to bylo možné obsahuje přechodová funkce datovou strukturu *Color Vector*.

Color Vector je šablonovaná třída, kde parametr šablony určuje dimenzi. Tato třída obsahuje standardní vektor, velikosti první dimenze, kde každý prvek je Color Vector, jehož šablonový parametr je dimenze o jedna nižší

(`vector<Color Vector<dimenze - 1>>`). Takto rekurzivně definovaná datová struktura má tedy podobu vícerozměrného pole, kde počet rozměrů je určen parametrem šablony.

Aby tato rekurzivní definice fungovala, musí systém obsahovat také její explicitní specializaci pro dimenzi 1. Tato specializace již obsahuje přímo standardní vektor hodnot (`vector<Color>`). Tímto je rekurze ukončena a lze tuto datovou strukturu korektně použít.

Přístup k hodnotě na daných souřadnicích probíhá tak, že si každá dimenze zjistí ze souřadnic (podle souřadnice jí příslušící), který prvek je žádán a rekurzivně zavolá přístup k hodnotě na daném prvku. Specializace pro dimenzi 1 pouze vrátí hodnotu na dané souřadnici a ta je vrácena přes všechny dimenze.

Pro potřeby systému bylo nutné třídu Color Vector rozšířit o metodu *recalculate* pro přepočet rozsahů jednotlivých dimenzí. Tento přepočet si zjistí poměr mezi novou a starou velikostí, pokud se rozsah zvětšuje, pak pouze nakopíruje jednu hodnotu do několika nových podle tohoto poměru. Pokud se rozsah zmenšuje, několik starých hodnot, jejichž počet závisí na poměru se průměruje do jedné nové hodnoty. Tento algoritmus je implementován rovněž rekurzivně a proto funguje i pro změnu rozsahů více-dimenzionálních přechodových funkcí.

2.5 Histogram

V systému existuje třída reprezentující histogram. Ten nám určuje, kolikrát se která hodnota vyskytuje v aktuálně zobrazovaných objemových datech. Je nutné, aby mohl být histogram také více-dimenzionální, proto obsahuje obdobnou datovou strukturu jako u přechodové funkce, která je popsána v kapitole 2.4.3. Jedná se také o rekurzivní šablonovanou třídu *Histogram*. Zde již však nebylo třeba funkce přepočítávání rozsahů a jednotlivé položky této struktury nejsou barvy, ale hodnoty histogramu.

Složitost použité datové struktury vyžaduje, aby byla již při vytváření histogramu inicializována na požadované rozsahy. Změna hodnot poté probíhá pomocí metody *set*. Třída histogramu neumožňuje přímý přístup k jednotlivým hodnotám. Získávání hodnot je realizováno metodou *get*, která vrací konstantní odkaz na požadovanou hodnotu. Aby bylo dosaženo konstantnosti histogramu po jeho vytvoření, je zde metoda *seal*, po jejímž zavolání již bude metoda set nefunkční. To také dává prostor pro případné zpracovávání hodnot histogramu po jeho vytvoření. V metodě seal pak může být například počítáno maximum, průměr a podobně.

2.6 Rozhraní pro GUI

Pracovní kopie je třída obalující přechodovou funkci, která řeší přepočítávání funkce ze zobrazených rozměrů, ve kterých se navrhuje, do rozsahů přechodové funkce. Uchovává si informace o aktuálním zoomu a používá je pro tento přepočet.

2.6.1 Přepočet

Přepočítávání probíhá lokálně, tzn. nepřepočítává se celá přechodová funkce při každé změně. Při změně bodu ve zobrazené navrhované funkci se tento bod zapíše postupně do intervalu jehož velikost odpovídá poměru rozsahu přechodové funkce ku zobrazovanému rozměru. Pokud je tento poměr menší než jedna, zapíše se pouze do jednoho bodu přechodové funkce. Tím je zajištěno, že je možné při vetším zoomu modifikovat přechodovou funkci v poměru 1:1 a jeden bod přechodové funkce se zobrazí na několik bodů zobrazovaného rozměru (viz. následující ukázka pseudo kódu). Při získávání hodnoty bodu přechodové funkce se aplikuje opačný přepočet, kterým se získá hodnota ve zobrazovaném rozsahu.

```
float pomer = rozsahFunkce/(zobrazovanyRozmer*zoom);
float indexZaklad = index*pomer + posunIndexu;
float indexPolomer = pomer/2;
float indexSpodni = indexZaklad - indexPolomer;
float indexHorni = indexZaklad + indexPolomer;
for(int i = indexSpodni; i < indexHorni ; ++i)
{
    if(i je v mezich rozsahu funkce)
    {
        data[i] = hodnota;
    }
}
```

Umístění těchto přepočtů do pracovní kopie umožňuje, že se painter ani modifier již nemusí starat o to, jestli je náhled zoomovaný. Přepočet je tedy v jedné třídě (lze ho v případě potřeby změnit pouze změnou jednoho zdrojového kódu) a umožňuje pracovat s přechodovou funkcí stejně při různých zobrazovaných rozměrech a při různých hodnotách zoomu.

Pracovní kopie si také uchovává histogram, dovoluje jeho zoom jako u přechodové funkce a navíc poskytuje rozhraní pro úpravu vydávání hodnot histogramu. Hodnoty histogramu jsou před vydáním, kromě přepočtu z rozsahu histogramu na zobrazovaný rozměr, také přepočítávány podle vzorce 2.1, aby je bylo možné zobrazit do intervalu $< 0 - 1 >$.

$$dispValue = 1 - e^{-\log_{logBase}(histValue)} \tag{2.1}$$

kde dispValue je zobrazovaná hodnotu v intervalu $< 0 - 1 >$, histValue je hodnota histogramu v daném bodě a logBase je základ logaritmu, který pracovní kopie dovoluje měnit a tak upravovat zobrazení histogramu.

2.6.2 Rozhraní

Rozhraní pracovní kopie tedy poskytuje pro účely komunikace s uživatelským rozhraním několik metod. Metody pro získání informace, jestli se přechodová funkce či histogram změnily. Dále pro úpravu základu logaritmu pro přepočet hodnot histogramu. Metody pro nastavení jednotlivých komponent barvy, kde se aplikuje přepočet funkce a pro získání barvy, kde se aplikuje opačný přepočet. Navíc lze získat barvy barevného modelu RGB, která se také přepočítává.

Možnost získání barvy barevného modelu RGB je zde proto, že painter nemá představu o tom, s jakým typem přechodové funkce pracuje (kromě její dimenze) a může chtít vykreslit výslednou barvu. To nelze aniž by věděl o jaký typ přechodové funkce se jedná, například RGB, HSV (Hue, Saturation, Value) a podobně.

Pro potřeby modifieru poskytuje rozhraní pracovní kopie také metodu pro zpřístupnění přechodové funkce přímo (zpřístupňuje nešablonované rozhraní přechodové funkce), modifier tedy má možnost, jak měnit přechodovou funkci přímo, aniž by se aplikoval přepočet zohledňující zobrazovaný rozměr a zoom.

Pracovní kopie je navržena, aby byla schopna plnit svou funkci i pro vícedimenzionální přechodové funkce.

2.7 Modifier

Modifier je část editoru, která řeší modifikaci přechodové funkce. *Abstract Modifier* je potomkem Qt třídy QWidget, tudíž se jedná o samostatnou grafickou komponentu, která dostává signály se vstupy uživatele (vstupy počítačové myši a klávesnice). Modifier obsahuje metody pro obsluhu těchto signálů a závisí na dané implementaci, jakým způsobem umožní přechodovou funkci modifikovat (např. free-hand kreslení křivky).

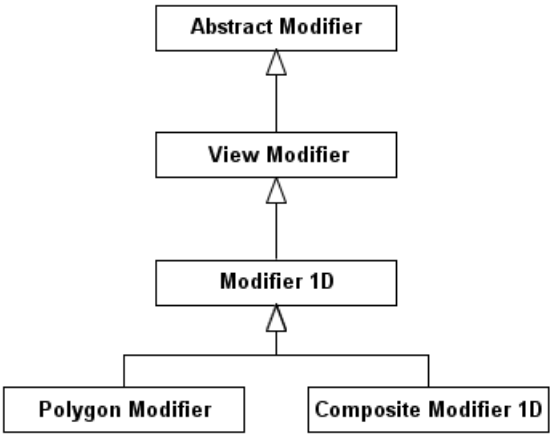
Modifier je jako QWidget přidáván do grafického rozhraní editoru. Navíc obsahuje metodu *getTools*, která vrací dokovací okno, kde je grafické rozhraní, které používá modifier navíc oproti tomu základnímu (dále pouze tools). Toto dokovací okno je připojeno k editoru a umožňuje tak rozšíření grafického rozhraní se zachováním jednotného vzhledu (základní + rozšířené v dokovacím okně). Data pro vykreslení modifieru jako grafické komponenty získává modifier od painteru. Jelikož může být vhodné k vykreslované přechodové funkci ještě něco doplnit (např. stupnici, náhled výsledné barvy, atd.) získává modifier oblast, která slouží pro modifikaci od painteru. Při změně velikosti tedy modifier informuje painter o tom, že se změnila velikost a poté si od painteru aktualizuje vstupní oblast.

Modifier obsahuje přechodovou funkci, kterou může používat přímo, pokud je to nutné, ale implicitně je z ní v systému vytvářena pracovní kopie a tu modifier používá.

V systému existuje hierarchie modifierů znázorněná obrázkem 2.2.

2.7.1 Abstract Modifier

Abstract Modifier je základ společný pro každý modifier, obsahuje rozhraní pro komunikaci s editorem, řeší základ ukládání a načítání, obsahuje pracovní kopii a řeší základní komunikaci mezi modifierem a painterem. Navíc je zde implicitní prázdná implementace metod zpracovávajících vstupní signály. Potomek tedy



Obrázek 2.2: Hierarchie modifierů

zpracuje signály, které potřebuje a nemusí řešit ostatní. Abstraktní Modifier obsahuje také implicitní implementaci metody pro získání dokovacího okna s tools, která vrací hodnotu NULL.

2.7.2 View Modifier

Zobrazení přechodové funkce je možné přibližovat – funkce zoom. Základní zoomování je implementováno v metodách abstraktní třídy *View Modifier*. Tím je zajištěno jednotné ovládání a chování zoomu. Tato třída také definuje svoje tools, které vrací v dokovacím okně v metodě getTools.

2.7.3 Modifier 1D

Modifier 1D je základní implementace jedno-dimenzionálního modifieru. Jedná se o free-hand kreslení křivky, která je přepočítávána do přechodové funkce. Tato třída definuje vlastní tools pro free-hand kreslení (viz. obrázek A.7). V metodě getTools pak do dokovacího okna vloží tools definované ve View Mofifier a vlastní tools. Tím je zajištěno, že je dostupné grafické rozhraní pro všechny funkce včetně toho definovaného v předkovi.

2.7.4 Polygon Modifier

Polygon Modifier rozšiřuje implementaci Modifier 1D tím, že při free-hand kreslení vykresluje místo příslušného bodu polygon s určitou velikostí základen. Definuje také tools, které umožňují nastavovat velikost základen a v metodě getTools do dokovacího okna vkládá tools definované ve View Modifier, tools definované v Modifier 1D a svoje vlastní tools. Ukázku navržené přechodové funkce pomocí tohoto modifieru můžeme vidět na obrázku A.8.

2.7.5 Composite Modifier 1D

Composite Modifier 1D je modifier, který ke své funkci využívá paletu. Umožňuje skládat několik přechodových funkcí do jedné výsledné, tu zobrazit a používat ji

na modifikaci dat. Tento modifier definuje vlastní tools (viz. obrázek A.9), které slouží k nastavení intervalu časovače aktualizujícího přechodovou funkci. Tools obsahuje také grafické rozhraní pro zobrazení seznamu skládaných přechodových funkcí a také pro vyvolání dialogu, kde lze nastavit, které přechodové funkce aktuálně otevřené v paletě budou použity pro skládání.

Aktualizace

Obsahuje časovač, který kontroluje jestli se skládané přechodové funkce změnily a případně aktualizuje složenou přechodovou funkci. Tento postup nám umožní zachovat plynulosti programu a současně sledovat změny skládaných funkcí bez nutnosti manuální aktualizace. Hodnotu periody časovače lze navíc měnit v tools a tím přizpůsobit četnost aktualizací například počtu skládaných funkcí či výkonnosti počítače.

Přidávání/odebírání

Přídávací dialog umožňuje zobrazit přechodové funkce v paletě jako seznam jmen, ale lze také zapnout náhledy k těmto jménům. Modifier umožňuje skládat pouze přechodové funkce s jemu odpovídající dimenzí a pouze ty, které nejsou skládané (mohlo by dojít k zacyklení aktualizace při přidání sebe sama).

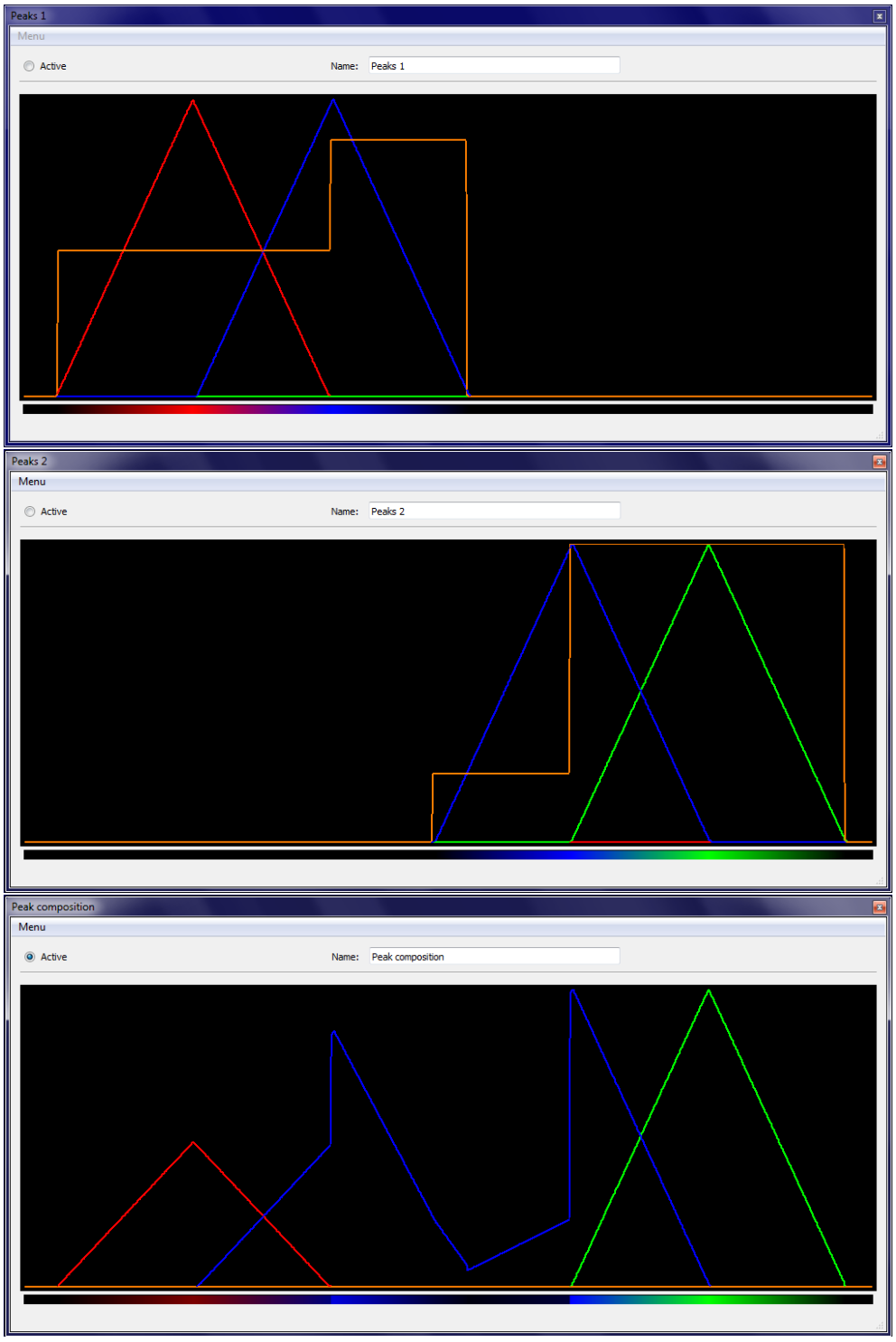
Výpočet

Výpočet skládání přechodových funkcí je řešen ve virtuální metodě, lze tedy změnit tento přepočet poděděním od Composite Modifier 1D a implementovat jiný algoritmus pro tento výpočet. Defaultní implementace počítá složenou přechodovou funkci jako součet všech přechodových funkcí ovlivněný jejich průhledností. Průhlednost nabývá reálných hodnot z intervalu $< 0 - 1 >$ a těmito hodnotami se násobí ostatní komponenty barvy před jejich přičtením. Průhlednost výsledné skládané přechodové funkce definuje uživatel free-hand kreslením křivky stejně jako u Modifier 1D.

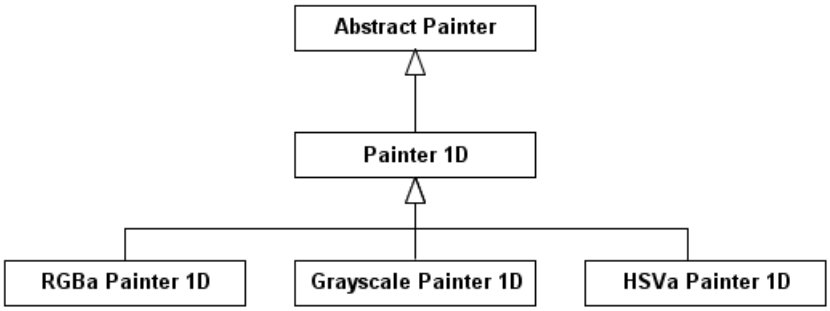
Příklad tohoto skládání můžeme vidět na obrázku 2.3. Nejprve jsme navrhli dvě přechodové funkce s barevnými špičkami a různými hodnotami průhlednosti. Následuje jejich kompozice kde můžeme vidět výsledek složení těchto dvou funkcí podle výše popsaného postupu.

2.7.6 Použití painteru

Jelikož je painter součástí modifieru a rozhraní painterů různých dimenzí budou typicky různá a přitom bude třeba, aby modifier tato rozhraní používal, může si modifier v konstruktoru vyžádat konkrétního potomka či potomka, který je zároveň předkem například pro danou dimenzi. Tuto možnost lze využít, pokud je modifier korektně registrován. Registrací je vynuceno, že vytvářecí metoda (viz. kapitola 2.11) painteru vrátí daného potomka a lze bez problémů použit standardní operátor dynamic_cast pro přetypování na konkrétního potomka.



Obrázek 2.3: Skládání funkcí v kompozici



Obrázek 2.4: Hierarchie painterů

2.8 Painter

Painter řeší vykreslování přechodové funkce. Obsahuje metodu *getView*, která dostane jako parametr pracovní kopii a vrací buffer s vykreslenou přechodovou funkcí. Modifier tento buffer vykreslí v metodě obsluhující signál pro vykreslení. Pracovní kopie obsahuje dotazovací metody, aby bylo možné zjistit, jestli se přechodová funkce či histogram změnily a je tedy možné toto využít pro aktualizaci pouze části, která se změnila.

Abstraktní předek, třída *Abstract Painter*, definuje pouze rozhraní pro ukládání, načítání a základní komunikaci s modifierem. Je zde metoda *setArea* pro nastavení oblasti, která slouží pro vykreslování, to umožní nastavit velikost bufferu a také to umožní spočítat oblast, která bude použita pro vstup. Metoda *getInputArea* umožňuje získat oblast pro vstup a modifier ji použije, aby mohl akceptovat pouze vstupy týkající se dané oblasti.

Metoda getView umožňuje práci s painterem nezávisle na jeho implementaci a vždy poskytne modifieru buffer, který je třeba vykreslit. Jednotliví potomci pak mohou implementovat různé způsoby vykreslování a budou vykreslovat vždy do bufferu, který poskytnou při volání metody getView.

V systému existuje hierarchie painterů znázorněná obrázkem 2.4.

2.8.1 Abstract Painter

Abstract Painter má metody pro komunikaci s modifierem čistě virtuální (neimplementuje je) a obsahuje implicitní prázdnou implementaci metod pro ukládání a načítání.

2.8.2 Painter 1D

Painter 1D obsahuje navíc metody *updateFunctionView*, *updateHistogramView* a *updateBottomColorBarView* a ke každé této metodě si uchovává buffer.

Metoda updateFunctionView vykresluje pouze přechodovou funkci do svého bufferu. Metoda updateHistogramView vykresluje histogram do svého bufferu a metoda updateBottomColorBarView vykresluje do svého bufferu náhled výsledné barvy přechodové funkce na příslušném místě.

Painter 1D vykresluje přechodovou funkci do obdélníku jako křivky jednotlivých komponent barev, kde osu y tvoří hodnota komponenty z intervalu $< 0 - 1 >$

a osu x tvoří hodnoty příslušející datům. Buffery se před kreslením do nich vyplní průhlednou barvou a to umožní jejich jednoduché vykreslení přes sebe v metodě getView. Před vykreslením se daný buffer aktualizuje, pokud je třeba.

Painter 1D také obsahuje barvy, které se používají pro vykreslení jednotlivých komponent barvy.

2.8.3 RGBa Painter 1D

RGBa Painter 1D pouze specifikuje barvy, které jsou použity pro vykreslování jednotlivých komponent barev a veškerá funkčnost je poděděna od Painter 1D. Ukázku můžeme vidět na obrázku A.7.

2.8.4 Grayscale Painter 1D

Grayscale Painter 1D specifikuje barvu, která je použita pro vykreslování, ale také má svou implementaci metody updateFunctionView, jelikož vykresluje pouze jednu komponentu a to jako odstín šedé (všechny komponenty mají stejnou hodnotu).

2.8.5 HSVa Painter 1D

HSVa Painter 1D specifikuje barvy, které jsou použity pro vykreslování a obsahuje navíc metodu *updateSideBar* a jí příslušící buffer, do kterého vykresluje boční náhled, který poskytuje uživateli představu, jaká barva přísluší na ose y komponentě hue. Implementace metody getView pak aktualizuje buffer pro boční náhled je-li třeba a volá metodu getView předka. Do výsledného bufferu vykreslí buffer vykreslený předkem a buffer s bočním náhledem. Ukázku můžeme vidět na obrázku A.8.

2.9 Editor

Základ editoru je abstraktní předek *Editor*. Ten poskytuje rozhraní pro komunikaci s paletou (viz. kapitola 2.10), obsahuje informaci o svoji struktuře, atributy, jméno, unikátní identifikátor, řeší ukládání a obsahuje modifier.

Rozhraní umožňuje informovat paletu, že je editor aktivován a že je zavírán. To je důležité pro její aktualizaci. Také zprostředkovává předání přechodové funkce, ze které vytváří konstantní funkci a předá ji paletě.

Pro potřebu ukládání si editor uchovává název souboru a časovou známku posledního uložení. Pokud je název souboru prázdný, nabídne editor uživateli při uložení dialog, kde uživatel určí, do kterého souboru chce ukládat. Editor také poskytuje funkci „uložit jako“, ta vyvolá vždy otevření dialogu pro uložení. Editor umožňuje uložit celý editor, kdy uloží přechodovou funkci včetně struktury editoru (z jakých částí se skládá) a dalších dat a nastavení všech částí, nebo uložit pouze přechodovou funkci, kdy uloží informaci o dimenzi a typu přechodové funkce a samotnou přechodovou funkci.

Editor také umožňuje načíst přechodovou funkci. Při načítání si editor ověří, jestli odpovídá dimenze a typ a pokud ano, pak načte přechodovou funkci ze souboru.

2.9.1 Grafické rozhraní

Veškerou funkci editoru řeší abstraktní předek Editor. Potomek *Editor GUI*, který se v systému používá, rozšiřuje funkčnost o základní grafické uživatelské rozhraní popsané spolu s ukázkou v kapitole chap:editorGUI. Tím je zajištěno jednotné základní grafické rozhraní pro všechny editory v systému a také lze díky tomu snadno změnit toto jednotné grafické rozhraní změnou používaného potomka. Není nutné zasahovat do systému, stačí pouze vytvořit potomka s novým grafickým rozhraním, upravit vytvořující metodu creatoru a všechny již existující editory budou fungovat s novým jednotným grafickým rozhraním.

Editor je potomkem Qt třídy QMainWindow a je připojován k aplikaci jako dokovací okno.

2.10 Paleta

Paleta zastřešuje celý systém. Spravuje editory, umožňuje přidat nový editor, uchovává si všechny otevřené editory. Řeší komunikaci s aplikací. Zpracovává veškeré změny a distribuuje je editorům. Umožňuje nastavení struktury dat pro přechodovou funkci a nastavení histogramu. Obsahuje také metody, které se vyskytují v rozhraní editoru. Ty pak převolávají odpovídající metody aktivního editoru. Tím se paleta chová jako by byla současně aktivním editorem a umožňuje tak získat přechodovou funkci pro modifikaci dat.

2.10.1 Chování

Paleta při vyžádání přechodové funkce vrací konstantní funkci a pokud není žádný editor aktivní, pak porovnání konstantní funkce s hodnotou NULL vrací hodnotu true.

Aktivním editorem je implicitně naposledy přidaný přípustný editor. Uživatel může změnit aktivní editor v grafickém rozhraní. Pokud není otevřený žádný přípustný editor, není žádný editor aktivní.

Jelikož systém podporuje také více-dimenzionální editory a je možné mít současně otevřené editory s různou dimenzí, musí paleta určovat, které editory jsou přípustné editory a které ne. Editor je přípustný editor, pokud jeho dimenze odpovídá dimenzi struktury dat. Také veškeré změny nastavení, které paleta distribuuje na editory, předává pouze přípustným editorům.

Struktura dat určuje parametry, které musí mít přechodová funkce, aby ji aplikace mohla použít pro modifikaci otevřených dat.

Vytváření nově přidaného editoru je delegováno na creator (viz. kapitola 2.11), který je součástí palety.

2.10.2 Grafické rozhraní

Paleta obsahuje grafické uživatelské rozhraní, které zobrazuje seznam všech otevřených editorů. Tento seznam může být zobrazen jako seznam jmen, ale lze také povolit zobrazení náhledů, které poskytují uživateli představu o tom, jak by přechodová funkce navržená v daném editoru vypadala při aplikaci na otevřená data. Náhledy jsou k dispozici pouze u přípustných editorů. Aktivní editor, přípustné

editory a ostatní editory jsou v grafickém rozhraní viditelně odlišeny. Kliknutím na přípustný editor v seznamu se z něj stane aktivní editor. Grafické rozhraní také umožňuje uživateli přidat nový editor do palety.

Náhledy jsou pravidelně aktualizovány, aby měl uživatel představu, jak bude použití přechodové funkce vypadat, pokud edituje přechodovou funkci editoru, který není aktivním editorem.

Paleta je potomkem Qt třídy QMainWindow a k aplikaci je připojována jako dokovací okno. Její vzhled můžeme vidět v kapitole A.4.

2.11 Creator

Creator je součástí palety a řeší vytváření editorů. Jelikož se editory skládají z několika částí a nový typ editoru může vzniknout pouze změnou jedné části, tzn. použitím jiného potomka dané části, je nutné na nějakém místě vytvořit konkrétního potomka, který pak bude již dále používán jako předek. K tomu slouží creator, který obsahuje vytvářecí metodu pro každou část editoru.

Protože vytváření editoru závisí na volbě uživatele aplikace, slouží creator zároveň jako dialog, který uživateli nabídne všechny dostupné druhy editorů a volbu potom použije při vytváření editoru. Příklady dialogového okna pro přidání editoru do palety můžeme vidět na obrázcích A.4, A.5 a A.6.

2.11.1 Registrace

Aby bylo možné nabídnout uživateli aplikace všechny dostupné editory a všechny jejich části, je nutné, aby bylo vše registrováno. Pro tento účel je vyhrazeno několik hlavičkových souborů, konkrétně TFDimensions.h, TFModifiers.h, TFFunctions.h, TFPainters.h a TFPredefined.h.

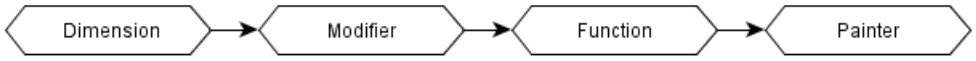
Každý z těchto souborů obsahuje výčet, dvě specializace šablonované konverzní metody a jednu statickou *get* metodu. Pokud je vytvořen nový typ jedné části editoru, je nutné mu přidělit identifikátor v podobě příslušného výčtu. Dále je nutné doplnit konverzní specializace, které konvertují dané výčty na řetězce se jmény zobrazovanými v dialogovém okně (viz. kapitola A.5) a opačně. Tím je zajištěno, že se nový typ korektně zobrazí v nabídce dialogu. Nakonec je nutné upravit get metodu, která vrací dostupné typy v podobě seznamu výčtů. Ukázka registrace je popsána v kapitole 2.13.

Registrační soubory

TFDimensions.h umožňuje registrovat novou dimenzi přechodové funkce. To znamená oznámit systému, že existuje editor, ve kterém lze přechodovou funkci s touto dimenzí navrhnout. Dimenze je velmi důležitá, je základem celého editoru, protože jednotlivé části, které pracují s různou dimenzí typicky nemohou pracovat spolu. Get metoda zde vrací všechny dostupné dimenze.

TFModifiers.h umožňuje registrovat nový typ modifieru. Get metoda zde dostane jako parametr dimenzi a podle ní se vrátí seznam těch typů modifieru, které pracují s touto dimenzí.

TFFunctions.h umožňuje registrovat nový typ přechodové funkce. Get metoda zde dostane jako parametr modifier (ten je již závislý na dimenzi, takže je známá



Obrázek 2.5: Závislost registrovaných částí

i použitá dimenze) a podle něj vrátí seznam typů přechodové funkce, které jsou vhodné pro tento modifier.

TFPainters.h umožňuje registrovat nový typ painteru. Get metoda zde dostane jako parametr typ přechodové funkce (ten rovněž určuje dimenzi) a podle něj vrátí seznam typů painteru, které jsou pro tuto přechodovou funkci vhodné.

TFPredefined.h umožňuje registrovat předefinovaný editor. Get metoda zde vrací seznam všech dostupných typů předdefinovaného editoru. Navíc oproti ostatním registračním souborům obsahuje tento druhou get metodu která podle typu předdefinovaného editoru vrací strukturu, kde jsou uloženy typy jednotlivých částí editoru.

Registrační úpravy

Závislost registrovaných částí je znázorněna obrázkem 2.5 (část je závislá na té nejblíže vlevo od ní). Pro korektní registraci je nutné také upravit get metodu bezprostředně následující části a tím označit, se kterými typy umí registrovaná část pracovat.

Vytvářecí metody creatoru, kde je jedna pro každou část editoru, je tedy nutné doplnit o vytvoření příslušného potomka dané části podle vybraného typu. Tyto metody umožňují také definovat atributy, které bude obsahovat výsledný editor. Úprava příslušné metody je poslední krok registrace dané části.

2.11.2 Módy

Creator pracuje ve třech módech. První je vytvoření uživatelem definovaného editoru, kde dialog postupně nabízí dostupné typy části podle předchozí uživatelovy volby. Druhý je výběr předdefinovaného editoru. Po jeho zvolení creator rovnou vytvoří daný editor. Poslední mód je načtení editoru ze souboru, v tomto módu otevře creator dialog pro výběr souboru a po zvolení souboru uživatelem vytvoří a načte editor z tohoto souboru.

2.11.3 Přínos

Soustředění vytvářecích metod do creatoru spolu s výše popsanou registrací umožňuje vytvoření a registraci nového typu části editoru aniž by bylo nutné zasahovat do systému. Registrace zajistí, že dialog již nabídne tento nový typ, korektně ho vytvoří a systém s ním bude pracovat. Stejně snadno lze přidávat předdefinované typy editorů.

2.11.4 Grafické rozhraní

Creator je potomkem Qt třídy Qdialog a chová se tedy jako dialogové okno, do kterého se dynamicky, v závislosti na volbě uživatele aplikace a registračních

souborech, generují možnosti pro přidání nového editoru do palety. Vzhled tohoto dialogového okna můžeme vidět v kapitole A.5

2.12 Ukládání a načítání

V systému je navrženo rozhraní pro ukládání a rozhraní pro načítání, které umožňuje používat writer a reader jednotným způsobem v celém systému. Rozhraní odpovídá ukládání a načítání dat v XML (Extensible Markup Language) formátu.

Samotná implementace ukládání či načítání nemusí nutně odpovídat XML formátu, může být libovolná a lze ji snadno změnit vytvořením potomka *XML Writer Interface* a *Xml Reader Interface*. Kód je třeba potom upravit pouze v místech, kde se vytváří writer a reader. Celý systém již potom používá jednotné rozhranípro ukládání a načítání a bude proto pracovat korektně s novou implementací bez nutnosti měnit další kód.

Rozhraní pro ukládání i pro načítání obsahuje metody *begin* a *end*. Metoda begin dostane název souboru, který je použit implementace může v této metodě otevřít soubor a připravit vše potřebné pro ukládání/načítání. Metoda vrací hodnotu true, pokud je možné začít ukládat/načítat a hodnotu false, pokud došlo k nějaké chybě. Metoda end slouží pro oznámení writeru/readeru, že již bylo ukládání/načítání ukončeno a ten může provést kroky potřebné pro korektní ukončení ukládání/načítání (typicky minimálně zavření souboru).

Rozhraní také obsahuje metody pro signalizaci chyb. Metoda *error* vrací hodnotu true, pokud nastala chyba a metoda *errorMessage* vrací řetězec obsahující textový popis chyby.

V současné verzi je používána implementace využívající prostředky knihovny Qt pro práci s XML soubory a to třídy QXmlStreamWriter a QXmlStreamReader.

2.12.1 XML struktura

Jak bylo již zmíněno v kapitole 2.9, je možné uložit buďto celý editor nebo pouze přechodovou funkci. Tyto dva způsoby uložení se liší příponou souboru a také jeho strukturou.

Editor

Soubor s uloženým editorem má příponu tfe a má následující strukturu.

<?**xml version**="1.0" **encoding**="UTF–8"?>

<!**DOCTYPE** TransferFunctionEditorFile>

<**Editor Name**="jméno_editoru"
Predefined="typ_předdefinované_struktury"
Dimension="typ_dimenze"
Function="typ_přechodové_funkce"
Painter="typ_painteru"
Modifier="typ_modifieru">

<!−−zde je místo pro uložení dodatečných nastavení painteru>

<**WorkCopy MaxZoom**="maximální_přiblížení"
HistLogBase="logBase 2.1">

<**ZoomProperty Dimension**="číslo_dimenze"
Zoom="hodnota_přiblížení"
Center="posunIndexu refcode:prepocet"/>
<!−−počet elementů ZoomProperty odpovídá dimenzi přechodové funkce>

</**WorkCopy**>
<**Function Dimensions**="dimenze"
Dimension#="rozsah_dimenze_#">
<!−−počet atributů Dimension# v elementu Function jeroven dimenzi přechodové funkce
a # označuje číslo dimenze, jejíž rozsah následuje>

<**Color Coordinates**="souřadnice"
Component1="hodnota"
Component2="hodnota"
Component3="hodnota"
Alpha="hodnota"/>
<!−−souřadnice mají tvar [dimenze1,dimenze2,...]>
<!−−počet elementů Color odpovídá počtu položek v přechodové funkci>

</**Function**>
<!−−zde je místo pro uložení dodatečných nastavení přechodové funkce>
<!−−zde je místo pro uložení dodatečných nastavení modifieru>

</**Editor**>

Přechodová funkce

Soubor s uloženou přechodovou funkcí má příponu tf a má následující strukturu.

<?**xml version**="1.0" **encoding**="UTF-8"?>

<!**DOCTYPE** TransferFunctionFile>

<**FunctionInfo Name**="jméno_přechodové_funkce"
Dimension="typ_dimenze"
Function="typ_přechodové_funkce">

<**Function Dimensions**="dimenze"
Dimension#="rozsah_dimenze_#">
<!−−počet atributů Dimension# v elementu Function jeroven dimenzi přechodové funkce a # označuje
číslo dimenze, jejíž rozsah následuje>

<**Color Coordinates**="souřadnice"
Component1="hodnota"
Component2="hodnota"
Component3="hodnota"
Alpha="hodnota"/>

<!−−souřadnice mají tvar [dimenze1,dimenze2,...]>
<!−−počet elementů Color odpovídá počtu položek v přechodové funkci>

</**Function**>

<!—zde je místo pro uložení dodatečných nastavení přechodové funkce>

</**FunctionInfo**>

2.13 Návod na rozšíření

Příklad rozšíření: Přidání nového typu přechodové funkce pro barevný model CMYK (Cyan, Magenta, Yellow, Key)

Kroky pro rozšíření systému:

1. Vytvoření nové třídy (v tomto příkladě přechodové funkce)
   - Vytvoření nové šablonované třídy *CMYk Function* jako potomka třídy Abstract Function, kde parametr šablony bude předán předkovi.
   - Implementace metod getRGBfColor a setRGBfColor – například je možné použít Qt třídu QColor, která umožňuje inicializovat barvu podle barevného modelu CMYK a poté získat barvu odpovídající barevnému modelu RGB.
2. Registrace: soubor TFFunctions.h
   - Přidání *FunctionCMYk* do výčtu *Function*
   - Doplnění konverzní metody *convert<Function, std::string>* o case větev výčtu FunctionCMYk a vracení řetězce ČMYK functionž této case větve.
   - Doplnění konverzní metody *convert<std::string, Function>* o case větev řetězce ČMYK functionä vracení výčtu FunctionCMYk z této case větve.
   - Doplnění metody *getAllowedFunctions* o vložení FunctionCMYk do seznamu pro modifiery, které budou moct přechodovou funkci používat.
3. Registrace: soubor TFPainters.h
   - Doplnění metody *getAllowedPainters* o case větev výčtu FunctionCMYk, kde budou vloženy do seznamu všechny paintery, které budou povoleny pro tuto přechodovou funkci. (Při současném stavu je možné například povolit PainterRGBa1D, případně lze doplnit systém o painter odpovídající přímo CMYK. Také se nabízí možnost registrovat CMYK painter s tím, že se místo vytvoření nové třídy pouze v case větvi vytvořující metody pro paintery vytvoří instance RGBa Painter a v konstruktoru se jí nastaví barvy, které by byly vhodné pro barevný model CMYK.)

4. Registrace: soubor TFCreator.h

    - Doplnění metody *createFunction_* o case větev výčtu FunctionCMYk, kde bude vrácena instance CMYk Function

Kroky 2., 3. a 4. představují kompletní korektní registraci nového typu přechodové funkce do systému.

3. Experimenty

V této kapitole si ukážeme možnosti naší implementace a také si zkusíme potvrdit některá tvrzení uvedená v literatuře, ze které jsme čerpali.

Každý experiment bude obsahovat tři části:

- popis experimentu
- výsledek experimentu
- zhodnocení výsledku

Výsledkem experimentu je jeden či více obrázků aplikace s navrženou přechodovou funkcí a zobrazenými objemovými daty po použití této funkce. Na všech obrázcích (naší aplikace) výsledků budou přechodové funkce založené na barevném modelu RGB. Budou tedy obsahovat tyto čtyři křivky definující výslednou funkci:

- červená – určuje červenou komponentu barvy
- zelená – určuje zelenou komponentu barvy
- modrá – určuje modrou komponentu barvy
- oranžová – určuje viditelnost/průhlednost

Na všech těchto obrázcích bude také zobrazen šedou plochou histogram zobrazovaných dat.

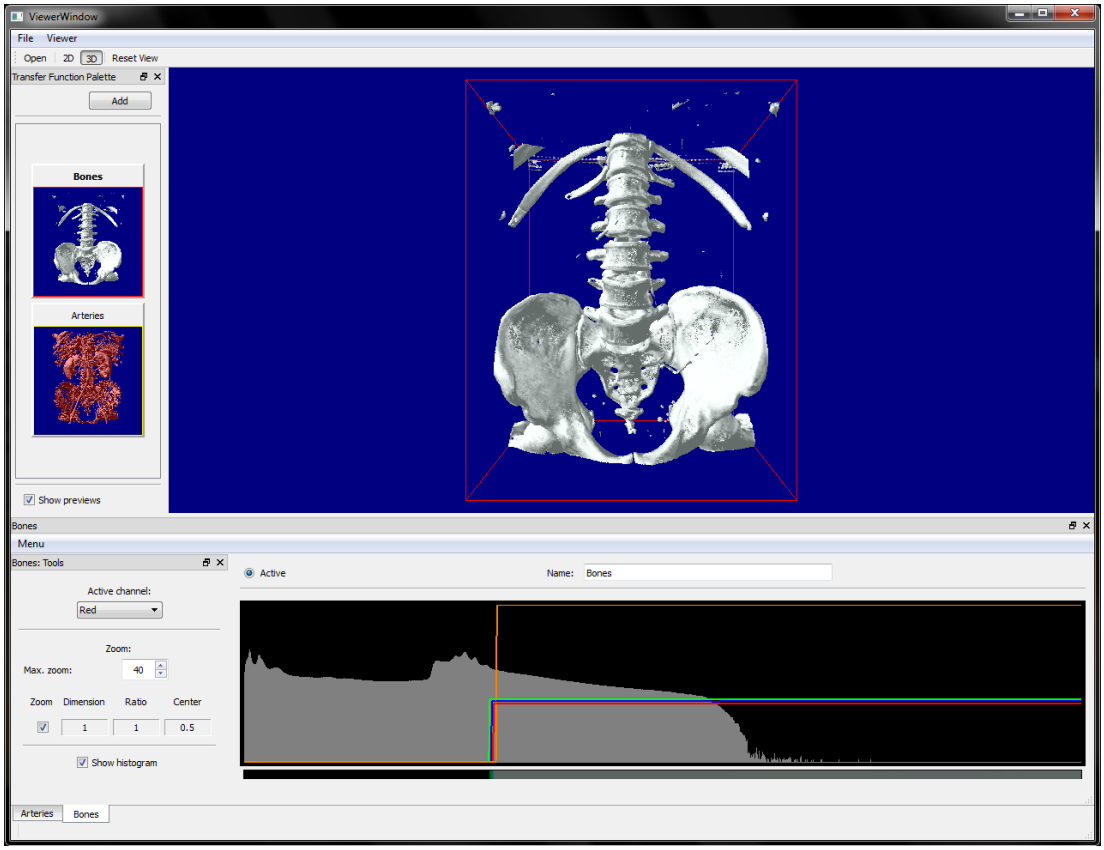
3.1 Prostorová závislost

3.1.1 Popis experimentu

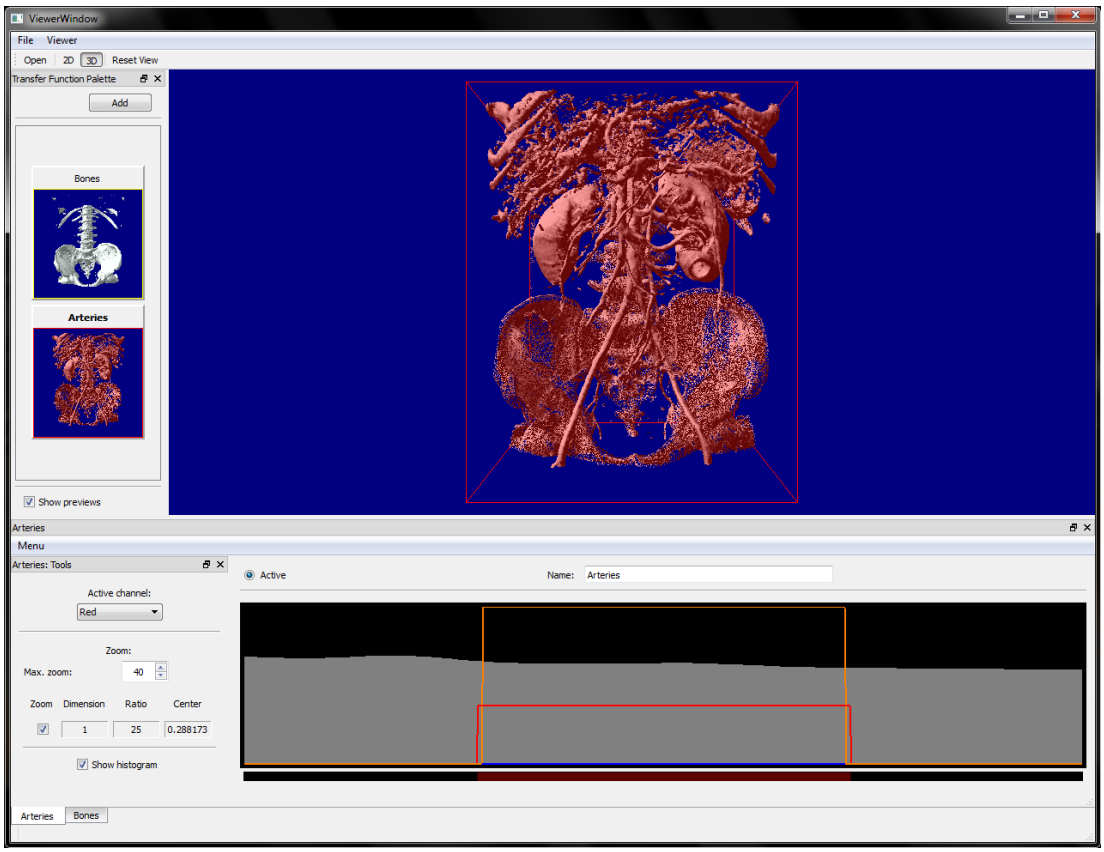
V tomto experimentu se zaměříme na prostorovou závislost jedno-dimenzionálních přechodových funkcí uvedenou v [5]. Jedná se o problém, kdy oblast kterou chceme zvýraznit obsahuje hodnoty vyskytující se také v jiných oblastech dat. Jednorozměrná přechodová funkce tedy zvýrazní i tyto oblasti. Kvůli této nevýhodě jsou používány více-dimenzionální přechodové funkce. My si vyzkoušíme navrhnout funkce pro znázornění pouze jednoho druhu tkáně (napodobení segmentace) a zhodnotíme výsledek.

3.1.2 Zhodnocení výsledku

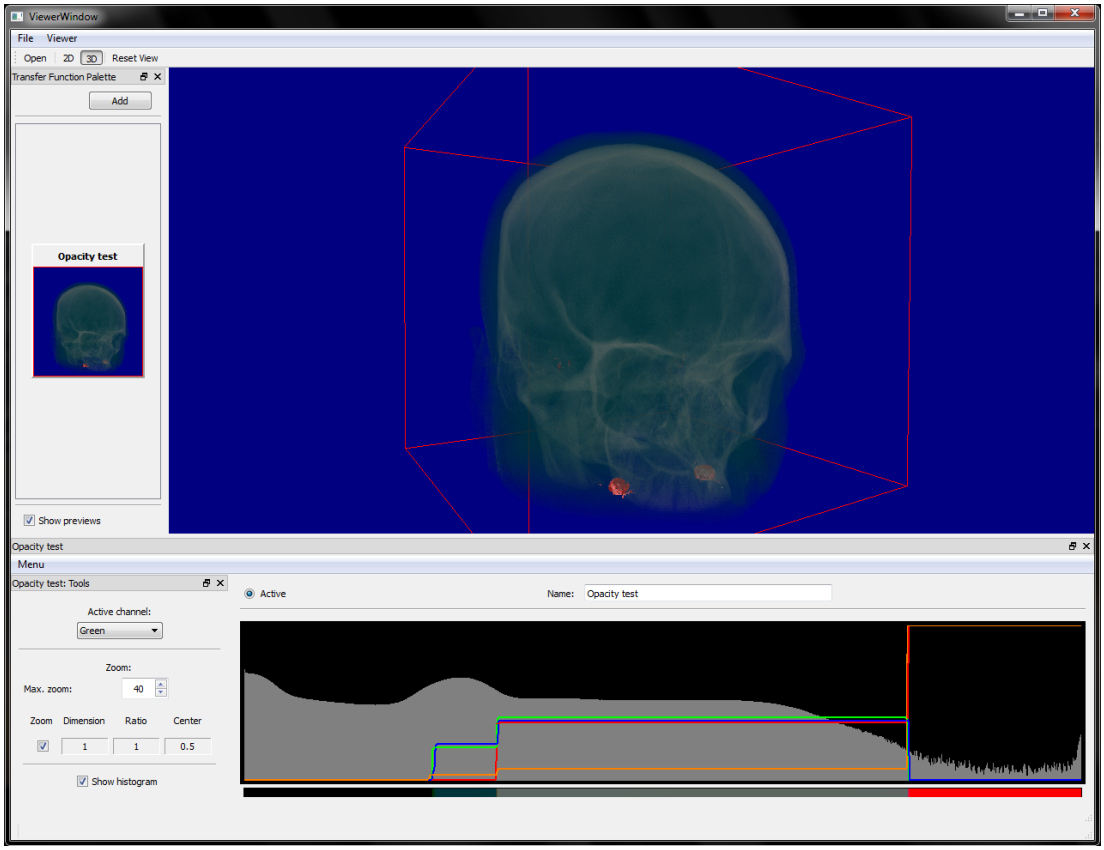
V první části experimentu (obrázek 3.1) jsme zkoušeli zobrazení pouze kostí v šedé barvě. V tomto případě, kromě několika malých artefaktů, které však výsledek nijak významně neovlivňují, se prostorová závislost neprojevila. V druhé části experimentu (obrázek 3.2) jsme se pokusili zobrazit pouze artérie červenou barvou. Zde se však prostorová závislost jedno-dimenzionálních funkcí projevila poměrně výrazně a to i při návrhu s 25-ti násobným přiblížením. Jak můžeme vidět na obrázku 3.2, kromě artérií lze také vidět částečně kosti, což dosti narušuje pohled na námi požadovaný druh tkáně.



Obrázek 3.1: Prostorová závislost - kosti



Obrázek 3.2: Prostorová závislost - arterie



Obrázek 3.3: Třívrstvé obarvení

3.2 Využití logaritmické stupnice

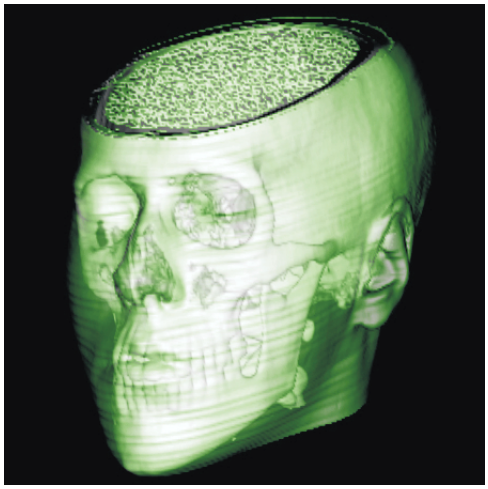
3.2.1 Popis experimentu

Publikace [7] pojednává o výhodě logaritmické stupnici pro navrhování viditelnosti/průhlednosti zobrazovaných dat. V tomto experimentu si vyzkoušíme navrhnout přechodovou funkci náročnější na průhlednost. Vyzkoušíme si postupně zprůhlednit dvě oblasti hodnot zobrazovaných dat, abych mohli i přes tyto vrstvy pozorovat třetí plně viditelnou oblast a zhodnotíme výsledek.

3.2.2 Zhodnocení výsledku

Jak můžeme vidět ve výsledku (obrázek 3.3), navrhli jsme přechodovou funkci, která obarvuje zobrazovaná data ve třech vrstvách. První tvoří měkká tkáň, obarvená modro-zelenou barvou a téměř průhledná. Druhá vrstva jsou kosti obarvené bílou barvou a zobrazené o něco méně průhledně než měkkou tkáň. Třetí vrstvu tvoří oblast nejvyšších hodnot CT, kterou jsme obarvili červeně a je plně viditelná. Tyto vysoké hodnoty náleží látkám s vysokou hustotou a bude se nejspíše jednat o zubní plomby.

Křivka určující viditelnost/průhlednost je pro plně viditelnou oblast na svém maximu. Aby bylo možné vidět třetí vrstvu, musí mít první dvě vrstvy velmi nízké hodnoty viditelnosti. Rozdíl viditelnosti mezi první a druhou vrstvou je opravdu malý a obě hodnoty se nacházejí ve velmi malém rozsahu ve srovnání s třetí vrstvou a to činí návrh komplikovanější.



Obrázek 3.4: Napodobovaný obrázek [8]

Obrázek 3.5: Přechodová funkce uvedená u napodobovaného obrázku [8]

Použití logaritmické stupnice pro křivku viditelnosti/průhlednosti by tedy nejspíše usnadnilo návrh podobných přechodových funkcí. Vyvstává však otázka, jestli by to v současné implementaci nezpůsobilo zkomplikování uživatelského rozhraní, jelikož způsob návrhu viditelnosti/průhlednosti je shodný se způsobem návrhu barev. Odlišné stupnice pro tyto křivky by vedly ke zmatení a zavedení logaritmické stupnice pro barvy je nevhodné.

3.3 Napodobení obrázku

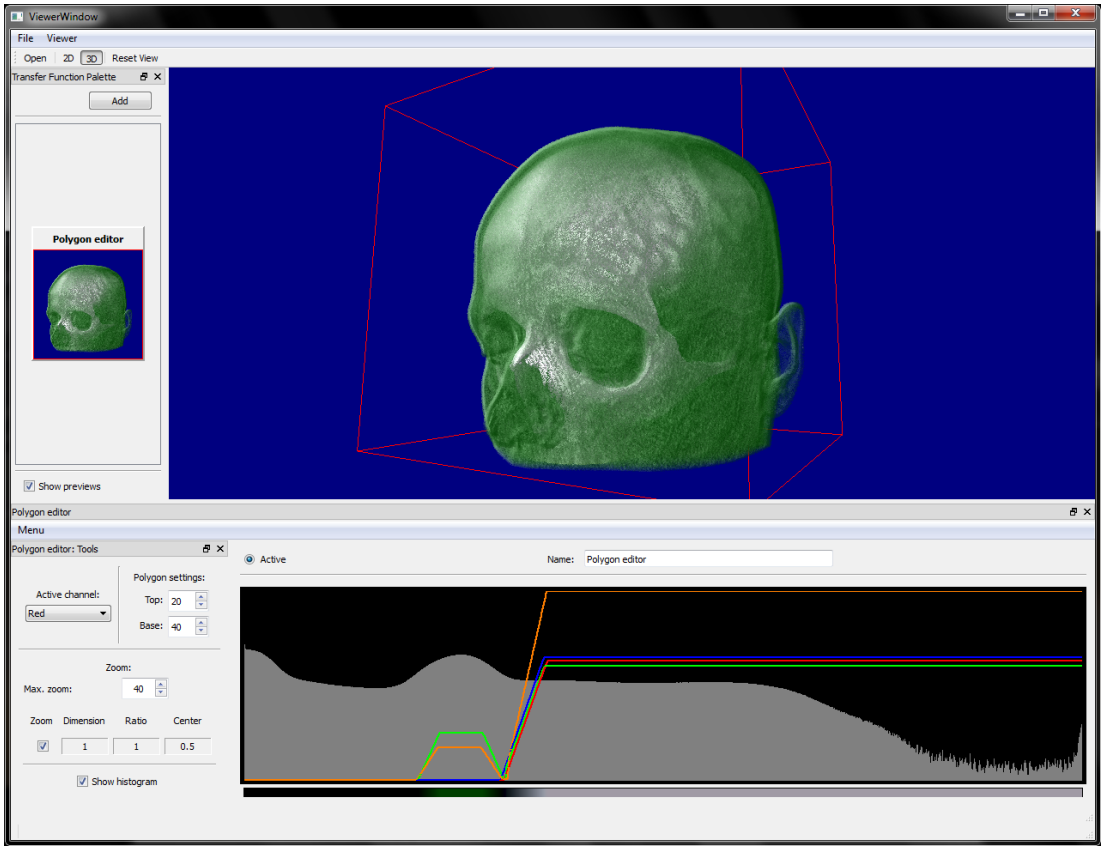
3.3.1 Popis experimentu

V literatuře se vyskytuje mnoho obrázků objemových dat obarvených přechodovou funkcí. V tomto experimentu jsme se rozhodli navrhnout přechodovou funkci tak, abychom jeden takový obrázek napodobili. Na závěr zhodnotíme výsledek.

3.3.2 Zhodnocení výsledku

Na obrázku 3.4 je obrázek, který jsme se rozhodli napodobit. Jedná se o obrázek z publikace [8]. U tohoto obrázku byla také zobrazena přechodová funkce, kterou můžeme vidět na obrázku 3.5. Námi navrženou funkci společně se zobrazenými daty, na které byla funkce aplikována můžeme vidět na obrázku 3.6.

Návrh přechodové funkce napodobující cílový obrázek byl jednoduchý, stačilo jen několik „pokus-omyl“ kroků a získali jsme podobný výsledek (výsledek se bude vždy lišit kvůli použití rozdílných zobrazovacích enginů a různých objemových



Obrázek 3.6: Výsledek napodobení obrázku

dat). Zajímavé však bylo, že se námi navržená přechodová funkce poměrně dost lišila od té uvedené v publikaci [8]. Důvod této odlišnosti bude nejspíše pouze ilustrativní záměr přechodové funkce 3.5.

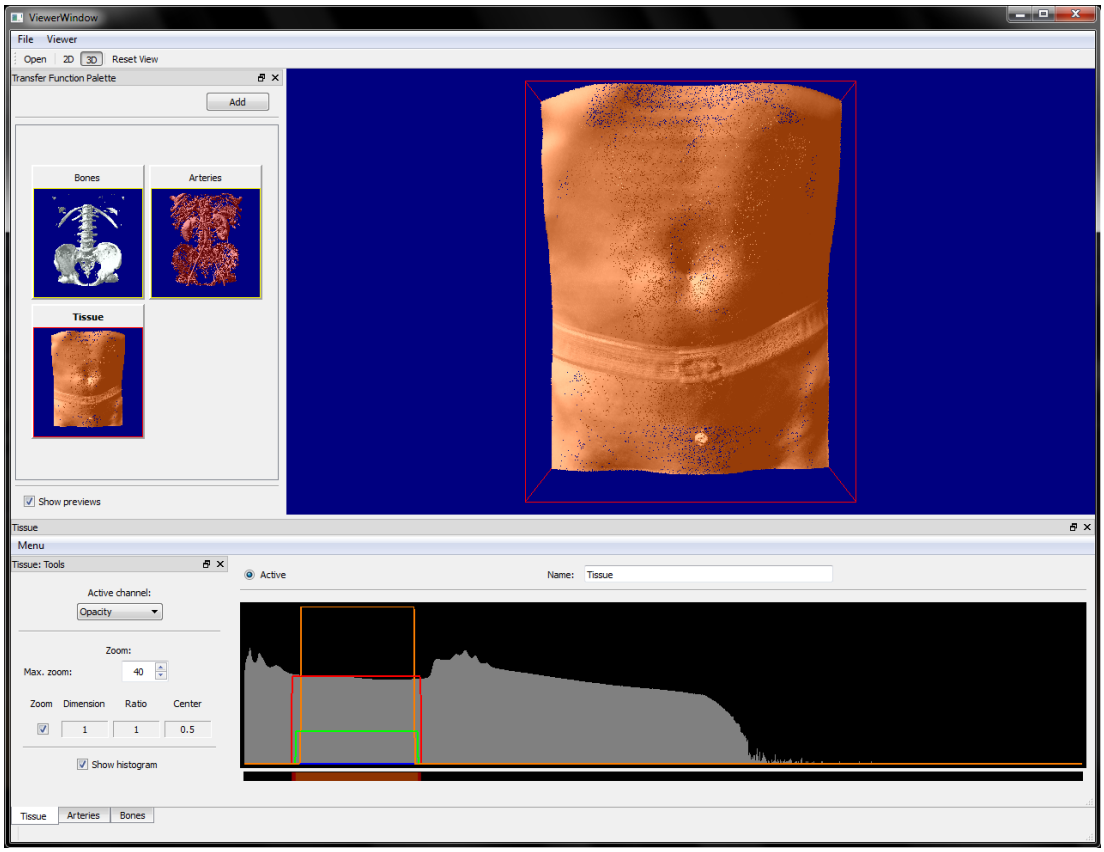
3.4 Použití kompozice

3.4.1 Popis experimentu

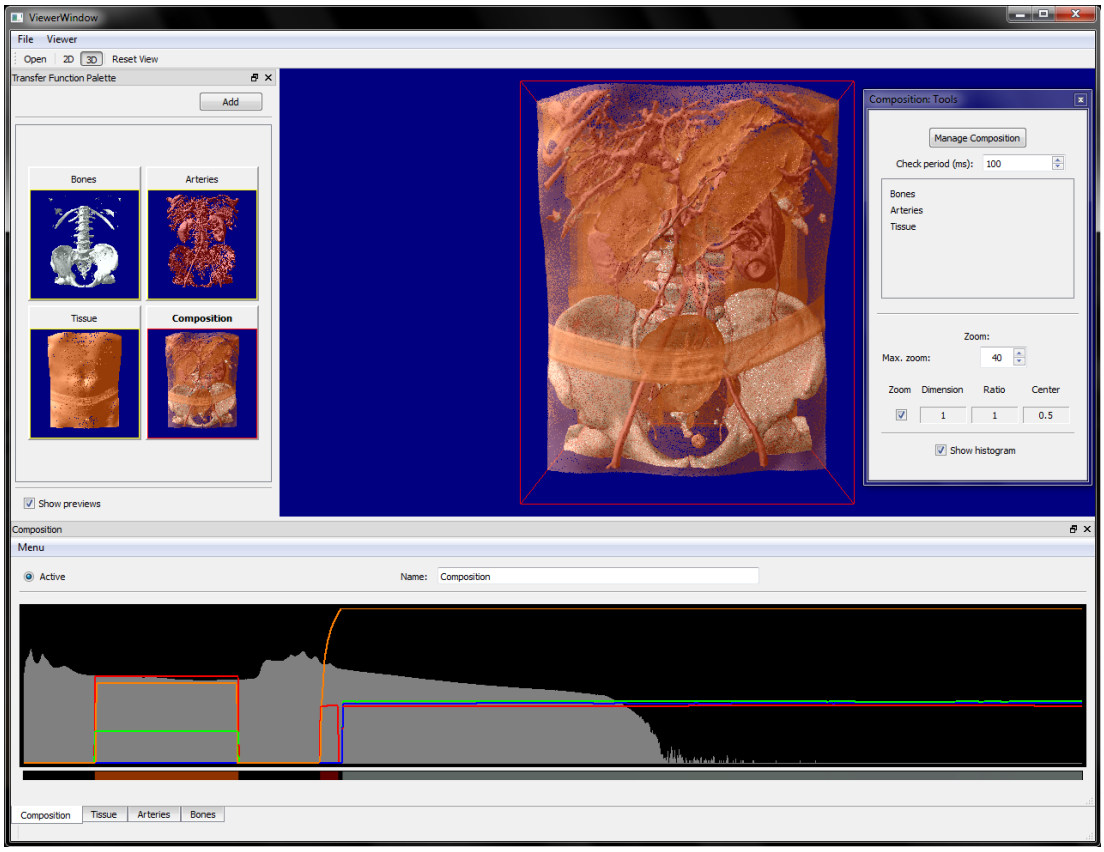
Tento experiment jsme se rozhodli uskutečnit, abychom vyzkoušeli užitečnost pokročilejšího způsobu editace v naší implementaci. Jedná se kompozici několika přechodových funkcí popsanou v kapitolách 2.7.5 (implementace) a A.6.4 (uživatelský manuál). Použijeme přechodové funkce navržené pro experiment 3.1 a přidáme ještě jednu navíc. Tyto funkce pak zkusíme složit pomocí kompozice a zhodnotíme výsledek.

3.4.2 Zhodnocení výsledku

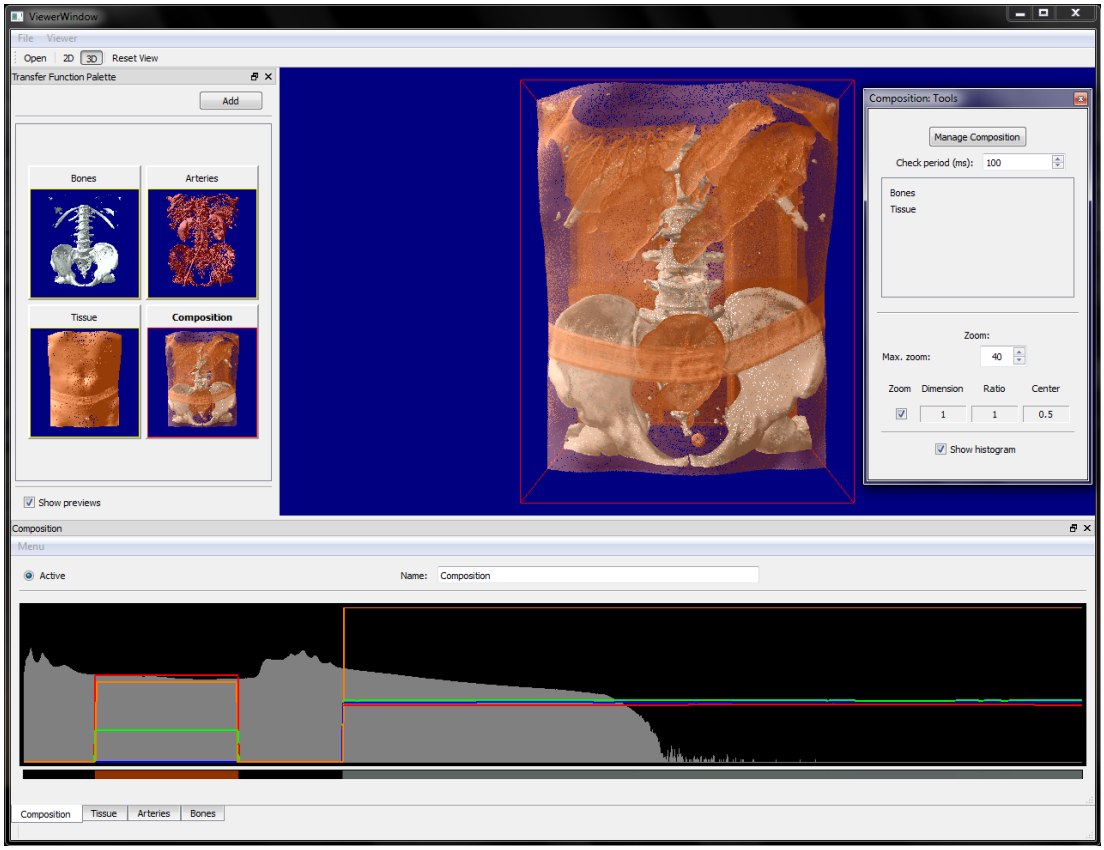
Na obrázku 3.7 můžeme vidět třetí přechodovou funkci zobrazující měkkou tkáň oranžovou barvou, kterou jsme vytvořili pro přidání do kompozice (první dvě můžeme vidět na obrázcích 3.1) a 3.2]. Tímto nám vznikli tři různé přechodové funkce, první zobrazuje pouze kosti, druhá arterie a třetí měkkou tkáň. Po jejich složení dostaneme funkce, které můžeme vidět na obrázcích 3.8 a 3.9. Doplněním



Obrázek 3.7: Třetí přechodová funkce pro kompozici



Obrázek 3.8: Kompozice - kosti, arterie, tkáň



Obrázek 3.9: Kompozice - kosti, tkáň

o křivku výsledné viditelnosti/průhlednosti jsme získali zobrazená data na těchto obrázcích.

Způsob skládání funkcí popsaný v kapitole 2.7.5 zajišťuje, že skládané funkce ovlivňují pouze tu oblast, ve které jsou samy viditelné. V tomto případě jsou tyto oblasti jednotlivých funkcí disjunktní a proto lze vidět všechny tři v nezměněné podobě.

Kompozice nám zjednodušuje návrh složitějších funkcí. Můžeme navrhnout přechodové funkce pro jednotlivé oblasti hodnot dat zvlášť a poté je libovolně skládat. Průběžná aktualizace kompozice zajišťuje pohodlnou úpravu jednotlivých funkcí a promítnutí změn do výsledku.

Závěr

Možná rozšíření

Rozšíření systému o nové typy editorů je jeden z hlavních způsobů použití a je popsán v předchozích kapitolách, nebudeme jej tedy již v této kapitole řešit. Za rozšíření zde budeme považovat rozšíření sytému jako frameworku pro tvorbu editorů nebo rozšíření základů (abstraktních předků) datových struktur či rozhraní sytému, jež používá zobrazovací engine.

Přechodová funkce

V tomto ohledu se nabízí možnost rozšířit datovou strukturu pro uchovávání přechodové funkce (a s ní úzce související pracovní kopii), respektive informaci, která definuje výstup pro jednu hodnotu zobrazovaných dat. Rozšíření o další údaje, jako například schopnost odrážet světlo, index lomu světla a podobně. Hlavním důvodem pro vznik této práce však byla potřeba přechodových funkcí pro zobrazování lékařských dat, kde tyto atributy nemají využití a proto je tato implementace neobsahuje. Toto rozšíření by však mohlo doplnit systém o možnost vytváření ilustrativních obrázků, které mají využití například v lékařské literatuře. Pokud bychom takovéto rozšíření zavedli, mohli bychom systém rozšířit například editorem, který by využíval postup popsaný v článku Style Transfer Functions [9].

Histogram

Možnosti rozšíření se nabízí také u třídy Histogram, která by mohla být rozšířena o zpracování histogramu a výpočet doplňujících informací jako například průměr, maximum, minimum a podobně. To by mohlo být užitečné při jeho zobrazování případně při ulehčování návrhu přechodové funkce. Pokud by byly tyto výpočty potřebné ve více editorech, bylo by vhodnější je centralizovat v třídě Histogram.

Paleta

Další oblast možného rozšíření je paleta. Je možné, že budeme chtít v budoucnu implementovat editor se speciálními požadavky a bude proto nutné rozšířit rozhraní či funkčnost palety.

Užitečným rozšířením by byla možnost ukládat pracovní prostředí. V podstatě by šlo o ukládání celých sad editorů (přechodových funkcí). Tím bychom se mohli posunout o úroveň výše a nabídnout také GUI pro správu těchto sad a vytvořit tak paletu palet. Otázkou však je, jestli by to bylo v praxi využitelné. Možnost uložení pracovního prostředí by nám poskytla například možnost pokračovat při spuštění aplikace tam, kde jsme skončili před jejím posledním ukončením.

S tím souvisí možnost mít implicitní sadu přechodových funkcí. Jednalo by se o předtvořené funkce, které by byly vždy součástí palety jako základ. Pro tyto účely by bylo vhodné vytvořit modifier, který by umožnil tyto funkce pouze prohlížet a nedovoloval by jejich změnu. Toto řešení je běžně používáno v aplikacích pro

zobrazování lékařských dat a umožňuje prohlížet si objemová data bez nutnosti navrhovat přechodovou funkci.

Shrnutí

Podařilo se nám vytvořit systém pro tvorbu editorů přechodových funkcí. Díky šablonované rekurzivní datové struktuře popsané v kapitole 2.4.3 je tento systém zobecněn na libovolnou dimenzi přechodové funkce. Použitím nešablonovaného rozhraní jsem zachovali požadavek na jednotné rozhraní a možnost práce s více editory najednou, kdy nám nezáleží na jejich typu. Jednoduché rozšíření jsme zajistili registračním systémem pro jednotlivé části editorů a soustředěním vytvářecích metod na jednom místě.

Pro ukázku použití našeho systému jsme implementovali několik druhů editorů, které nám zároveň posloužily pro potvrzení splnění některých kladených požadavků. V kapitole o experimentech 3 jsme ověřovali několik vlastnosti přechodových funkcí a porovnávali naše výsledky s těmi publikovanými.

Implementovali jsme také dvě novinky, které nejsou publikovány a chtěli jsme vyzkoušet, zda přináší nějaké výhody. První je kompozice, neboli editor, který umožňuje skládání více již navržených přechodových funkcí. Výhody tohoto editoru jsme ověřovali v experimentu 3.4. Druhou novinkou je navrhování přechodové funkce modifikací komponent barevného modelu HSV (obrázek A.8).

Při testování návrhu pomocí modelu HSV jsme zjistili, že umožňuje přirozeným způsobem přidělovat barvu jednotlivým hodnotám. Úroveň křivky *hue* určuje barevný tón, stačí tedy hodnotám, které chceme barevně odlišit přiřadit jinou úroveň komponenty hue. U jednotlivých barev pak již stačí doplnit jejich sytost (komponenta saturation) a jas (komponenta value). Pro zdůraznění této výhody jsme přidali postranní náhled, který nám dává představu o tom, která úroveň komponenty hue odpovídá kterému barevnému tónu. Tento způsob návrhu se velice hodí pro navrhovatele, kteří nemají dobrou představu o tom, jak složit požadovanou barvu ze základních barev modelu RGB (například lékaři).

Seznam použité literatury

[1] Dokumentace knihovny Qt. *http://doc.qt.nokia.com*. Květen 2011.

[2] Dokumentace knihovny Boost. *http://www.boost.org*. Květen 2011.

[3] Projekt MedV4D. *http://cgg.mff.cuni.cz/trac/medv4d/wiki*. Květen 2011.

[4] David Bednárek. *Pokročilé programování v C++ (část B)*. Prezentace k přednáškám, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze, 2011.

[5] Michal Svoboda. *Uživatelská rozhraní pro editaci přechodových funkcí*. Bakalářská práce, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze, 2009.

[6] Pavel Campr. *Získávání 3D modelů lidských tkání z obrazových dat CT*. Diplomová práce, Fakulta aplikovaných věd, Západočeská univerzita v Plzni, 2005.

[7] Potts, Simeon and Möller, Torsten. *Transfer functions on a logarithmic scale for volume rendering*. GI '04 Proceedings of Graphics Interface 2004, strany 57–63, 2004.

[8] Andreas H. König, Eduard M. Gröller. *Mastering Transfer Function Specification by using VolumePro Technology*. Spring Conference on Computer Graphics, 1999.

[9] Stefan Bruckner and Meister Eduard Gröller. *Style Transfer Functions for Illustrative Volume Rendering*. Computer Graphics Forum, Volume 26, Number 3, strany 715–724, září 2007.

[10] Ropinski, Timo and Praßni, Jörg-Stefan and Steinicke, Frank and Hinrichs, Klaus H.. *Stroke-Based Transfer Function Design*. IEEE/EG International Symposium on Volume and Point-Based Graphics, strany 41–48, 2008.

A. Uživatelský manuál

A.1 Úvod

Aplikace slouží pro navrhování, ukládání a uprovavání uložených přechodových funkcí. Cílem tohoto dokumentu je vysvětlit, jakým způsobem lze aplikaci používat a popsat použité grafické rozhraní. Pro názornost jsou k textu připojeny obrázky se vzhledem grafického rozhraní, vzhled se však může mírně lišit v závislosti na operačním systému a jeho nastaveních.

A.1.1 Přechodové funkce

Přechodové funkce obecně slouží pro přidělení barev jednotlivým hodnotám zobrazovaných dat. Při použití na lékařská data, což je záměrem této aplikace, lze pomocí přechodových funkcí například přidělit různým druhům tkání různou barvu, některé oblasti dat částečně zprůhlednit nebo je nezobrazit vůbec (oblast bude úplně průhledná).

A.1.2 Prohlížeč

Aby jsme měli představu, jak použití navržených přechodových funkcí vypadá, je součástí aplikace prohlížeč, který umožňuje zobrazení objemových dat. Data lze zobrazit dvojrozměrně, kdy si prohlížíme dvojrozměrný snímek nebo vždy jeden konkrétní průřez tojrozměrným snímkem, nebo tojrozměrně, kde můžeme vidět celý tojrozměrný snímek najednou.

A.1.3 Lokalizace

Jazyk používaný v aplikaci je anglický. Všechny použité specifické pojmy jsou vysvětleny v tomto textu. Vícejazyčnost nebyla důležitá pro tento projekt a pokud by se ukázala jako žádoucí, může být dodatečně implementována.

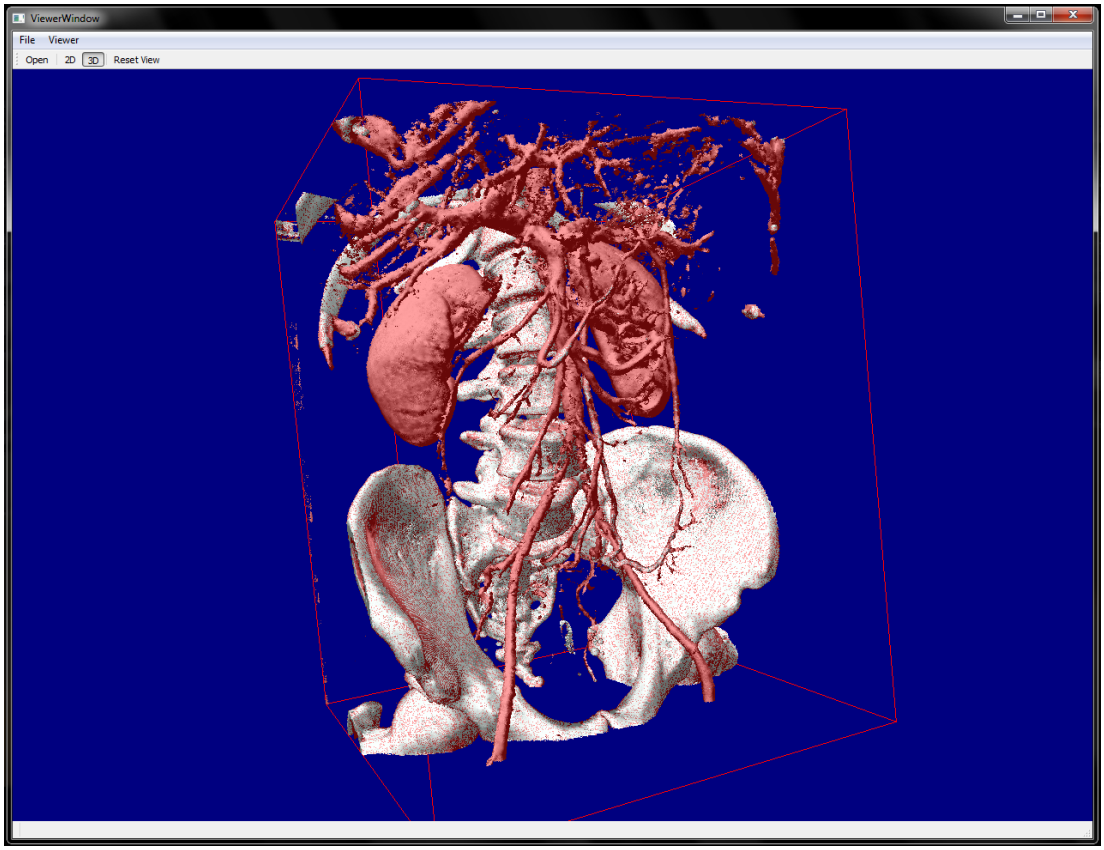
A.1.4 Formát vstupních dat

Pro práci s objemovými daty používáme pouze formát dump. Ten je generován pomocí projektu MedV4D z objemových dat v DICOM formátu. Používání jiných formátů je technicky možné, avšak používání pouze dumpů je snazší a zadání této práce nám nekladlo žádné požadavky na používané formáty.

A.2 Instalace

V připadě 32-bitového operačního systému Windows (XP nebo novější) je k dispozici samorozbalovací archiv TransferFunctions_install32.exe, ve kterém se nachází spustitelný soubor a všechny knihovny a dodatečné soubory potřebné pro spuštění aplikace.

V případě jiného operačního systému je k dispozici adresářová struktura (viz. příloha B) obsahující zdrojové kódy, knihovny a dodatečné soubory a také cmake



Obrázek A.1: Prohlížeč zobrazující lékařská objemová data po použití přechodové funkce

soubor CMakeList.txt. Pomocí tohoto cmake souboru je možné sestavit projekt a přeložit jej pro konkrétní operační systém. Tento postup je popsán v příloze C.

Jazyk i použité knihovny jsou multiplatformní a projekt MedaV4D je také běžně vyvíjen a používán na unixových operačních systémech (Linux). Aplikace je tedy v principu také multiplatformní, avšak byla testována pouze pro 32-bitový operační systém Windows 7. Primární cílovou platformou je proto Windows.

A.2.1 Hardwarové nároky

Aplikace byla testována na konfiguraci:

Procesor: Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz 2.40GHz

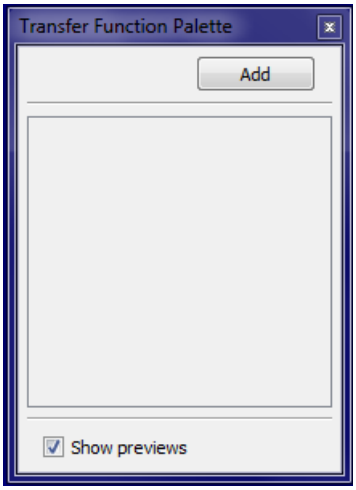
RAM: 4GB

Grafická karta: ATI Radeon HD 5770

Aplikace vyžaduje grafickou kartu s shadery verze 4.0 nebo novější.

A.3 Prohlížeč

Vzhled prohlížeče při použití přechodové funkce pro zobrazení lékařských dat můžeme vidět na obrázku A.1.



Obrázek A.2: Prázdná paleta

Základní funkce, které zobrazovač nabízí jsou vyvedeny do panelu nástrojů, který se nachází pod menu nabídkou. Open – spustí dialogové okno, pro výběr souboru s daty, která chceme zobrazit. Pokud ještě není v paletě žádný editor použitelný (viz. kapitola A.4) pro načtená data, vyvolá se po načtení dat dialogové okno popsané v kapitole A.5.

- 2D – přepne zobrazovač do módu dvou rozměrného zobrazování
- 3D – přepne zobrazovač do módu tří rozměrného zobrazování
- Reset View – anuluje otočení a přiblížení/oddálení náhledu
- Menu Viewer dále nabízí další pokročilé funkce a to povolení/zakázání stínování a povolení/zakázání funkce jittering.

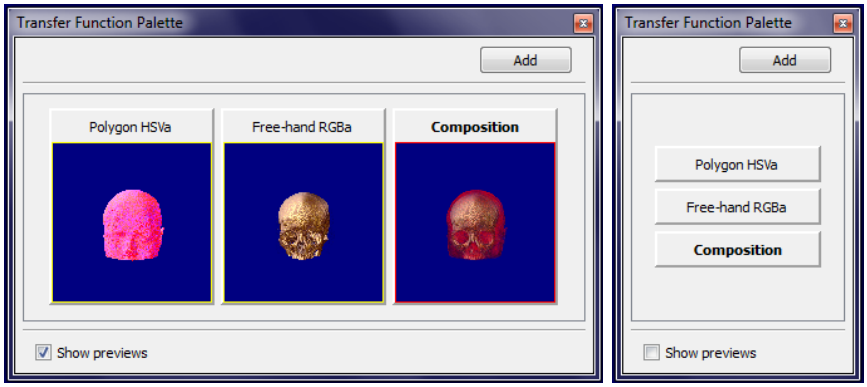
Navíc při zobrazení náhledu dat je možno jej kolečkem myši přibližovat/oddalovat. Stisknutím a držením levého tlačítka myši a následným pohybem je možno náhled otáčet.

A.4 Paleta

Při spuštění aplikace se současně se zobrazovačem otevře také paleta aktuálně otevřených editorů. Po spuštění je tato paleta prázdná, můžeme ji vidět na obrázku A.2.

Pro přidání nového editoru nebo pro načtení editoru ze souboru slouží tlačítko Add. Po kliknutí na toto tlačítko paleta vyvolá dialogové okno, které nás provede tvorbou či načtením editoru přechodové funkce a bude popsáno v kapitole A.5.

Okenní křížek (v pravém horním rohu) způsobí pouze skrytí okna a lze ho znovu zobrazit v menu, které se zobrazí při kliknutí pravým tlačítkem myši na menu lištu zobrazovače.



Obrázek A.3: Paleta se zapnutými (vlevo)/ vypnutými (vpravo) náhledy

A.4.1 Položky palety

Položky v paletě fungují jako tlačítka pro zvolení editoru, jehož přechodová funkce se bude používat pro zobrazení dat. Kliknutí na položku editoru, jehož přechodovou funkci nelze použít, nezpůsobí žádnou změnu. Paleta obsahuje zaškrtávací tlačítko Show previews, které zapíná/vypíná zobrazení náhledů. Na obrázku A.3 můžeme vidět, jak vypadá paleta s náhledy (vlevo) a bez náhledů (vpravo).

Pokud nejsou načtena žádná data, nelze ani vytvořit náhledy a při jejich povolení se zobrazí na jejich místě pouze zpráva, že náhled není dostupný.

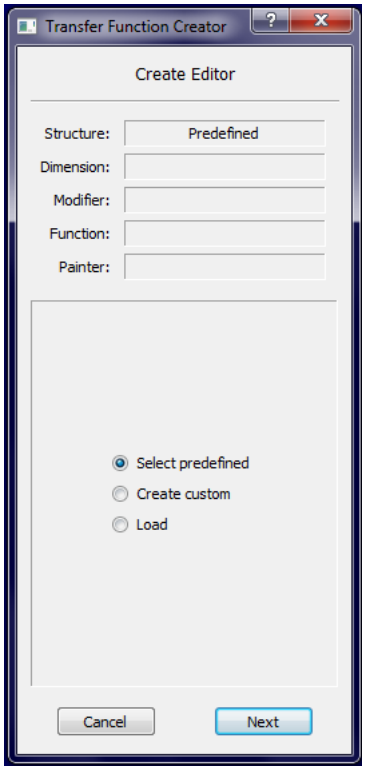
Editor, jehož přechodová funkce je právě používána na pro zobrazení dat, je zvýrazněn tučně psaným názvem a v případě náhledu také červeným rámečkem okolo něj. Editory, jejichž přechodovou funkci lze použít na aktuálně otevřená data (dimenze editoru a jeho přechodové funkce odpovídá požadavkům zobrazovače), mají názvy psány obyčejným písmem a v případě náhledu je okolo nich zobrazen žlutý rámeček. Editory, jejichž přechodovou funkci nelze použít na aktuálně otevřená data, mají názvy psány kurzívou a náhledy jsou bez rámečku. Pro tento případ ovšem nelze vygenerovat náhled a proto v náhledu bude pouze zpráva, že náhlední není k dispozici. První dva případy lze vidět na obrázku A.3.

Pro ilustraci, editory v paletě na obrázku A.3 můžeme vidět na obrázcích A.7 (RGBa 1D), A.8 (HSVa 1D) a A.9 (Composition 1D).

A.5 Průvodce přidáním do palety

V této kapitole si popíšeme dialogové okno, které nás provádí přidáním editoru do palety. Objeví se po stisknutí tlačítka Add na paletě a můžeme ho vidět na obrázku A.4.

První část okna tvoří nadpis, který poskytuje informaci, ve kterém kroku přidávání se právě nacházíme. Druhá část obsahuje 5 pojmenovaných polí, která se vyplňují v závislosti našich volbách při přidávání editoru. Třetí část je seznam možností, které máme v aktuálním kroku přidávání a poslední část tvoří tlačítka, která umožňují pohyb mezi jednotlivými kroky, konečné potvrzení přidání nebo jeho zrušení. Funkce tlačítka závisí na kroku přidávání, v prvním kroku lze místo přechodu do předchozího kroku zrušit přidávání, v posledním kroku lze místo přechodu do následujícího kroku potvrdit přidání zvoleného editoru. Nejprve si popíšeme pole v druhé části.



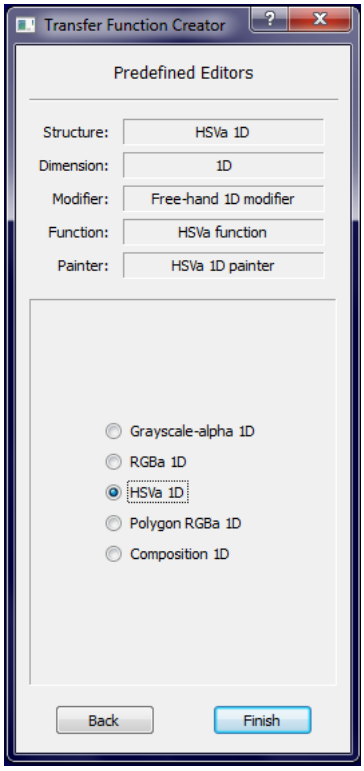
Obrázek A.4: Dialogové okno pro přidání editoru do palety

- Structure – jaký způsob přidání editoru jsme zvolili, případně název zvolené předdefinované struktury
- Dimension – zvolená dimenze přidávaného editoru (a jeho přechodové funkce)
- Modifier – zvolený způsob editace přechodové funkce
- Function – zvolený typ přechodové funkce
- Painter – zvolený způsob vykreslování

A.5.1 Způsob přidání editoru

V tomto kroku se můžeme rozhodnout, jak chceme přidat editor. Popíšeme si možnosti, které můžeme také vidět na obrázku A.4.

- Select predefined – přidat již předdefinovaný editor, při této volbě následuje krok popsaný v kapitole A.5.2
- Create custom – vytvořit respektive složit vlastní editor, po této volbě následují kroky popsané v kapitole A.5.3
- Load – načíst editor ze souboru, pro tuto volbu již nejsou další kroky a po jejím zvolení a potvrzení se vyvolá dialogové okno pro výběr souboru s editorem. Po úspěšném načtení se tento editor přidá do palety



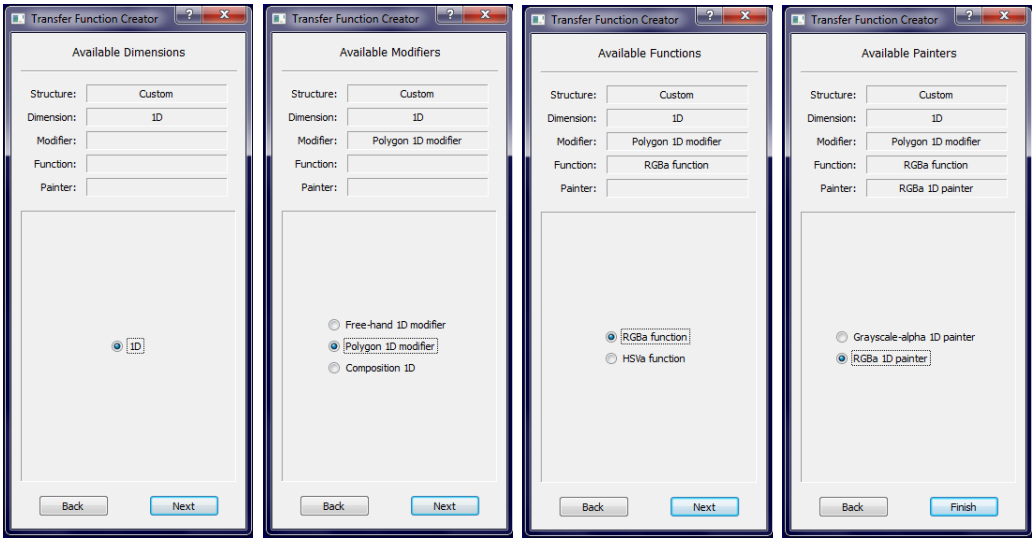
Obrázek A.5: Výběr předdefinovaného editoru

A.5.2 Předdefinované editory

Po zvolení Select predefined v kroku Create Editor (a přechodu dále) se dostaneme do kroku, kde vybíráme ze sady předdefinovaných editorů, které jsou v aktuální verzi dostupné. Na obrázku A.5 můžeme vidět tento výběr pro současnou verzi aplikace. Při zvolení námi požadované možnosti se změní pole Structure z Predefined na název předdefinovaného editoru a zbývající pole se vyplní názvy příslušných částí podle jeho složení. Pokud máme zvolen editor, stačí pouze potvrdit volbu a do palety se přidá daný editor.

A.5.3 Vytvoření vlastního editoru

Po zvolení Create custom v kroku Create Editor následují kroky, které nám postupně podle polí v druhé části dialogového okna (viz. kapitola A.5) nabídnou možnosti dostupné v aktuální verzi. Na obrázku A.6 můžeme vidět posloupnost těchto kroků (v pořadí zleva do prava) v současné verzi. Při přechodu do následujícího kroku se v závislosti na naší volbě vyberou dostupné části, podle pravidel aktuální verze a výběr tedy bude obsahovat pouze ty části, které umí pracovat s těmi již vybranými.



Obrázek A.6: Posloupnost kroků při přidávání vlastního editoru

A.6 Editory

A.6.1 Základní grafické rozhraní

Všechny editory mají společné jednotné grafické rozhraní a navíc své dodatečné GUI, které umožňuje ovládat funkce, lišící se podle způsobu návrhu přechodové funkce. Individuální ovládací prvky jsou umístěny do zvláštního dokovacího okna.

Nejprve si popíšeme základní GUI společné pro všechny editory. To můžeme vidět na obrázcích A.7 a A.8. V podstatě se jedná o tři části a to menu, záhlaví a oblast přecodové funkce.

Menu

Zde máme možnosti Editor, Function a Close.

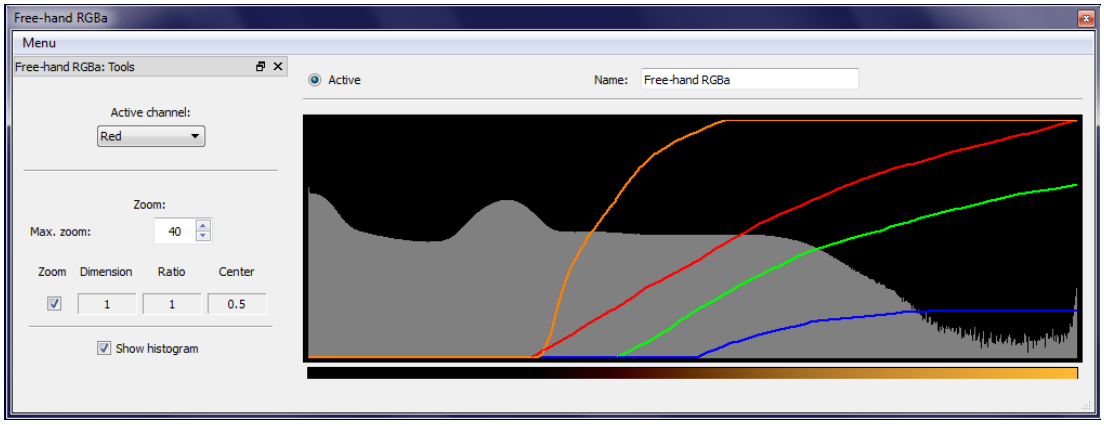
Možnost Close zavře daný editor a odebere ho z palety. Pokud byla provedena neuložená změna, budeme dotázáni, zda chceme uložit změny. Okenní křížek (v pravém horním rohu) způsobí pouze skrytí okna a lze ho znovu zobrazit v menu, které se zobrazí při kliknutí pravým tlačítkem myši na menu lištu zobrazovače.

Po kliknutí na Function se zobrazí podmenu umožňující uložit či načíst pouze přechodovou funkci. Tento způsob umožňuje ulkládání u speciálních editorů jako například kompozice popsaná v kapitole A.6.4 a následné upravení funkce po jejím načtení do jiného editoru. Při načítání funkce se kontroluje, zda odpovídá její dimenze a typ té aktuální. Pokud se liší, objeví se dialogové okno s chybou a nic se nezmění.

Podmenu položky Editor dovoluje uložit celý editor. To znamená, že se uloží přechodová funkce, ale také dodatečné informace a nastavení všech částí editoru.

Záhlaví

Záhlaví obsahuje tlačítko Active, které funguje jako kliknutí na položku v paletě odpovídající tomuto editoru. Tedy způsobí, že se zde navržená přechodová



Obrázek A.7: Free-hand editor s vykreslováním podle modelu RGB

funkce bude používat pro zobrazení dat. Pokud dimenze neodpovídá požadavkům zobrazovače je toto tlačítko zakázáno.

Pole s názvem Name lze editovat a můžeme zde pojmenovat tento editor (tuto přechodovou funkci). Toto jméno se pak zobrazuje v paletě a také jako titulek oken editoru a jeho dodatečného GUI.

Oblast přechodové funkce

Jedná se o oblast pro zobrazení a editaci samotné přechodové funkce. Její modifikace závisí na typu editoru (jeho modifieru) a jednotlivé způsoby jsou popsány v kapitolách A.6.2, A.6.3 a A.6.4.

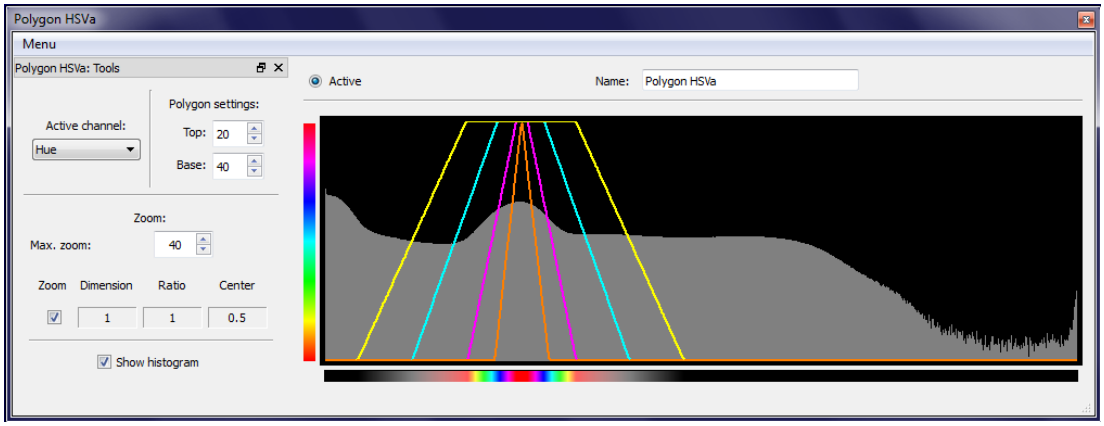
A.6.2 Free-hand editor

Jedna ze současných možností návrhu přechodové funkce je free-hand kreslení (volné kreslení od ruky) křivek jednotlivých komponent barvy. Na obrázku A.7 můžeme vidět editor s tímto zbůsobem návrhu přechodové funkce a jeho grafické rozhraní (GUI).

Editace

Způsob návrhu spočívá v tom, že pro každou komponentu barvy (u RGB je to červená, zelená, modrá a viditelnost) klikneme do editační oblasti (černý obdélník pod záhlavím na obrázku A.7) levým tlačítkem myši a tahem kreslíme křivku určující její hodnotu. To, kterou komponentu modifikujeme můžeme měnit kliknutím pravého tlačítka myši do editační oblasti. Obdélník pod editační oblastí znázorňuje výslednou barvu (bez komponenty průhlednosti).

Otáčením kolečka myši v editační oblasti můžeme přechodovou funkci přibližovat či oddalovat. Pokud při použití kolečka navíc zmáčkneme klávesu alt, budeme namísto přibližování/oddalování upravovat zobrazení histogramu (mění se logaritmický základ přepočtu hodnot logaritmu).



Obrázek A.8: Editor s návrhem přechodové funkce pomocí lichoběžníků a s vykreslováním podle modelu HSV

Dodatečné GUI

Na obrázku A.7 můžeme vidět vlevo dokovací okno, kde se nachází grafické rozhraní pro ovládání tohoto editoru. V horní části (části jsou oddělený vodorovnou čarou) se nachází výběr komponenty pro modifikaci. To nám umožní její jednoduché výbrání a také nám dává přehled o tom, kterou komponentu modifikujeme.

Ve střední části se nachází vstup pro zadání maximálního přiblížení a informace o aktuálním přiblížení. Pole Dimension říká, o kterou dimenzi se jedná (toto GUI je použitelné i pro více-dimenzionální editory), Ratio udává aktální násobek přiblížení a Center je poloha středu zobrazeného výřezu přechodové funkce. Je zde také zaškrtávací tlačítko, kterým můžeme povolit/zakázat přibližování v příslušné dimenzi.

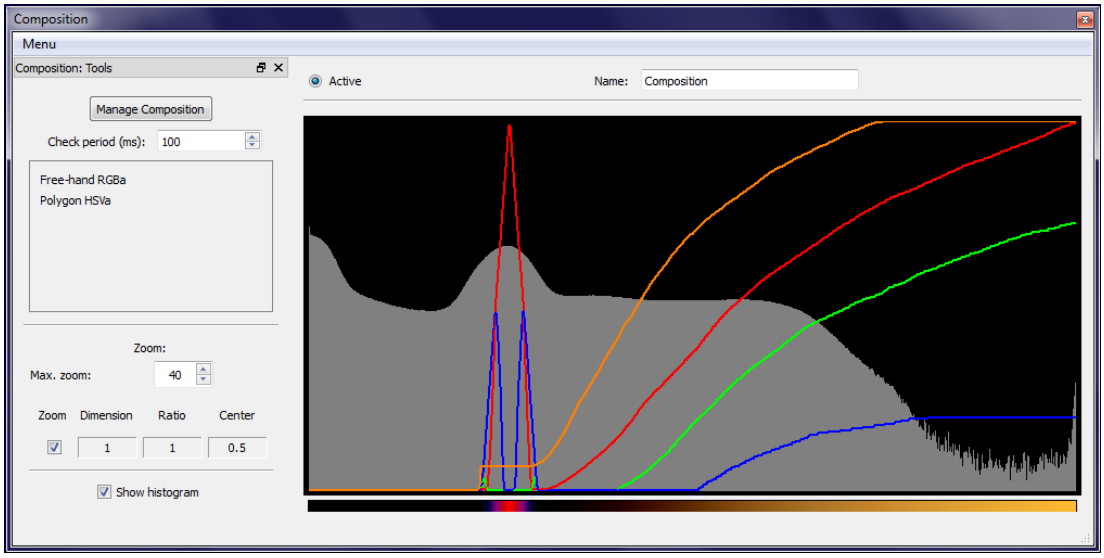
Ve spodní části je pak pouze zaškrtávací tlačítko pro povolení/zakázání zobrazení histogramu.

A.6.3 Návrh pomocí lichoběžníků

Tento způsosob návrhu pouze rozšiřuje způsob editace popsaný v kapitole A.6.2 proto si zde uvedeme pouze to, co je navíc a budeme používat porovnání oproti zmíněnému způsobu. Návrhování u tohoto způsobu modifikace se líší tím, že místo jediného bodu v místě kliknutí se modifikuje (a také vykreslí) lichoběžník, kde poloha ukazatele myši je středem jeho horní základny. Velikost základen tohoto lichoběžníku lze měnit v dodatečném GUI, které je rozšířeno jen o prvky pro změnu těchto dvou hodnot. Vzhled editoru včetně jeho dodatečného grafického rozhraní můžeme vidět na obrázku A.8.

A.6.4 Kompozice

Pokročilejší součást aplikace je kompozice přechodových funkcí. Jde o editor, který výslednou funkci vytváří skládáním funkcí editorů, které jsou již v paletě. Příspěvek jednotlivých skládaných funkcí závisí na jejich křivce viditelnosti. Ta nabývá hodnot 0-1 a hodnoty ostatních komponent jsou touto hodnotou vynásobeny a přičteny do výsledné funkce. Skládáním tedy získáme všechny komponenty kromě viditelnosti, tu musíme navrhnout free-hand kreslením (tento způsob je



Obrázek A.9: Editor s kompozicí přechodových funkcí a s vykreslováním podle modelu RGB

popsán v kapitole A.6.2).

Na obrázku A.9 můžeme vidět, jak takovýto editor vypadá. Obsahuje navíc GUI specifické pro kompozici (ostatní části GUI jsou již popsány v kapitole A.6.2). To se zkládá ze seznamu jmen skládaných funkcí, vstupu pro nastavení periody časovače a tlačítka pro přidávání/odebírání funkcí do/z kompozice.

Aktualizace

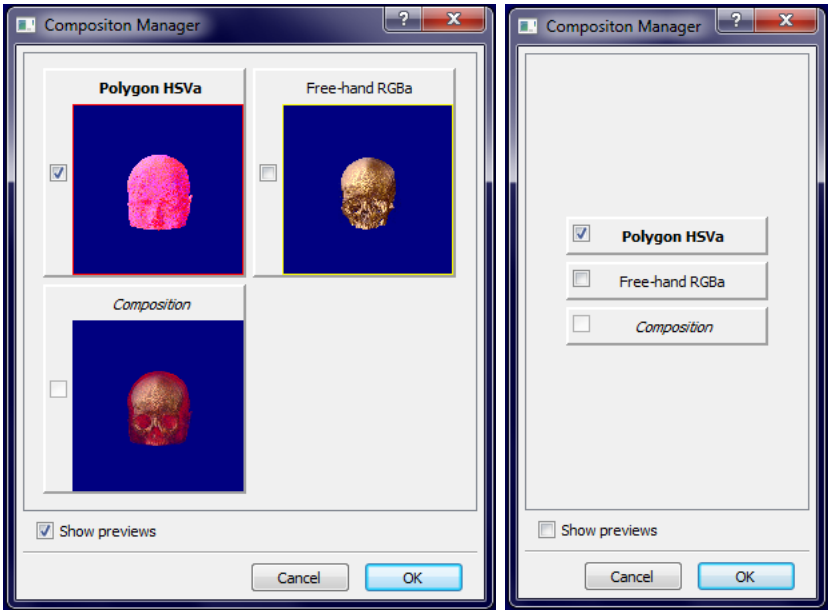
Časovač, který je součástí kompozice aktualizuje výslednou funkci, tím je zajištěno, že se změny provedené na funkciích v kompozici projeví také na výsledné funkci. Ovládání periody zajistí, že můžeme upravit četnost aktualizací například v závislosti na počtu skládaných funkcí či na výkonnosti našeho počítače.

Úprava kompozice

Pro pohodlnou práci a aby dodatečné GUI nebylo příliš obsáhlé, probíhá přídávání/odebírání do/z kompozice prostřednictvím dialogového okna. V tomto okně se zobrazí všechny editory (funkce), které jsou aktuálně v paletě. Okno má stejně jako paleta dva módy zobrazení a to s náhledy nebo bez nich. Oba tyto módy můžeme vidět na obrázku A.10 (s náhledy vlevo, bez nich vpravo).

Každá položka v seznamu je zaškrtávacím tlačítkem. Jsou zde tři stavy tlačítek:

- Povolené – název normálním písmem a náhled se žlutým rámečkem, lze na něj kliknout a zaškrtnout ho a tím přidat do kompozice
- Zaškrtnuté – název tučným písmem a náhled s červeným rámečkem, je zaškrtnuté a přidané do kompozice
- Zakázané – název kurzívou a náhled bez rámečku, nelze jej zaškrtnout a tedy přidat do kompozice



Obrázek A.10: Výběr editorů (funkcí) pro přidání do kompozice – s náhledy (vlevo)/ bez náhledů(vpravo)

Zakázaná jsou ta tlačítka, která odpovídají editorům (funkcím) s jinou dimenzí nebo kompozici (aby nemohlo dojít k zacyklení). Když vybereme a zaškrtneme, co bychom chtěli mít v kompozici, potvrdíme volbu tlačítkem OK. Rozmyslíme-li si to, zrušíme volbu tlačítkem Cancel a kompozice zůstane v původním stavu.

A.7 Vykreslování

V současné verzi aplikace jsou tři typy vykreslování, jedná se o RGBa, Grayscale a HSVa. RGBa můžeme vidět na obrázku A.7. Grayscale se liší tím, že vykresluje pouze jednu komponentu barvy (jedná se o editor, který přiděluje hodnotám dat pouze odstíny šedi a viditelnost/průhlednost) a křivka je kreslena jinou barvou. HSVa můžeme vidět na obrázku A.8, liší se oproti ostatním barvami křivek jednotlivých komponent a navíc je zde pomocný boční náhled zobrazující škálu barevného modelu HSV.

B. Obsah DVD

- `TransferFunctions_install32.exe` - samorozbalovací archiv obsahující spustitelný soubor
- `source`
  - `MedV4D` - složka obsahující projekt MedV4D potřebný pro kompilaci aplikace
  - `TransferFunctions` - složka obsahující zdrojové kódy aplikace
  - `CMakeList.txt` - soubor pro konfiguraci programu CMake (viz. příloha C)
- `libs` - složka obsahující dynamicky linkované knihovny potřebné pro běh aplikace, jsou zde v debug i release módu
- `shader` - složka obsahující shader potřebný pro běh aplikace
- `data` - složka obsahující testovací objemová data
- `examples` - složka obsahující uložené editory a přechodové funkce, jejichž ukázku můžeme najít v textu práce
- `doc` - složka obsahující text bakalářské práce
- `README.txt` - soubor s postupem, jak nainstalovat aplikaci

C. Překlad projektu

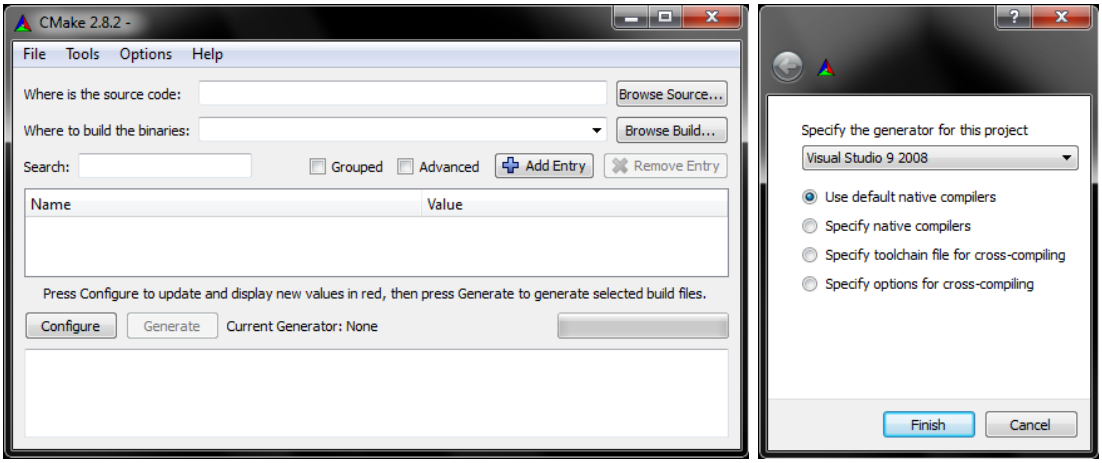
CMake je multiplatformní program pro generování projektů (například pro Visual Studio na platformě Windows) či vytváření *make* souborů (například pro překladač GCC používaný na unixových platformách) potřebných pro přeložení aplikace. K tomu je třeba poskytnout *CMakeList*, což je soubor, který konfiguruje tento program, aby poskytoval požadované výsledky.

CMake je standardní prostředek pro tvorbu projektů vyvíjených pro MedV4D a jednotlivé CMakeListy používají moduly společné pro všechny projekty.

Pro úspěšné přeložení naší aplikace je potřeba použít CMake. Popíšeme si v několika bodech postup pro vygenerování projektu nebo make souboru pro námi zvolený překladač. V textu se budeme odkazovat na GUI programu CMake, které můžeme vidět na obrázku C.1.

- stáhneme si z webu http://www.cmake.org/cmake/resources/software.html aktuální verzi CMake pro námi zvolenou platformu
- nainstalujeme CMake
- do pole označeného *Where is the source code:* nastavíme cestu k našemu souboru CMakeList.txt (viz. příloha B)
- do pole označeného *Where to build binaries* nastavíme cestu, kde chceme vygenerovat projekt či make soubor pro náš překladač
- stiskneme tlačítko *Configure*, objeví se dialogové okno, kde zvolíme generátor z nabídky a potvrdíme tlačítkem *Finish*
- počkáme, dokud CMake neprovede potřebné kroky pro nastavení generátoru
- stiskneme dvakrát tlačítko Configure, po každém stisknutí však musíme chvíli počkat, než se tlačítko změní ze *Stop* zpět na Configure
- nyní se zpřístupnilo tlačítko *Generate*, které stiskneme
- poté, co CMake dokončí generování jej vypneme
- vygenerovaný výsledek přeložíme zvoleným překladačem

Pokud by se stalo, že CMake nenajde knihovny boost a Cg, musíme tyto knihovny doinstalovat.



Obrázek C.1: Uživatelské rozhraní programu CMake