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

Diplomová práce

Objemová haptická vizualizace

Plzeň, 2011 Zdeněk Hamák

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

V Plzni dne 19. 5. 2011 Zdeněk Hamák

Title: Volume Haptic Rendering
Author: Zdeněk Hamák
Supervisor: Ing. Petr Vaněček, Ph.D.
Supervisor's e-mail address: pvanecek@kiv.zcu.cz

Abstract: This thesis presents a method for better exploration of volume data. To better exploration is used velocity of haptic device and detection map of volume data. Both information is compute together by line equation, where velocity determine the angel of the line. The main principle is, when user move fast with haptic device, force applied to haptic device is computed only from edges (changes) in volume data. User can quick find all important areas. As they slow down, important of original data grow up. When user move slow, haptic force is computed only from original data. All theory is implement in H3DAPI. It is open-source framework for quick and easy implementation haptic scene.

Keywords: haptic rendering, velocity driven exploration, H3DAPI, volume change detection

Obsah

Seznam obrázků

Kapitola 1

Úvod

Vizualizace volumetrických dat je důležitou součástí počítačové grafiky. Umožňuje zobrazit různá vědecká data a výsledky simulací. Lidé díky nim dokáží lépe porozumět danému problému. V moderní počítačové době se také využívá virtuální reality, která nejen poskytuje vizuální zobrazení dat, ale také hmatové vnímání. To se nazývá haptika. Pomocí haptického zařízení je možné se dotýkat virtuálních objektů a může působit zpětně síla, která působí na povrchu objektu (tření), v jeho okolí nebo prostoru (proudění vzduchu nebo kapalin, magnetické síly, atd.).
Hmatové vnímání je důležitá schopnost v poznávání objektů a působení sil. Tato práce se zaměřuje na prozkoumání možnosti zlepšení vnímání pomocí haptického zařízení ve vektorových a skalárních volumetrických datech v závislosti na rychlosti pohybu haptického zařízení.

1.1 Cíl práce

Zpětná vazba a haptické zařízení jsou velice užitečným nástrojem při prozkoumávání trojrozměrného prostředí a k lepšímu pochopení zkoumaných dat. Při velkém množství informace není člověk schopen se zorientovat ve volumetrických datech, protože nedůležité informace ho mohou odklonit od důležitých. Tato práce má dva hlavní cíle:

- **Vymyslet metodu, která se pokusí zvýraznit důležitá volumetrická**

data, a tím se v nich lépe zorientovat.

- **Tuto metodu naimplementovat v prostředí H3DAPI s využitím haptického zařízení PHANTOM a ověřit její přínos.**

1.2 Motivace a myšlenky

V počítačové grafice a haptice jsme pracovali s volumetrickými daty hurikánu, která jsme prozkoumávali pomocí haptického zařízení. Když byl pohyb pomalý, bylo možné dobře sledovat vektorové pole a získávat informace. Při rychlém pohybu byl pohyb nepřehledný a působící síly se rychle měnily. Bylo také obtížné se dostat na konkrétní místo, pokud vektorové pole směřovalo proti směru pohybu.

- *První požadavek* - Nejdůležitějším požadavkem na budoucí metodu bylo, aby byla závislá na rychlosti pohybu haptickým zařízením.

Nejprve vedla úvaha k metodě Level of Detail (LoD), která se velice často používá v počítačové grafice a urychluje vykreslování výsledné scény, protože vzdálenější předměty mohou být méně detailní. Tato metoda tedy podle vzdálenosti od kamery používá různé úrovně detailů. Myšlenka byla, metodu upravit pro haptické zařízení a v závislosti na rychlosti pohybu měnit LoD. Problémem se stalo jak určit jednotlivé stupně LoD ve volumetrických datech. Bylo by zapotřebí použít dělení prostoru a mohlo by se stát, že by se ztratila důležitá místa nebo vektory.

- *Druhý požadavek* - Při rychlém pohybu působit na haptické zařízení významnými vektory, aby uživatel rychle našel požadované místo.

Hlavní myšlenkou tohoto požadavku je, aby v místech, kde je stejné vektorové pole nepůsobila na haptické zařízení při velké rychlosti žádná síla. V místech, kde

je změna ve směrech vektorů, působí síla, která upozorní uživatele na změnu ve vektorovém poli.
Příklad takového působení můžeme najít ve skutečném světě. Například při hledání okraje izolepící pásky. Pohybujeme-li se po povrchu, nepůsobí na nás žádná síla. Při přejezdu přes hranu zaznamenáme změnu a o kousek se vrátíme. Při pomalém prozkoumání nalezneme okraj izolepy.

1.3 Nástin práce

Tato práce začíná uvedením do problému. V Kapitole 2 jsou zmíněna haptická zařízení, jejich druhy a základní metody haptického renderování. Následně jsou v Kapitole 3 představeny různé knihovny a prostředí, které se používají pro haptická zařízení. Podrobně je popsáno prostředí H3DAPI, které bylo využito k této práci. Hlavní část práce je představena v Kapitole 4, kde jsou hlavní principy a postupy. V následující Kapitole 5 je implementace v prostředí H3DAPI. V Kapitolách 6 a 7 jsou výsledky a shrnutí práce společně s testovacími daty.

Kapitola 2

Haptická zařízení

Haptické zařízení, nebo-li Haptika, je hmatové zařízení, které využívá uživatelova hmatu pro aplikování sil, vibrací a pohybů. Mechanické působení může být použito k pomoci při vytváření virtuálních objektů, pro kontrolu virtuálních objektů nebo pro vylepšení vzdáleného ovládání strojů a zařízení. Tato technologie také umožnila lépe prozkoumat, jak pracuje lidský hmat.

Existuje velké množství různých haptických zařízení. Nejznámější oblastí jsou haptická zařízení určená pro hraní her. Jedná se například o volanty nebo joysticky, které vibrují, nebo kladou odpor v závislosti na počítačové hře. Tím se snaží zlepšit herní zážitek. Mnohem kvalitnější a specifičtější zařízení jsou používána v profesionálních aplikacích, kde poskytují uživateli silovou zpětnou vazbu (force feedback) pro lepší pochopení simulace nebo i pro reálnější vnímání. V porovnání s herními zařízeními, profesionální poskytují mnohem přesnější a detailnější silovou zpětnou vazbu. Také mají více stupňů volnosti. Herní volant má pouze 1 stupeň volnosti (rotaci) a joystick má 2 (pohyb dopředu/dozadu a doleva/doprava). Profesionální zařízení mají nejčastěji alespoň 3 stupně volnosti (pro pohyb uživatele ve všech směrech). Další stupně volnosti přidávají zařízení možnosti rotace.

Dále zde bude ukázáno několik haptických zařízení. V sekci 2.2 si podrobněji představíme haptické zařízení, na kterém se tato práce testovala. V sekci 2.3 budou představeny různé metody aplikovaní a výpočtu síly pro haptické zařízení.

Obrázek 2.1.1: Haptické zařízení Delta.3 od firmy Force Dimension

2.1 Přehled Haptických zařízeních

Bylo by možné najít mnoho firem, které ve svých produktech mají něco společného s technologií zpětné vazby. V této kapitole bude zmíněno pouze několik firem zabývající se profesionálními haptickými zařízeními.

Force Dimension

Jedná se o Švýcarskou firmu působící na trhu od roku 2001. Nabízí různá haptická zařízení. Všechny modely využívají paralelního mechanismu, který poskytuje větší tuhost a tím i reálnější působení sil. Základní modely jako Omega.3 nebo Delta.3 (Obrázek 2.1.1) poskytují pouze 3 aktivní stupně volnosti. To znamená, že se lze pohybovat v osách x, y, z. V těchto osách působí i zpětné síly. Kvalitnější Omega.6 poskytuje 3 aktivní stupně volnosti pro pohyb a navíc 3 pasivní stupně volnosti pro rotaci. Rotaci zařízení pouze snímá a nepůsobí zpětná vazba. Zařízení Delta.6 poskytuje navíc 3 aktivní stupně volnosti pro rotaci. Jedno z nejlepší zařízení této firmy je Omega.7 (Obrázek 2.1.2). Od Omega.6 navíc obsahuje aktivní možnost stisknutí. Vrcholem nabídky je zařízení Sigma.7, které má 7 aktivních stupňů volnosti. Zařízení se také liší rozsahem sil a velikostí rozsahu pohybu.

Obrázek 2.1.2: Haptické zařízení Omega.7 od firmy Force Dimension

Haption

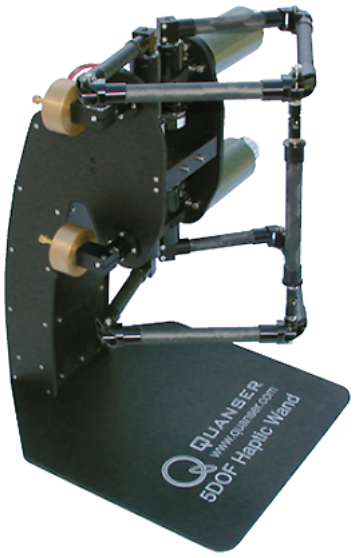
Francouzská firma vytvářející haptická zařízení se sériovým mechanismem. Ve své nabídce má jedno zařízení s 3-mi stupni volnosti a několik zařízení s 6-ti stupni volnosti.

Quanser

Kanadská firma s velkou nabídkou různých haptických zařízení. V nabídce jsou zařízení poskytující volnost od 2 stupňů až do 5-ti stupňů volnosti. Zařízení s 5-ti stupni volnosti (Obrázek 2.1.3) poskytuje pohyb ve 3 osách a rotaci ve 2 osách.

Sensable Technologies PHANTOM

Produktová řada nabízí zařízení pro potřeby výzkumných i komerčních zákazníků. Zařízení PHANTOM Premium jsou vysoce přesná a poskytují největší síly i největší pracovní prostor. Některé modely poskytují možnost 6-ti stupňů volnosti. Stolní zařízení PHANTOM Desktop poskytuje lepší možnosti. Nabízí větší sílu, nižší tření a větší pracovní prostor než PHANTOM Omni, který je cenově nejefektivnějším zařízením na trhu.
PHANTOM Omni (Obrázek 2.2.1) bude podrobně popsán v následující sekci, protože se jedná o zařízení, na kterém byla tato práce testována.



Obrázek 2.1.3: Haptické zařízení 5 DOF Haptic Wand System od firmy Quanser

Obrázek 2.2.1: Haptické zařízení PHANTOM Omni od firmy Sensable

2.2 PHANTOM Omni

Jedná se o relativně malé zařízení s IEEE-1394 a FireWire portem, které zajišťuje rychlou instalaci a snadné použití. PHANTOM Omni disponuje 6-ti stupni volnosti. Pohyb v osách x, y, z je doplněn o zpětnou vazbu. Rotaci v těchto osách je možné pouze snímat. Přesné hodnoty zařízení jsou uvedeny v tabulce 2.1

Model	PHANTOM Omni
Pracovní plocha zpětné vazby	> 160 W x 120 H x 70 D mm
Podstava	∼ 168 W x 203 D mm
Hmotnost	1,78 kg
Rozsah pohybu	Pohyb rukou a v zápěstí
Jmenovité rozlišení polohy	∼ 0.055 mm (450 dpi)
Tření	< 0.26N
Maximální síla	3.3 N
Trvalá síla (po 24h)	> (0.88 N)
Tuhost	osa X > 1.26 N/mm osa Y > 2.31 N/mm osa Z > 1.02 N/mm
Setrvačnost	∼ 45 g
Zpětná vazba	osy x, y, z (digitální snímače)
Pohyb stylusu	rotace podle os x, y, z
Rozhraní	IEEE-1394 FireWire port

Tabulka 2.1: Specifikace haptického zařízení PHANTOM Omni

2.3 Haptic Rendering

Chceme-li využívat jakékoliv haptické zařízení je důležité správně vypočítat a přidělit (namapovat) síly na konkrétní zařízení. Data obsahují skalární nebo i vektorové veličiny. Nejjednodušší případ je, když obsahují pouze jednu hodnotu pro daný voxel. Obvykle obsahují více parametrů pro jeden voxel. Například v hurikánu je vektor, který určuje směr a velikost proudění větru, teplota, tlak a další. Vizualizace více rozměrného datového setu je náročný problém. Řešením může být použití více vizualizačních technik, ale hrozí nebezpečí, že výsledný obraz se stane nepřehledným.

Využitím haptického zařízení můžeme některá data reprezentovat silou nebo vibrací. Pokud to haptické zařízení umožňuje, je možné využívat i rotaci, nebo také sílu stisku. Visuální informace prezentuje celkový pohled na objekty. Haptické zařízení prezentuje atributy v lokální části dat.

Metod haptizace je několik a záleží na potřebách, která metoda se pro dané použití hodí nejvíce. Existují různé možnosti, jak vypočítávat sílu působící na zařízení. Většina metod řeší problémy s dotýkáním plochy objektů. Nejjednodušší metodou je přímý výpočet síly, pak následují metody založené na haptických primitivech a

omezující metody (Constraint-Based metods) do kterých například patří Proxy-Based metody založené na virtuálním bodu.
V následující části budou popsány jednotlivé metody aplikace síly na haptické zařízení.

2.3.1 Přímý výpočet síly

Základní a nejjednodušší metodou výpočtu síly pro haptické zařízení je přímé namapování. V závislosti na typu dat je několik možností, jak hodnoty namapovat na sílu.

Skalární data

Skalární data můžeme namapovat na haptické zařízení dvěma způsoby. První způsob je použití skalární hodnoty k definování síly vibrace (2.1) :

$$O = \alpha[S(x, y, z)] \tag{2.1}$$

kde $S(x, y, z)$ je skalární hodnota ve voxelu, α je vektor určující změnu měřítka. V tomto případě bude v každém voxelu vibrovat zařízení podle velikosti skalární hodnoty. Druhou metodou je vypočítat gradient skalárních dat a ten namapovat na sílu podle rovnice (2.2).

$$F = \alpha[-gradS(x, y, z)] \tag{2.2}$$

Tím se převede skalární pole na třírozměrné potenciálové pole. Zařízení bude tlačeno do míst s nízkým potenciálem.

Vektorová data

Vektorová data obsahují tři složky, takže mohou být přímo namapovány na sílu:

$$F = \alpha[V(x, y, z)] \tag{2.3}$$

Při reprezentaci víceparametrového datového setu je nutné správně namapovat jednotlivá data. Například vektorová data budou namapována na sílu a skalární

data na vibrace. Pokud by bylo více vektorových dat, nelze je najednou namapovat na jedno haptické zařízení.

2.3.2 Omezující metody (Constraint-Based)

Tento přístup byl poprvé představen v [LYG02] a následně byl upravován a použit v [IBHJ03], [VTNB04], [LSCY05], [LSCY05]. Byl vyvinut, aby poskytl přirozenější reprezentaci silového působení v haptických datech, než u přímé metody výpočtu síly. Omezující metoda používá lokální data k určení lokálního pasivního omezení, které reprezentuje data v každé pozici ve volumetrických datech. Tak je možné nastavit povrch chování odpovídající reálným materiálům, jako je tření nebo síla potřebná k proniknutí skrz povrch. Metody umožňují projít skrz povrch, aniž by bylo nutné vypínat a znova zapínat působení síly na haptické zařízení. Omezující přístup byl využit i při generování síly z vektorového pole [IBHJ03], [LSCY05]. Většinou je zapotřebí vizualizovat statická data jako CT snímky. V [PCY07] je bráno v úvahu, že se volumetrický objekt může pohybovat a ovlivňovat sílu, kterou působí na haptické zařízení. Zástupcem těchto metod je metoda Proxy-Based a další odvozené metody.

2.3.3 Proxy-Based (God-object)

Tato metoda simuluje, jak se přes virtuální pružinu ovlivňuje haptické zařízení (probe) a nahrazující bod (proxy point/Haptic Interface point - HIT) v prostoru. Výpočet této metody může být rozdělen do tří částí (Obr. 2.3.1).

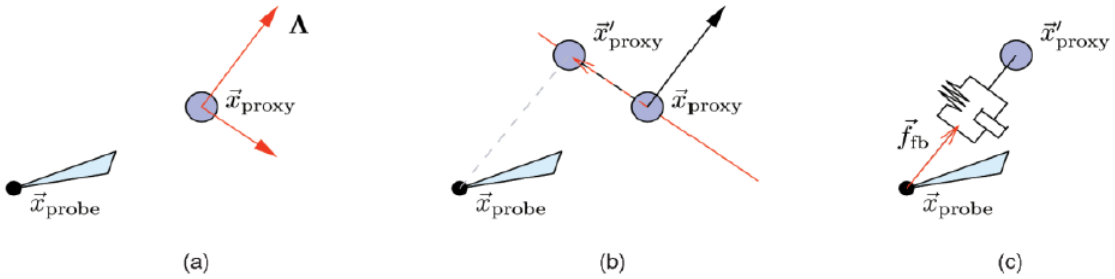
1. Získat data v nahrazujícím bodě, $\vec{x}_{proxy}$:

$$\Lambda = \Lambda(T^{-1}\vec{x}_{proxy}) \tag{2.4}$$

   kde Λ jsou volumetrická data objektu, který nás zajímá a T je transformace popisující velikost a pozici volumetrických dat ve světových souřadnicích.

2. Použít data k určení posunu náhradního bodu:

$$\vec{x}'_{proxy} = \vec{x}_{proxy} + \vec{F}(\Lambda, \vec{x}_{proxy}) \tag{2.5}$$



Obrázek 2.3.1: Tři kroky použité k určení pozice náhradního bodu a vypočítání síly působící na haptické zařízení.(a) První krok, získány data o $\vec{x}_{proxy}$ z poslední pozice. (b) V druhém kroku se náhradní bod posune. (c) Nakonec se vypočítá síla působící na haptické zařízení pomocí virtuálního tlumiče.

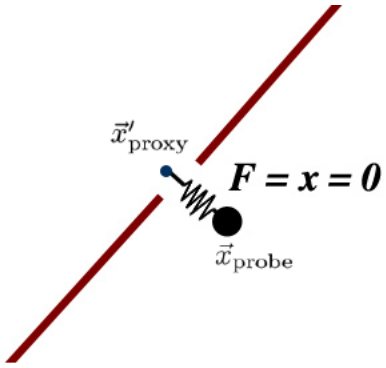
kde $\vec{F}$ je funkce , určující chování, jak se má posunout nahrazující bod v závislosti na simulované vlastnosti.

3. Vypočítat sílu $\vec{f}_{fb}$ z pozice nového nahrazujícího bodu vzhledem k haptickému zařízení:

$$\vec{f}_{fb} = -k_s \left(\vec{x}_{proxy} - \vec{x}_{probe}\right) - k_d \left(\dot{\vec{x}}_{proxy} - \dot{\vec{x}}_{probe}\right) \tag{2.6}$$

kde k_s je konstanta tuhosti a k_d je tlumící konstanta.

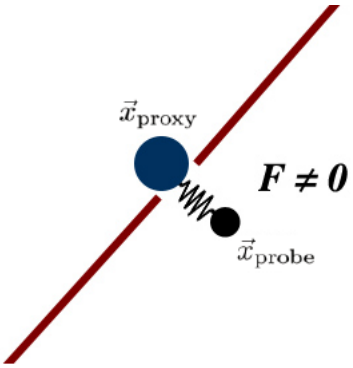
Tyto tři kroky jsou shodné pro různé metody založené na tomto principu. Jediný rozdíl je v druhém kroku, kde se určuje posun virtuálního bodu.
Při pohybu v prostoru bez interakce s objekty je virtuální bod ve stejné pozici jako haptika. Při dotyku s objektem zůstane virtuální bod na povrchu objektu a pozice haptiky bude uvnitř objektu. Přičemž síla se vypočítá jako vzdálenost pozice haptiky a virtuálního bodu.
Problémem algoritmu je, pokud geometrie obsahuje malé díry. Virtuální bod projde dírou a zasekne se na druhé straně (Obrázek 2.3.2). Nazpět se dostane teprve tehdy, když uživatel nalezne další díru. Tento problém řeší následující přístup.



Obrázek 2.3.2: Problém metody proxy-point (God-object)

2.3.4 Ruspini

V základě je metoda stejná jako metoda Proxy-point (God-Object). Jediný rozdíl je, že místo virtuálního bodu je použita virtuální koule. Ta zabrání proniknutí skrz stěnu objektu (Obrázek 2.3.3). Tato metoda také poskytuje reálnější výsledky, protože dotykovou plochu prstu si lze představit jako část koule. Je to lepší nahrazení než jediným bodem. (Obrázek 2.3.3)



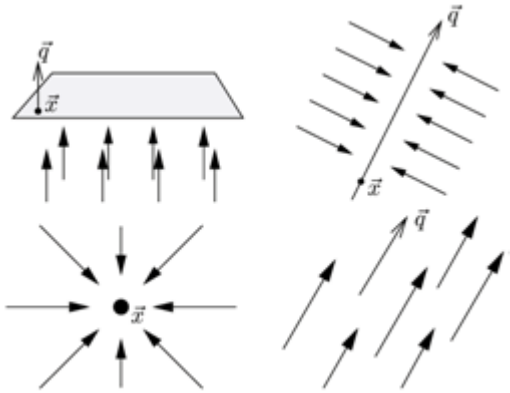
Obrázek 2.3.3: Metoda Ruspini - řešení problému metody proxy-point

2.3.5 Primitives-Based

Tato metoda využívá proxy-based přístupu i přímého výpočtu síly. Je založena na třech krocích, stejně jako proxy-based metoda, přičemž v druhém kroku definuje funkci $\vec{F}$ podle rovnice (2.5). Výpočet se neprovádí přímo, ale je skryt

za abstraktní vrstvu haptických primitiv. Tato metoda byla poprvé popsána v [LSCY05].

Haptická primitiva jsou čtyři. Jedná se o bod, přímku, rovinu a směrovou sílu (Obr. 2.3.4). První tři se využívají k definování omezení na povrchu. Směrová síla se používá na aktivní síly a ostatní silové funkce. Superpozicí těchto čtyř primitiv je možné složit jakoukoliv silovou odezvu potřebnou v haptických zařízeních.



Obrázek 2.3.4: Haptická primitiva: bod, přímka, rovina, směrová síla.

Haptická primitiva jsou charakterizována jednoduchou silovou funkcí s parametry síly s_j, pozice $\vec{x}_j$ a směru reprezentovaného jednotkovým vektorem $\vec{q}_j$. Každé primitivum generuje sílu s velikostí s_j směrem k nejbližšímu bodu na primitivu definovaném pozicí a orientací. Primitivum rovina je pouze jednosměrná. Poskytuje odezvu, pouze když se tlačí proti povrchu roviny, ale už ne tehdy, když se pohybuje od roviny.

Jednotlivá primitiva jsou definována funkcemi:

- *Bod* - Produkuje sílu, která směřuje k bodu v prostoru.

$$\vec{f}_j(\vec{x}_{proxy}) = \begin{cases} 0, & |\vec{x}_j - \vec{x}_{proxy}| = 0 \\ s_j \frac{\vec{m}}{|\vec{m}|}, & |\vec{m}| \neq 0 \end{cases} \qquad \vec{m} = \vec{x}_j - \vec{x}_{proxy} \tag{2.7}$$

- *Přímka* - Produkuje sílu, která směřuje k nejbližšímu bodu na přímce.

$$\vec{f_j}(\vec{x}_{proxy}) = \begin{cases} 0, & |\vec{m}| = 0 \\ s_j \frac{\vec{m}}{|\vec{m}|}, & |\vec{m}| \neq 0 \end{cases}$$
$$\vec{m} = \vec{q_j}\left[\vec{q_j} \cdot (\vec{x}_{proxy} - \vec{x}_j)\right] - (\vec{x}_{proxy} - \vec{x}_j) \tag{2.8}$$

- *Rovina* - Přímá síla, která existuje pouze na jedné straně roviny, která je definována pomocí $\vec{x}_j$ a $\vec{q}_j$.

$$\vec{f_j}(\vec{x}_{proxy}) = \begin{cases} 0, & (\vec{x}_{proxy} - \vec{x}_j) \cdot \vec{q_j} \geq 0 \\ s_j\vec{q_j}, & (\vec{x}_{proxy} - \vec{x}_j) \cdot \vec{q_j} < 0 \end{cases} \tag{2.9}$$

- *Směrová síla* - Síla, která nezávisí na pozici, určující je pouze směr.

$$\vec{f_j}(\vec{x}_{proxy}) = s_j\vec{q_j} \tag{2.10}$$

Kapitola 3

Programové prostředí

K programování aplikací se využívají různé nástroje a prostředí, které usnadňují jejich vývoj. Poskytují nástroje, abstraktní třídy nebo hotové moduly, které je možné volně využít. Tyto nástroje se nazývají programové prostředí (framework). Cílem frameworku je převzetí typických problémů dané oblasti, čímž se usnadní vývoj tak, aby se návrháři a vývojáři mohli soustředit pouze na své zadání. Při počátečním použití frameworku se může zdát, že je použití neefektivní, protože je nutné framework nastudovat a naučit se jej používat. Tento čas je možné investovat do napsání vlastního kódu a framework nevyužít. Výhody se projeví při pravidelném používání, kdy postupně přidáváme moduly a programujeme vždy pouze chybějící část.

V následujících kapitolách budou popsány 3 nejvýznamnější frameworky pro haptické zařízení. Všechny jsou naprogramovány v programovacím jazyku C++. Jedná se postupně o Chai3D, Sensable OpenHaptics a H3DAPI. Všechny tři prostředí jsou optimalizovány pro grafické prostředí OpenGL. Chai3D a Sensable OpenHaptics jsou založeny pouze na programovacím jazyku C++. Prostředí H3DAPI kombinuje několik programovacích jazyků. Základem je C++, ve kterém jsou napsány všechny třídy a metody. Ty se využívají při definování scény v jazyku X3D (viz. sekce 3.3.2). Třetí možností je skriptovací jazyk Python (viz. sekce 3.3.3), který umožňuje snadné nadefinování interakce se scénou a další propojení jednotlivých tříd a metod.

3.1 Chai3D

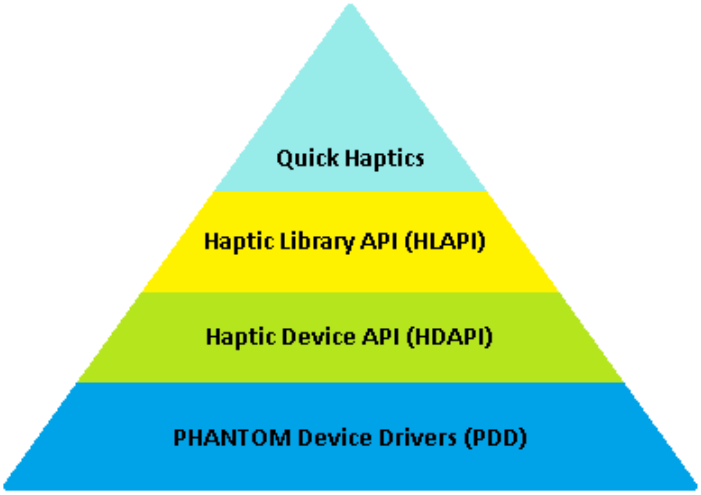
Jedná se o volně dostupnou sadu knihoven C++ se zdrojovými kódy pro počítačovou haptiku, vizualizaci a interaktivní real-time simulaci. Podporuje velkou řadu komerčních zařízení se 3-mi, 6-ti a 7-mi stupni volnosti a jednoduše umožňuje přidat další nová zařízení. Chai3D je velice vhodný pro výukové a výzkumné účely. Poskytuje jednoduchý základ, nad který mohou být vyvinuty další rozšíření. Umožňuje připojení více haptických zařízeních současně. Možnosti Chai3D se dají dále rozšířit pomocí modulů přidávající další funkce a vlastnosti, například fyzický model.

Na hlavních stránkách Chai3D je popis prostředí a příklady v jakých oblastech se používá a jakých výsledků lze dosáhnout. V případě zájmu o prostředí je možné si stáhnout potřebné knihovny, podívat se na příklady a návody.

3.2 OpenHaptics

Prostředí OpenHaptics je vyvinuto firmou Sensable a je připraveno k použití s výrobky této firmy. Snaží se programování zjednodušit tím, že sjednotí základní kroky haptických/grafických aplikací. V poslední verzi 3.0 byla přidána možnost využít novou část prostředí nazvanou QuickHaptics micro API. Umožňuje napsat jednoduchou grafickou/haptickou aplikaci za pomocí 8-mi řádek kódu namísto 300 řádek! QuickHaptics umožňuje rychlý programový návrh a vývoj. Je ideální pro vkládání haptiky do již existujících aplikací, testování nových nápadů, a pro vytváření ukázek a příkladů.

Hlavní dvě knihovny OpenHaptics jsou HDAPI a HLAPI. Celá struktura OpenHaptics je znázorněna na obrázku 3.2.1. Je zde znázorněno jak jsou jednotlivé knihovny postaveny na sobě. Na nejnižší úrovni jsou ovladače PDD (PHANTOM Device Drivers). Nad ovladači je knihovna HDAPI, která poskytuje nízkoúrovňový přístup k haptickému zařízení. Knihovna HLAPI je postavena na HDAPI a zjednodušuje přístup k haptickému zařízení. Nejvýše je nová knihovna QuickHaptics umožňující velice jednoduchou tvorbu haptických programů.



Obrázek 3.2.1: Struktura prostředí OpenHaptics

3.2.1 HDAPI

Knihovna HDAPI poskytuje nízko-úrovňovou vrstvu pro přístup k haptickému zařízení. Je navržena pro vývojáře, kteří jsou obeznámeni se základy a vlastnostmi haptických zařízení. Znají také metody haptické vizualizace a chtějí vypočtenou sílu přímo posílat na haptické zařízení.
Dále jsou uvedeny možnosti knihovny HDAPI. Jedná se o zúžený výběr. Celý seznam je uveden na stránkách firmy SENSABLE.

- *Informace o zařízení:*
  - Pozice, orientace, rychlost a stav tlačítek.
  - Tři souřadné systémy: Kartézský, kloubový(joint), přímá data ze snímačů v zařízení.
- *Charakteristiky o zařízení:*
  - Model, verze a sériové číslo.
  - Dimenze pracovního prostředí.
  - Maximální tuhost, síla a rychlost.

  - Možnost kalibrace.

- *Nastavení stavu zařízení:*
  - Síla/rotace v Kartézském prostoru.
- *Vlastní plánovač pro haptický cyklus.*
- *Hlášení a obsluha chyb.*
- *Kalibrace zařízení.*

3.2.2 HLAPI

Jedná se o vysoko-úrovňovou knihovnu postavenou nad HDAPI. Tato knihovna poskytuje snadnější implementování OpenGL grafiky. Podporuje událostmi řízený programovací model. To znamená, že při určité události se vykoná určitý programový kód. Knihovna je zaměřena na vývojáře, kteří jsou méně obeznámeni s programováním haptického zařízení, ale chtějí rychle a snadno přidat haptické zařízení do již existující grafické aplikace.
Knihovna je navržena pro použití s ostatními knihovnami, rozšiřující možnosti v oblasti fyziky nebo dynamiky. Rozšiřitelná a flexibilní architektura umožňuje použití ostatních grafických knihoven.
Následně jsou uvedeny některé možnosti knihovny HLAPI. Celý seznam je na stránkách firmy SENSABLE.

- *Tvary:*
  - OpenGL primitiva (polygony, body, přímky).
  - Vlastní rozšíření.
- *Efekty síly:*
  - Konstantní (např.: gravitace).
  - Viskozita a 3D tření.

  - Pružina.

- *Vlastnosti povrchů:*
  - Tření.
  - Tuhost a pružnost.
- *Rozšíření moduly od jiných vývojářů.*
- *Události.*

3.3 H3DAPI

H3DAPI je volně dostupná vývojová platforma, která využívá otevřené standary OpenGL a X3D s haptickou vizualizací v jednotném scénovém grafu pro definování grafické i haptické vizualizace. H3DAPI je nezávislá na platformě a na haptickém zařízení. Umožňuje implementovat zvuk a stejně tak stereografii na podporovaných monitorech.
Stejně jako ostatní prostředí využívající scénový graf, je H3DAPI primárně navržen pro velice rychlý vývojový proces. Kombinací X3D, C++ a skriptovacího jazyku Python nabízí tři možné varianty programování aplikace, které nabízí to nejlepší z vlastností jednotlivých jazyků. C++ pro výpočetní rychlost, kde je výkonnost velice důležitá. X3D a Python v oblastech, kde je zapotřebí rychlý vývoj a není zapotřebí veliké výkonnosti.
H3DAPI je napsáno v C++ a je navrženo tak, aby se snadno dalo rozšiřovat. To umožňuje vývojářům volnost v přidávání dalších grafických nebo haptických funkcí do H3DAPI pro potřeby jejich aplikací.

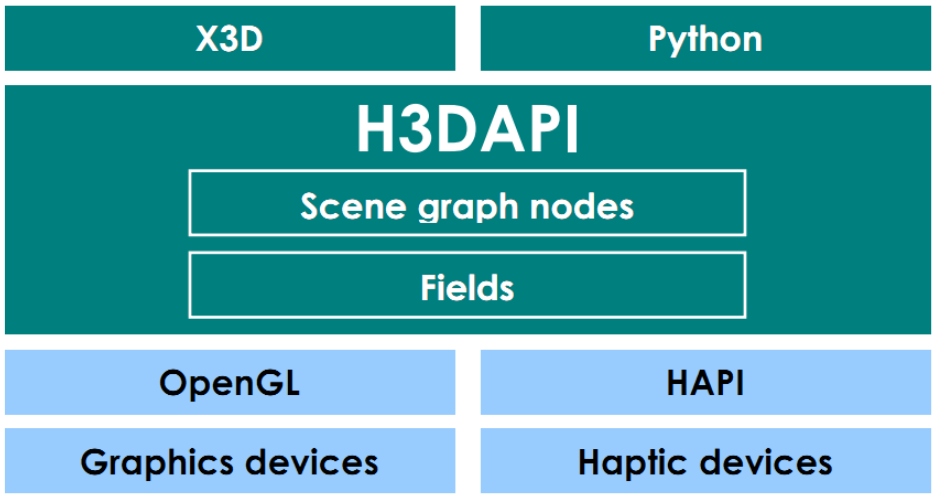
Hlavní výhody použití H3DAPI

- Umožňuje velice rychlý vývoj haptických aplikací použitím X3D a scriptovacího jakyka Python.

- Snadná rozšiřitelnost haptickými nebo visuálními doplňky použitím programovacího jazyka C++.
- Podpora velkého množství haptických zařízení dostupných na trhu.
- Platformní nezávislost.
- Možnost zvolit si haptický renderer včetně H3D bodového a kulového rendereru, SensAble OpenHaptics a Chai3D.

3.3.1 Architektura H3DAPI

H3DAPI je v podstatě vrstva, která zaobaluje grafiku a haptiku do jednoho prostředí a implementuje X3D a Python jako rozhraní pro vývoj aplikací (Obrázek 3.3.1). Zkombinováním haptiky a grafiky do jednoho scénového grafu zajišťuje, že vývojář aplikace se nemusí zbytečně zabývat rozšiřováním grafických nebo haptických částí. Může se plně věnovat vývoji programu.



Obrázek 3.3.1: Architektura prostředí H3DAPI

Velkou výhodou H3DAPI je poskytnutí možnosti využít X3D a skriptovací jazyk Python. H3DAPI je navrhnuto jako modulární a snadno rozšiřitelné prostředí. Zkušení vývojáři mohou využít nízko-úrovňový přístup k grafice a haptice použitím

programovacího jazyka C++.

V následujících sekcích budou popsány jednotlivé programovací jazyky H3DAPI a na příkladech ukázány možnosti využití.

3.3.2 X3D - Extensible 3D

Grafický formát X3D vychází z formátu VRML. Slouží pro popis plošných i prostorových scén. Pro popis využívá stromové struktury, ve které je popsaná celá scéna (scénový graf). Scénový graf je struktura, která logicky řadí popis prostorové scény. Jedná se o sadu uzlů (Nodes) v grafu nebo stromové struktuře. Uzel může mít mnoho dětí, ale pouze jednoho rodiče. Scénový graf v 3D aplikacích a hrách popisuje celé scény, ve kterých uzly popisují jednotlivé objekty.

X3D je otevřený formát popisující pomocí scénového grafu grafickou scénu pomocí jazyka XML. Jedná se o ISO-schválený standard, který poskytuje systém pro ukládání, získávání a přehrávání real-time grafiky. V programu 1 je znázorněný popis jednoduché scény s použitím X3D v prostředí H3DAPI.

Program 1 Jednoduché vytvoření scény s koulí v X3D

```
<Group>
    <Shape>
        <Sphere radius="0.1" />
        <Appearance>
            <Material DEF="MATERIAL" diffuseColor="1 0 0" />
        </Appearance>
    </Shape>
</Group>
```

Nejprve je uvedeno, že se jedná o skupinu. V této skupině vytvoříme objekt. Definujeme, že objekt bude koule s poloměrem "0.1". Ve vlastnostech objektu je nastavena difusní barva na červenou a materiál pojmenovaný jako MATERIAL.

3.3.3 Python

Python je dynamický, objektově orientovaný skriptovací programovací jazyk. Je vyvíjen jako otevřený projekt, který zdarma nabízí instalační balíčky pro většinu běžných platforem. Byl navržen tak, aby umožňoval tvorbu rozsáhlých a plnohodnotných aplikací. K význačným vlastnostem patří jeho jednoduchost a produktivnost z hlediska rychlosti psaní programů. Vlastností, která je využita právě u H3DAPI je, že se Python snadno vkládá do jiných aplikací, kde slouží jako jejich skriptovací jazyk. Tím lze aplikacím dodávat chybějící pružnost. Jiné aplikace nebo aplikační knihovny mohou naopak implementovat rozhraní, které umožní jejich použití v roli pythonského modulu.

V H3DAPI je Python využit jako skriptovací jazyk a umožňuje rozšířit chování objektů a animací ve scéně. Na následujícím programu je ukázka jednoduché aplikace v H3DAPI, kde se při stisknutí tlačítka myši změní barva koule. V programu 2 je zobrazeno propojení X3D s Pythonem.

Program 2 Přidání k programu 1 propojení s Pythonem

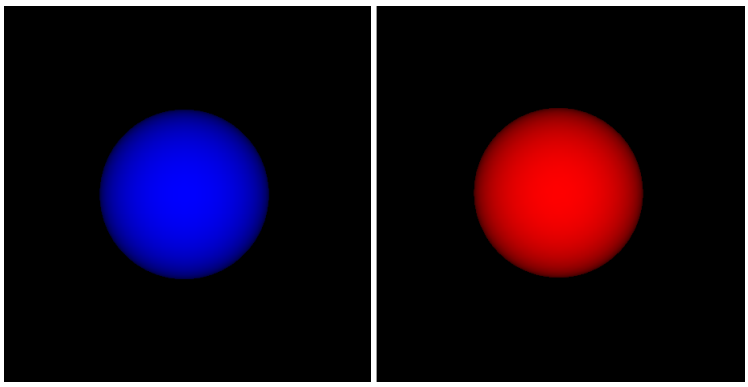
```
<Group>
    <Shape>
        <Sphere radius="0.1" />
        <Appearance>
            <Material DEF="MATERIAL" />
        </Appearance>
    </Shape>
    <PythonScript DEF="PS" url="Sphere.py" />
    <MouseSensor DEF="MS" />
    <ROUTE  fromNode="MS" fromField="leftButton"
            toNode="PS" toField="color" />
    <ROUTE  fromNode="PS" fromField="color"
            toNode="MATERIAL" toField="diffuseColor" />
</Group>
```

Začátek je stejný jako u programu 1 pro X3D. Definovali jsme uzel *MouseSensor*, který nám poskytuje informace o myši. Pomocí *ROUTE* se propojí informace o levém tlačítku myši s naším skriptem. Při stisknutí tlačítka vznikne událost, která zavolá náš skript. Druhý *ROUTE* propojí náš skript s difusním materiálem koule, aby se při vykonání skriptu změnila barva koule. Samotný skript napsaný v jazyce Python je v programu 3.

Program 3 Skript v jazyce Python pro změnu barvy krychle při dotyku

```
from H3DInterface import *

class Color(TypedField(SFColor, SFBool)):

    def update(self, event):

        if ( event.getValue() ):

            return RGB(1, 0, 0)

        else:

            return RGB(0, 0, 1)

color = Color()
```

Nejprve je importováno rozhraní pro komunikaci s H3DAPI. Následně je naprogramována změna barvy v případě stisknutí tlačítka myši. Na obrázku 3.3.2 je zobrazen výsledek pokud nenastane nebo nastane stisk tlačítka.



Obrázek 3.3.2: Výsledek jednoduchého programu. Vlevo bez stisknutého tlačítka myši, vpravo při stisknutém tlačítku myši

Kombinací X3D a Pythonu se eliminuje nutnost použít programovací cyklus na-

psat kód, zkompilovat a otestovat. Tím se velice zkrátí vývojový čas, protože je možné napsaný kód rovnou spustit.

3.3.4 C++

C++ je objektově orientovaný programovací jazyk. Je vyvinut jako rozšíření jazyka C. V jazyku C++ jsou napsány všechny knihovny a uzly v H3DAPI. Vše, co se dá vytvořit pomocí X3D a Pythonu, je také možné napsat pomocí C++ s využitím H3D uzlů a knihoven.
V H3DAPI je C++ využíváno jako rozšiřující a upravující jazyk, se kterým je možné:

- Upravovat existující uzly a pole pro grafiku i haptiku.
- Rozšiřovat H3DAPI dalšími novými uzly.
- Upravovat nebo přidávat haptické renderovací algoritmy, silové efekty a haptické povrchy.

Nevýhodou C++ je celkem pomalý vývojový čas, protože napsání kódu, kompilace a testování zaberou hodně času.
V programu 4 je znázorněn stejný příklad jako v programu 2 a 3. Je tedy možné porovnat složitost a délku programového kódu.

Cmake

Cmake je multiplatformní make. Make je nástroj, který se stará o generování spustitelných programů nebo jiných souborů ze zdrojových kódů programu. CMake se používá k ovládání kompilačního procesu pomocí konfiguračních souborů nezávislých na kompilátoru.

V H3DAPI je Cmake použit pro vygenerování projektu ze zdrojových souborů

Program 4 Ukázkový program v jazyce C++ s využitím knihoven H3DAPI

```
\\ Připojené soubory potřebné pro natavení scene grafu
#include <H3D/X3D.h>
#include <H3D/Scene.h>
#include <H3D/MouseSensor.h>
#include <H3D/Material.h>
#include <H3D/GLUTWindow.h>

using namespace H3D;

class Color : public TypedField< SFColor, SFBool > {
    virtual void update() {
        if( static_cast< SFBool * >(event.ptr)->getValue() )
            value = RGB( 1, 0, 0 );
        else
            value = RGB( 0, 0, 1 );
    }
};

int main(int argc, char* argv[]) {

// Nastavení scény pomocí stringu a metody createX3DNodeFromString
    string theSceneGraphString = "<Group>"
                                 "<Viewpoint position=\"0 0 1\" />"
                                 "     <Shape>"
                                 "        <Appearance>"
                                 "           <Material DEF=\"MATERIAL\" />"
                                 "        </Appearance>"
                                 "       <Sphere radius=\"0.1\" />"
                                 "     </Shape>"
                                 "   <MouseSensor DEF=\"MS\" />"
                                 "</Group>";

// myDefNodes obsahuje funkci pro získání uzlu ze scén grafu poskytnutím DEF jména.
X3D::DEFNodes *myDefNodes = new X3D::DEFNodes();

// createX3DNodeFromString vrátí ukazatel na nejvyšší uzel v stringu
AutoRef< Node > myGroup( X3D::createX3DNodeFromString(
theSceneGraphString, myDefNodes ) );

// Získání uzlů potřebných pro propojení s Pyhonem
MouseSensor *myMouseSensor = static_cast< MouseSensor * > (myDefNodes->getNode("MS"));
Material *myMaterial = static_cast< Material * > (myDefNodes->getNode("MATERIAL"));

// Vytvoření instance třídy Color pro obarvování.
Color *myColor = new Color();

// Nastavení propojení.
myMouseSensor->leftButton->route( myColor );
myColor->route( myMaterial->diffuseColor );

// Vytvoření nové scény.
AutoRef< Scene > scene( new Scene );

// Vytvoření okna pro zobrazení.
GLUTWindow *glwindow = new GLUTWindow;

// přidání okna do scény.
scene->window->push_back( glwindow );

// přidání našeho uzlu, který má být zobrazen na scénu.
scene->sceneRoot->setValue( myGroup.get() );

// start hlavní smyčky
Scene::mainLoop();
```

pro zvolený systém a kompilátor. Po vygenerování projektu je možné vytvořit knihovny a spustitelné soubory.

3.3.5 Dostupné knihovny

Celé H3DAPI se skládá z několika knihoven, kde každá knihovna přidává další rozšiřující funkce do základního balíku.

HAPI

Je knihovna starající se o haptickou vizualizaci a přenášení síly na haptické zařízení. Je možné ji využít samostatně bez H3DAPI jako knihovnu, se kterou je možné rozšířit stávající aplikaci o haptickou vizualizaci. HAPI knihovna je navržena jako modulární, a je velice jednoduché ji rozšířit o další funkce.
Knihovna je nezávislá na zařízení a podporuje velké množství haptických zařízeních dostupných na trhu. Podporované zařízení jsou:

- Zařízení od firmy Sensable.
- Zařízení od firmy Force Dimension.
- Zařízení Falcon od firmy NovInt.
- Zařízení HapticMaster od firmy Moog FCS Robotics.

HAPI obsahuje několik možností jak generovat sílu z 3D objektů. Pro generování síly z 3D grafiky je zapotřebí nejprve zvolit haptický renderer (GodObject, Ruspiny), dále zvolit tvar a vlastnosti povrchu a zaslat informace haptickému zařízení.
Efekty síly v HAPI generují sílu na haptické zařízení z jiného zdroje, než je povrch objektu. Nejčastěji se jedná o závislosti na pozici haptického zařízení. HAPI obsahuje několik naprogramovaných silových efektů, které mohou být použity, a jsou dostačující pro velké množství aplikací.

Volume Haptic Toolkit - VHTK

Knihovna VHTK byla vyvinuta během výzkumného projektu zaměřujícího se na použití haptického zařízení ve volumetrických datech. VHTK rozšiřuje H3DAPI přidáním uzlů pro načítání a zpracování volumetrických dat a vytvoření visuální i haptické vizualizace.
VHTK byl vyvinut od základu tak, aby využil všech vlastností H3DAPI. Nástroj může být použit v souborech X3D, Python nízko-úrovňové programování a další rozšíření v programovacím jazyku C++.
Specifické vlastnosti VHTK:

- Víceúrovňové programování použitím X3D, Pythonu a C++.
- Volumetrická haptika založená na haptických primitivech, které umožňují současné použití aktivních i pasivních haptických uzlů.
- Osm předefinovaných haptických uzlů pro haptickou vizualizaci skalárních i vektorových dat.
- Několik metod volumetrické vizualizace, včetně přímého vykreslení (DVR), hedge-hog a iso-surface.
- Velké množství nastavení haptické i grafické vizualizace.
- Široké spektrum přenosových funkcí pro určení haptických vlastností.

Knihovna VHTK byla vyvinuta k poskytnutí velkého množství haptické zpětné vazby, která není spojena s geometrickým povrchem. To znamená, že není vhodná pro většinu aplikací zaměřujících se na virtuální realitu. Existují ale různá uplatnění, jako vědecká a medicínská vizualizace, simulace píchání jehlou, nebo simulace biopsie a další.

Kapitola 4

Vylepšení haptické detekce volumetrických objektů

Prvním nápadem bylo vytvořit funkci, která by vytvořila ve volumetrických datech stromovou strukturu. Ta by zachovávala důležitá místa jako jsou změny směru vektorů a změny viskozity. Místa shodná by se nahradila jednou hodnotou, která by platila pro danou oblast. Tato struktura by se následně využila při haptické vizualizaci, kde v závislosti na rychlosti pohybu haptického zařízení by se volil stupeň zjednodušení. Při pomalém pohybu by byla zobrazena skutečná data. Při nejrychlejším pohybu by byly zobrazeny pouze důležitá místa. Uživatel by je tak mohl lépe a rychleji identifikovat a pomalým pohybem důkladně prozkoumat. Podobný princip využili pro objekty složené z trojúhelníkové sítě v Brně [KS05], kde využili Level of Detail [1] na objekty. V závislosti na rychlosti pohybu volí stupeň detailu. Pro první prozkoumání stačí uživateli vědět, že se v místě něco nachází a při následném pomalém pohybu se mu zobrazí i detaily.

Popisovaná stromová struktura by byla příliš složitá, mohla by se ztratit nějaká důležitá informace, a nebo by nebylo možné ji dostatečně zjednodušit. Řešením bylo detekovat změny ve volumetrických datech, jako se detekují hrany v obrázku. Pomocí této informace zvýraznit místa změny a zmenšit působení v místech, kde

[1] Level of Detail - v počítačové grafice se jedná o representaci objektů, která při větší vzdálenosti od promítací roviny snižuje množství detailů na objektu.

jsou data neměnná při rychlém pohybu haptického zařízení. Při pomalém pohybu by volumetrická data měla být zobrazena beze změny.
V následujících kapitolách budou uvedeny teoretické postupy, jak bylo dosaženo požadovaného výsledku.

4.1 Detekce změny v datech

Nejprve je nutné detekovat změny v datech. Významné změny se nejčastěji vyskytují v místech, kde se skokově mění hodnota dat (změna směru vektoru, změna skalární hodnoty).
V oblasti zpracování obrazu se pro tento účel používají hranové detektory. Ty detekují velké změny mezi hodnotami pixelů. Výsledkem detektorů je mapa, která označuje místa v obraze, kde se vyskytují změny (hrany).
V našem případě chceme detekovat změny ve volumetrických datech. Pro tento účel lze použít různé hranové detektory. V této práci je použit Sobelův hranový operátor, protože je velice jednoduchý na implementaci.

4.2 Sobelův hranový operátor

Je použit v oblasti zpracování obrazu jako hranový detektor. Technicky se jedná o diskrétní diferenciální operátor, se kterým se vypočítá přibližná hodnota gradientu intenzity v obrazu. Sobelův hranový operátor je založen na konvoluci obrazu s malým, separovatelným filtrem. Pro 2D se používají 2 filtry 3x3, jeden pro detekci v horizontálním směru a druhý pro detekci ve vertikálním směru. Výpočet pomocí Sobelova operátoru je relativně nenáročný.
K výpočtu jsou použity dva 3x3 filtry, které jsou konvolovány s originálním obrazem. Pokud definujeme A jako originální obraz a G_x a G_y jako 2 filtry, bude

výpočet následující:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \tag{4.1}$$

kde $*$ označuje 2-dimensionální konvoluční operátor. Souřadnice v ose x jsou zde definovány jako rostoucí ve směru zleva doprava a souřadnice v ose y jsou definovány jako rostoucí ve směru shora dolů. Pro každý bod obrazu vypočítáme přibližný gradient jako kombinace velikosti gradientů:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{4.2}$$

Pro výpočet v 3D budeme potřebovat rozšířit základní operátor o další filtr, který bude hledat gradient v třetí ose. Následně jsou uvedeny jednotlivé filtry:

$$\underset{x-1}{\begin{bmatrix} -1 & -3 & -1 \\ -3 & -6 & -3 \\ -1 & -3 & -1 \end{bmatrix}} \quad \underset{x}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \quad \underset{x+1}{\begin{bmatrix} 1 & 3 & 1 \\ 3 & 6 & 3 \\ 1 & 3 & 1 \end{bmatrix}}$$

$$\underset{y-1}{\begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix}} \quad \underset{y}{\begin{bmatrix} 3 & 6 & 3 \\ 0 & 0 & 0 \\ -3 & -6 & -3 \end{bmatrix}} \quad \underset{y+1}{\begin{bmatrix} 1 & 3 & 1 \\ 0 & 0 & 0 \\ -1 & -3 & -1 \end{bmatrix}} \tag{4.3}$$

$$\underset{z-1}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{z}{\begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix}} \quad \underset{z+1}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}}$$

Pro každý bod volumetrických dat vypočítáme přibližný gradient, jako kombinace velikosti gradientů, podobně jako u rovnice 4.2:

$$G = \sqrt{{G_x}^2 + {G_y}^2 + {G_z}^2} \tag{4.4}$$

Skalární data

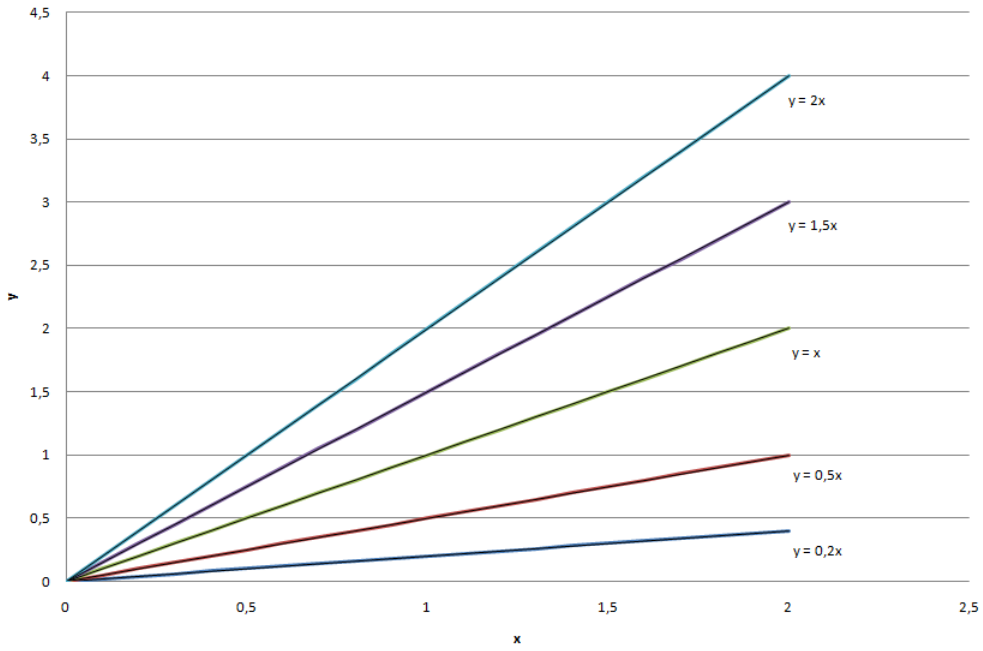
V případě hledání změn ve volumetrických datech se na skalární hodnoty aplikují jednotlivé filtry 4.3 a vypočítá se pro každý voxel rovnice 4.4. Před koncem výpočtu se převedou hodnoty na rozsah $0 - 1$. Výsledkem je volumetrická mapa, kde hodnota 1 znamená největší přechod mezi daty a hodnota 0 znamená, že jsou všechna data v okolí bodu stejná. Tyto hodnoty se uplatní při výpočtu velikosti síly na haptické zařízení.

Vektorová data

U volumetrických dat je situace složitější, protože jeden voxel obsahuje 3 složky vektoru. Pro zjištění změn nad vektorovými daty se provede aplikování filtrů a výpočet 4.4 na každou složku vektoru. Na konci výpočtu se převedou hodnoty na rozsah $0 - 1$. Tímto výpočtem dostaneme vektorová volumetrická data, kde každý voxel obsahuje informaci, jak se mění data v osách x,y,z.

4.3 Aplikace detekovaných změn a rychlosti pohybu

Po detekování změn, je zapotřebí nalézt funkci, která v závislosti na rychlosti pohybu haptického zařízení a detekovaných změnách bude upravovat výslednou haptickou sílu. Úprava by měla být taková, aby při rychlém pohybu byly zvýrazněné pouze změny (hrany). Aby při pomalém pohybu byly původní data zobrazeny beze změny. S rychlostí pohybu se mají data postupně měnit, jelikož může docházet ke skokové změně, která zmate uživatele.



Obrázek 4.3.1: Vliv parametru k na sklon přímky.

Úkolem bylo vymyslet funkci, která má jako vstup dva parametry (rychlost pohybu a detekovanou mapu). Výsledkem měla být jedna hodnota, kterou by se vynásobila originální data a podle hodnot parametrů by se na haptické zařízení aplikovala požadovaná síla. Postupně vznikalo několik funkcí, které neposkytovali požadované vlastnosti a vyžadovali pomocné výpočty. Nakonec byla vymyšlena velice jednoduchá a efektivní funkce, která využívá směrnicovou rovnici přímky. Základní směrnicová rovnice přímky má tvar:

$$y = k * x + q \tag{4.5}$$

kde, x je hodnota na ose x, y je výsledná hodnota na ose y, k je směrnice přímky určující její sklon a q je proměnná určující, kde přímka protne osu y, nebo-li se jedná o horizontální posun přímky.

Pokud budeme měnit hodnotu proměnné k, bude se měnit sklon přímky a přímka bude stále protínat osu y ve stejném bodě (viz. Obrázek 4.3.1). Pokud místo hodnoty k dosadíme hodnotu aktuální rychlosti pohybu haptického zařízení, kterou označíme *velocity* a na osu x naneseme hodnoty z detekované mapy, které

označíme jako *detect*, dostaneme následující rovnici:

$$y = velocity * detect \tag{4.6}$$

Při maximální rychlosti se na haptické zařízení aplikují data v místech, kde je změna. Tam kde je změna největší (v detekované mapě má hodnotu 1) aplikují se data v plné velikosti. V místech, kde je menší změna se originální data úměrně zmenší podle detekovaných změn. V místech, kde jsou data stejná, nebude na zařízení působit žádná síla. Při rychlém pohybu se funkce chová správně. Problém nastane při pomalém pohybu. Proměnná k se bude blížit k nule, přímka bude ležet na ose x a protínat osy y v bodě nula. Tím všechny výstupní hodnoty budou nula a žádná data se nezobrazí. V našem případě potřebujeme, aby se při malé hodnotě rychlosti použila celá originální data.

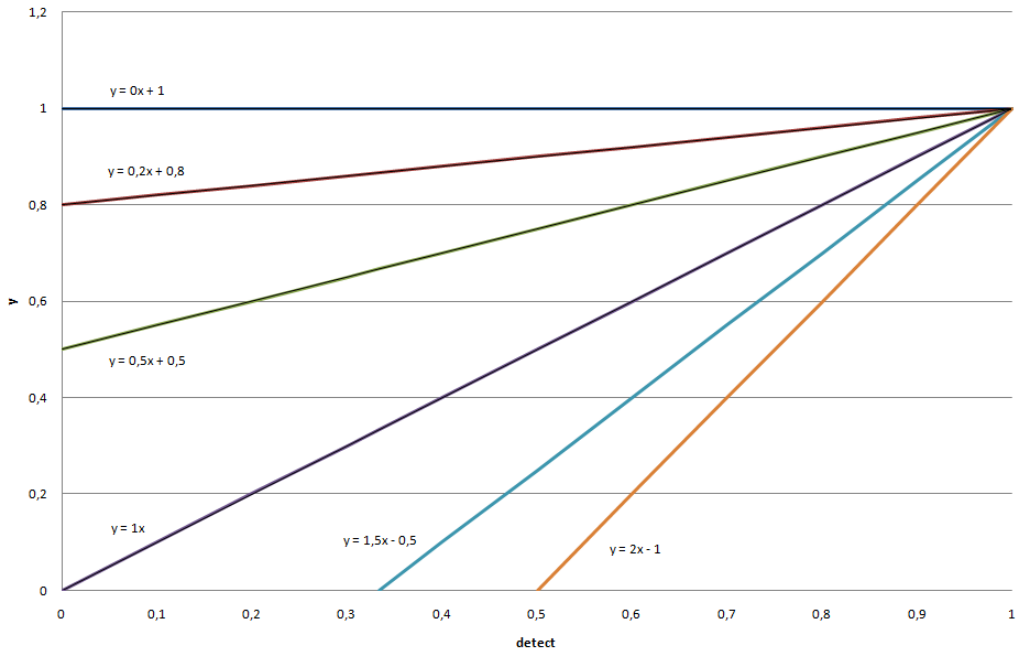
Zatím jsme nebrali v úvahu proměnou q, která zajišťuje posun v ose y. Potřebujeme upravit rovnici 4.6 tak, aby při nulové rychlosti byl výsledek roven 1. Toho dosáhneme tak, že k rovnici přičteme hodnotu $1 - velocity$. Nová rovnice bude vypadat následovně:

$$y = velocity * detect + (1 - velocity) \tag{4.7}$$

Při změně rychlosti se bude měnit sklon přímky i posun po ose y. Dosadíme-li si do rovnice za rychlost hodnotu 0, znamenající pomalý pohyb bude výsledkem rovnice 1. Při rychlém pohybu (hodnota 1) se přičtený člen vyruší a výsledek bude stejný jako u rovnice 4.6. Rychlost haptického zařízení může být ve skutečnosti větší než 1. To by znamenalo, že výsledek může být záporný a při aplikaci na haptické zařízení by se objevil opačný efekt než požadujeme. Tomu ale jednoduše můžeme zabránit tak, že přidáme podmínku:

$$if\ y < 0\ then\ y = 0 \tag{4.8}$$

Chování výsledné rovnice 4.7 je znázorněno na následujícím obrázku 4.3.2. Z obrázku 4.3.2 je vidět, že se křivka mění podle požadavků, tzn. při pomalém pohybu (velocity rovno 0), je hodnota funkce 1. Původní hodnoty dat se nemění. Při rychlém pohybu se zobrazí pouze přechody.

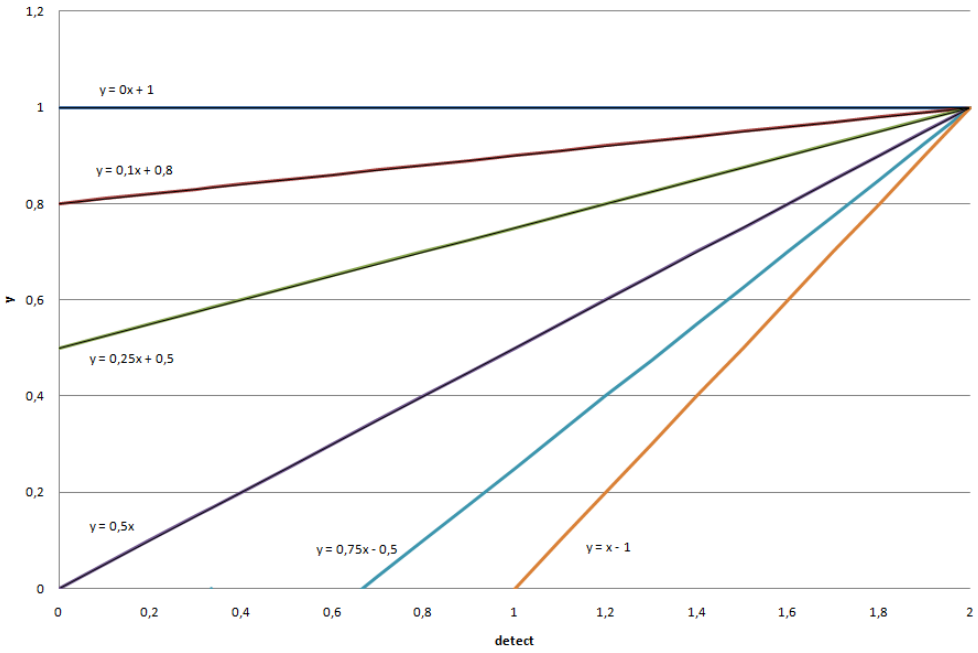


Obrázek 4.3.2: Rovnice 4.7 při různých parametrech rychlosti

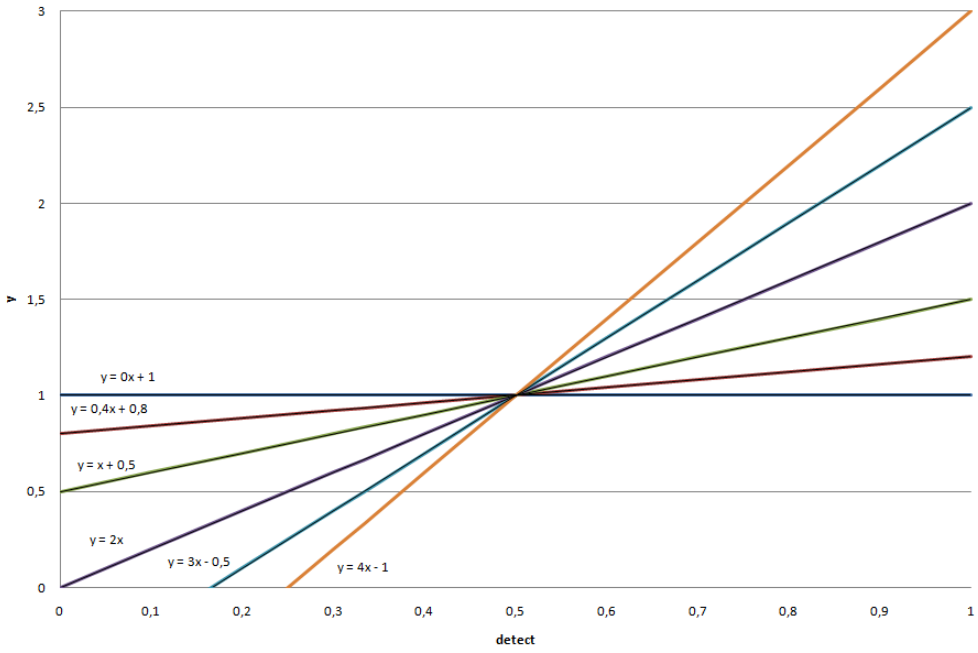
Do výsledné rovnice 4.9 přibude ještě jeden nastavitelný parametr *strength*, který mění strmost křivek, a tím zvětšuje nebo zmenšuje znatelnost přechodů v datech.

$$y = strength * velocity * detect + (1 - velocity) \tag{4.9}$$

V obrázku 4.3.3 je hodnota $strength = 0,5$, v obrázku 4.3.2 je hodnota $strength = 1$ a v obrázku 4.3.4 je hodnota $strength = 2$.



Obrázek 4.3.3: Rovnice 4.9 při různých parametrech rychlosti a parametru $strength = 0,5$



Obrázek 4.3.4: Rovnice 4.9 při různých parametrech rychlosti a parametru $strength = 2$

Kapitola 5

Implementace v H3DAPI

Programové prostředí bylo zvoleno H3DAPI s knihovnou VHTK, která obsahuje uzly pro práci se skalárními i vektorovými uzly. Prvním úkolem bylo zjištění možností knihovny VHTK a funkčnosti jejích uzlů, a které podle popisu mohly být důležité pro naši implementaci. Hledaly se uzly, které mají na starost ukládání skalárních a vektorových volumetrických dat pro načtení vstupních dat. Dále uzly, které dokáží přečíst hodnotu z volumetrických dat v místě, kde je haptické zařízení a danou sílu použít. Pro skalární data se hledal uzel, který simuluje pohyb v různě hustém prostředí. To definujeme skalární hodnotou načtenou z volumetrických dat. Všechny požadované uzly se podařilo v knihovně VHTK nalézt a zjistit jejich funkčnost a možnosti. Jejich popis je v sekci 5.1.
Dále v této kapitole v sekci 5.2 popíšeme postup vytvoření knihoven pro H3DAPI, které přidávají nové uzly do formátu X3D. Postup ukážeme na skalárních datech. Pro vektorová data platí velice podobný postup. Rozdíl je, že se dědí od jiných tříd a program je odlišný z důvodu použití na vektorech, ale v principu stejný.

5.1 Důležité VHTK uzly

Pokud chceme vytvořit nový uzel, můžeme začít hned. Bude zde ukázán postup, který vývoj zjednoduší, protože nebude nutné psát velké množství řádek programového kódu. Přesně to je výhodou H3DAPI, které se tímto přístupem snaží

zrychlit vývojový cyklus.

Dále popíšeme základní funkčnost uzlů. Přesný popis je v dostupné dokumentaci.

5.1.1 ScalarVolume, VectorVolume

Jako první je nutné umět načíst skalární, nebo vektorová data. K tomu slouží uzly **ScalarVolume**, případně **VectorVolume**. Uzly mají parametr *url*, který definuje jméno souboru, který se má načíst do pole. Oba uzly musí ve svém těle obsahovat uzel **ImageLoader**. Ten zajišťuje načtení souboru. V H3DAPI existuje několik druhů ImageLoadrů, které umožňují načíst různé typy souborů. V našem případě byl využit **RawImageLoader**, který má na starost načítání z binárních souborů. V části programu 5 je ukázáno použití **ImageLoaderu** s uzlem **ScalarVolume**.

Program 5 Použití uzlu *RawImageLoader* s uzlem *ScalarVolume*

```
<Group>
    <ScalarViscosityMode>
        <ScalarVolume url="file.bin">
             <RawImageLoader width="100" height="100" depth="100"
                      pixelType="LUMINANCE"
                             bitsPerPixel="32"
                      pixelComponentType="RATIONAL"
                      pixelSize="0.003 0.003 0.003" />
 </ScalarVolume>

        <PiecewiseFunction containerField="viscosity"
                           continuous="true"
                 segments="     0.00 0.0
                                1.00 1.0
                                2.00 2.0 "/>
    </ScalarViscosityMode>
</Group>
```

Uzel **ScalarVolume** nemůže být použit bez nadřazeného uzlu, kterým je **ScalarViscosityMode** popsaný v následující sekci 5.1.2. Uzel **RawImageLoader** má několik parametrů popisující data. První tři parametry určují počet prvků v každé ose. Parametr *pixelType* určuje, jakého druhu je pixel. Hodnoty, které lze nastavit jsou: LUMINANCE, LUMINANCE_ALPHA, RGB, RGBA, BGR, BGRA a VEC3. Se **ScalarVolume** používáme nejčastěji hodnotu LUMINANCE. Parametr *bitsPerPixel* učuje, kolik místa v paměti zabere jeden pixel. Parametrem *pixelComponentType* definuje o jaký typ hodnoty se jedná, zda jde o SIGNED,

UNSIGNED nebo RATIONAL. Parametr *pixelSize* je volitelný parametr, který určuje rozměr každého pixelu v metrech. To může být užitečné pro volume rendering, aby se dali vypočítat rozměry v prostoru.

5.1.2 ScalarViscosityMode

Uzel **ScalarViscosityMode** simuluje viskózní prostředí při pohybu haptickým zařízením uvnitř volumetrických dat. Je použit v programu 5, kde slouží jako rodičovský uzel pro **ScalarVolume**. Sám má dva vstupní uzly. Jedním je **ScalarVolume**, který obsahuje volumetrická skalární data. Druhý uzel je **PiecewiseFunction** definující převodní funkci ze skalárních hodnot na sílu pro haptické zařízení. Parametr *containerField* definuje, že mapovací funkce se má uložit do proměnné *viscosity* v uzlu **ScalarViscosityMode**.
Při průchodu volumetrickými daty, se podle pozice haptického zařízení načte hodnota z dat. Mapovací funkce určí viskozitu prostředí, a ta se použije proti směru pohybu haptického zařízení.

5.1.3 VectorForceMode

Uzel **VectorForceNode** je v použití velice podobný uzlu **ScalarViscosityVolume**. Potřebuje pro svoji činnost dva uzly, **VectorVolume** pro uložení vektorových dat a mapovací funkci pro určení síly vektorů. Uzel pracuje tak, že podle pozice haptického zařízení načte vektor z volumetrických dat, upraví jeho velikost podle mapovací funkce a aplikuje na haptické zařízení.

5.2 Tvorba knihoven

K vytvoření knihovny potřebujeme znát, od jakých tříd budeme dědit, a jaké parametry je potřeba předávat, aby se vytvořená knihovna chovala jako uzel v X3D struktuře. My jsme si nalezli čtyři uzly, které jsou funkčností podobné našim požadavkům a podle nich vytvoříme nové uzly s požadovanou funkčností.

Program 6 Ukázka CMake souboru pro vytvoření nového uzlu pro knihovnu VHTK

```
cmake_minimum_required(VERSION 2.6.0)
project(ScalarViscosityVelocity)

# Kde hledat zdrojove soubory
SET( ScalarViscosityVelocity_SRCS
         "${ScalarViscosityVelocity_SOURCE_DIR}/../src/ScalarViscosityVelocity.cpp")
SET(optionalLibs)                    #volitelné knihovny
INCLUDE_DIRECTORIES( ../src )        #zahrnout adresáře.
INCLUDE_DIRECTORIES( ../include )
SET(requiredLibs)                    #potřebné knihovny

SET( CONVERTED_H3D_ROOT "" )         # převod \ na / ve win
IF( EXISTS $ENV{H3D_ROOT} )
FILE( TO_CMAKE_PATH $ENV{H3D_ROOT} CONVERTED_H3D_ROOT )
ENDIF( EXISTS $ENV{H3D_ROOT} )

#kde nalezt moduly pro vyhledani knihoven
SET(CMAKE_MODULE_PATH "${CONVERTED_H3D_ROOT}/build/modules"
                      "${ScalarViscosityVelocity_SOURCE_DIR}/modules" )

FIND_PACKAGE(H3DAPI REQUIRED)        #H3DAPI
IF(H3DAPI_FOUND)
INCLUDE_DIRECTORIES( ${H3DAPI_INCLUDE_DIR} )
SET(requiredLibs ${requiredLibs} ${H3DAPI_LIBRARIES} )
ENDIF(H3DAPI_FOUND)

FIND_PACKAGE(H3DUTIL REQUIRED)       #H3DUtil
IF(H3DUTIL_FOUND)
INCLUDE_DIRECTORIES( ${H3DUTIL_INCLUDE_DIR} )
SET(requiredLibs ${requiredLibs} ${H3DUTIL_LIBRARIES} )
ENDIF(H3DUTIL_FOUND)

FIND_PACKAGE(HAPI REQUIRED)          #HAPI
IF(HAPI_FOUND)
INCLUDE_DIRECTORIES( ${HAPI_INCLUDE_DIR} )
SET(requiredLibs ${requiredLibs} ${HAPI_LIBRARIES} )
ENDIF(HAPI_FOUND)

FIND_PACKAGE(VHTK REQUIRED)          #VHTK
IF(VHTK_FOUND)
INCLUDE_DIRECTORIES( ${VHTK_INCLUDE_DIR} )
SET(requiredLibs ${requiredLibs} ${VHTK_LIBRARIES} )
ENDIF(VHTK_FOUND)

# Definice kompilačních flagů pro všechny nastavení, a pak přidání do nich
SET( ScalarViscosityVelocity_COMPILE_FLAGS "" )
SET( ScalarViscosityVelocity_COMPILE_FLAGS "${ScalarViscosityVelocity_COMPILE_FLAGS}
         -DCUTTING_EXPORTS" )

# sdílené knihovny
ADD_LIBRARY( ScalarViscosityVelocity SHARED ${ScalarViscosityVelocity_SRCS} )
#propojit proti knihovnám
TARGET_LINK_LIBRARIES( ScalarViscosityVelocity ${requiredLibs} ${optionalLibs} ${EXTRA_LIBS} )

#nastavení debug verse, změna jména knihovny
SET_TARGET_PROPERTIES( ScalarViscosityVelocity PROPERTIES DEBUG_POSTFIX "_d")
#nastavení instalačního adresáře
SET( CMAKE_INSTALL_PREFIX ${ScalarViscosityVelocity_SOURCE_DIR}/../.. CACHE PATH
                  "Install path prefix, prepended onto install directories." FORCE )
# nastavení kompilačních flags pro projekt
SET_TARGET_PROPERTIES( ScalarViscosityVelocity PROPERTIES COMPILE_FLAGS
                      "${ScalarViscosityVelocity_COMPILE_FLAGS}" )

# Nainstalovat do následujících adresářů
INSTALL( TARGETS ScalarViscosityVelocity
         LIBRARY DESTINATION lib
         RUNTIME DESTINATION bin
         ARCHIVE DESTINATION lib )
```

Protože je H3DAPI otevřený projekt, jsou všechny zdrojové soubory volně dostupné a je možné se z nich inspirovat a naučit jak H3DAPI pracuje. Dále popíšeme postup vytvoření dvou knihoven. Jedna se stará o detekci změn ve volumetrických datech, druhá tyto data využívá společně s informací o rychlosti a generuje haptickou sílu.
Jako první krok je ale nutné vytvořit projekty. K tomuto účelu se používá nástroj CMake.

5.2.1 CMake

Cmake byl už krátce popsán v sekci 3.3.4. Abychom mohli nástroj CMake použít, musíme vytvořit textový soubor s názvem *CMakeLists.txt*. V něm jsou definovány cesty ke knihovnám, cesty ke zdrojovým souborům a nastavení projektu. V programu 6 je ukázán soubor pro vytvoření uzlu **ScalarViscosityNode**. Na první řádce jsou minimální požadavky na verzi programu Cmake. Na druhém řádku je založení projektu a jeho jméno. Jednotlivé příkazy jsou podrobně popsány pomocí komentářů.
Po vytvoření textového souboru použijeme program CMake, kde zadáme cestu k vytvořenému souboru a vybereme adresář, do kterého se má projekt vytvořit. V našem případě byl použit kompilátor od Microsof Visual Studio 2010. Nástroj CMake vytvoří všechny potřebné soubory a nastaví projekt tak, že ho po vytvoření jde rovnou načíst do Visual Studia a začít programovat. Všechny závislosti a cesty ke knihovnám jsou nastaveny.
Máme-li vytvořen projekt, můžeme začít vytvářet nový uzel. Jako první potřebujeme vytvořit uzel, který detekuje ve volumetrických datech.

5.2.2 Detekování změn

Princip detekce Sobelovým hranovým operátorem byl popsán v sekci 4.2. V této sekci ukážeme jeho implementace v C++ tak, že jej můžeme použít i v X3D souboru. Programový kód je rozdělen na hlavičkový soubor .h a zdrojový soubor .cpp.

Program 7 Detekce změn pomocí Sobelova hranového operátoru, hlavičkový soubor .h

```
#ifndef _VHTK_SCALARDETECTORNODE_HH_
#define _VHTK_SCALARDETECTORNODE_HH_

#include <VHTK/VHTKScalarImageNode.hh>
#include <H3D/H3DImageLoaderNode.h>
#include <H3D/X3DUrlObject.h>
#include <H3D/MFNode.h>

namespace VHTK {

  class ScalarDetectorNode :
    public VHTKScalarImageNode,
    public H3D::X3DUrlObject {
  public:
    // CouldNotLoadImage vyjímka je vyhozena, pokud žádny Loader nedokáže načíst volumetrická data.
    H3D_API_EXCEPTION( CouldNotLoadImage );

    // MFNode obsahující ImageLoader.
    typedef H3D::TypedMFNode< H3D::H3DImageLoaderNode > MFImageLoader;

    // SFImage je přepsán, aby aktualizoval hodnoty z volumetrických dat načtené z url pomocí ImageLoadru
    class SFImage:
      public H3D::TypedField< VHTKScalarImageNode::SFImage,
                              H3D::Types< H3D::MFString, MFImageLoader > > {
    protected:
      // Aktualizuje hodnoty z Image3DTexture načtené ImageLoaderem z url
      virtual void update();
    private:
static const unsigned int kernelLength = 3; //velikost vektoru
static const unsigned int kernelSize = 9; //velikost jadra
    };

    // Constructor.
    ScalarDetectorNode( H3D::Inst< H3D::SFNode   > _metadata     = 0,
                  H3D::Inst< H3D::Field    > _dataChanged  = 0,
                  H3D::Inst< H3D::SFBool   > _scaleVectors = 0,
                  H3D::Inst< SFImage       > _image        = 0,
                  H3D::Inst< H3D::MFString > _url          = 0,
                  H3D::Inst< MFImageLoader > _imageLoader  = 0 );

    // Uzel ImageLoader pro čtení dat.
    auto_ptr< MFImageLoader > imageLoader;

    // H3DNodeDatabase pro tento uzel.
    static H3D::H3DNodeDatabase database;
  };
}
#endif
```

Zdrojový soubor je rozdělen do třech menších částí pro lepší přehlednost a lepší popis.

Začneme s hlavičkovým souborem. Ten je zobrazen v programu 7. Na začátku jsou připojeny potřebné knihovny z H3D a VHTK knihovny. Vytvoříme naši třídu, kterou dědíme od abstraktní třídy **VHTKScalarImageNode** a **X3DUrlObject**. Dále jsou definovány potřebné proměnné a ukazatele.

V následujícím programu 8 je první část zdrojového kódu pro detekci změn ve volumetrických datech. Na začátku je připojen hlavičkový soubor, který je uveden v programu 7. Jsou použity dva jmenné prostory H3D a VHTK. Důležité je přidat náš nový uzel do databáze všech uzlů, aby mohl být použit v X3D. Dále následuje vytvoření vstupně výstupních polí pro parametry. První parametr obsahuje url pro zdrojová data, druhý parametr bude obsahovat zvolený **ImageLoader** pro načítání dat.

První část programu obsahovala konstruktor třídy, vstupní parametry a přidání našeho uzlu do databáze uzlů. Druhá část programu 9 obsahuje na začátku deklaraci filtrů Sobelova hranového operátoru. Pak následuje získání ukazatele na **ImageLoader** a *url*. Načítání je prováděno tak, že je možné zvolit více cest a více **imageLoaderů**, použije se ale pouze jeden a to ten, který dokáže data přečíst. Dále se vytvoří prázdný obrázek, do kterého se uloží načtené hodnoty.

Pro rozsáhlost kódu není uveden výpočet konvoluce. Jedná se o postupné násobení Sobelovým hranovým 3D operátorem. Výsledkem jsou volumetrická data, která v místech, kde jsou data stejná mají hodnotu 0 a v místech, kde dochází ke změně mají hodnotu v rozmezí $0 - 1$ podle velikosti změny.

Detekce změny pro vektorová data je velice podobná. Sobelovým hranovým operátorem se vynásobí každá složka vektoru, výsledkem jsou vektorová volumetrická data, kde složky vektoru obsahují hodnotu, jak se daná složka mění.

5.2.3 Aplikace na haptické zařízení

Vypočtením změn ve volumetrických datech můžeme přistoupit k aplikaci těchto změn na haptické zařízení. Jak bylo popsáno v Kapitole 4 sekci 4.3 použijeme

Program 8 Detekce změn pomocí Sobelova hranového operátoru, zdrojový soubor .cpp, 1. část

```
#include "scalarDetectorNode.h"
#include <VHTK/Debug.hh>

using namespace H3D;
using namespace VHTK;

// Přidání tohoto uzlu do seznamu všech uzlů
H3DNodeDatabase ScalarDetectorNode::database( "ScalarDetectorNode", &(newInstance<ScalarDetectorNode>),
                                        typeid( ScalarDetectorNode ), &VHTKScalarImageNode::database );

// vstupn2 v7stupn9 parametry
namespace ScalarDetectorNodeInternals {
  FIELDDB_ELEMENT( ScalarDetectorNode, url, INPUT_OUTPUT );
  FIELDDB_ELEMENT( ScalarDetectorNode, imageLoader, INPUT_OUTPUT );
}

//konstruktor
ScalarDetectorNode::ScalarDetectorNode
( Inst< SFNode        > _metadata,
  Inst< Field         > _dataChanged,
  Inst< SFBool        > _scaleVectors,
  Inst< SFImage       > _image,
  Inst< MFString      > _url,
  Inst< MFImageLoader > _imageLoader ) :
  VHTKScalarImageNode( _metadata,
                       _dataChanged,
                       _scaleVectors,
                       _image ),
  X3DUrlObject( _url ),
  imageLoader( _imageLoader ) {

  type_name = "ScalarVolume";
  database.initFields( this );

  url->routeNoEvent( image );
  imageLoader->routeNoEvent( image );
}


  ...
```

Program 9 Detekce změn pomocí Sobelova hranového operátoru, zdrojový soubor .cpp, 2. část

```
void ScalarDetectorNode::SFImage::update() {
// 3 Sobel filters (kernels)
Vec3f kernel3y[kernelSize] = {Vec3f(-1, -3, -1), Vec3f(-3, -6, -3), Vec3f(-1, -3, -1),
                              Vec3f(0, 0, 0), Vec3f(0, 0, 0), Vec3f(0, 0, 0),
                              Vec3f(1, 3, 1), Vec3f(3, 6, 3), Vec3f(1, 3, 1)};
Vec3f kernel3x[kernelSize]= { Vec3f(-1, -3, -1), Vec3f(0, 0, 0), Vec3f(1, 3, 1),
                              Vec3f(-3, -6, -3), Vec3f(0, 0, 0), Vec3f(3, 6, 3),
                              Vec3f(-1, -3, -1), Vec3f(0, 0, 0), Vec3f(1, 3, 1) };
Vec3f kernel3z[kernelSize]= { Vec3f(-1, 0, 1), Vec3f(-3, 0, 3), Vec3f(-1, 0, 1),
                              Vec3f(-3, 0, 3), Vec3f(-6, 0, 6), Vec3f(-3, 0, 3),
                              Vec3f(-1, 0, 1), Vec3f(-3, 0, 3), Vec3f(-1, 0, 1) };

  //ukazatel na imageLoader a na url
  MFImageLoader *image_loaders = static_cast< MFImageLoader * >( routes_in[1] );
  MFString *urls = static_cast< MFString * >( routes_in[0] );

  for( MFString::const_iterator i = urls->begin(); i != urls->end(); ++i )
  {
      for( MFImageLoader::const_iterator il = image_loaders->begin();
                      il != image_loaders->end(); il++ )
      {
            Image *image = static_cast< H3DImageLoaderNode * >(*il)->loadImage( *i );
//velikost zabrané paměti
            unsigned int size = image->width() * image->height() *
                                      image->depth() * ( image->bitsPerPixel() / 8 );
//data pro uložení
            unsigned char * data = new unsigned char[size];
            PixelImage *px = new PixelImage(image->width(), image->height(),
                                      image->depth(), image->bitsPerPixel(),
                                      image->pixelType(), image->pixelComponentType(),
                                      data, false, image->pixelSize());

            size = image->width() * image->height() * image->depth();

            int indexData, indexKernel; //index v 1d poli

//pomocné proměné pro výpočet
            Vec3f *tempX = new Vec3f[size];
            H3DFloat *out = new H3DFloat[size];
            H3DFloat v;

            int x,y,z; //pozice
            H3DFloat maximum = 0;

... // konvoluce nactenych dat a Sobelova 3D operatoru

            if( image )
            {
               VHTK_DEBUG_LOG_I("Image value updated");
               value = px;   //ulozeni detekovanych zmen
               return;
            }
        }
    }
} //end program
```

k výpočtu rovnici 4.9. Potřebujeme načíst originální a detekovaná volumetrická data, potom zjistit hodnotu rychlosti pohybu, upravit výslednou sílu a aplikovat ji na haptické zařízení.

Hlavičkový soubor pro aplikaci detekovaných dat na haptické zařízení je velice krátký (Program 10). Na začátku se připojí potřebné hlavičkové soubory. Vytvoří se třída, která je děděná od abstraktní třídy **VHTKScalarModeNode**. Deklarují se ukazatele na potřebné vstupní proměnné.

Program 10 Aplikace na haptické zařízení,hlavičkový soubor .cpp

```
#ifndef _VHTK_SCALARVISCOSITYVELOCITY_HH_
#define _VHTK_SCALARVISCOSITYVELOCITY_HH_

#include <VHTK/VHTK.hh>

#include <VHTK/VolumeHaptics.hh>
#include <VHTK/VHTKScalarModeNode.hh>

namespace VHTK
{

class ScalarViscosityVelocity : public VHTKScalarModeNode {
  public:
    // Initializace vnitřních promněných
    ScalarViscosityVelocity();

    // systémová funkce
    void getPrimitives( primitives_t &r, const state &s );
    //mapovací funkce pro sílu viskozity
    auto_ptr< TSSFNode( SFTransferFunction ) > viscosity;
    auto_ptr< H3D::SFFloat > edgeStrenght;
    auto_ptr< H3D::SFBool > withEdgeVelocity;
    auto_ptr< MFScalarDataNode > detectVolume;

    // X3D interface
    static H3D::H3DNodeDatabase database;
    // Initializace funkcionality uzlu */
    void initialize();
  };
}
#endif
```

Program 11 zobrazuje kód starající se o aplikaci výpočtů na haptické zařízení. V první části programu je konstruktor a metoda pro inicializaci, která se spustí pouze jednou na začátku. Nejdůležitější částí je metoda *GetPrimitives*, která haptickému zařízení odešle sílu, kterou má aplikovat. V této metodě se nejprve získá hodnota mapovací síly pro viskozitu. Zjistí se jestli se má použít zvýraznění hran. Uloží se aktuální hodnota o rychlosti pohybu. Získá se hodnota z detekovaných volumetrických dat a hodnota určující rychlost změny sklonu křivky. Všechny

hodnoty se dosadí do rovnice 4.9. Výslednou hodnotou se vynásobí původní volumetrická hodnota. Haptickému zařízení se odešle primitivum, které obsahuje směr a sílu působení. V případě vektorových dat je situace velice podobná. Rovnicí se vynásobí každá složka původního vektoru a výsledek se aplikuje na haptické zařízení.

Po seznámení se s prostředím H3DAPI a knihovnou VHTK je implementace velice jednoduchá a rychlá. Detekce změn v datech se provádí pouze při spuštění programu. Během vykonávání se pouze čtou hodnoty z dat a pomocí rovnice 4.9 se přepočítávají a aplikují na původní volumetrická data. Výpočet je jednoduchý a nezpomaluje vykreslování a aplikování síly na haptické zařízení.

Vygenerované knihovny nelze přenášet, protože záleží na umístění souborů z H3DAPI a dalších knihoven. Je proto důležité pomocí CMake vytvořit projekt a zkompilovat knihovny na počítači, kde se mají použít. Musí také být zkompilovány stejným kompilátorem jako H3DAPI a další knihovny.

5.3 Použití v H3DAPI

Knihovny lze použít v H3DAPI jako nové uzly. V programu 12 je zobrazen výsledný program, který využívá obě vytvořené knihovny. Na začátku programu jsou informace o XML souboru. Hned po vytvoření scény se importují vytvořené knihovny. Následuje nastavení použití volumetrické haptické vizualizace. Dále vytvoříme náš uzel **ScalarViscosityVelocity** a nastavíme mu potřebné parametry. Parametrem *edgeStrenght* se nastaví rychlost změny sklonu křivky. Parametrem *withEdgeVelocity* se zapíná použití zvýraznění hran. V našem uzlu musí být uzel načítající data a uzel pro detekci změn. Uzel pro detekci změn se jmenuje **ScalarDetectorNode**, který obsahuje adresu na data a **ImageLoader**. Druhý uzel **ScalarVolume** uchovává původní volumetrická data. Uzel **PiecewiseFunction** definuje mapovací funkci volumetrických dat na sílu určující viskozitu.

Program 11 Aplikace na haptické zařízení, zdrojový soubor .cpp

```
#include "scalarViscosityVelocity.h"

using namespace H3D;
using namespace HAPI;
using namespace VHTK;

H3DNodeDatabase ScalarViscosityVelocity::database
( "ScalarViscosityVelocity",
  &(newInstance<ScalarViscosityVelocity>),
  typeid( ScalarViscosityVelocity ),
  &VHTKScalarModeNode::database );

namespace ScalarViscosityVelocityInternals {
  FIELDDB_ELEMENT( ScalarViscosityVelocity, viscosity, INPUT_OUTPUT );
  FIELDDB_ELEMENT( ScalarViscosityVelocity, edgeStrenght, INPUT_OUTPUT );
  FIELDDB_ELEMENT( ScalarViscosityVelocity, detectVolume, INPUT_OUTPUT );
  FIELDDB_ELEMENT( ScalarViscosityVelocity, withEdgeVelocity, INPUT_OUTPUT );
}

ScalarViscosityVelocity::ScalarViscosityVelocity()
  : viscosity( new TSSFNode( SFTransferFunction ) ),
    edgeStrenght( new SFFloat(1)  ),
    withEdgeVelocity( new SFBool(1)),
    detectVolume( new MFScalarDataNode ) {
  type_name = "ScalarViscosityMode";
  database.initFields( this );
  viscosity->setValue( NULL );
  edgeStrenght->setValue( 1 );
}

void ScalarViscosityVelocity::initialize(){
  VHTKScalarModeNode::initialize();

  if( !viscosity->getValue() ){
    VHTK_RUNTIME_WARNING( "ScalarViscosityMode::initialize()",
                          "Transfer Function 'viscosity' not specified" ); }
}

void ScalarViscosityVelocity::getPrimitives( primitives_t &plist, const state &s )
{
  VHTKTransferFunctionNode *visc_transfer = viscosity->getValue();

  if( visc_transfer ) {
      float scalar = getScalarAtProxy(s);

      if(withEdgeVelocity->getValue())
  {
        HAPIFloat vel = s.probe_v.length();
        HAPIFloat detect = detectVolume->getValueByIndex(int(s.frame))->
         ->getScalar(Matrix4f(s.T), Vec3f(s.proxy));

        HAPIFloat veloc = edgeStrenght->getValue() * vel*detect + (1 - vel);
        if (veloc < 0)
            veloc = 0;

        plist.push_back( primitive( s.proxy,
                         visc_transfer->get(scalar)*(veloc),
                         PRIMITIVE_POINT ) );
      }
      else
        plist.push_back( primitive( s.proxy,
                         visc_transfer->get(scalar),
                         PRIMITIVE_POINT ) );
  }
}
```

Program 12 Použití knihoven v H3DAPI

```
<?xml version="1.0" encoding="utf-8"?>
<X3D profile='Full' version='3.2'>
    <head>
        <meta name='title' content='HaptNode.x3d'/>
        <meta name='author' content='Z.Hamak 2011'/>
    </head>
    <Scene>
        <!-- import vytvorenych knohoven -->
        <ImportLibrary library="bin\\ScalarViscosityVelocity" />
        <ImportLibrary library="bin\\ScalarDetectorNode" />

        <VolumeHaptics DEF="HAPTICS" />

        <!-- Detekce hran pri rychlem pohybu -->
        <ScalarViscosityVelocity active="true" edgeStrenght="5" withEdgeVelocity="true">
            <!-- Pouziti Sobel3D operatoru -->
            <ScalarDetectorNode DEF="SDVOLUME" containerField="detectVolume" url='"data/testa.bin"'>
                <RawImageLoader DEF="LOAD" width="100" height="100" depth="100"
                                bitsPerPixel = "32"
                                pixelType = "LUMINANCE"
                                pixelComponentType = "RATIONAL"
                                pixelSize="0.003 0.003 0.003" />

            </ScalarDetectorNode>
            <!-- Skalarni data -->
            <ScalarVolume DEF="SVOLUME" url='"data/testa.bin"'>
                <RawImageLoader USE="LOAD"/>

            </ScalarVolume>

            <PiecewiseFunction containerField="viscosity" continuous="true"
                        segments="  0.00 1.0
                                    1.00 1.0
                                    2.00 2.0
                                    3.00 3.0"/>

        </ScalarViscosityVelocity>
    </Scene>
</X3D>
```

Kapitola 6

Testovaná data a výsledky

Po implementaci, otestování funkčnosti a doladění programu bylo zapotřebí vymyslet metodu testování, pro ověření správnosti předpokladů. Zamýšlelo se, že navrhovaná metoda umožní uživatelům lepší rozeznání objektů a umožní jim rychleji prozkoumat volumetrická data.

Pro účel otestování byly vymyšleny tři typy jednoduchých příkladů, které jsou popsány v sekci 6.1. Každý příklad se vykonal dvakrát. Jednou s využitím funkce detekce a zvýraznění hran a podruhé bez této funkce. Pořadí provádění se střídalo, aby byla zajištěna nezávislost. Tyto příklady se pustili 12-ti osobám, aby splnili požadovaný úkol. Během vykonání se měřil čas a pozice haptického zařízení. Na konci měření vznikl soubor dat. Zjištěné výsledky jsou uvedeny v sekci 6.2. Po vykonání příkladů uživatelé odpověděli na tři otázky. Jaký mají názor na detekci a zvýraznění změn a jestli jim při vyhledávání objektů pomohla.

Před začátkem měření měli uživatelé možnost si vyzkoušet práci s haptickým zařízením na jednoduchém příkladě, aby si vyzkoušeli chování haptického zařízení. Následovaly dva příklady, které se chovaly jako testovací. První z nich obsahoval krychli přesně uprostřed dat. Druhý příklad obsahoval tři krychle rozmístěné v prostoru. Rozdíl od třetího příkladu byl v tom, že krychle bylo možné vizualizovat a ověřit si správné nalezení jiné viskozity.

6.1 Testovací data

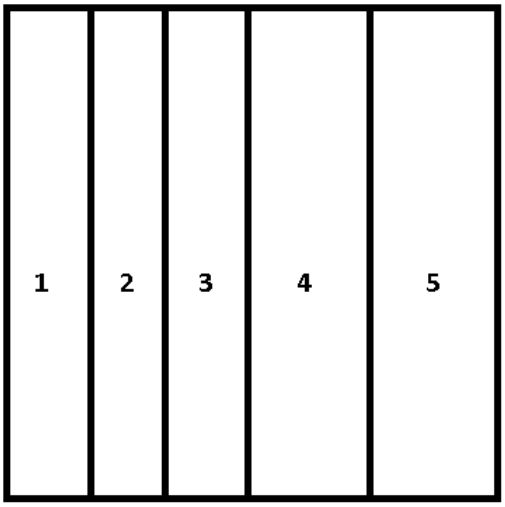
Všechny testovací programy jsou bez volumetrické vizualizace. Je to z důvodu, aby se pro prozkoumávání využil pouze hmat. Úkoly jsou založeny na vyhledání objektů ve volumetrických datech. Pokud jsou volumetrická data vizualizována, uživatelé pro vyhledání objektů využijí zrak a testovanou metodu pouze pocítí a nejspíše ji nevyužijí.
Příklady jsou seřazeny postupně podle obtížnosti. Zdrojové soubory jsou na přiloženém CD.

6.1.1 První příklad

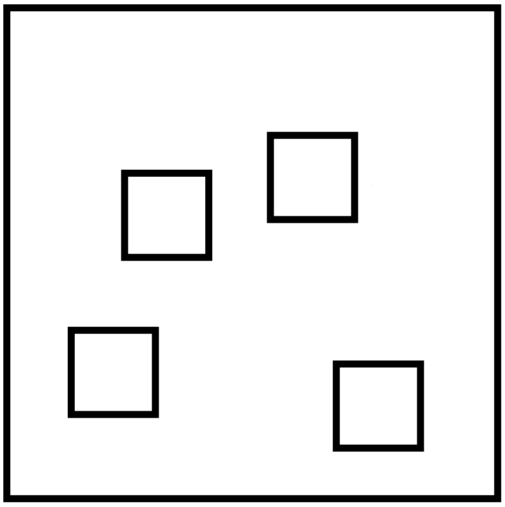
První příklad se zaměřuje na vyhledání míst, kde se mění hodnota skalárních dat, nebo-li kde dochází k přeměně viskozity prostředí. Pro co největší zjednodušení se data mění pouze v ose x. V ostatních osách jsou data stejná. Obrázek 6.1.1 znázorňuje vzhled volumetrických dat a čísla označují různou viskozitu. Viskozitu pro haptickou vizualizaci definujeme podle mapovací funkce. Každá oblast představovala rozdílnou viskozitu. Rozdíl viskozit byl v některých přechodech velký a dobře znatelný, ale vyskytovaly se i místa, kde byl rozdíl velice malý. Úkolem bylo nalézt co nejvíce změn a označit je pomocí tlačítka na haptickém zařízení. Po stisknutí tlačítka se v místě stisku objevila koule. Zároveň se do logovacího souboru uložila informace o času a pozici haptického zařízení.

6.1.2 Druhý příklad

Druhý příklad rozšiřuje složitost o osu y. Jedná se tedy o hledání dat v 2D prostoru. Na ploše se vyskytují 4 čtverce. Hledání probíhá jen v osách x a y, na ose z nezávisí. Chování v ose z je stejné. Rozmístění znázorňuje obrázek 6.1.2. V prostoru mezi čtverci byla konstantní viskozita. Každá krychle představovala odlišnou viskozitu. Některé obsahovali hustší prostředí a jiné řidší. Uživatel měl za úkol nalézt a označit čtverce s jinou viskozitou a označit je kliknutím tlačítkem na haptickém zařízení uvnitř čtverce.



Obrázek 6.1.1: Znázornění dat pro první testovací příklad

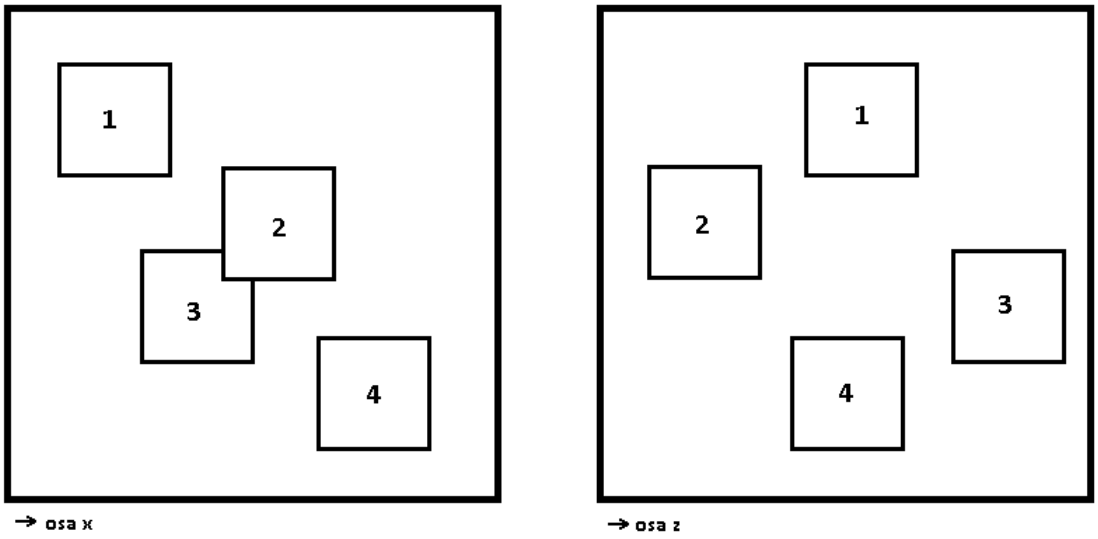


Obrázek 6.1.2: Znázornění dat pro druhý testovací příklad

6.1.3 Třetí příklad

Třetí příklad byl nejsložitější, protože se jednalo o nalezení 4 krychliček v celém prostoru. Rozmístění krychliček je znázorněno na obrázku 6.1.3. Tento obrázek měli uživatelé při hledání dat k dispozici. Levý obrázek označuje pohled zepředu.

Pravý obrázek zobrazuje pohled zboku a je v něm znázorněno v jaké hloubce se krychlička vyskytuje. Ten jim poskytl informaci, kde přibližně mají data hledat. V prostoru mezi krychličkami byla konstantní viskozita. Každá krychlička představovala jinou viskozitu. Úkolem bylo opět nalézt krychličku a označit ji



Obrázek 6.1.3: Znázornění dat pro třetí testovací příklad

stisknutím tlačítka na haptickém zařízení.

6.2 Zjištěné výsledky

Pro ověření předpokladů bylo pozváno 12 osob, které se pokusili splnit požadované úkoly. Testování ovlivnila vibrace haptického zařízení, která pochází z nepřesného snímání polohy. Při aplikaci viskózní síly začalo haptické zařízení silně vibrovat. Vibrace se nepodařilo odstranit. Uživatelé byli předem upozorněni a poučeni, že při vyskytnutí vibrací je potřeba přesunout haptické zařízení do jiné polohy a následně pokračovat v prozkoumávání dat.

Postupně zveřejníme výsledky ze všech měření. Porovnávány budou pouze počty nalezení objektů. Měřený čas se přitom velice lišil. Každý uživatel objevil objekty

za různý čas. Bez detekce a zvýraznění většinou nalezli nejvýraznější rozdíl v datech a skončili s hledáním. Při použití detekce a zvýraznění byl čas hledání delší, protože uživatelé zaznamenali i jemnější přechody, snažili se je detekovat a označit.

6.2.1 První příklad

V prvním příkladu bylo za úkol lokalizovat přechody mezi prostředími. Celkem bylo v datech 6 přechodů. V tabulce 6.1 je zobrazeno kolik uživatelů nalezlo daný počet přechodů.

počet přechodů	s detekcí a zvýrazněním	bez detekce a zvýraznění
6	3	0
5	4	1
4	3	6
3	2	5

Tabulka 6.1: Nalezené přechody u příkladu 1

Z tabulky je vidět, že při zapnuté detekci a zvýraznění hran byl počet nalezených přechodových hran vyšší. Vážený průměr pro hodnoty s detekcí je $4,66$ a pro hodnoty bez detekce je 3.66. To znamená, že se průměrně nalezlo o 1 hranu více, pokud byla detekce a zvýraznění zapnuto.

6.2.2 Druhý příklad

Ve druhém příkladu se hledali čtverce. V testovaných datech byly 4 čtverce. V tabulce 6.2 je zobrazeno kolik uživatelů nalezlo daný počet čtverců.

počet čtverců	s detekcí a zvýrazněním	bez detekce a zvýraznění
4	5	3
3	5	6
2	2	3

Tabulka 6.2: Nalezené čtverce u příkladu 2

Z tabulky je vidět, že při zapnuté detekci a zvýraznění hran byl počet nalezených čtverců vyšší. Vážený průměr pro hodnoty s detekcí je $3,25$ a pro hodnoty bez

detekce je 3. Rozdíl není tak velký jako v prvním příkladě. Rozdíl závisí na zvolených hodnotách viskozity a mohli ho ovlivnit nežádoucí vibrace. Zvolení hodnot viskozit je velice důležité. Pokud je rozdíl dostatečně velký, je možné snadno detekovat rozdíly i bez zvýraznění. Detekce a zvýraznění nám tedy umožní použít menší rozdíly a rozeznat tak více detailů.

6.2.3 Třetí příklad

Ve třetím příkladu bylo za úkol hledat krychličky v prostoru. V testovaných datech se nacházely 4 krychličky. V tabulce 6.3 je zobrazeno kolik uživatelů nalezlo daný počet krychliček.

počet krychliček	s detekcí a zvýrazněním	bez detekce a zvýraznění
4	7	5
3	4	2
2	0	3
1	0	1

Tabulka 6.3: Nalezené krychličky u příkladu 3

Z tabulky je vidět, že při zapnuté detekci a zvýraznění hran byl počet nalezených čtverců vyšší. Vážený průměr pro hodnoty s detekcí je $3,63$ a pro hodnoty bez detekce je 3. U tohoto příkladu měli uživatelé k dispozici obrázek s přibližným rozmístěním krychlí v prostoru. Z výsledků je vidět, že i když uživatelé přibližně věděli, kde by se měl objekt nacházet, nebylo snadné jej nalézt. Detekce a zvýraznění hran umožnila zvětšení šance na nalezení objektů.

6.2.4 Dojmy uživatelů

Po provedení testů byli uživatelé dotázání na tři následující otázky:

- Co vám činilo největší potíže při manipulaci s haptickým zařízením?
- Byla znatelná funkce zvýraznění rozdílu mezi různými prostředími v závislosti na rychlosti pohybu?
- Pomohla vám funkce zvýraznění v hledání objektů? A jak?

Na první otázku uživatelé odpovídali velice podobně. Velice rušivá byla vibrace zařízení způsobená nepřesným snímáním polohy. Dále uživatelé uváděli chybějící vizuální informaci. Vizuální informace je pro průzkum dat velice důležitá. Pro otestování přístupu prezentovaného v této práci bylo zapotřebí vizuální informaci odstranit a zjistit, zda zvýraznění pomůže při hledání objektů. Pokud bychom vizualizovali všechny objekty, uživatelé by je nalezli podle vizuální informace a konstatovali, že zvýraznění cítí.

Na druhou otázku odpovídali všichni uživatelé stejně a to, že funkce byla znatelná.

Na třetí otázku byly odpovědi také velice podobné. Uživatelé uváděli, že mohli lépe rozeznat, kde se přibližně objekty nachází a hledat pouze v místech, kde zaznamenali zvýrazněnou změnu.

Kapitola 7

Závěr

V této práci jsem popsal metodu pro detekci a zvýraznění hran ve volumetrických datech pro haptickou vizualizaci. Metoda je použitelná pro skalární data s použitím viskozity a pro vektorová data. Podle výsledků prezentovaných v předchozí kapitole je vidět zlepšení rozpoznávání objektů ve volumetrických datech. Uživatelé hodnotili tuto funkci kladně. Pomohla jim také v objevování objektů.

Hlavní princip této metody spočívá v detekování změn ve volumetrických datech pomocí Sobel 3D operátoru. To se provede při inicializaci aplikace. Výsledkem je volumetrická mapa obsahující hodnoty jak velká změna je v datech. Během výpočtu se z mapy načítají hodnoty v místě, kde je haptické zařízení a společně s rychlostí pohybu se dosadí do rovnice 4.9. Výsledkem rovnice se vynásobí původní volumetrická data a tím se dosáhne zvýraznění nebo potlačení určitých hodnot v závislosti na rychlosti a detekované mapě.

Značnou část práce mi zabralo prozkoumání a naučení se prostředí H3DAPI, které je velice rozsáhlé a kombinuje několik programovacích stylů. Po prozkoumání a zjištění, jak správně využívat všech předností tohoto prostředí, byla implementace výsledné metody velice rychlá. Bylo důležité prozkoumat zdrojové kódy a nalézt třídy, od kterých můžeme nejlépe dědit. Ze začátku mi činilo největší problém hledání uzlů, které jsou potřebné jako rodičovské uzly pro mnou zvolenou funkci. Dokumentace k H3DAPI je dobrým začátkem k pochopení základů.

Pro podrobnější pochopení je důležité využít Doxygen nápovědu dostupnou na internetových stránkách H3DAPI a zdrojové soubory dostupné po stažení aplikace.

Možností dalšího vývoje a vylepšení metody by mohlo být porovnání více metod pro detekci změn ve volumetrických datech. Dále pak vyzkoušení nějaké jiné funkce počítající převod z rychlosti a detekovaných změn na sílu pro haptické zařízení. Nová funkce by mohla i při malé rychlosti zachovat částečné zvýraznění dat, aby i při pomalém pohybu bylo zvýraznění znatelné a pomohlo tak uživatelům v rozpoznávání změn.

Pro nasazení v reálné aplikaci je vhodné, aby při nalezením nějakého objektu či změny se volumetrická data ořízla a stala se viditelná jen určitá část, kterou by pak uživatel mohl lépe prozkoumat. To by mohlo být použito i při velkých datech, kde by se pomocí metody našla zajímavá místa a následně došlo k prozkoumání jen daných míst.

Literatura

[AB01] M. W. Asghar, K. E. Barner *Nonlinear Multiresolution Techniques with Applications to Scientific Visualization in a Haptic Environment* IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 7, NO. 1, January/March 2001

[H3DAPI] H3DAPI framework URL: <`http://www.h3dapi.org`>

[H3DAPIT] H3DAPI Tutorials URL: <`http://sites.google.com/site/h3dtutorials/home`>

[IBHJ03] M. Ikits, J. D. Brederson, C.D. Hansen, C.R. Johnson, *A Constraint-Based Technique for Haptic Volume Exploration*. Proc. IEEE Visualization, pp. 263-269 2003

[IN93] H. IWATA and H. NOMA, *Volume Haptization*. Proc. IEEE Symp. Research Frontiers in Virtual Reality, pp.16-23, Oct. 1993.

[K06] Pavel Kolčárek *Multiresolution Methods for Haptic Rendering* Ph.D. Thesis, Brno 2006

[KS05] P. Kolčárek, J. Sochor *Haptic Rendering using Velocity Driven Level of Detail* GRAPHITE '05 2005

[LSCY05] K.Lundin, M.Sillén, M.Cooper and A. Ynnerman, *Haptic visualization of computational fluid dynamics data using reactive forces*. Proc. Conf. Visualization and Data Analysis, pp. 31-41, Jan. 2005.

[LGY05] K. Lundin, B. Gudmundsson and A. Ynnerman, *General Proxy-Based Haptics for Volume Visualisation*. Proc. IEEE World Haptics Conf., pp. 557-560, Mar. 2005.

[LYG02] K. Lundin, A. Ynnerman and B. Gudmundsson, *Proxy-Based Haptic Feedback from Volumetric Density Data*. Proc. Eurohaptics, pp. 104-109, 2002.

[PCY07] K. L. Palmerius, M. Cooper and A. Ynnerman, *Haptic Rendering of Dynamic Volumetric Data*. IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 14, NO. 2, March/April 2007.

[PHANTOM] Phantom Omni Haptic device, URL: <http://www.sensable.com/haptic-phantom-omni.htm>

[PN01] D. Parkhurst, E. Niebur *Evaluating Velocity-based Level of Detail Rendering of Virtual Environments using Visual Search* The Johns Hopkins University, Baltimore, Maryland 2001

[VTNB04] E. Vidholm, X. Tizon, I. Nystrom and E. Bengtsson, *Haptic Guided Seeding of MRA Images for Semi-Automatic Segmentation*. Proc. IEEE Symp. Biomedical Imaging, 2004.

[X3D] X3D specification URL: <http://www.web3d.org/x3d/>

[WS10] L. Wei and A. Sourin *Haptic Rendering of Mixed Haptic Effects* 2010 International Conference on Cyberworlds 2010

[ZPD02] J. Zhang, S. Payandeh, John. Dill *Haptic Subdivision: an Approach to Defining Level-of-detail in Haptic Rendering* Proceedings 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems 2002

[ZPD03] J. Zhang, S. Payandeh, J. Dill *Levels of Detail in Reducing Cost of Haptic Rendering: a Preliminary User Study* HAPTICS '03 Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment 2003

Příloha A

Knihovny pro vektorová data

A.1 Detekce změn

Zdrojový soubor VectorDetectorVolume.h

```
#include "scalarDetectorNode.h"
#include <VHTK/Debug.hh>
using namespace H3D;
using namespace VHTK;
// Přidání tohoto uzlu do seznamu všech uzlů
H3DNodeDatabase ScalarDetectorNode::database( "ScalarDetectorNode", &(newInstance<ScalarDetectorNode>),
                                        typeid( ScalarDetectorNode ), &VHTKScalarImageNode::database );
// vstupně výstupní parametry
namespace ScalarDetectorNodeInternals {
  FIELDDB_ELEMENT( ScalarDetectorNode, url, INPUT_OUTPUT );
  FIELDDB_ELEMENT( ScalarDetectorNode, imageLoader, INPUT_OUTPUT );
}
//konstruktor
ScalarDetectorNode::ScalarDetectorNode
( Inst< SFNode        > _metadata,
  Inst< Field         > _dataChanged,
  Inst< SFBool        > _scaleVectors,
  Inst< SFImage       > _image,
  Inst< MFString      > _url,
  Inst< MFImageLoader > _imageLoader ) :
  VHTKScalarImageNode( _metadata,
                       _dataChanged,
                       _scaleVectors,
                       _image ),
  X3DUrlObject( _url ),
  imageLoader( _imageLoader ) {
  type_name = "ScalarVolume";
  database.initFields( this );
  url->routeNoEvent( image );
  imageLoader->routeNoEvent( image );
}
```

Zdrojový soubor VectorDetectorVolume.cpp

```
#include "VectorDetectorVolume.h"
#include <VHTK/Debug.hh>
using namespace H3D;
using namespace VHTK;
using namespace H3DUtil;
// Přidání tohoto uzlu
H3DNodeDatabase VectorDetectorVolume::database( "VectorDetectorVolume",
                                        &(newInstance<VectorDetectorVolume>),
                                        typeid( VectorDetectorVolume ),
                                        &VHTKVectorImageNode::database );
namespace VectorDetectorVolumeInternals {
```

```
  FIELDDB_ELEMENT( VectorDetectorVolume, url, INPUT_OUTPUT );
  FIELDDB_ELEMENT( VectorDetectorVolume, imageLoader, INPUT_OUTPUT );
}
VectorDetectorVolume::VectorDetectorVolume
( Inst< SFNode        > _metadata,
  Inst< Field         > _dataChanged,
  Inst< SFBool        > _scaleVectors,
  Inst< SFImage       > _image,
  Inst< MFString      > _url,
  Inst< MFImageLoader > _imageLoader) :
  VHTKVectorImageNode( _metadata,
                       _dataChanged,
                       _scaleVectors,
                       _image ),
  \\deklarace proměných
  X3DUrlObject( _url ),
  imageLoader( _imageLoader ){
  type_name = "VectorDetectorVolume";
  database.initFields( this );
  url->routeNoEvent( image );
  imageLoader->routeNoEvent( image );
}
void VectorDetectorVolume::SFImage::update() {
//Sobeluv hranovy operator
Vec3f kernel3x[kernelSize] = {Vec3f(-1, -3, -1), Vec3f(-3, -6, -3), Vec3f(-1, -3, -1),
                        Vec3f(0, 0, 0), Vec3f(0, 0, 0), Vec3f(0, 0, 0),
                        Vec3f(1, 3, 1), Vec3f(3, 6, 3), Vec3f(1, 3, 1)};
Vec3f kernel3y[kernelSize]= { Vec3f(1, 3, 1), Vec3f(0, 0, 0), Vec3f(-1, -3, -1),
                        Vec3f(3, 6, 3), Vec3f(0, 0, 0), Vec3f(-3, -6, -3),
                        Vec3f(1, 3, 1), Vec3f(0, 0, 0), Vec3f(-1, -3, -1) };
Vec3f kernel3z[kernelSize]= { Vec3f(-1, 0, 1), Vec3f(-3, 0, 3), Vec3f(-1, 0, 1),
                        Vec3f(-3, 0, 3), Vec3f(-6, 0, 6), Vec3f(-3, 0, 3),
                        Vec3f(-1, 0, 1), Vec3f(-3, 0, 3), Vec3f(-1, 0, 1) };
VHTK_DEBUG_LOG();
//získání ukazatele na imageLoader a cesty k souborům
MFImageLoader *image_loaders =
static_cast< MFImageLoader * >( routes_in[1] );
MFString *urls = static_cast< MFString * >( routes_in[0] );
//Použít postupně všechny adresy a ImageLoadery
    for( MFString::const_iterator i = urls->begin(); i != urls->end(); ++i ) {
        for( MFImageLoader::const_iterator il = image_loaders->begin();
            il != image_loaders->end();
            il++ )
{
        //uložení dat do obrázku
        Image *image = static_cast< H3DImageLoaderNode * >(*il)->loadImage( *i );
         //výpočet velikosti dat
        unsigned int size =
               image->width() * image->height() * image->depth() *
                ( image->bitsPerPixel() / 8 );
        vytvoření nového pole pro detekci
        unsigned char * data = new unsigned char[size];
        PixelImage *px = new PixelImage(image->width(), image->height(), image->depth(),
         image->bitsPerPixel(),image->pixelType(), image->pixelComponentType(),
          data, false, image->pixelSize());
size = image->width() * image->height() * image->depth();
        Vec3f *tempX = new Vec3f[size];
        Vec3f *tempY = new Vec3f[size];
        Vec3f *tempZ = new Vec3f[size];
Vec3f *out = new Vec3f[size];
Vec3f v;
int indexData, indexKernel; //index in 1dimension array
int x,y,z; //pozition
H3DFloat maximum = 0;
//3dimensional data, aplikace Sobelova hranoveho aplikatu
        for(unsigned int idataZ = 0; idataZ < image->depth(); idataZ++)
        {
            for(unsigned int idataY = 0; idataY < image->height(); idataY++)
            {
                for(unsigned int idataX = 0; idataX < image->width(); idataX++)
                {
                     indexData = idataX + image->width() * (idataY + image->height() * idataZ);
                   //dimensional kernel
                    for(unsigned int ikernelZ = 0; ikernelZ < kernelLength; ikernelZ++)
                    {
                        for(unsigned int ikernelY = 0; ikernelY < kernelLength; ikernelY++)
                        {
                            for(unsigned int ikernelX = 0; ikernelX < kernelLength; ikernelX++)
                            {
                                x = ikernelX - 1 + idataX;
```

```
                                y = ikernelY - 1 + idataX;
                                z = ikernelZ - 1 + idataZ;
                                if (x > -1 && y > -1 && z > -1 && x < (int)image->width() &&
                                        y < (int)image->height() && z < (int)image->depth())
                                {
                                   //indexDataKernel = x + image->width() * (y + image->height() * z);
                                    indexKernel =  ikernelX + kernelLength * (ikernelY);
                                    image->getElement(&v,x,y,z);
                                    tempX[indexData].x += v.x * kernel3x[indexKernel][ikernelZ];
                                    tempX[indexData].y += v.x * kernel3y[indexKernel][ikernelZ];
                                    tempX[indexData].z += v.x * kernel3z[indexKernel][ikernelZ];
                                    tempY[indexData].x += v.y * kernel3x[indexKernel][ikernelZ];
                                    tempY[indexData].y += v.y * kernel3y[indexKernel][ikernelZ];
                                    tempY[indexData].z += v.y * kernel3z[indexKernel][ikernelZ];
                                    tempZ[indexData].x += v.z * kernel3x[indexKernel][ikernelZ];
                                    tempZ[indexData].y += v.z * kernel3y[indexKernel][ikernelZ];
                                    tempZ[indexData].z += v.z * kernel3z[indexKernel][ikernelZ];
                                }
                            }
                        }
                    }//end kernel
                   out[indexData].x = sqrt(tempX[indexData].x * tempX[indexData].x +
                    tempX[indexData].y * tempX[indexData].y + tempX[indexData].z * tempX[indexData].z);
                   out[indexData].y = sqrt(tempY[indexData].x * tempY[indexData].x +
                    tempY[indexData].y * tempY[indexData].y + tempY[indexData].z * tempY[indexData].z);
                   out[indexData].z = sqrt(tempZ[indexData].x * tempZ[indexData].x +
                tempZ[indexData].y * tempZ[indexData].y + tempZ[indexData].z * tempZ[indexData].z);
                   if(out[indexData].x > maximum)
                       maximum = out[indexData].x;
                   if(out[indexData].y > maximum)
                       maximum = out[indexData].y;
                   if(out[indexData].z > maximum)
                       maximum = out[indexData].z;
                }
            }
        }//end data
       v.x = 0; v.y = 0; v.z = 0;
       //set vectorVolume
       //převedení na rozsha 0 - 1
        for(unsigned int z = 0; z < image->depth(); z++)
        {
           for(unsigned int y = 0; y < image->height(); y++)
           {
               for(unsigned int x = 0; x < image->width(); x++)
               {
                   indexData = x + image->width() * (y + image->height() * z);
                   v.x = (out[indexData].x / maximum);
                   v.y = (out[indexData].y / maximum);
                   v.z = (out[indexData].z / maximum);
                   px->setElement(&v, x,y,z);
               }
           }
         }
       //pokud existuje obrazek, uložího
       if( image ) {
            VHTK_DEBUG_LOG_I("Image value updated");
            value = px;
             return;
        }
    }
  }
}
```

A.2 Aplikace na haptické zařízení

A.2.1 Zdrojový program, HaptNode.h

```
#ifndef _VHTK_HAPTMODE_HH_
#define _VHTK_HAPTMODE_HH_
#include <VHTK/VolumeHaptics.hh>
#include <VHTK/VHTKVectorModeNode.hh>
namespace VHTK {
  class HaptForceMode : public VHTKVectorModeNode {
  public:
    // inicializace vnitrnich dat
    HaptForceMode( );
    // Systemova funkce
    void getPrimitives( primitives_t &r, const state &s );
    // sila aplikovaneho vektoru podle mapovaci funkce
    auto_ptr< TSSFNode( SFTransferFunction ) > strength;
    auto_ptr< TSSFNode( SFTransferFunction ) > velocityStrength;
auto_ptr< MFVectorDataNode > velocityVolume;
    // databaze uzlu
    static H3D::H3DNodeDatabase database;
    // inicializace
    void initialize();
  };
}
#endif
```

Zdrojový program HaptNode.cpp

```
#include "HaptNode.h"
using namespace H3D;
using namespace HAPI;
using namespace VHTK;
//pridani do databaze
H3DNodeDatabase HaptForceMode::database
    ( "HaptForceMode",
      &(newInstance<HaptForceMode>),
      typeid( HaptForceMode ),
      &VHTKVectorModeNode::database );
namespace HaptForceModeInternals {
  FIELDDB_ELEMENT( HaptForceMode, strength, INPUT_OUTPUT );
  FIELDDB_ELEMENT( HaptForceMode, velocityStrength, INPUT_OUTPUT);
  FIELDDB_ELEMENT( HaptForceMode, velocityVolume, INPUT_OUTPUT);
}
//konstruktor
HaptForceMode::HaptForceMode( )
  : strength( new TSSFNode( SFTransferFunction ) ),
  velocityStrength (new TSSFNode(SFTransferFunction) ),
  velocityVolume ( new MFVectorDataNode )
{
  type_name = "HaptForceMode";
  database.initFields( this );
}
//inicializace
void HaptForceMode::initialize(){
  VHTKVectorModeNode::initialize();
  //test zda je nastavena mapovaci funkce
  if( !strength->getValue() ){
    VHTK_RUNTIME_WARNING( "HaptForceMode::initialize()",
                          "Transfer Function 'strength' not specified" ); }
  if( !velocityStrength->getValue() ){
    VHTK_RUNTIME_WARNING( "HaptForceMode::initialize()",
                          "Transfer Function 'strength' not specified" ); }
}
//vytvoreni primitiva
void HaptForceMode::getPrimitives( primitives_t &plist,
                                   const state &s ){
  //mapovaci funkce
  VHTKTransferFunctionNode *strength_transfer = strength->getValue();
  VHTKTransferFunctionNode *velocityTransfer = velocityStrength->getValue();
  if( strength_transfer && velocityTransfer) {
// get vector at Probe position
Vec3 V = getVectorAtProbe(s);
    HAPIFloat V_l = V.length();
HAPIFloat s_l = strength_transfer->get( (float)V_l );
Vec3 newV = V / V_l; //normalizace
newV = newV * s_l; //velikost podle mapovaci funkce
//data z volumetricke mapy
```

```
Vec3 velVolume = velocityVolume->getValueByIndex(int(s.frame))->
                 getVector(Matrix4f(s.T), Vec3f(s.proxy));
//vypocet podle navrzene rovnice
HAPIFloat vel = s.probe_v.length();
Vec3 pom;
pom.x = (vel ) * velVolume.x + (1.0 - vel);
pom.y = (vel ) * velVolume.y + (1.0 - vel);
pom.z = (vel ) * velVolume.z + (1.0 - vel);
//HAPIFloat pom_l = velVolume.length();//pom.length();
if(pom.x < 0 )
pom.x = 0;
if(pom.y < 0 )
pom.y = 0;
if(pom.z < 0 )
pom.z = 0;
//vypocet vysledku
Vec3 result;
result.x = newV.x * pom.x;
result.y = newV.y * pom.y;
result.z = newV.z * pom.z;
HAPIFloat pom_l = result.length();
//predani hodnot haptice
    if( pom_l > H3DUtil::Constants::f_epsilon )
      plist.push_back( primitive( pom_l,
                                  result / pom_l, PRIMITIVE_FORCE ) );
  }
}
```

A.3 Zdrojový soubor .x3d

```
<?xml version="1.0" encoding="utf-8"?>
<X3D profile='Full' version='3.2'>
    <Scene>
        <!-- Import knihoven -->
        <ImportLibrary library="bin\\HaptNode" />
        <ImportLibrary library="bin\\VectorDetectorVolume" />
        <IMPORT inlineDEF="H3D_EXPORTS" exportedDEF="HDEV" AS="HDEV" />
        <Background skyColor="1.0 1.0 1.0"/>
        <Group DEF="STREAMS_GROUP" />
        <LocalInfo DEF="INFO" />
        <VolumeHaptics stiffness="800" DEF="HAPTICS" />
<!-- Uzel pro vektory se zvyraznenim -->
        <HaptForceMode active="true">
             <VectorVolume DEF="VVOLUME" url='"data/vec2f.bin"'>
                 <RawImageLoader DEF="LOADER" width="10" height="10" depth="10"
                                 bitsPerPixel = "96"
                                 pixelType = "VEC3"
                                 pixelComponentType = "RATIONAL"
                                 pixelSize="0.06 0.06 0.06" />
                </VectorVolume>
            <VectorDetectorVolume containerField="velocityVolume" url='"data/vec2f.bin"'>
                <RawImageLoader USE="LOADER" />
            </VectorDetectorVolume>
            <!-- mapovaci funkce -->
            <PiecewiseFunction containerField="strength" continuous="true"
                    segments="  0.00 2.0
                                1.00 2.0
                                100  2.0"/>
            <PiecewiseFunction containerField="velocityStrength" continuous="true"
                    segments="  0.00 3.0
                                1.00 3.0
                                100  3.0"/>
        </HaptForceMode>
<!-- Skript pridavajici stream tube -->
<PythonScript DEF="PUT" url="python/PutStreamTubes.py">
<Group USE="STREAMS_GROUP" containerField="references" />
<Appearance
containerField="references">
<Material
ambientIntensity="1"/>
</Appearance>
    <StreamTubes
        containerField="references"
        step="0.01"
        maxLength="5.0"
        maxRadius="0.01">
      <VHTKVectorDataNode USE="VVOLUME" />
    </StreamTubes>
  </PythonScript>
  <ROUTE
      fromNode="HDEV" fromField="mainButton"
```

```
      toNode="PUT" toField="button" />
  <ROUTE
      fromNode="HAPTICS" fromField="proxy"
      toNode="INFO" toField="position" />
  <ROUTE
      fromNode="INFO" fromField="position"
      toNode="PUT" toField="position" />
</Scene>
</X3D>
```