

Univerzita Karlova v Praze  
Matematicko-fyzikální fakulta

## BAKALÁŘSKÁ PRÁCE



Josef Talaš

### Interaktivní deformace objemových dat

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Václav Krajíček

Studijní program: Informatika, Programování

2009

Rád bych poděkoval mému vedoucímu Mgr. Václavu Krajíčkovi za četné rady při psaní mé bakalářské práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 1.8.2009

Josef Talaš

# Obsah

<b>1</b>	<b>Úvod</b>	<b>6</b>
1.1	Cíle práce . . . . .	6
1.2	Struktura textu . . . . .	7
<b>2</b>	<b>Existující metody</b>	<b>8</b>
2.1	Mass Spring System . . . . .	8
2.2	Free-form Deformation . . . . .	10
2.3	Finite Element Method . . . . .	10
2.4	3D Chainmail Algorithm . . . . .	12
<b>3</b>	<b>Implementace</b>	<b>14</b>
3.1	VL Framework . . . . .	14
3.2	Algoritmus . . . . .	15
3.2.1	Základní Chainmail algoritmus . . . . .	15
3.2.2	Popis algoritmu . . . . .	16
3.2.3	Paralelizace algoritmu . . . . .	18
3.2.4	Elastic Relaxation . . . . .	20
3.3	Interpolace . . . . .	21
<b>4</b>	<b>Výsledky</b>	<b>24</b>
4.1	Reálná data . . . . .	24
4.2	Umělá data . . . . .	25
4.3	Shrnutí . . . . .	27
<b>5</b>	<b>Uživatelská dokumentace</b>	<b>29</b>
5.1	Úvod . . . . .	29
5.2	Požadavky . . . . .	29
5.3	Ovládání . . . . .	30
5.3.1	Klávesové zkratky . . . . .	30

5.4	Hlavní okno programu . . . . .	30
5.4.1	Zobrazovací část . . . . .	30
5.4.2	Nástrojová část . . . . .	31
5.5	Tutorial . . . . .	34
5.5.1	Deformace celého objemu dat . . . . .	34
5.5.2	Deformace vybrané části dat . . . . .	34
<b>6</b>	<b>Programátorská dokumentace</b>	<b>36</b>
6.1	Požadavky . . . . .	36
6.1.1	Potřebné knihovny . . . . .	36
6.2	Program . . . . .	37
6.2.1	Testovací interface . . . . .	37
6.2.2	Funkce reconnect . . . . .	38
6.2.3	Části programu . . . . .	38
6.3	Vstupní a výstupní data . . . . .	40
<b>7</b>	<b>Závěr</b>	<b>41</b>
7.1	Možná rozšíření . . . . .	41
<b>A</b>	<b>Obsah CD</b>	<b>45</b>

Název práce: Interaktivní deformace objemových dat

Autor: Josef Talaš

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Václav Krajíček

e-mail vedoucího: Vaclav.Krajicek@mff.cuni.cz

Abstrakt: Předložená práce popisuje implementaci 3D Chainmail algoritmu pro deformace objemových dat. Tento algoritmus je zvolen po porovnání několika obvyklých přístupů k deformacím. Nejedná se o algoritmus vhodný pro fyzikálně věrné deformace, je kompromisem mezi rychlostí, věrností a jednoduchostí. V další části práce je zvolený algoritmus podrobně popsán. Je také rozebrána možnost jeho paralelizace za použití moderní knihovny Microsoft Parallel Extensions. Část práce je též věnována VL Frameworku, který slouží k zobrazování objemových dat v našem programu. Výsledný program je otestován na reálných i umělých datech. Cílovou platformou programu je operační systém Microsoft Windows.

Klíčová slova: objemová data, deformace, algoritmus Chainmail

Title: Interactive deformation of volume data

Author: Josef Talaš

Department: Department of Software and Computer Science Education

Supervisor: Mgr. Václav Krajíček

Supervisor's e-mail address: Vaclav.Krajicek@mff.cuni.cz

Abstract: This thesis describes an implementation of 3D Chainmail Algorithm for volume data deformation. This algorithm is chosen after a comparison of several approaches to volume data deformation. This is not an algorithm suitable for physically plausible deformations, it is a compromise between performance, plausibility and simplicity. In another part of this thesis the algorithm is closely described. There is also analysed the possibility of algorithm parallelization using advanced Microsoft Parallel Extensions library. A part of this thesis is devoted to VL framework, which is in our program used to display volume data. The final program is tested on both real and artificial data. The target platform is Microsoft Windows operating system.

Keywords: volume data, deformation, Chainmail algorithm

# Kapitola 1

## Úvod

V počítačové grafice se pro zobrazování objektů obvykle používají povrchová data. Povrch zobrazovaného objektu je modelován pomocí velkého počtu polygonů, které jsou následně pokryty texturou. Tuto metodu používají vývojáři počítačových her, a proto jsou dnešní grafické karty prakticky vyvíjené pro renderování takovýchto dat. Mnohé aplikace ale požadují informace nejen o povrchu objektů, ale i o jejich vnitřní struktuře. Proto se začala používat objemová data. Objemová data se díky zvyšování výpočetní kapacity počítačů dostávají do popředí více než tomu bylo doposud. Pro jejich zobrazování byly v minulosti potřeba drahé specializované počítače, ale díky vývoji hardware se dnes tato data dají zobrazovat s dostatečným počtem snímků za sekundu i na průměrně výkonném domácím stroji.

Objemová data mají široké využití. Používají se v lékařství (diagnostika pacientů pomocí počítačového tomografu nebo magnetické rezonance), strojírenství (návrhy různých součástí) nebo již i herním průmyslu (např. pro zobrazování terénu nebo i objektů). Proto roste potřeba existující data deformovat - měnit jejich tvar. Existuje mnoho přístupů pro deformování objemových dat s různými vlastnostmi a způsoby použití.

### 1.1 Cíle práce

Cílem naší práce je vytvořit program, pomocí kterého se budou provádět deformace objemových dat na průměrně výkonném počítači. Tyto deformace by měly být pokud možno interaktivní a pro uživatele snadno proveditelné.

Objemová data se skládají z voxelů. Voxel reprezentuje jednu hodnotu v pravidelné trojrozměrné mřížce, do které jsou objemová data umístěna.

Jedná se o analogii s pixelem v dvourozměrném obrázku. Objemová data se často skládají i z několika milionů voxelů. Požadavek na fyzikální věrnost při deformacích by proto znamenal, že by uživatel musel na výpočet deformace čekat často dlouhou dobu. To ale odporuje našemu požadavku na interaktivitu, proto nebudeme po řešení deformací chtít fyzikální věrnost - bude stačit, aby deformace dat byly dostatečně věrné. Naším cílem tedy jsou deformace, které budou kompromisem mezi věrností, rychlostí (zaručující vhodnou míru interaktivity) a uživatelskou přívětivostí.

## 1.2 Struktura textu

V následující kapitole jsou představeny některé existující metody pro deformování objemových dat. Třetí kapitola podrobně rozebírá zvolenou metodu a použitý algoritmus. Ve čtvrté kapitole je testována rychlost algoritmů na reálných i umělých datech. Pátá kapitola obsahuje uživatelskou dokumentaci, šestá pak dokumentaci programátorskou. Poslední, sedmá, kapitola shrnuje a hodnotí dosažené výsledky a navrhuje možnou další práci.

# Kapitola 2

## Existující metody

V této kapitole jsou rozebrány obvyklé přístupy při řešení deformací objemových dat.

### 2.1 Mass Spring System

Mass Spring System patří mezi metody založené na fyzikálních zákonech. Objekt je modelován jako mřížka hmotných bodů propojených pružinami (viz obrázek 2.1). Tyto pružiny způsobují přenos působící síly na další body. Síly vyvolané pružinami jsou většinou lineární, ale dají se využít i nelineární, např. pro modelování lidské kůže.

Chování pružiny (lineární) je fyzikálně popsáno pomocí Hookova zákona:

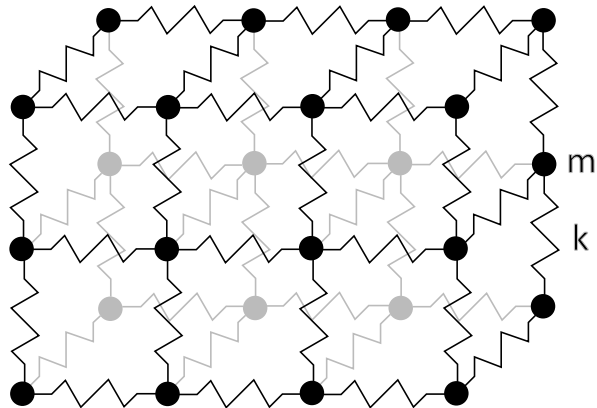
$$F = -Kx$$

$K$  - konstanta udávající tuhost pružiny, čím vyšší má hodnotu, tím je těžší pružinu natáhnout

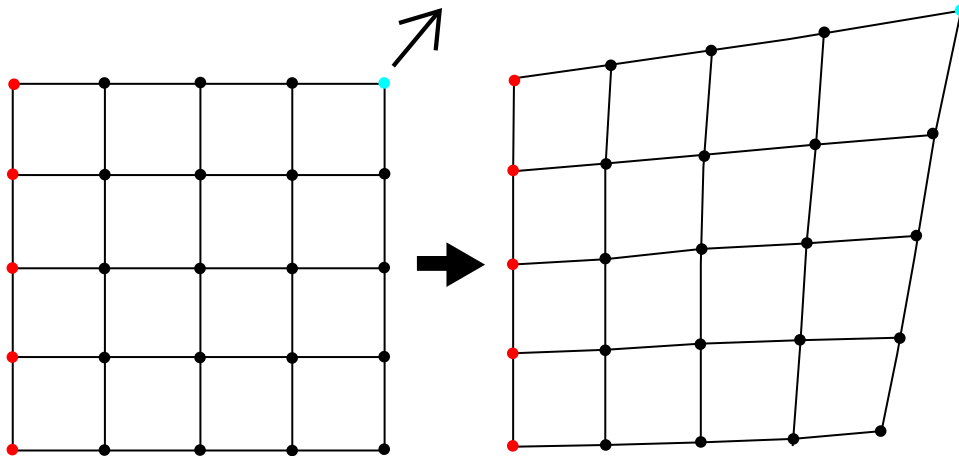
$x$  - vzdálenost, o kterou byla pružina vychýlena ze své rovnovážné polohy, tedy takové polohy, kdy na pružinu nepůsobí žádná síla

$F$  - síla potřebná k navrácení pružiny do rovnovážné polohy

Síla má zápornou hodnotu. To je z toho důvodu, že působí opačným směrem než síla, která přesunula pružinu z rovnovážné polohy (tedy pokud pružinu natahujeme, tato síla pružinu smršťuje a naopak). Pomocí různých hodnot tuhostí pružin se dá simulovat více druhů materiálů, např. při použití u části lidského těla by kůže měla menší hodnotu tuhosti než kost.



Obrázek 2.1: Příklad MSS. Pružiny mezi body přenášejí působící síly.



Obrázek 2.2: Jednoduchá deformace čtvercové mřížky pomocí metody Mass Spring System. Červené body jsou pevné body, které se nemohou pohybovat.

Pohyb bodu v mřížce je dán Druhým Newtonovým zákonem [5]:

$$m_i \ddot{x}_i = -\gamma_i \dot{x}_i + \sum_j g_{ij} + f_i$$

kde  $m_i$  je hmotnost bodu,  $x_i \in \mathfrak{R}^3$  pozice bodu a pravá strana rovnice jsou

síly působící na bod. První výraz na pravé straně je tlumicí síla,  $g_{ij}$  je síla působící na bod  $i$  pružinou mezi body  $i$  a  $j$ , a  $f_i$  je součet všech ostatních sil, které působí na bod  $i$  (např. gravitace nebo síla vzniklá v důsledku interakce uživatele). Rovnice pohybu celého systému se skládají z pohybů všech bodů v mřížce. Výsledné pozice bodů v mřížce jsou vypočítány pomocí diferenciálních rovnic.

Tato metoda se často používá pro modelování látek, vlasů nebo animací obličejů. Je to jedna z nejčastěji používaných technik, která využívá fyzikálních principů (i když stále zjednodušeně). Její výhodou je relativně snadný způsob použití, a to i pro laika - několik objektů spojených pružinami si obvykle lze snadno představit. Příklad jednoduché deformace je na obrázku 2.2.

## 2.2 Free-form Deformation

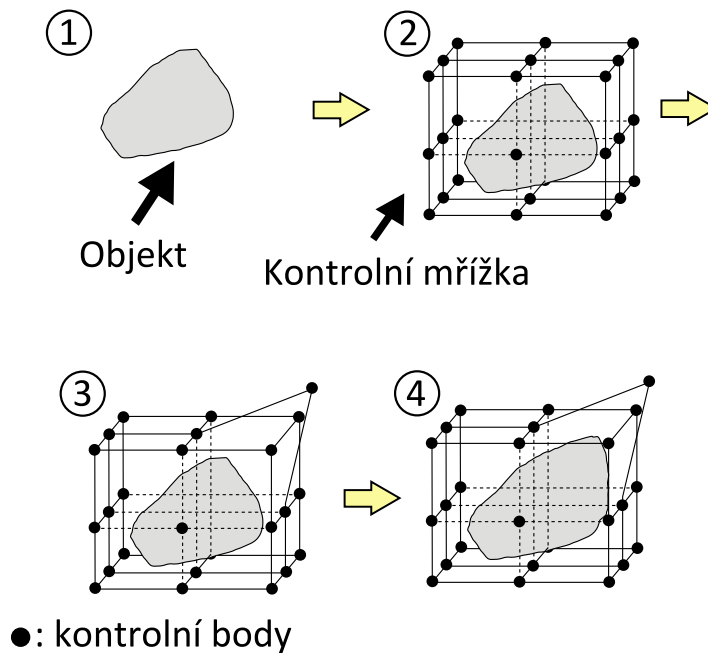
V této metodě je deformovaný objekt umístěn do kontrolní mřížky. Pomocí přemístování kontrolních bodů v této mřížce dochází k příslušné deformaci objektu (příklad je na obrázku 2.3). Free-form Deformation je metoda často využívána v počítačové animaci a také při modelování složitějších 3D objektů. Její výhodou je možnost deformovat pouze určitou část objektu.

Tuto metodu prezentovali již v roce 1986 Thomas W. Sederberg a Scott R. Parry [14]. Vznikla vylepšením metody Alana H. Barra, která se týkala deformací objektů pomocí matematických funkcí [1]. Sabine Coquillart prezentovala v roce 1990 rozšíření této metody na Extended Free-form Deformation [2]. Toto vylepšení umožňovalo libovolný tvar kontrolní sítě, což zvětšilo možnosti použití této metody. Na druhou stranu to ale také zvýšilo náročnost jejího vytvoření. Tato metoda byla původně čistě grafická - nedovolovala zachovávat určité fyzikální vlastnosti deformovaného objektu. Toto omezení bylo překonáno např. v práci [8], kde autoři prezentují algoritmus umožňující intuitivnější modelování a zachování proporcí deformovaných objektů.

## 2.3 Finite Element Method

Tato metoda pracuje tím způsobem, že se snaží udržovat deformované těleso v určité rovnováze [5]. Tato rovnováha je dána vlastnostmi objektu a vnějšími silami, které na tento objekt působí. Objekt je vyjádřen jako spojitá funkce, kterou se algoritmus snaží udržovat v rovnovážném stavu.

## Free-form Deformation



Obrázek 2.3: 1)deformovaný objekt 2)objekt s kontrolní mřížkou 3)deformace kontrolní mřížky 4) odpovídající deformace objektu

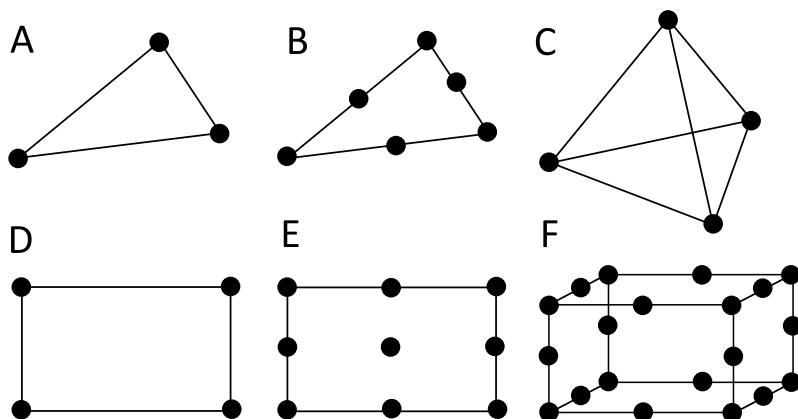
Rovnovázný stav je popsán pomocí systému rovnic. Celý objekt se rozdělí na soustavu elementů, které se navzájem pospojují pomocí krajních bodů. Některé používané elementy jsou na obrázku 2.4. Pro tyto elementy jsou nalezeny příslušné interpolační rovnice, pomocí kterých získáváme hodnoty funkce v těchto elementech. Řešení deformace spočívá v nalezení aproximace této funkce splňující systém rovnic, který popisuje rovnovážný stav.

Pro požadovanou funkci  $\Phi(x, y, z)$  je hodnota  $\Phi$  v bodě  $(x, y, z)$  aproximována jako:

$$\Phi(x, y, z) \approx \sum_i h_i(x, y, z)\Phi_i,$$

kde  $h_i$  jsou interpolační funkce elementu, který obsahuje bod  $(x, y, z)$ , a  $\Phi_i$  jsou hodnoty funkce  $\Phi(x, y, z)$  v krajních bodech elementu.

Tato metoda je oproti Mass Spring systémům fyzikálně realističtější,



Obrázek 2.4: Některé obvyklé elementy používané ve Finite Element Method, na které je celý objekt rozdělen.

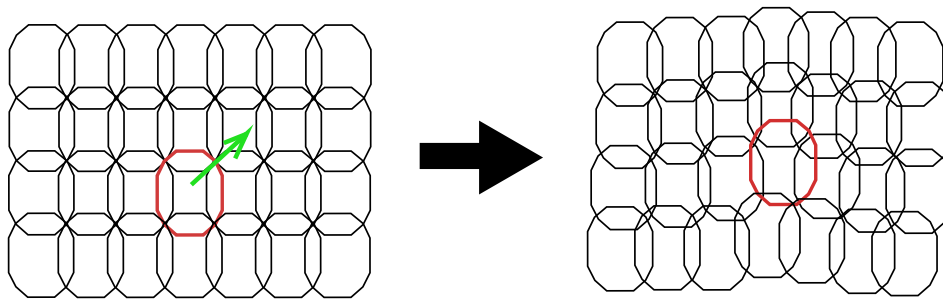
proto se často používá např. v mechanice nebo biologii. Kvůli větší reálnosti je ale také více výpočetně náročná. U této metody se při deformování složitějšího objektu požaduje nadefinovat jeho rozdělení na jednoduchá tělesa, což může být pro uživatele nesnadný úkol.

## 2.4 3D Chainmail Algorithm

Algoritmus 3D Chainmail byl vytvořen přímo pro deformace objemových dat [6]. Tento algoritmus pracuje s objektem rozděleným na elementy spojené se svými sousedy, vzhledem k nimž musí splňovat jednoduché geometrické podmínky. Jednotlivé elementy obvykle odpovídají přímo voxelům objemových dat. Díky napojení elementů se svými sousedy může pohyb jednoho elementu vést k přesunutí jeho sousedů, což může opět vyvolat další přesuny (příklad deformace je na obrázku 2.5).

Tato metoda je poměrně intuitivní - nastartování deformace se obvykle provádí přesunem jednoho elementu. Také se po uživateli nepožadují žádné další vstupy, jako tomu bylo u některých předchozích metod. Rychlost metody závisí na počtu ovlivněných elementů - čím menší počáteční posun, tím menší počet přesunutých elementů a tedy rychlejší výpočet.

Tato metoda byla v [6] použita pro deformace měkkých tkání. Jako základ při simulaci endoskopie sloužila v práci [3]. Je to tedy metoda vhodná



Obrázek 2.5: Příklad jednoduchého použití 2D verze Chainmail algoritmu

i pro použití v programech, které vyžadují určitou fyzikální věrnost. Pro deformace nehomogenních dat je ovšem potřeba použít algoritmus Enhanced Chainmail [13].

Algoritmus Chainmail umožňuje rychlé deformace i velkých objektů, u kterých dokáže simulovat velký rozsah materiálů. Oproti jiným metodám, které modelují deformace složitými výpočty na malém množství elementů, tento algoritmus provádí jednoduché výpočty na velkých datech.

# Kapitola 3

## Implementace

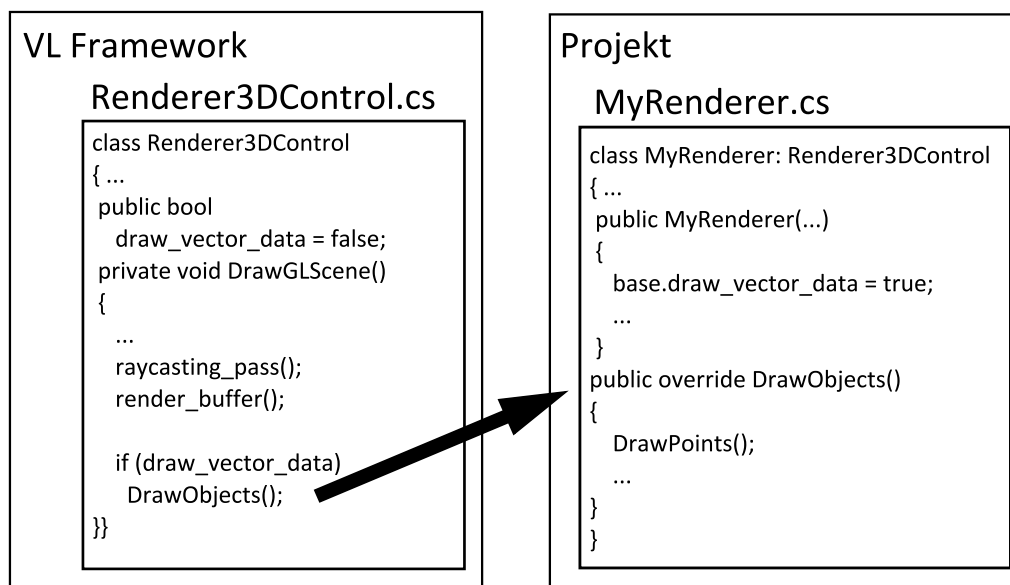
Požadavek na jednoduchost ovládání a dostatečnou věrnost a rychlost deformací nejlépe splňuje algoritmus 3D Chainmail. To byl také důvod, proč byl zvolen k implementaci v naší práci.

### 3.1 VL Framework

Pro zobrazování objemových dat se používá VL Framework [9]. Je to kompaktní framework pro zobrazování a práci s objemovými daty, který je psaný v programovacím jazyce C#. Jeho důležitou vlastností je rychlé renderování dat pomocí GPU.

Pro použití v naší práci byl mírně upraven kvůli nemožnosti vykreslování vektorových dat do scény v původní verzi. Do třídy `Renderer3DControl`, která slouží k 3D vizualizaci objemových dat, je přidána proměnná typu `boolean` a metoda s prázdným tělem. Proměnná má výchozí hodnotu `false`. Přidaná metoda je volána po zobrazení objemových dat za podmínky, že nová proměnná má hodnotu `true`. Od třídy `Renderer3DControl` je v projektu odvozena třída `MyRenderer`. Ve třídě `MyRenderer` je nová metoda z třídy `Renderer3DControl` předefinována kódem pro zobrazování vektorových dat. Tento způsob rozšíření VL Frameworku je kompatibilní s programy využívajícími jeho původní verzi. Ilustrace této modifikace je na obrázku 3.1.

Objemová data se standardně zobrazují pomocí izoploch (ISO surfaces). Používání této zobrazovací metody v programu znamená zvýraznění povrchu objemových dat. Tato zobrazovací metoda tedy nezobrazuje hodnoty voxelů uvnitř těchto dat. Z tohoto důvodu (a také pro zjednodušení) se v programu



Obrázek 3.1: Úprava VL Frameworku.

nepoužívá celý šedotónový rozsah voxelů, ale pouze hodnoty 0 nebo 1.

## 3.2 Algoritmus

### 3.2.1 Základní Chainmail algoritmus

Deformace objemových dat jsou založeny na základním algoritmu 3D Chainmail publikovaném Sarah F.F.Gibson [6], dále rozšířeném o funkci zvanou elastická relaxace (Elastic Relaxation), která je popsána v práci [7].

Tento algoritmus počítá se skutečným tvarem dat, tedy s trojrozměrnou mřížkou elementů. Každý element má až 6 ukazatelů na své přímé sousedy. Každý element musí splňovat jednoduché geometrické omezení - musí ležet v oblasti vymezené polohou svých sousedů (viz obrázek 3.2). Pokud toto omezení není splněno, dochází k tzv. "constraint violation" a element je přesunut tak, aby tato omezení opět splňoval. To ale může způsobit porušení těchto omezení u jeho sousedních prvků - deformace se tedy takto šíří objektem dál. Malé deformace (tedy posun o několik voxelů) se mohou zastavit po přesunu několika elementů, oproti tomu velké se mohou rozšířit klidně



Obrázek 3.2: Vymezený prostor sousedního elementu

přes celý objekt. Příklad jednoduché deformace je na obrázku 3.3. Hlavní výhodou tohoto algoritmu je velká rychlost - v práci [6] bylo dokázáno, že každý element je v průběhu deformace zpracován maximálně jednou.

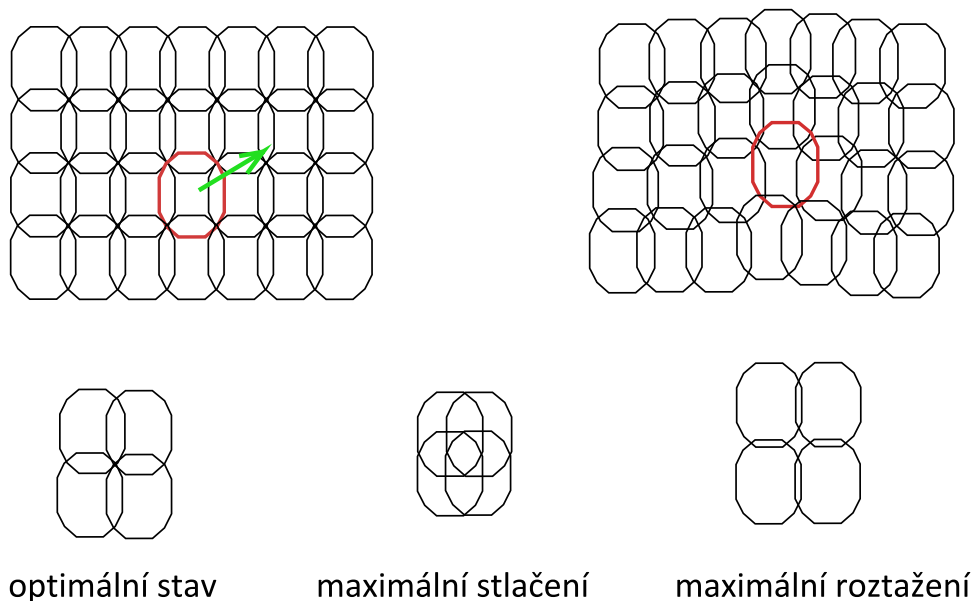
Hodnota MAXDIST (MINDIST) určuje maximální (minimální) vzdálenost sousedního prvku v hlavní ose (tj. osa, ve které je daný prvek sousedem) - např. pro pravého (nebo levého) souseda je to osa x. Hodnota MAXDELTA určuje maximální odchylku sousedního prvku ve dvou zbývajících osách - např. osy y a z. Hodnoty MAXDIST a MINDIST se dají v programu měnit. Hodnota MAXDELTA se vypočítá z těchto dvou hodnot jako  $MAXDELTA = (MAXDIST - MINDIST)/2$ .

Algoritmus používá 6 seznamů sousedů - pro každý typ souseda jeden (podle 3 os, pro každou osu 2 směry). Do těchto seznamů se ukládají elementy, u kterých existuje možnost, že se budou přesunovat - proto jim říkáme seznamy kandidátů. Navíc ještě potřebujeme i seznam všech elementů, u nichž došlo k přesunutí.

### 3.2.2 Popis algoritmu

1. Uživatel přesune element na novou pozici, element je přidán do seznamu přesunutých elementů a jeho souřadnice jsou upraveny na nové hodnoty. Jeho sousedé jsou vloženi do příslušných seznamů kandidátů.
2. Prochází se postupně všech 6 seznamů kandidátů. Seznam pravých sousedů se prochází následovně:

Kontroluje se, zda element leží ve svém vymezeném prostoru oproti jednomu z jeho sousedů. Oproti kterému sousedovi je poloha testována,



Obrázek 3.3: Příklad deformace 2D objektu (červený element je posunut ve směru šipky) a různé stavy sousedních elementů.

záleží na tom, do kterého seznamu element patří. U seznamu pravých sousedů je to vždy jejich levý soused (je to vždy soused "opačný" k typu seznamu). Kontrola a případná korekce probíhá tímto způsobem:

```

if (x - levý.x < MINDIST)
    x = levý.x + MINDIST;
else if (x - levý.x > MAXDIST)
    x = levý.x + MAXDIST;

if (y - levý.y < - MAXDELTA)
    y = levý.y - MAXDELTA;
else if (y - levý.y > MAXDELTA)
    y = levý.y + MAXDELTA;

if (z - levý.z < - MAXDELTA)
    z = levý.z - MAXDELTA;
else if (z - levý.z > MAXDELTA)

```

$z = \text{levý}.z + \text{MAXDELTA};$

Pokud dojde k přesunu elementu, všichni jeho sousedé kromě levého (tedy toho, oproti kterému se testuje poloha) jsou přidáni do seznamů kandidátů. Postupně jsou takto zpracovány všechny elementy z daného seznamu.

3. Seznam levých sousedů je zpracován podobně jako je popsáno v předchozím bodu, jenom jejich poloha se testuje vůči jejich pravým sousedům a při přesunu se do seznamů kandidátů přidávají všichni sousedé kromě pravého.
4. Ostatní seznamy jsou zpracovány obdobně v pořadí: horní, spodní, přední a zadní.

Aby se dal tento algoritmus použít i na nekonvexní objekty, je třeba jej trochu upravit, resp. přidat kus kódu. Pokud pravý nebo levý soused elementu ze seznamu horních (dolních) kandidátů nemá spodního (horního) souseda, tak by se měl přidat do seznamu kandidátů. Obdobně i pro elementy z ostatních seznamů kandidátů.

### 3.2.3 Paralelizace algoritmu

Kvůli zvýšení rychlosti deformací a čím dál většímu rozšíření vícejádrových procesorů byla přidána možnost nechat běžet část algoritmu paralelně.

K paralelizaci je použita knihovna Microsoft Parallel Extensions [11], která slouží ke zjednodušení paralelizace aplikací. Tato knihovna zahrnuje spoustu existujících konceptů paralelizace algoritmů připravených k použití. To značně redukuje možnost vzniku chyby v důsledku programátorova nesprávného použití vláken. Koncept, který je použit v programu, se nazývá "paralelní for-cyklus". Jedná se o obdobu klasického for-cyklu z programovacích jazyků s tím rozdílem, že u paralelního for-cyklu nejsou jednotlivé kroky prováděny postupně, ale vykonává se jich pomocí více vláken najednou několik. K tomu je nutné, aby tyto kroky byly navzájem nezávislé - aby nezáleželo na jejich vzájemném pořadí. Navíc je potřeba zajistit, aby proměnné používané ve for-cyklech byly buď lokální, a nebo aby k nim mohlo přistupovat najednou maximálně jedno vlákno.

For-cykly v algoritmu požadavek na nezávislost splňují. Ve for-cyklech dochází k přidávání elementů do seznamů kandidátů - operace *Add* na kolekci *List*  $\langle$ *Element* $\rangle$ . Jednou možností by bylo jednotlivé seznamy zamykat pomocí mutexů. Používání mutexů je ale velmi pomalé, protože zahrnuje složité volání do jádra operačního systému. Proto se používá složitější verze paralelního for-cyklu, která umožňuje sdílení dat v rámci jednoho vlákna. Při vzniku vlákna se vytvoří objekt obsahující lokální seznamy kandidátů, které se při vykonávání iterací for-cyklu naplní elementy. Tyto elementy se při zániku vlákna zkopírují do globálních seznamů kandidátů (to je jediné místo, kde je nutné mutex použít).

### Použití Microsoft Parallel Extensions

V algoritmu dochází k postupnému procházení všech seznamů kandidátů pomocí for-cyklů. Procházení levých sousedů v sériové verzi probíhá stylem:

```
count = LeftCandidate.Count;
for (i = 0; i < count; ++i)
{
    ...
}
```

Paralelní verze:

```
count = LeftCandidate.Count;
Parallel.For<Lists>(
    //první index, index o iteraci za posledním indexem
    0, count,
    //velikost iterace
    1,
    //metoda, kterou provede vlákno po svém vzniku
    delegate()
    {
        return new Lists();
    },
    //metoda zpracovávající jednu iteraci
    delegate(int i, ParallelState<Lists> state)
    {
        ...
    }
}
```

```

    },
    //metoda, kterou provede vlákno před zánikem
    delegate(Lists lists)
    {
        lock (mutex)
            MoveToGlobal(lists);
    },
    task_manager, TaskCreationOptions.None
);

```

Proměnná *task\_manager* slouží mj. k nastavení optimálního počtu procesorů a vláken. V programu je vytvořena následovně:

```

TaskManagerPolicy policy = new TaskManagerPolicy(1, 1, 2);
task_manager = new TaskManager(policy);

```

Konstruktor *TaskManagerPolicy* má parametry zleva: minimální počet procesorů, ideální počet procesorů a ideální počet vláken na procesor.

Při využití anonymních metod je přepsání do paralelní verze poměrně rychlé. Obdobně se paralelizace používá i u ostatních seznamů kandidátů. Paralelizaci je možno zapnout pomocí definování symbolu `USE_PARALLEL_FOR` v souboru se třídou *Deformation* (*Deformation.cs*). Pokud chceme použít sériovou verzi, tak stačí tuto definici zakomentovat. Testy rychlosti a porovnání paralelní a sériové verze jsou v kapitole 4.

### 3.2.4 Elastic Relaxation

Algoritmus 3D Chainmail produkuje data, která sice splňují určitá geometrická omezení, ale ne nutně mají optimální energetický stav. Elastic Relaxation vylepšuje výsledek spočítaný popsáním algoritmem tím způsobem, že přesune elementy do vhodných pozic tak, aby docházelo ke snižování energie celého systému. Energie systému závisí na vzdálenostech mezi elementy. Pokud se tyto vzdálenosti blíží optimální hodnotě, je celková energie systému nízká. Pokud jsou vzdálené od optimální hodnoty, energie systému roste.

Jde si to představit tak, že mezi jednotlivými elementy jsou pružiny, jejichž rovnovážný stav je při délce 1. Jsou-li sousední elementy blíž, pružiny se snaží sousedy od elementu odsunout, pokud jsou naopak dál, snaží se je

přítáhnout. Součet těchto sil (přesněji jejich absolutních hodnot) dohromady tvoří celkovou energii systému.

Během elastické relaxace je pozice každého elementu postupně přesunována tak, aby energie systému klesala.

Je několik možností, jak definovat optimální pozici elementu vzhledem k jeho sousedům. Nejjednodušší by mohlo být vzít za optimální pozici průměr pozic okolních sousedů. Snadno se ale dá představit, že by to způsobovalo smršťování okrajů dovnitř objektu.

Počítání optimální pozice elementu podle [7] probíhá tedy následujícím způsobem:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{opt} = \frac{1}{N} \sum_{sousedé} \begin{pmatrix} x_n - \Delta x_n \\ y_n - \Delta y_n \\ z_n - \Delta z_n \end{pmatrix}$$

$$\Delta x_n = \begin{cases} -\Delta x : & n \text{ je levý soused,} \\ +\Delta x : & n \text{ je pravý soused,} \\ 0 : & \text{pro ostatní sousedy.} \end{cases}$$

$$\Delta y_n = \begin{cases} -\Delta y : & n \text{ je spodní soused,} \\ +\Delta y : & n \text{ je vrchní soused,} \\ 0 : & \text{pro ostatní sousedy.} \end{cases}$$

$$\Delta z_n = \begin{cases} -\Delta z : & n \text{ je zadní soused,} \\ +\Delta z : & n \text{ je přední soused,} \\ 0 : & \text{pro ostatní sousedy.} \end{cases}$$

kde  $N$  je počet sousedů elementu a hodnoty  $\Delta x$ ,  $\Delta y$  a  $\Delta z$  jsou optimální vzdálenosti dvou vedlejších elementů v daných osách (v programu se pro všechny tři používá hodnota 1).

### 3.3 Interpolace

Při provádění deformací se mohou dva sousední elementy od sebe vzdálit i o několik voxelů. Četnost tohoto jevu závisí také na hodnotě konstanty MAXDIST. Pokud bude mít konstanta MAXDIST velkou hodnotu, tak dva sousední prvky budou moci být od sebe daleko bez toho, že by došlo

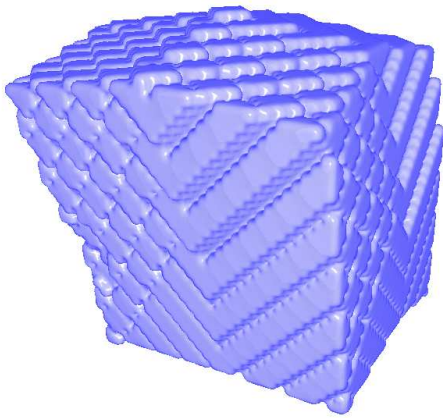
k porušení geometrických omezení a spuštění deformace. Proto v datech vznikají díry, které nevypadají hezky.

Snaha tedy byla tomuto jevu zamezit nebo alespoň jej omezit pomocí rychlého algoritmu, který by nezpomaloval již tak dost náročnou deformaci. Během interpolace se prochází seznam všech elementů a pokud má element pravého souseda, provede se následující pseudokód:

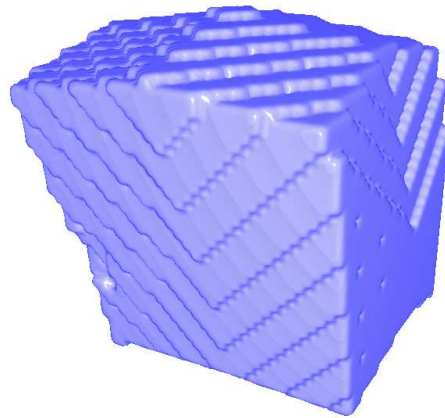
```
if ( pravýSoused.VoxelX > element.VoxelX + 1 && voxel na
    pozici (element.VoxelX + 1) je prázdný)
{
    přidejElement(element.VoxelX + 1, element.VoxelY,
        element.VoxelZ);
}
else if (pravýSoused.VoxelX == element.VoxelX+1 &&
    (y-ová nebo z-ová souřadnice elementu a pravého souseda
    se liší) &&
    voxel na pozici (element.VoxelX+1) je prázdný
    )
{
    přidejElement(element.VoxelX + 1, element.VoxelY,
        element.VoxelZ);
}
```

Obdobným způsobem se zkontrolují i pozice vrchního a předního souseda. Rozdíl mezi daty při použití interpolace je vidět na obrázku 3.4.

Vypnutá interpolace



Zapnutá interpolace



Obrázek 3.4: Porovnání deformované krychle se zapnutou a vypnutou interpolací.

# Kapitola 4

## Výsledky

V této kapitole se testuje rychlost algoritmu s různým nastavením. Testy byly prováděny na počítači s konfigurací Core 2 Duo 2.0 GHz, 2 GB RAM paměti a nVidia GeForce 8600M GT. Naměřené výsledky jsou průměrem ze 3 provedených testů.

### 4.1 Reálná data

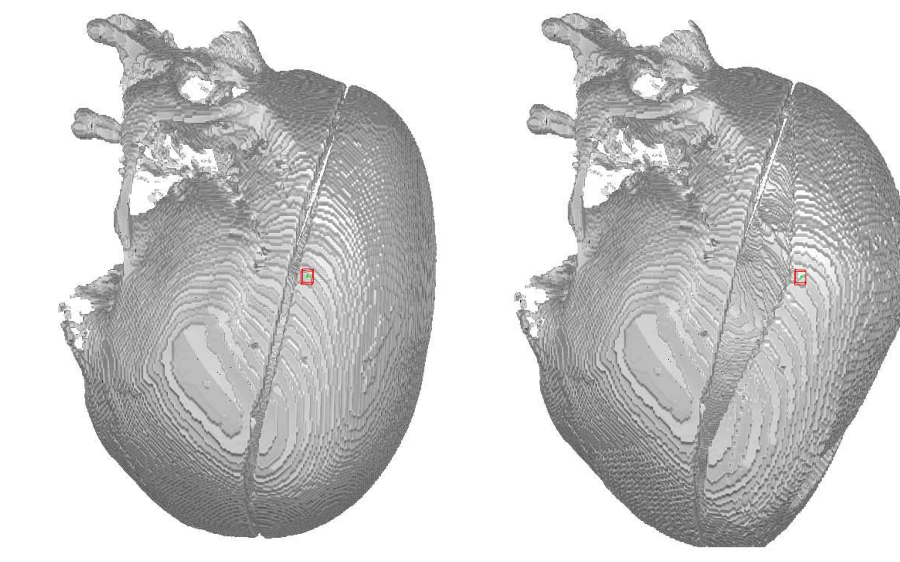
Nejprve program otestujeme na reálných lékařských datech získaných pomocí počítačového tomografu a převedených do formátu vox (tento formát je popsán na konci kapitoly 6). Tato data jsou anonymní.

V testu A se deformuje lidská lebka. Element, který je nad vodorovným řezem, je posunut o vzdálenost přibližně 26 voxelů směrem kolmo od něj. Celková velikost dat je 256 x 256 x 323 voxelů. Průběh testu je znázorněn na obrázku 4.1.

Pro deformaci v testu B slouží objemová data znázorňující lidskou hlavu. Deformuje se pravé ucho, které je posunuto o cca 6 voxelů směrem od hlavy (viz obrázek 4.2). Celková velikost dat je 170 x 170 x 101 voxelů.

Provedená operace	Doba výpočtu
Test A	
Deformace - sériová	330 ms
Deformace - paralelní	431 ms
Deformace - sériová plus jedna elastická relaxace	537 ms
Deformace - sériová plus dvě elastické relaxace	715 ms
Test B	
Deformace - sériová	74 ms
Deformace - paralelní	79 ms
Deformace - sériová plus jedna elastická relaxace	325 ms
Deformace - sériová plus dvě elastické relaxace	472 ms

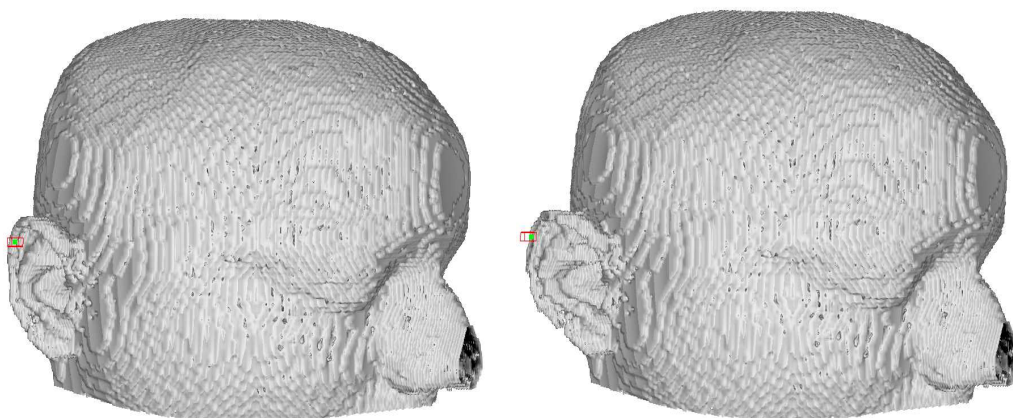
Tabulka 4.1: Výsledky testů A a B.



Obrázek 4.1: Průběh testu A, původní (levý obrázek) a výsledný tvar dat.

## 4.2 Umělá data

V testu C je deformován kvádr o velikosti 149 x 89 x 149 voxelů (je umístěn v mřížce o velikosti 256 x 256 x 256 voxelů). Element, který tvoří jeden



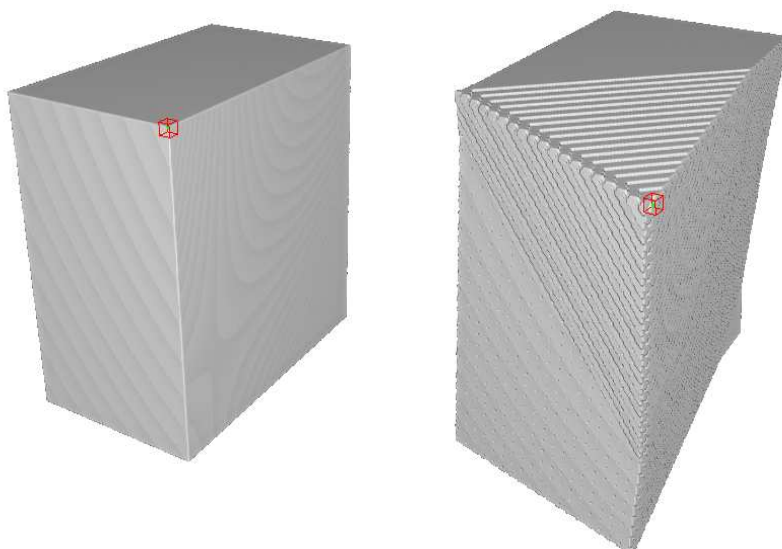
Obrázek 4.2: Průběh testu B, původní (levý obrázek) a výsledný tvar dat.

z horních rohů kvádru, je posunut o 60 voxelů směrem mimo kvádr o něco níž než byla jeho původní poloha (viz obrázek 4.3).

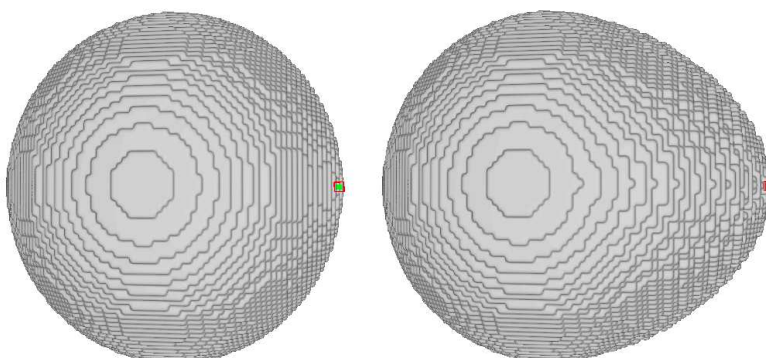
V testu D je deformována koule o průměru 75 voxelů, která je součástí dat o velikosti 100 x 100 x 100 voxelů. Element je posunut ve vodorovné rovině cca 13 voxelů směrem od středu koule (obrázek 4.4).

Provedená operace	Doba výpočtu
Test C	
Deformace - sériová	1641 ms
Deformace - paralelní	1857 ms
Deformace - sériová plus jedna elastická relaxace	1989 ms
Deformace - sériová plus dvě elastické relaxace	2290 ms
Test D	
Deformace - sériová	138 ms
Deformace - paralelní	204 ms
Deformace - sériová plus jedna elastická relaxace	205 ms
Deformace - sériová plus dvě elastické relaxace	256 ms

Tabulka 4.2: Výsledky testů C a D



Obrázek 4.3: Průběh testu C, původní (levý obrázek) a výsledný tvar dat.



Obrázek 4.4: Průběh testu D, původní (levý obrázek) a výsledný tvar dat.

### 4.3 Shrnutí

Paralelní verze je ve všech testech o něco pomalejší než sériová. Zdá se, že režie na překopírovávání lokálních seznamů vláken do globálních zabírá více

času než jaký je celkový přínos plynoucí z paralelizace.

Výsledky ukazují, že na rychlost algoritmu nemá vliv celkový objem dat, ale pouze těch dat, která byla deformací ovlivněna. Je to vidět v testu B, kdy jsou použita poměrně velká data, ale algoritmus bez elastické relaxace proběhl za nejkratší dobu ze všech testů.

Naopak elastická relaxace se provádí na celých datech. Proto u již zmiňovaného testu B zabírá velkou část výpočetního času elastická relaxace. Opak lze vidět u testu D, kdy jsou použita poměrně malá data.

# Kapitola 5

## Uživatelská dokumentace

### 5.1 Úvod

Objemová data jsou prostředek, kterým lze přinést mnohem více informace do 3D počítačové grafiky než dovoluje tradiční povrchový model. Ze zpracování objemových dat těží například medicína, materiálový či chemický průmysl. S rostoucí kapacitou a rychlostí počítačů nachází objemová data čím dál větší uplatnění. Proto je také třeba vyvíjet nástroje, které nám umožní s těmito daty snadno manipulovat.

Tento program slouží k deformování objemových dat. Tyto deformace se dají přirovnat k warpingu 2D obrázku, oproti kterému jsou ovšem prováděny na trojrozměrných datech. Program se snaží zachovat si vlastnosti důležité pro interaktivní editační nástroj - tedy rychlost a uživatelskou přívětivost. Zároveň však usiluje o rozumnou věrnost prováděných deformací.

### 5.2 Požadavky

Doporučená konfigurace:

- dvoujádrový procesor na 2.0 GHz
- 2 GB RAM
- grafická karta nVidia řady 6000 nebo lepší

Pozn: Program byl testován na konfiguraci Core 2 Duo 2.0 GHz, 2 GB RAM a nVidia GeForce 8600M GT. Obdobná sestava je proto pro běh programu

doporučována, i když program může pro menší data svižně pracovat i na pomalejších stroji.

## 5.3 Ovládání

Otáčení a zoomování scény se provádí pomocí myši klasicky jako ve většině grafických programů (tj. "chycení" levým tlačítkem a pohybem myši se provede otočení, zoomování je pomocí kolečka myši).

### 5.3.1 Klávesové zkratky

- Alt + q - otevření souboru
- Alt + s - uložení souboru
- F5 - quicksave
- F6 - quickload

## 5.4 Hlavní okno programu

Hlavní okno se dělí na 2 hlavní části - část zobrazovací vlevo a část nástrojovou vpravo.

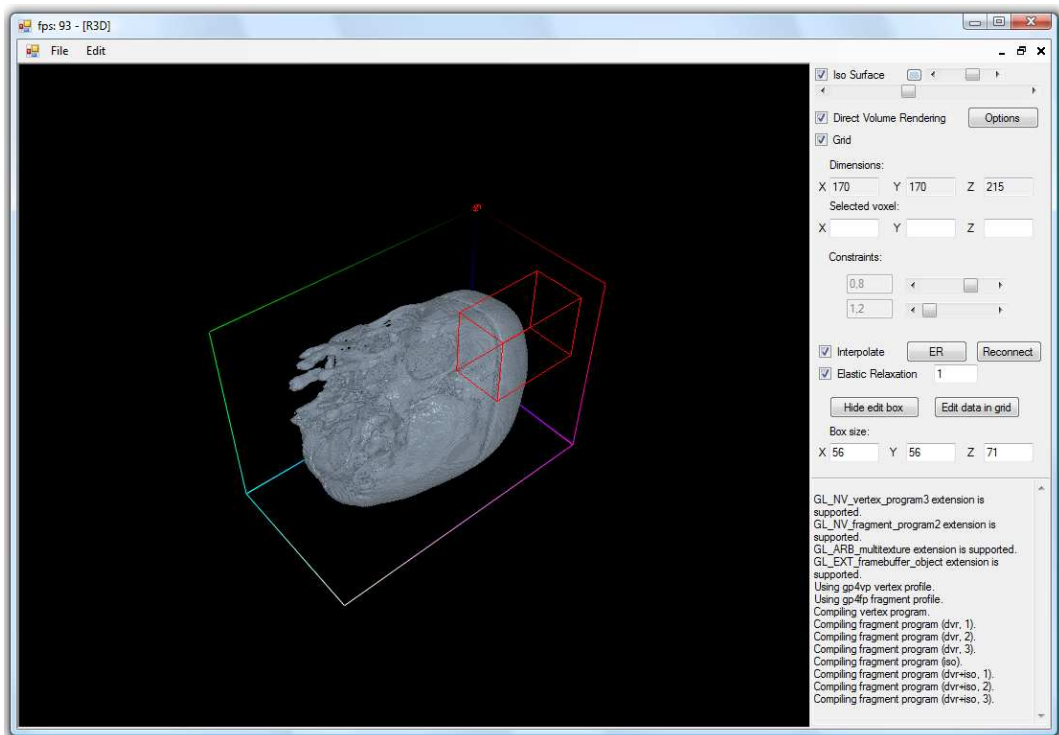
### 5.4.1 Zobrazovací část

Zobrazovací část programu slouží, jak již název napovídá, k zobrazování dat. V této části se též provádějí jejich deformace a v horní části se nachází hlavní menu.

#### Hlavní menu

File -

- Open File - zobrazí okno s výběrem souboru, který chceme deformovat.
- Save - uloží aktuální stav dat do stejného souboru, ze kterého byla data načtena.



Obrázek 5.1: Hlavní okno programu s načtenými daty

- Save as ... - uloží aktuální data do souboru, který si sami zvolíme.
- Exit - ukončí aplikaci (bez uložení dat).

Edit -

- Undo - zruší poslední provedenou deformaci. Nelze použít opakovaně.
- Redo - zruší právě provedené Undo.
- Quicksave - uloží aktuální tvar objemových dat.
- Quickload - načte uložený tvar objemových dat a vytvoří k nim nové napojení elementů (provede reconnect).

#### 5.4.2 Nástrojová část

V nástrojové části se nastavuje způsob zobrazování dat a různé možnosti ovlivňující provádění deformací. Tato část také poskytuje různé informace,

např. o dimenzi zobrazovaných dat nebo souřadnice právě vybraného voxelu.

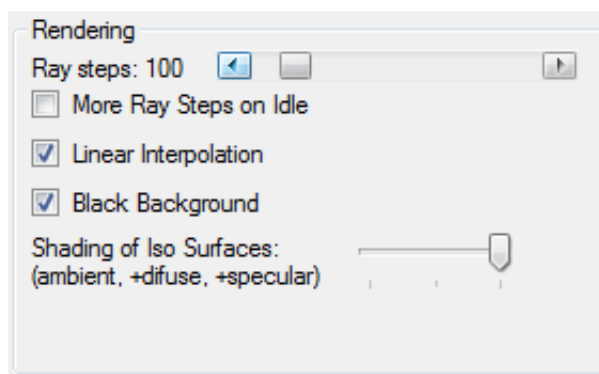
- **Iso surface** - zapíná zobrazování objemových dat pomocí izoploch, tlačítko napravo zobrazuje paletu barev pro izoplochy, posuvníkem u tohoto tlačítka se nastavuje alfa kanál barvy a spodním posuvníkem hustota izoploch.
- **Direct Volume Rendering** - zapíná Direct volume rendering - trochu mění vzhled zobrazovaných dat.
- tlačítko **Options** - zobrazí okno s pokročilejšími možnostmi nastavení. Popis tohoto okna je dále v textu.
- **Grid** - zapíná zobrazování mřížky na povrchu objemových dat.
- **Dimensions** - zobrazuje počty voxelů v daných osách.
- **Selected voxel** - pokud je vybrán nějaký voxel, tak se zde zobrazují jeho souřadnice.
- **Constraints** - zobrazuje aktuální hodnoty MINDIST a MAXDIST, u obou hodnot jsou i posuvníky, kterými se tyto hodnoty dají měnit.
- **Interpolate** - zapíná interpolaci po provedení deformace.
- tlačítko **ER** - provede elastickou relaxaci dat.
- tlačítko **Reconnect** - přenastaví napojení jednotlivých elementů.
- **Elastic Relaxation** - zapíná relaxaci po provedení deformace, v políčku se nastavuje počet prováděných relaxací.
- **Show/Hide edit box** - zobrazí/skryje kvádr výběru dat.
- **Edit data in grid/End part-def** - spustí/ukončí omezení deformace na část dat.
- **Box size** - obsahuje rozměry kvádru pro výběr části dat.
- V dolní pravé části se nachází okno pro výpisy programu.

Pro uživatele jsou tyto výpisy pouze informační a v optimální situaci se jimi vůbec nemusí zatěžovat (jinak se tam např. vypisují použítá OpenGL rozšíření grafické karty, také se tam vypisují informace

v případě, že karta nepodporuje rozšíření, která jsou pro běh programu potřeba).

Všechny ovládací prvky v nástrojové části mají v programu tooltips - stačí chvíli podržet myš nad daným prvkem.

## Okno options



Obrázek 5.2: Okno options

- posuvník **Ray steps** - nastavuje počet kroků jednoho paprsku během ray-tracingu. Vyšší hodnoty znamenají hezčí vzhled dat, ale za cenu méně snímků za sekundu.
- **More Ray Steps on Idle** - pokud je zapnuta tato možnost, tak se automaticky zvýší počet kroků ray-tracingu během doby, kdy uživatel neprovádí otáčení kamery, přiblížení, apod.
- **Linear Interpolation** - zapíná lineární interpolaci zobrazovaných dat, která vylepšuje vzhled zobrazovaných dat.
- **Black Background** - zapnuto znamená černé pozadí dat, vypnuto bílé.
- **Shading of Iso Surfaces** - nastavuje stínování izoploch. Zleva doprava od nejméně k nejvíce hezkému.

## 5.5 Tutorial

### 5.5.1 Deformace celého objemu dat

1. V menu File vybereme položku Open File.
2. Pomocí dialogu, který se nám otevřel, vybereme zvolený soubor obsahující objemová data ve formátu vox.
3. Dojde k zobrazení dat, tato data lze libovolně otáčet, zoomovat apod.
4. Pomocí pravého tlačítka myši a při stisknuté klávesy Ctrl vybereme element, kterým budeme pohybovat, ten se po vybrání zobrazí zeleně.
5. Elementem se dá pohybovat pouze ve směru kolmém k pohledu kamery, proto kameru vhodně natočíme.
6. Za držení pravého tlačítka myši posuneme kurzor v požadovaném směru - souřadnice, kam dojde k posunutí elementu, jsou vyznačeny červenou krychlí.
7. Až budeme mít vybrány požadované souřadnice, tak pravé tlačítko myši pustíme a tím dojde ke spuštění deformace.

### 5.5.2 Deformace vybrané části dat

1. Načteme data stejně jako při editaci celých dat
2. Klikneme na tlačítko "Show edit block" - tím dojde ke zobrazení kvádrů výběru dat a v "Box size" se zobrazí aktuální velikost kvádrů. Při deformaci části dat se budou deformovat jenom ta data, která leží v tomto kvádru.
3. Nastavíme požadovanou velikost a polohu kvádrů.

Velikost kvádrů se nastavuje pomocí políček v "Box size" - jednoduše se požadovaná délka v daném rozměru napíše do příslušného pole. Poloha kvádrů se dá nastavit pomocí myši i klávesnice. Klávesy  $z$  a  $u$  slouží k posunu kvádrů ve směru osy  $x$ , klávesy  $h$  a  $j$  ve směru osy  $y$ ,  $n$  a  $m$  ve směru osy  $z$ . Posun myši za držení pravého tlačítka pohybuje kvádrem ve směru kolmém k pohledovému vektoru kamery.

4. Klikneme na tlačítko "Start part-def". Nyní se dají deformovat pouze data uvnitř zvoleného kvádrů.
5. Provedeme požadovanou deformaci stejně jako při deformování celých dat.
6. Ukončíme režim deformace vybrané části dat kliknutím na tlačítko "End part-def". Nyní se budou deformovat opět celá data.

# Kapitola 6

## Programátorská dokumentace

Programovací jazyk: C#

Vývojové prostředí: Microsoft Visual Studio 2005

Operační systém: Windows Vista

### 6.1 Požadavky

Kvůli použitému VL frameworku je nutná grafická karta od firmy nVidia, a to řady alespoň 6000.

#### 6.1.1 Potřebné knihovny

Ke správnému překladu a běhu programu jsou potřeba následující knihovny:

- VL framework  
Je součástí diplomové práce [9], používá se na zobrazování objemových dat. Tento framework byl lehce upraven pro vykreslování vektorových dat do scény.
- TAO Framework [4]  
Nutný k běhu VL frameworku.
- nVidia Cg Toolkit (konkrétně cg.dll a cgGL.dll) [12]  
Knihovny nutné k běhu VL frameworku.

- Microsoft Parallel Extensions [11]  
Zjednodušuje psaní vícevláknových aplikací. Bude součástí .Netu 4.0.
- Microsoft .NET Framework [10]  
Ve verzi 3.5 nebo vyšší.

## 6.2 Program

### 6.2.1 Testovací interface

Program obsahuje interface pro testování rychlosti deformací. Tento interface se použije pokud je program spuštěn s argumenty. Tvar argumentů:

```
vox_file zobrazit_výsledek počet_relaxací x1 y1 z1 x2 y2 z2
```

- vox\_file - název souboru s daty
- zobrazit\_výsledek - pokud bude "yes" nebo "true" tak se zobrazí výsledná data po provedení deformace
- počet\_relaxací - počet provedených elastických relaxací po deformaci
- x1, y1, z1 - souřadnice voxelu, který bude přemístěn
- x2, y2, z2 - výsledné souřadnice voxelu

V případě, že program není spuštěn s parametry, je vhodné okno s konzolí skryt. To je provedeno pomocí WinAPI funkcí ze sdílené knihovny user32.dll. Nejprve je nutno tyto funkce naimportovat:

```
[DllImport("user32.dll")]
public static extern IntPtr FindWindow(string lpClassName,
    string lpWindowName);
```

```
[DllImport("user32.dll")]
static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
```

Potom se nastaví jméno okna s konzolí na "Voldeform", na toto okno se pomocí funkce *FindWindow* najde handle a pomocí funkce *ShowWindow* se skryje.

```

Console.Title = "Voldeform";
IntPtr hWnd = FindWindow(null, "Voldeform");
if (hWnd != IntPtr.Zero)
{
    ShowWindow(hWnd, 0); // 0 = SW_HIDE
}

```

## 6.2.2 Funkce reconnect

Funkce reconnect způsobuje znovuvytvoření mřížky elementů a vzájemné napojení těchto elementů následujícím způsobem.

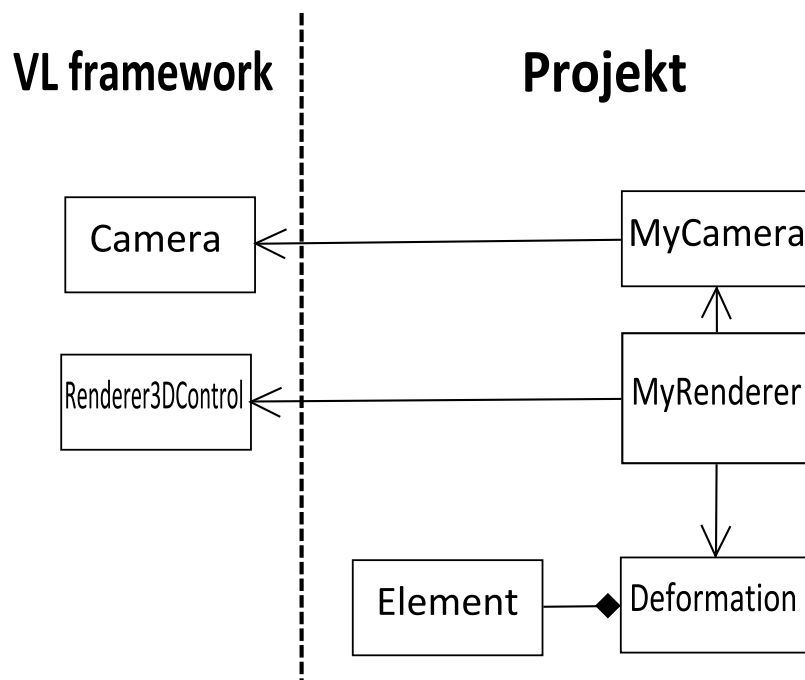
Pro každý nenulový voxel na scéně se vytvoří element. Pro každý element se najdou všichni jeho sousedé. Za souseda je považován element ležící ve vzdálenosti 1 voxel od elementu. Tedy např. pravým sousedem elementu na souřadnicích  $(x,y,z)$  by mohl být element  $(x+1,y,z)$ . Pokud takový element na těchto souřadnicích neexistuje, pak element pravého souseda nemá.

Pokud se někde vyskytne element nebo skupina propojených elementů, které ale s ostatními propojeny nejsou, dojde k rozpadu dat na více částí.

## 6.2.3 Části programu

Diagram znázorňující vztahy důležitých tříd programu je na obrázku 6.1. Zdrojový kód programu je rozdělen do několika souborů:

- Program.cs - vstupní bod programu - zobrazení hlavního okna a zpracování nezachycených výjimek.
- Main.cs - hlavní okno programu (s oknem třídy MyRenderer), obsahuje kód starající se o načítání, ukládání dat a několik handlerů pro uživatelské interakce (tlačítka, zatrhávací boxy, posuvníky).
- SplashScreen.cs - okno zobrazující se při načítání a zpracování dat.
- MyCamera.cs - obsahuje třídu MyCamera, která je odvozená od třídy Camera z VL frameworku. Obsahuje proměnnou obsahující informaci, zda je stisknuta klávesa Ctrl (používá se při výběrání voxelu).
- MyRenderer.cs - obsahuje třídu MyRenderer odvozenou od třídy Rendere3DControl z VL frameworku, ta se používá především kvůli



Obrázek 6.1: Class diagram

vykreslování vlastních dat do scény a uživatelským interakcím na datech. Obsahuje reference na třídy Deformation a MyCamera. Třída MyRenderer zobrazuje data, po zásahu uživatele vedoucím na deformaci připraví datové struktury a zavolá metodu ze třídy Deformation. Po skončení deformace znovu vyrenderuje data a zobrazí je.

- Deformation.cs - obsahuje třídy Deformation a Element. Třída Element odpovídá jednomu elementu v 3D mřížce deformovaného objektu. Obsahuje jeho souřadnice, souřadnice odpovídajícího voxelu, ukazatele na sousedy a další. Třída Deformation obsahuje datové struktury a metody potřebné na provádění deformací - je to např. seznam všech elementů (tj. instancí třídy Element) a reference na ně uspořádané do trojrozměrné mřížky (kvůli rychlejšímu nalezení sousedů).
- Other.cs - obsahuje pomocné datové struktury používané v programu.

## 6.3 Vstupní a výstupní data

Program pracuje se soubory typu "vox", což je jednoduchý textový soubor vytvořený pro použití v tomto projektu. Tento typ souboru má následující obsah:

První řádek obsahuje dimenze objemových dat oddělené středníkem nebo mezerou, za touto trojicí ještě mohou následovat rozměry voxelu (opět odděleny středníkem nebo mezerou). Dimenze objemových dat musí být celá kladná čísla, rozměry voxelu mohou být čísla s plovoucí desetinnou čárkou. Správný tvar prvního řádku je tedy např. "100;60;70", "100 60 70", "100 60 70 0.330078 0.6 0.330078" apod.

Další řádky obsahují souřadnice bodů, které patří do objemových dat (např. "34,23,2"). Tyto souřadnice mohou být následovány dvojtečkou a hodnotou voxelu na dané souřadnici (např. ":145"), ale jako hodnota se použije vždy 255 - tato možnost je zde ponechána jen z důvodu kompatibility se starším typem souboru. Počet souřadnic na jednom řádku je libovolný a mezi sebou musí být souřadnice odděleny středníkem. Souřadnice, které jsou mimo dimenzi objemových dat (která byla na prvním řádku), se ignorují. Do stejného typu dat program pochopitelně objemová data i ukládá.

# Kapitola 7

## Závěr

Cílem této práce byl program na deformování objemových dat. Tyto deformace měly být kompromisem mezi věrností, rychlostí a uživatelskou přívětivostí.

V práci bylo rozebráno několik používaných přístupů deformace objemových dat a z nich zvolen algoritmus 3D Chainmail, který byl díky svým vlastnostem nejvhodnějším kandidátem. Splňuje požadavky pro věrnost a je pro uživatele poměrně intuitivní. Při porovnávání rychlosti deformací je potřeba vzít v potaz velikost dat, nebo přesněji velikost dat, která byla deformací ovlivněna (jak bylo vidět i při testování rychlosti algoritmu v kapitole 4).

Pro zobrazování objemových dat byl zvolen poměrně nový VL Framework, který vyniká velkým počtem snímků za sekundu. Aby se změna dat po provedení deformace projevila při zobrazování, je potřeba celá data znovu vyrenderovat. To může být u velkých dat poměrně časově náročné.

Dá se tedy říci, že u dat, která nejsou příliš velká, program všechny požadavky splňuje. Ovšem u velkých dat je již při deformaci potřeba počítat s určitou prodlevou, která celkovou interaktivitu programu poněkud snižuje.

### 7.1 Možná rozšíření

Další práce by se mohla zaměřit na takovou úpravu algoritmu, která by umožňovala provádět deformace spouštěné pomocí více elementů. Ať už pomocí několika dalších okolních voxelů nebo třeba voxelů ležících na nějaké křivce na povrchu objemových dat.

Již dnes se vývoj procesorů zaměřuje na zvyšování počtu jader, ale jejich

frekvence již několik let stagnují. Proto by bylo vhodné se zaměřit na úpravu paralelní verze algoritmu, která je v aktuální verzi o něco pomalejší než sériová.

Další možností by bylo přepsat program tak, aby se dal použít i na operačním systému Linux. To by ovšem nejspíš znamenalo předělat i část VL Frameworku. Porovnání rychlosti algoritmu pod různými operačními systémy by ale jistě bylo zajímavé.

# Literatura

- [1] Alan H. Barr. Global and local deformations of solid primitives. *ACM SIGGRAPH Computer Graphics, Volume 18*, 1984.
- [2] Sabine Coquillart. Extended free-form deformation: a sculpturing tool for 3D geometric modeling. *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990.
- [3] Christopher Dräger. A Chainmail Algorithm for Direct Volume Deformation in Virtual Endoscopy Applications. Master's thesis, Technische Universität Wien, 2005.
- [4] The Tao Framework. Tao framework. <http://www.taoframework.com> (30.7.2009), 2007.
- [5] Sarah F. F. Gibson and Brian Mirtich. A Survey of Deformable Modeling in Computer Graphics. Technical report, Mitsubishi Electric Research Laboratories, 1997.
- [6] Sarah F.F. Gibson. 3D chainmail: A Fast Algorithm for Deforming Volumetric Objects. *Proceedings of the 1997 symposium on Interactive 3D graphics*, 1997.
- [7] Sarah F.F. Gibson. Using Linked Volumes to Model Object Collisions, Deformation, Cutting, Carving, and Joining. *IEEE Transactions on Visualization and Computer Graphics*, 1999.
- [8] Gentaro Hirota, Renee Maheshwari, and Ming C. Lin. Fast Volume-Preserving Free Form Deformation Using Multi-Level Optimization. In *Proc. Symposium on Solid Modeling and Applications*, pages 234–245. ACM Press, 1999.

- [9] Jakub Hlaváček. Gpu-accelerated processing of medical data. Master's thesis, Charles University in Prague, Faculty of Mathematics and Physics, 2008.
- [10] Microsoft. Microsoft .NET Framework. <http://www.microsoft.com/net> (30.7.2009).
- [11] Microsoft. Microsoft Parallel Extensions. <https://connect.microsoft.com/site/sitehome.aspx?SiteID=516> (30.7.2009).
- [12] nVidia. nVidia Cg Toolkit. [http://developer.nvidia.com/object/cg\\_toolkit.html](http://developer.nvidia.com/object/cg_toolkit.html) (30.7.2009).
- [13] Markus A. Schill, Sarah F. F. Gibson, H.-J. Bender, and R. Manner. Biomechanical Simulation of the Vitreous Humor in the Eye Using an Enhanced Chainmail Algorithm. *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98*, pages 679–687, 1998.
- [14] Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 1986.

# Dodatek A

## Obsah CD

Součástí práce je i přiložené CD, které obsahuje následující adresáře:

- bin** Přeložený projekt a potřebné sdílené knihovny.
- data** Objemová data ve formátu vox.
- doc** Text práce v elektronickém formátu a dokumentace vygenerovaná pomocí Doxygenu.
- install** Instalátory pro některé potřebné knihovny.
- src** Projekt pro Microsoft Visual Studio 2005.