Design and Development of a VR System for Exploration of Medical Data Using Haptic Rendering and High Quality Visualization

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

DOKTOR DER NATURWISSENSCHAFTEN
Dr. rer. nat.

genehmigte Dissertation
von

M.Sc. Roman Vlasov

2016

Referent: Prof. Dr. Franz-Erich Wolter
Korreferent: Prof. Dr. Gabriel Zachmann
Tag der Promotion: 4. November 2016

Kurzzusammenfassung

Haptische Exploration fügt der Arbeit mit 3D Daten eine neue Dimension hinzu: Die Möglichkeit, Objekte zu berühren. Dies erlaubt neue Möglichkeiten in der medizinischen Simulation, Ausbildung und präoperativen Planung in einer Virtual Reality Umgebung.

Eine einzelne Momentaufnahme einer solchen haptischen Rückkopplung besteht aus drei Schritten: Kollisionserkennung, Kollisionsantwort und Kraftgenerierung. Um ein natürliches, verzögerungsfreies Arbeiten zu ermöglichen, wird eine Wiederholrate von mindestens 1 kHz benötigt, für die es unterschiedliche Ansätze von oberflächen- und voxelbasierten Renderingmethoden gibt. Ein Nachteil fast aller bisher verwendeten Verfahren ist dabei, dass entweder keine Garantien für die Einhaltung der Wiederholrate gegeben werden kann oder die simulierten Objekte einer speziellen topologischen Struktur entsprechen müssen. Dies ist besonders für sensible Prozesse wie die Operationsplanung kritisch. Um dies zu beheben, wurde eine neue, robuste und schnelle (150 kHz) Methode entwickelt, die Ansätze aus Ray Casting und Path Finding kombiniert und dabei nahezu konstante Zeitkomplexität hat. Da die Methode ohne zeitaufwendige Initialisierung arbeitet und auf impliziten Oberflächenmodellen eingesetzt werden kann, lassen sich auch dynamische Objekte abbilden. Darauf aufbauend präsentieren wir ein flexibles Deformation Framework, das es erlaubt, unsere haptische Renderingmethode mit verschiedenen Deformationsmodellen zu kombinieren. Es wird ein neues Echtzeit-Visualisierungs-Verfahren vorgestellt, um die graphische Darstellung der Segmente mit der Simulation zu synchronisieren und eine interaktive (bleibende) Echtzeit-Deformation der Objekte zu ermöglichen. Für diesen Zweck wurden zwei auf Potential Fields basierende Methoden für die lokale Deformations-Simulation entwickelt und eingesetzt. Die erste Methode verwendet reguläre Potential Fields. Die zweite Methode nutzt unsere neuen Cuboid Fields aus. Weiterhin zeigen wir, dass diese Cuboid Fields für das haptische Rendering von Volumendaten besser geeignet sind. Darüber hinaus schlagen wir einen Prototyp einer globalen Deformationsmethode vor. Der gesamte Ansatz aus den vorgeschlagenen Methoden zum haptischen Rendering, zur Visualisierung und Deformation (Deformation Framework) erfordert keine Vorkalkulation.

Das in dieser Arbeit vorgestellte Deformation Framework und alle Haptik Rendering-Visualisierungs und Deformations-Methoden wurden komplett neu entwickelt. Das Design und die Entwicklung dieser Methoden waren das Hauptziel dieser Arbeit. Diese Arbeit wurde durch das Siemens/DAAD Postgraduate Programme unterstützt.

Abstract

Haptic exploration adds an additional dimension to working with 3D data: a sense of touch. This is especially useful in areas such as medical simulation, training and pre-surgical planning, as well as in museum display, sculpting, CAD, military applications, assistive technology for blind and visually impaired people, entertainment and others.

Each haptic rendering frame consists of three stages: collision detection, collision response and force feedback generation. In order to feel the 3D data smoothly, an update rate of at least 1 kHz is required. There exist different surface- and voxel-based haptic rendering methods. Unaddressed practical problems for almost all of them are that no guarantees for collision detection could be given and/or that a special topological structure of the objects is required. Here we present a novel and robust approach based on employing the ray casting technique to collision detection and path finding for collision response. The approach is very fast (150 kHz) and does not have the aforementioned drawbacks while guaranteeing nearly constant time complexity, independent of data resolution. This is especially important for delicate procedures, e.g. pre-operation planning. The collision response uses an implicit surface representation, which can be used with dynamically changing objects, as no precalculation is needed. Further on, we present our flexible deformation framework allowing us to use our haptic rendering approach together with deformation models. We present our graphics approach which we use to keep the graphics representation of segments up-to-date during the deformation simulation. The challenge here is to reflect deformations of objects interactively. Further on, we propose two local deformation simulation approaches based on the method of potential fields. The first approach uses "regular" potential fields. The second approach uses our novel cuboid fields. Further on, we demonstrate that cuboid fields are better suited to haptic rendering of volumetric data. Additionally, we introduce the prototype of the global deformation approach. The resulting haptic rendering approach combined with our proposed approaches for deformation simulation within our deformation framework does not require any pre-calculated structure and works "on the fly".

Our deformation framework and all our haptic rendering and deformation simulation approaches were fully developed by us from scratch. Their design and development was the main aim of this work. This project was supported by a grant provided by Siemens/DAAD Postgraduate Programme.

German Keywords: Virtuelle Realität; Haptik Rendering; Visualizierung

English Keywords: Virtual Reality; Haptic Rendering; Visualisation

Preface

I would like to thank Prof. Dr. Franz-Erich Wolter for giving me the possibility to make the PhD at Welfenlab and for supervising and guiding me during the doctoral studies. I would like to thank Dr. Karl-Ingo Friese for helping me all the way during the doctoral studies, for giving me very useful advices, for his support and for helpful comments. I would like to thank Prof. Dr. Nadia Magnenat-Thalmann for a possibility to work at her research institute in Singapore and for giving me one day a week to work on my PhD there. Working there was an amazing and an important and useful experience. I would like to thank Dr. Alexander Vais, Hannes Thielhelm, Roman Burg, Victor Opilat, Neetha Das, Sergej Zerr, Philipp Blanke, Jan Rzepecki, Andreas Tarnowsky, Maximilian Klein, Martin Gutschke, Ricardo Millan, Benjamin Berger, Rasmus Buchmann and my other friends and colleagues for their support, for checking my dissertation and for giving helpful comments and advices. I would like to thank Ms. Ilona Esz and Mrs. Anca Vais for their support and for their help guiding me through all the formalities which one faces during the doctoral studies. I would like to thank Jakob Riga for his Bachelor thesis which I co-supervised and which became a section in this dissertation.

I would like to thank my parents and my family for their support and patience all this time.

And of course I would like to thank the DAAD for sponsoring the research by a grant provided by Siemens/DAAD Postgraduate Programme (DAAD is Deutscher Akademischer Austausch Dienst - German Academic Exchange Service).

Hannover, 2016 Roman Vlasov

Contents

List of Figures

List of Tables

List of Abbreviations

2D – Two-Dimensional

3D – Three-Dimensional

6D – Six-Dimensional

AABB – Axis Aligned Bounding Box

API – Application Program Interface

BD-Tree – Bounded Deformation Tree

CPU – Central Processing Unit

CT – Computed Tomography

DAAD – Deutscher Akademischer Austauschdienst (German Academic Exchange Service)

DoF – Degree-of-Freedom

DVR – Direct Volume Rendering

EU – European Union

FEM – Finite Element Method

FFC – Face-Centered Cubic

GPU – Graphics Processing Unit

GUI – Graphical User Interface

HIP – Haptic Interaction Point

HU – Hounsfield unit

HWS – Haptic Workspace

IP – Interaction Point

IST – Inner Sphere Trees

JNA – Java Native Access

LMD – Local Minimum Distance

MIP – Maximum Intensity Projection

MMCA – Modified Marching Cubes Algorithm

MRI – Magnetic Resonance Imaging

MRT – Magnetic Resonance Tomography

MWS – Workspace of Movement

PAFF – Point-Associated Finite Field

PC – Personal Computer

QSA – Quasi-Static Approximation

RAM – Random Access Memory

SNCH – Spatialized Normal Cone Hierarchies

SPH – Smoothed Particle Hydrodynamics

SWS – Scene Workspace

TCP/IP – Transmission Control Protocol/Internet Protocol

VM – Virtual Machine

VR – Virtual Reality

VWS – View Workspace

YADiV – Yet Another Dicom Viewer

YDMF – YaDiV Deformable Model Framework

Chapter 1

Introduction

With the evolution of medical scanning devices, especially Computed Tomography (CT) and Magnetic Resonance Imaging (MRI), 3D volume data is nowadays widely used in modern medicine. These modalities have become an integral part of a clinical practice. Resulting 3D images are used for diagnosis, therapy planning, interventional guidance, and follow-up. 3D volume data is also in use in geology, CAD-applications, entertainment and other areas.

In order to significantly increase usability and efficiency of work with 3D data, an additional dimension could be added to a virtual system - a sense of touch. This could be done using a haptic device. With a haptic device a user can both manipulate a virtual object and feel force feedback reactions. Source data could be in different representations (triangulated surface, hexahedrons, volumetric, ...), but we focus on a volumetric one, since it is a direct output from the scanning devices. Other data types can be transformed to this one, if necessary.

Our goal was to design and develop a VR system, which integrates all stages of (medical) volume data processing and provides a user with a haptic interface and high-quality visualization. Stages of the volume data processing are presented in detail in section 2.2.1.

The general challenges of haptic rendering are a huge amount of volumetric data per object, stability and that it requires and update rate of at least 1 kHz. There exist many haptic rendering methods, but almost all of them have drawbacks that (1) “thin” obstacles could be skipped or an interaction point could go inside them and/or (2) a certain topological structure of objects is needed, such as connectivity or number of holes. The last is also an important issue, since the real medical data we work with can have any

structure, especially if segmentation has been done automatically. The aforementioned drawbacks are not acceptable for precise procedures such as pre-operation planning in surgery.

Here we present a novel approach published in [227, 225] that does not have these problems while guaranteeing nearly constant time complexity independent of data resolution by employing ray casting for the collision detection and a "sliding along a surface" model for the collision response. Additionally, no precalculations or explicit surface representations are needed. This means that a virtual scene may be both dynamic and static and that objects can be dynamically changed. To our best knowledge, the use of ray casting in haptic rendering is a novel interdisciplinary approach being on the cutting edge of visualization and haptic rendering research areas. Our method was implemented and tested within our VR system based on the framework provided by the YaDiV platform [73, 72] – a powerful virtual system for working with 3D volume data, which was developed at our Institute. This allows us to combine novel haptic rendering methods for exploration of medical data with high-quality visualization. Our approach has nearly constant time complexity independent of data resolution and is very fast – up to 750 points can be simulated at haptic update rates (1 kHz) for the collision detection only and up to 150 points for the collision detection and collision response (both values are given for a moderate end-user PC). This allows to perform object-object collision detection at a sufficient speed. Further on, we present our improved haptic rendering approach published in [226], which employs local path finding for collision response and employs an improved force feedback generation scheme. We show that the path finding paradigm can be successfully employed in other research areas, such as haptic rendering in our case.

For the advanced contact resolution, we focus on a flexible framework which allows us to use our above mentioned improved approach of haptic rendering of volumetric data together with deformation models. We show that it is feasible to do so, since our haptic rendering approach adds its properties including collision detection guarantee and non-penetration guarantee to the selected deformation model. Furthermore, we present our graphics approach which we use to keep the graphics representation of objects up-to-date during the deformation simulation. The challenge here is to reflect deformations of objects interactively.

In order to validate our framework, we propose our local deformation simulation approach based on the method of potential fields, where potential fields can be considered as specific finite elements, i.e. discrete carriers of properties of the medium [99]. Further on, we introduce our novel cuboid potential fields (see remarks in section 5.18) and pro-

pose how to use them for the local deformation simulation. We demonstrate that cuboid potential fields are better suited to haptic rendering of volumetric data. Furthermore, we show how to establish the correspondence of parameters of our proposed deformation simulation models to parameters of real materials, and propose a way to take the heterogeneity of the simulated material into account. Additionally, we introduce the prototype of the global potential fields based deformation approach. The potential field based deformation simulation approaches are a good "illustration", because they initially do not have the "nice" properties of our haptic rendering approach. Additionally, the resulting combined haptic rendering approach with our proposed deformation simulation approaches within our deformation framework does not require any pre-calculated structure and works "on the fly". The haptic update rate of our deformation framework remains stable when a deformation simulation is added. It does not decrease for both local and global simulation approaches. Furthermore, the haptic update rate is still orders of magnitude higher than the required 1 kHz.

The deformation framework, as well as all our haptic rendering and deformation simulation approaches, was fully developed by us from scratch, without the use of any third party libraries.

The thesis is structured as follows. In chapter 2 we give basics, definitions and general overview of haptic rendering and of visualization of volumetric data. In chapter 3 we give an extensive overview and classification of existing visualization and haptic rendering methods and their advantages and disadvantages. In chapter 4 we present our haptic rendering approach for volumetric data being published in our works [227, 225], and its improvements being published in our work [226]. We also discuss implementation details, and give the results of tests with real volumetric data. In chapter 5 we propose our flexible deformation framework which allows us to use our improved approach of haptic rendering of volumetric data presented in chapter 4 together with deformation models. Furthermore, we present our graphics approach which we use to keep the graphics representation of segments up-to-date during the deformation simulation. Furthermore, we introduce our novel cuboid potential fields and our potential fields based deformation simulation approaches. Further on, we give the results of tests with real volumetric data. In chapter 6 we present the summary and future outlook.

Chapter 2

Basics and Definitions

In this chapter we give definitions and general overview of visualization and haptic rendering. It is divided into two main sections, respectively.

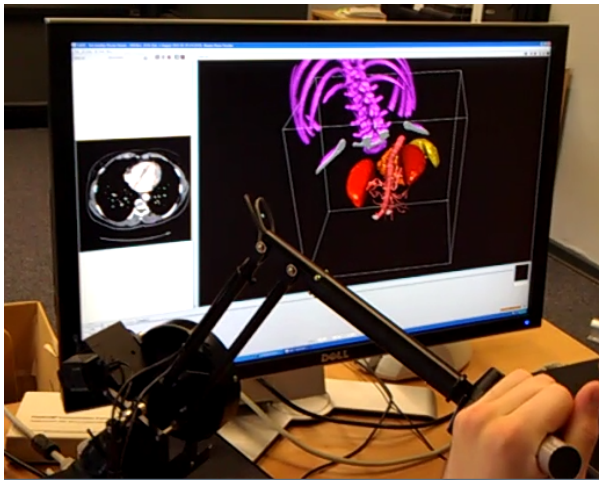


Figure 2.1: A user is manipulating the purple object using the Phantom haptic device and feeling force feedback reactions when collisions occur (source: our work [225])

2.1 Haptic Interaction

Haptic devices add a new dimension to simulation frameworks: feeling the objects. With a haptic device a user can both manipulate a virtual object and feel force feedback reactions.

Haptic devices are useful in a medical simulation and training, museum display, painting, sculpting, CAD, visualization, military applications, assistive technology for blind and visually impaired people, interaction technologies with scientific data [187], entertainment and other applications. For more details see e.g. [140, 97]. Additionally, user studies were performed showing that a training with haptic devices gives better results than a training without them [156, 205, 202].

Note [177]: The first haptic device, "an ultimate display with force feedback", was suggested by Ivan Sutherland in 1965.

2.1.1 Definitions

Definition [82, 174, 177]:
The term **haptic** (from the Greek *haptesthai*, meaning "to touch") is an adjective used to describe something relating to or based on the sense of touch. Haptic is to touching as visual is to seeing and as auditory is to hearing.

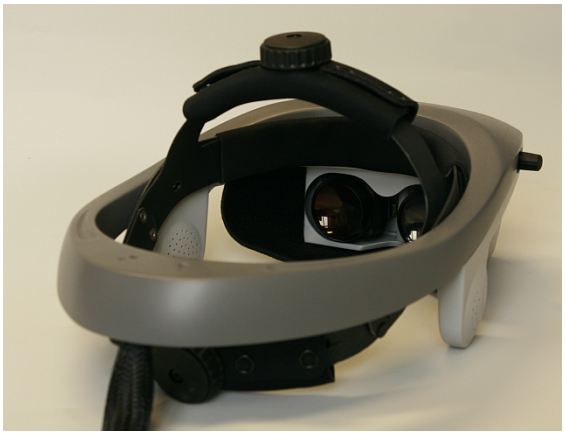


Figure 2.2: Head mounted display

Definition [82, 174, 177, 203]:
Haptic rendering is a process of computing and generating forces and torques in response to user interactions with virtual objects.
(See an example in figure 2.1.)

2.1.2 Types of Input/Output Devices

Definition:
Tracking devices – devices which track the position and/or orientation of objects in 3D space, i.e. track a specified number of degrees-of-freedom (DoFs).

Below we list commonly used tracking devices/types:

- A **Computer mouse** – probably, the most well-known and widespread tracking device. It tracks two DoFs
- A **head tracking device (or head mounted display)** – a helmet with small displays in front of eyes and sensors tracking the position and orientation of the head.
- **Full hand tracking devices** – usually a glove with sensors (e.g. *Wireless CyberGloveII* – see figure 2.3) tracking orientation of the user's hand including fingers. This allows to perform user interactions in a more natural way (e.g. in navigation tasks, see [82]). Position of the hand is also tracked by some models

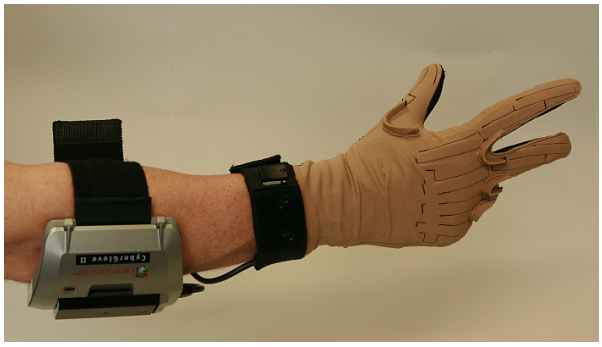


Figure 2.3: Sensor gloves

Figure 2.4: Limb tracking device (*Polhemus FastTrak*) (source: [100])

- **Full body tracking** – this is the most complicated tracking task and a subject of much ongoing research. Achieving highly accurate data in real-time is still an unsolved problem [82]. Example of a limb tracking device is shown in figure 2.4
- **Set of sensors**, which could be fixed on arbitrary objects at arbitrary places. Examples of such sets are optic sensors with a camera tracking system (like *IO-tracker/4*) and inertia sensors (like *InertiaCube3 Wireless*)
- **A haptic display** is also a tracking device. See the next section for details.

2.1.3 Haptic displays

Definition:
A **kinesthetic display** – a device which tracks its own position and/or orientation and stimulates the kinesthetic sense of the user via a programmable force feedback. It is allowed that the device has 0 DoFs, i.e. it doesn't track any DoFs.

Figure 2.5: Joystick with force feedback

Remarks:
The *kinesthetic sense* is e.g. a sense of rough surface features. In other words, it is a sense that tracks the positions of the limbs.

Definition:
A **haptic display (a haptic device)** – we define it as another name for the kinesthetic display.

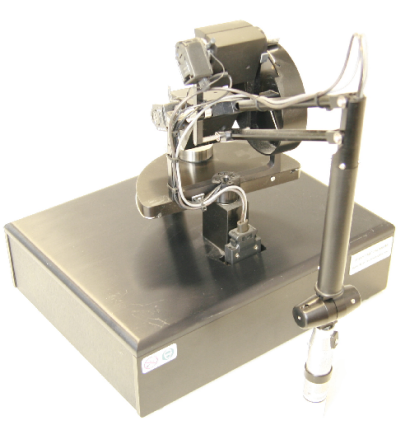
Remarks:

- Some authors (e.g. [72]) assume that haptic displays are not only kinesthetic displays but also cutaneous displays (see section 2.1.4 for definition of the last term)
- It is assumed in some works that tracking devices are also haptic displays.

Further we list common and important types of haptic displays:

- **Game manipulators with force feedback** – manipulators like joystick or wheel are actually haptic displays because they track the position and can give a programmable force feedback
- **3-DoFs haptic displays**
- **6-DoFs haptic displays** – 6-DoFs devices become more and more popular now

Figure 2.6: Example of a 3-DoFs haptic display: *Novint Falcon*



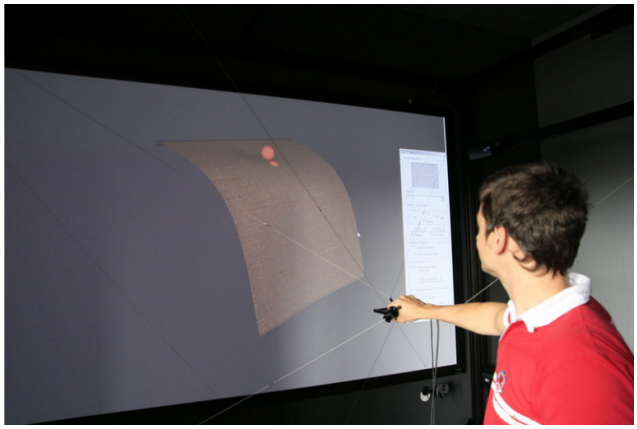


Figure 2.7: Examples of 6-DoFs haptic displays: *Phantom Premium 6DOF* and *INCA 6D*

Figure 2.8: Example of an n-DoFs device (source: [14])

- **n-DoFs haptic displays** – such devices still can be rarely seen, since they are usually produced for simulation of very specific tasks. Examples of general-purpose n-DoFs devices are a force feedback glove (figure 2.8) and a force feedback chair.

2.1.4 Cutaneous displays

Definition:
A **cutaneous display (tactile display)** is a device which can track its own position and/or orientation and stimulate the cutaneous perception of the user.

Such devices are used when it is necessary to give a perception of texture and roughness, e.g. a perception of fabrics [9, 10, 8].

Devices of this group could be based on different principles: electromagnetic displays, pneumatic displays, displays with electroactive polymers, air jet displays and others. An interested reader can find out more in the work of Allerkamp [8] devoted to the cutaneous perception.

2.1.5 Passive Haptics

Definition [100]:
Passive haptics is augmenting a high-fidelity visual virtual environment with low fidelity physical objects.

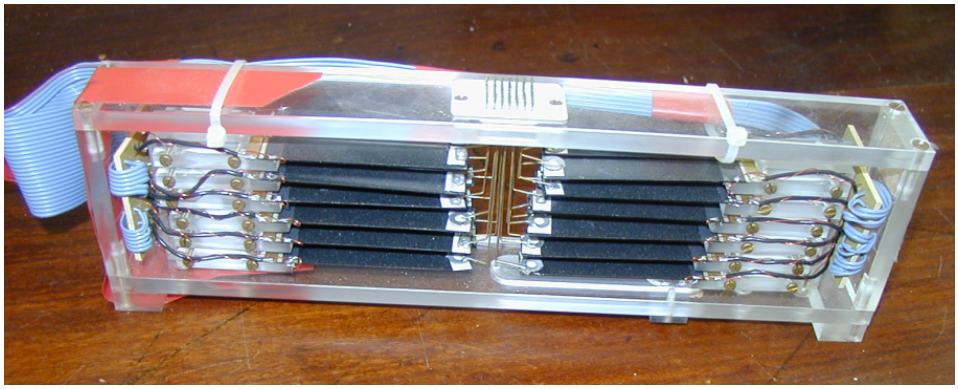


Figure 2.9: One of tactile displays used in [9] (source: [9])

Figure 2.10: Visual virtual kitchen (*left*) and passive haptic kitchen (*right*) (source: [100])

The PhD thesis of Insko [100] is devoted to this topic. The author considered situations when a user is immersed into a virtual emvironment with a head mounted display and can walk in there by walking in the real world. According to Insko, the most disturbing unnatural property of virtual environments is the ability of users to pass through visual obstacles. In order to eliminate this drawback the author proposed to add low-fidelity physical objects like styrofoam blocks and particle-board countertops to the real-world according to high-fidelity obstacles in the virtual environment (see figure 2.10).

2.1.6 Synchronization of Different Devices

Different types of input/output devices could be used together in one system, but synchronization problems could arise in this case. See e.g. the recent work of Hwang et al.

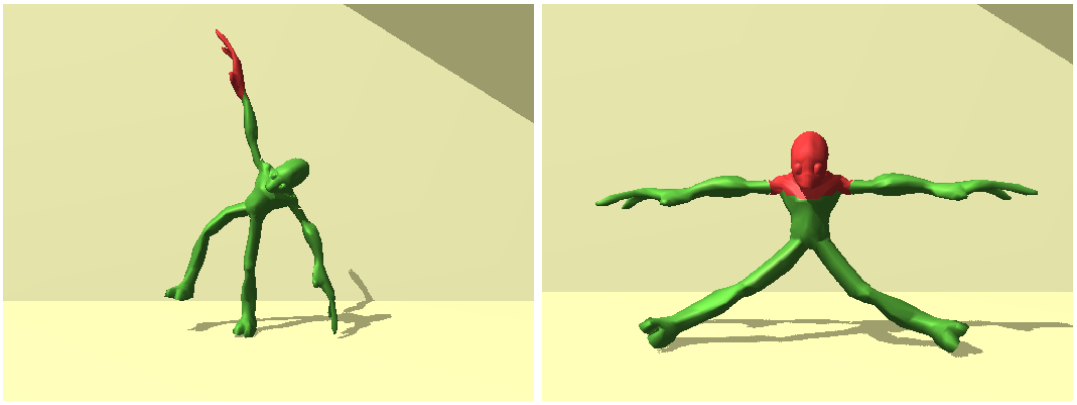


Figure 2.11: Difference between the tool and the handle: the tool is the whole alien, the handle is the red part of it. (source: [77])

[96] devoted to this issue in case of simultaneous use of a haptic display and a full-hand tracking device.

2.1.7 Additional Definitions

Definition:
A **probe (or end-effector) (of a haptic display)** is the part of the device the position/orientation is tracked for (passive /DoFs) and a force feedback is applied to (active DoFs).

Definition [**77**]:
A **tool (in a virtual world)** is an object in the virtual world, which the user manipulates via the probe. Further we will use the term **tool** if it is understandable by context what is meant. A particular case of the tool is the **(haptic) interaction point** (if the object is a 3D point).

Definition [**77**]:
A **handle (in a virtual world)** is a grasped part of the tool.

Remarks:
The difference between the tool and the handle is shown in figure 2.11.

2.1.8 Degrees-of-Freedom (DoFs)

One of the characteristics of any haptic display and therefore a haptic rendering algorithm is the number of DoFs. Generally, information about any particular DoF of the device can be processed in an arbitrary way. But as far as haptic displays were created to make working with a virtual reality (VR) environment more intuitive and convenient, manipulations with a device's probe usually correspond to those with an object (e.g. movements to movements, rotations to rotations). Further we will assume such a correspondence by default.

The following DoFs are commonly used in haptic rendering:

- **3-DoFs in 3D-space** – processing of translations along axes and synthesis of linear force feedback OR processing of rotations and synthesis of angular force feedback
- **6-DoFs in 3D-space** – processing of translations and rotations and synthesis of linear and angular force feedback. 6-DoFs haptic rendering became quite common and widespread nowadays
- **n-DoFs** – haptic rendering for devices, which have any other number of DoFs. The simpliest example is a game joystick (2-DoFs in 2D-space). A more interesting case is haptic rendering for $n > 6$, since this is a field of much ongoing research, because there exist no general methods and widespread devices.

Remarks: Further we consider devices with at least 1-DoF if not stated otherwise.

2.1.9 Haptic Rendering Pipeline

There exist different variations of the haptic rendering pipeline, but generally it looks as shown in figure 2.12. In this section we give an overview of each step. We consider 6-DoFs, but the pipeline could be generalized to n-DoFs.

A haptic rendering application should solve three main tasks: contact determination (also called collision detection), collision response and generation of force feedback. All stages are often tightly integrated in order to effectively use a solution of one task for solving others. In the sections below we consider them in more detail.

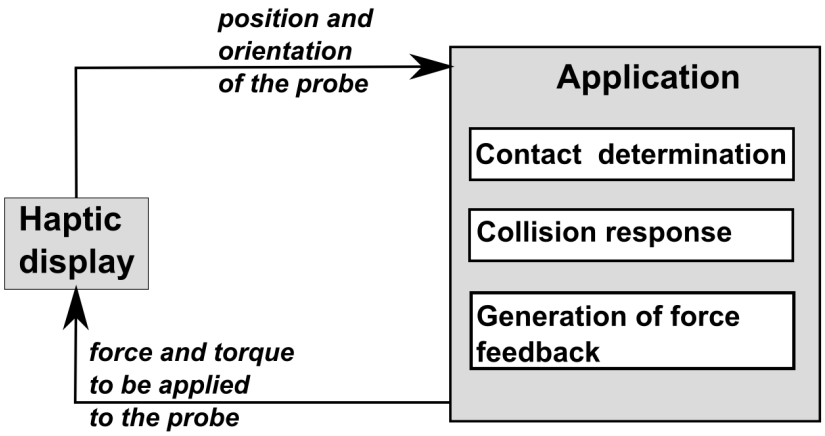


Figure 2.12: General haptic rendering pipeline (source: our work [227])

Contact Determination

An application should make contact determination between the tool and other objects in a virtual world according to the configuration of those and position/orientation of the probe. The application should not only detect colliding objects, but also find points/areas of the contact. Depending on how much the virtual environment is changing during the simulation process (e.g. whether objects can move or not, are they deformable or not) different methods should be applied. We consider them in more detail for every paper in the section 3.2, which is devoted to the detailed overview of haptic rendering approaches.

Various hierarchy structures could be used for contact determination, like those described in [82] (sensation preserving contact levels-of-detail), [84, 206] (OBB Trees), [106] (spatialized normal cone hierarchies), [188] (continuous collision detection), [198] (hierarchy of bounding spheres), [102] (bounded deformation tree – BD-Tree), [235] (inner sphere tree), [15, 16, 18] (point-based BD-Tree, nested point tree hierarchy).

Additionally, a review of publicly available collision detection *systems* can be found in [206].

Collision Response

Using information from the collision determination step, the application should make an appropriate collision response (i.e. physical simulation) between interacting objects (including the tool) in the virtual world. Different authors proposed different solutions to this task (see e.g. an overview [176]), and the solutions could be classified as follows:

- *constraint-based* (if a collision is found then stop the simulation and formulate a constraint problem in order to find collision forces, accelerations, velocities and positions)
- *impulse-based*
- *vector fields, including potential fields*
- *meshless method of finite spheres*
- *penalty-based* (apply collision forces based on the amount of objects' interpenetration)
  - local-penetration methods
  - pre-contact penalty forces.

The methods are discussed together with corresponding haptic rendering approaches in section 3.2.

Generation of Force Feedback

The application should generate a force feedback in order to give the user a feeling of the virtual world. Feedback forces and torques are generated according to the collision response and other forces in the system. "Other forces (and torques)" could be e.g. gravity or magnetic forces. For training purposes, there could also be forces (and torques) which e.g. track the user along a predefined way or let him/her do pre-recorded actions (for example, in surgical simulation). Additionally, force feedback may allow a user to feel different fields and to feel streamlines of vector data – see [98].

There are important stability issues concerning the force feedback generation. They are discussed in sections 2.1.13, 2.1.14.

2.1.10 Controlling a Haptic Display

According to [82], there are two ways of controlling a haptic display:

1. **admittance control** – a user applies a force to the device, and the application moves the probe according to the simulation

2. **impedance control** – a user moves the probe of the device, and the application produces forces. This scheme was firstly suggested in 1985 by Hogan [94] for contact tasks in manipulation in robotics.

The drawback of the admittance control scheme is that instabilities may arise

- during a free-space motion in the virtual world, because the probe must move at high accelerations under small applied forces
- when the probe rests on a stiff physical surface.

But this control scheme is stable for rendering of stiff virtual surfaces.

Conversely to the admittance control scheme, the drawback of the impedance control is that instabilities may arise in the simulation of stiff (rigid) virtual surfaces, because the device must react with large changes in force to small changes in position. Although a free-space motion is quite stable.

According to [82] and [177], the impedance control scheme is cheaper and easier to construct and is usually used nowadays.

Remarks:
Further we consider the impedance control scheme if not stated otherwise.

2.1.11 Passivity

Definition [82]:
A subsystem is passive if it does not add energy to the global system.

Remarks [82]:
A composite system obtained from two passive subsystems is always stable.

Colgate et al. [50] analyzed passivity (stability) conditions for 1-DoF haptic rendering of a virtual wall modeled as a viscoelastic (the virtual spring and damper) unilateral constraint and found the necessary and sufficient condition for the passivity:

$$b > \frac{KT}{2} + B,\ B \geq 0 \tag{2.1}$$

or if B is allowed to be negative then

$$b > \frac{KT}{2} + |B|, \tag{2.2}$$

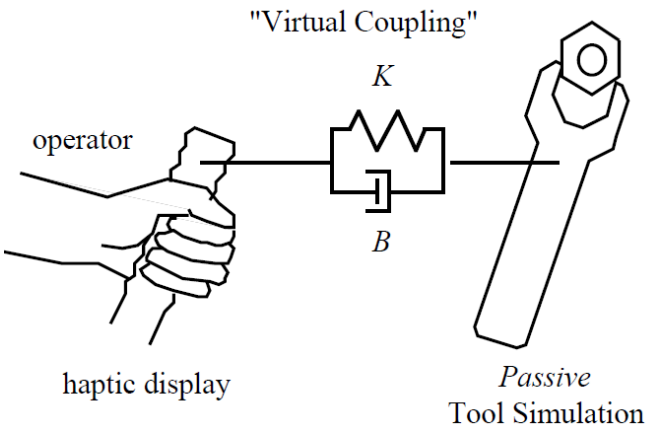


Figure 2.13: Virtual coupling (source: [51])

where:
K – the stiffness of the virtual wall;
B – the damping of the virtual wall;
T – the sampling period;
b – the inherent damping of the device.

These results give guidelines for a design of haptic interface: in order to implement very stiff constraints (high K, B) it is helpful to maximize b and minimize T.

2.1.12 Direct Rendering and Virtual Coupling

There exist two main techniques of handle manipulation:

- **direct rendering** – apply manipulations with the probe directly to the handle
- **virtual coupling** – connect the haptic probe to the handle through a virtual spring-damper connection – see figure 2.13. For 6-DoFs there are usually two such connections: one for translations and one for rotations. This technique was firstly proposed by Colgate et al. [51]. Additionally, the authors showed that haptic rendering will be still passive in this case (see section 2.1.11).

Direct rendering is useful if a haptic rendering method can perform all the stages (contact determination, collision response and force feedback generation) at an update rate sufficient for a stable user interaction (1 kHz).

But what should one do if e.g. contact determination or collision response can only perform at much lower frequencies? The solution is to decouple the synthesis of interaction

forces from the simulation of the virtual environment, i.e. to provide force feedback at 1 kHz but make the physics computations, say, at 30 Hz. Such approaches are called **multirate approaches**, and virtual coupling is good in these cases [176]. However direct rendering could also be used if e.g. an intermediate representation for fast force feedback calculations is built at a sufficiently high update rate.

2.1.13 Stability and Force Feedback Update Rate

As stated in [82], one can measure the **quality of haptic rendering** in terms of a dynamic range of forces (impedances) that can be simulated in a stable manner, that is a force should be very low for movements in free space and high for contacts between a rigid tool and rigid objects. So, the probe should stop quickly if rigid contact between the tool and a virtual obstacle occurs. But because of the sample and latency phenomena, unstable behavior of the probe could arise in such cases. Such instability is felt by a user in a form of disturbing oscillations. It is even possible that the tool passes through an obstacle because of a fast movement, or it appears at different sides of it in successive frames. See section 2.1.14 for a detailed description of the problems.

In connection with these issues Colgate et al. showed in the work [50] devoted to passivity and stability analysis, that a key factor for achieving a high dynamic range of forces, while ensuring stable haptic rendering is a computation of feedback forces at a high update rate. According to Brooks et al. [37] it should be at least 0.5-1 kHz. If the update rate is lower then in addition to stability problems a user can also feel motion "jerks" of the haptic device because of the high fidelity of the human kinesthetic system. In [82] Glengloss et al. wrote that sensing bandwidth for kinesthetic feeling can be as high as 400 Hz, and 5 - 10 kHz for cutaneous perception.

2.1.14 Stability Problems

A force update rate of 1 kHz is generally not sufficient for stable haptic rendering, as shown in the example below.

Example showing the insufficiency of a force update rate of 1 kHz for stable haptic rendering:
Let us assume that "the basic concept" (term from [82]), i.e. a naive method, is used. For simplicity we consider 3-DoFs only and a tool as a point. For each haptic iteration, the stages of the naive haptic rendering approach are as follows:

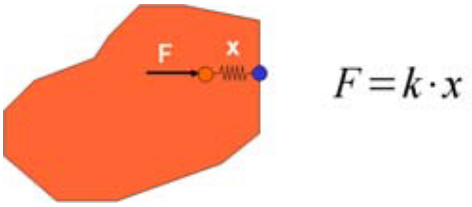


Figure 2.14: The naive haptic rendering algorithmn (source: [82])

1. Collision detection:

   if (the tool is inside some obstacle) **then**
   collision appeared
   else
   no collision
   end if

2. Collision response is penalty-based:

   if (collision appeared) **then**
   push the tool out to the closest surface of the obstacle
   end if

3. Force feedback generation is penetration-distance-based:

   if (collision appeared) **then**
   return a force, which is proportional to the distance at which the tool was pushed out
   else
   return 0
   end if

For the above naive method, the following stability problems could occur if the tool moves too fast ("too fast" depends on sizes of obstacles):

1. the tool passes through an obstacle, i.e. it appears at different sides of it in successive frames

2. unexpected force discontinuities when the tool crosses boundaries of internal Voronoi cells of the obstacle's surfaces, i.e. if the tool is inside one Voronoi cell at one frame and is inside other Voronoi cell at the next frame. See figures 2.15, 2.16.

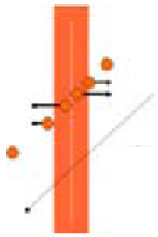


Figure 2.15: The force direction changes after crossing the middle line (source: [82])

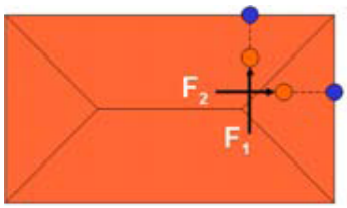


Figure 2.16: Unexpected force discountinuities in magnitude and direction (source: [82])

Even if the issues mentioned in the above example are solved, the following problems could still be presented:
If the tool has reached an obstacle and is located on its surface, but the user is still moving the probe against it, then the distance between the probe and the tool in the virtual world becomes larger and larger. If virtual coupling is used then the coupling force could drastically increase, and the same problems as with "too fast tool movement" will appear. Additionally, precision of all computations involving the coupling force will decrease because of the large force value, and this will lead to numerical problems. Another issue in this case is that an overflow of the coupling force value could happen.

Remarks:

1. For a 6-DoFs haptic rendering there are the same stability issues for linear movements and linear forces, and problems of the same nature for rotations and torques

2. Other issues may appear if the tool is not a point but an object (e.g. if the tool has some very big or very small parts compared to sizes of obstacles).

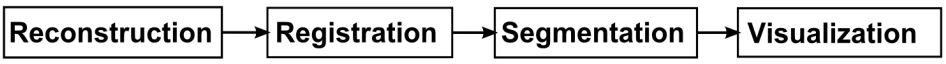
As the conclusion, "clever" collision detection, collision response and force feedback generation methods should be used in order to provide stable haptic rendering even at 1 kHz.

2.2 Visualization

2.2.1 Volumetric Data Processing Pipeline

Nowadays volumetric data processing and visualization, especially medical imaging, are widely used for analysis, diagnosis, illustration (Rößler et al. [195]) and other purposes such as neurosurgery planning and reconstruction of industrial CT (Computed Tomography).

Generally, four stages of a medical image processing and visualization pipeline are usually arranged (see Chen et al. [43] for details):



Further we give a brief explanation of each stage.

Reconstruction. This stage is also named "Geometry Processing or Construction" in [43]. It is the process to generate 3D volume data set from the data which lacks in geometrical, topological and semantical information. E.g. data which is acquired by the Computed Tomography procedure: the absorption along X-rays, which are sent through the observed object from the different positions. There are several different reconstruction algorithms for medical and industrial CT scanners.

Registration. A matching process of different reconstructed data sets, obtained from the same source, is called registration. Example: after the reconstruction of several different data sets of the same patient, which may be acquired under different conditions, the data sets usually do not match perfectly. For instance, this could happen because the patient had different positions in the different scanners or because of different parameters for the scanners (e.g. different distortions). Registration is needed in this case.

Segmentation. Segmentation is a process to extract certain structures from a volume data set. In medical context this can be anatomical organs, e.g. kidney, liver or bones, or pathological structures, like tumors. The direct volume rendering technique (see section 2.2.4 for details) provides some kind of implicit segmentation during rendering via a transfer function (the transfer function describes how the intensity values in the data set are mapped to colors and opacities; see definitions in section 2.2.2). E.g. in data sets obtained by CT it is easy to extract bone tissue with a transfer function. Unfortunately, there are several structures which can not be extracted by implicit segmentation. Explicit segmentation algorithms are used in this case, which apply to each voxel a tag

indicating if it belongs to a certain structure or not.

Visualization. Visualization is a representation of data in a native, intuitively clear and easily-understandable way. There are several different rendering techniques. For some techniques rendering time is a key criterion (real-time rendering), and for some other of them quality (realism) of the rendered image is a key one. Multi-volume rendering is also a very useful technique due to its ability to render different datasets at the same time, especially if rendering in real-time is performed [195, 108]. The stages above are usually realized in bounds of different projects, but a promising tendency of their incorporation in the bounds of one system could be currently seen, e.g. a system presented by Lundström in [133] and the YaDiV system of Friese et al. [73].

2.2.2 Data Representation

Definition:
We assume that **volumetric data** V is presented as a set of **volumetric elements (voxels)**
$\{\mathbf{x}, s, l\}$, where
$\mathbf{x}$ – coordinates of a voxel;
s – a scalar value associated with the voxel (*intensity value*, or *intensity*);
l (arbitrary type) – application specific data, such as data, that indicates whether this voxel corresponds to a certain segment or not.

In the field of medical visualization volume data is usually acquired with Computed Tomography (CT) or Magnetic Resonance Tomography (MRT). The result of such an acquisiting process is a data set consisting of pairs $< coordinates, intensity\ value >$, where the scalar value is a value measured by the scanning device (e.g. the value of unabsorbed X-rays) [43]. One can take a look for a description of volume data and related terms in [110].

From the geometrical point of view, volumetric data is a set of 3D points in 3D space. As far as it is usually acquired by medical scanning devices, the distances between the points along a coordinate axis are usually equal, i.e. they are positioned at nodes of some rectilinear grid. For easier imagination, it is common to think about the data as of "bricks" in a rectilinear grid – see figure 2.17 (but actually it is still a set of 3D points).

Remarks:
Further we assume that all voxels in a given data set are positioned in the nodes of some

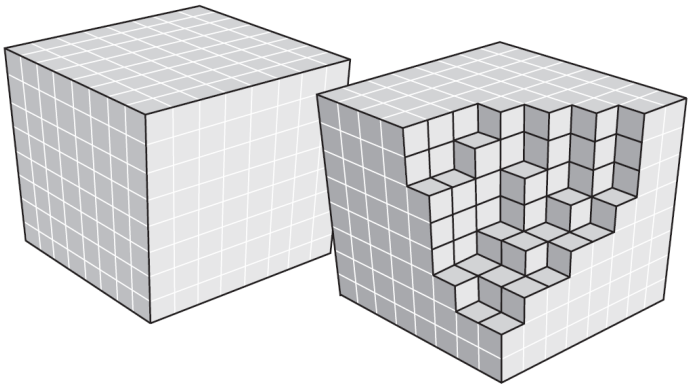


Figure 2.17: Volumetric data as "bricks" in a rectilinear grid (source: [89])

rectilinear grid, if not stated otherwise.

Since the scanned data has no color or tissue information, a segmentation step of the data could be further needed. That is, if explicit segmentation algorithms are used, a tag is applied to each voxel. This tag indicates if the voxel belongs to a certain structure or not and is denoted in our definition of volumetric data as l.

The segmentation process is a large field of research, and a lot of different approaches for different purposes have already been proposed (see e.g. [43] and [72] for an overview and suggested methods). As long as we assume that the scanned data is already segmented, segmentation is not in scope of our research.

Unformal definition:
A **transfer function** describes how $< coordinates, intensity\ value >$ pairs ($< \mathbf{x}, s >$ pairs in terms of our definitions) for voxels of a given data set are mapped to colors and opacities.

Remarks:
We need more than just the intensity as parameters of the transfer function because different segments of volumetric data could have different intensity-to-color-and-opacity mappings. For this we use the "coordinates" parameter in order to determine the segment we need to make the mapping for.

Definition:
A **transfer function** f for volumetric data V is the mapping

$$f : X \times S \mapsto C \times I, \tag{2.3}$$

where
$X \subset \mathbb{R}^3$ – a convex hull of the input coordinates of V;
$S \subset \mathbb{R}$ – a convex hull of the input intensities of V;
C – a set of colors;
I – a set of opacities.

Remarks:
We use a convex hull in the definitions of X and S, because a voxel can be sampled not only at one of the given discrete positions of V, but also between them. An interpolation is needed in such a case, leading to additional positions and intensities (see [138] for an evaluation of interpolation techniques). All such values are included into convex hulls of initial positions and intensities of V.

Below we present a classification of visualization techniques for volumetric data. A detailed overview of concrete approaches is given in section 3.1. One can also take a look at other overviews in [190, 89, 65, 88, 143].

In general, one could divide visualization techniques into polygonal rendering and volumetric rendering. Further we present each of them in more detail.

2.2.3 Surface Rendering

This type of rendering is well-known, quite widespread and fully supported by current graphics hardware due to the possibility to parallelize the rendering process. Data to be drawn is represented as polygons (usually, triangles). This type of rendering is not in focus of our work, and we would like to refer interested readers e.g. to a deep state-of-the-art overview by Akenine-Moller et al. [7].

Surface rendering is useful when an explicit surface representation for a volumetric object is given. But usually one just has a set of voxels, i.e. a set of points with additional parameters. Therefore if one wants to use surface rendering then polygons should be retrieved from the volumetric data. It could be easily done for iso-surfaces (an **iso-surface** is a level set of a continuous function whose domain is 3D-space; unformally, it could e.g. represent regions of a particular density in a 3D CT scan): polygonal iso-surfaces could be obtained e.g. by the Marching Cubes algorithm [129, 126] or one based on it [158]. But if one also wants e.g. to draw other voxels in semi-transparent

mode then problems will appear since almost all of them usually do not have any explicit surface representation [89, 88]. It would be very time and memory consuming to create such a polygonal representation.

In order to avoid the aforementioned problems, Direct Volume Rendering (DVR) techniques were proposed.

2.2.4 Direct Volume Rendering

The idea of DVR is to render volumetric data directly. Direct methods display voxel data by evaluating an optical model which describes how the volume emits, reflects, scatters, absorbs, refracts and occludes the light [139].

Optical Models

Almost all important optical models for DVR are described in a survey paper by Nelson Max [139], and we briefly summarize them here:

- **Absorption only**. The volume consists of particles, which only absorb the light and do not scatter or emit any. Partial absorption (attenuation) of the light is allowed
- **Emission only**. The volume consists of particles, which emit the light only
- **Absorption and emission**. The volume consists of particles which absorb and emit the light
  Remarks: According to [89, 88], the "Absorption and emission" model with a restriction that only a directional light parallel to the viewing direction is allowed, is nowadays the most common one in DVR
- **(Single) scattering and shading**. In addition to absorption and emission of particles this model includes scattering of illumination from light sources. *One light ray could be scattered by a particle only once*, any opaque and semi-transparent objects on the way of the ray to the particle are ignored. Additionally, normals of particles are needed for determination of direction of the scattered ray
- **Shadowing**. In addition to features of the previous model, opaque and semi-transparent objects between the light sources and the illuminated particles are

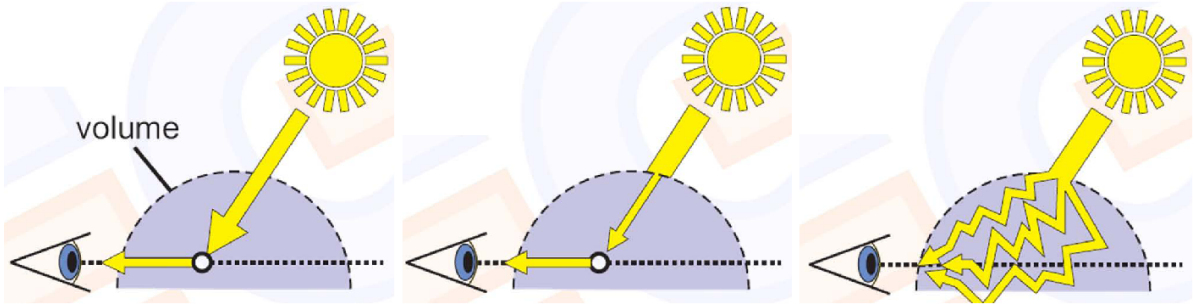


Figure 2.18: (*left*) (Single) scattering and shading, (*middle*) Shadowing and (*right*) Multiple scattering optical models (source: presentation for [65])

taken into account: a light ray from a light source could be stopped or attenuated on its way

- **Multiple scattering**. This model includes all features of the previous model and has a support for an incident light that has already been scattered by multiple particles before it is scattered toward the eye. One light ray could be scattered sequentially by several particles. One ray could also be reflected as several rays with smaller intensities. One ray could be partially reflected and partially go through a particle if the particle is semi-transparent and has a non-zero albedo
- **Multiple scattering and refraction**. This model is not described by Nelson Max [139]. In addition to features of the previous model refraction of light rays is allowed.

Since the "Absorption and emission" optical model with allowance of only a directional light parallel to the viewing direction is the most common one, we will take a look at a rendering integral for it. We will describe it in the way as it was done in [38] and [143], but adapt it to the selected optical model. The rendering integral $I_\lambda(\mathbf{x}, r)$, i.e. the amount of the light of wavelength λ coming from a ray direction r that is received at location $\mathbf{x}$ on the image plane, is:

$$I_\lambda(\mathbf{x}, r) = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt} ds, \tag{2.4}$$

where
L – the length of the ray r;
μ – absorption (extinction) coefficient at the specified position on the ray r;
C_λ – amount of the light of wavelength λ emitted at the specified position on the ray r.

Remarks: Further we assume that μ and C_λ have two different types of parameters:

- $\mu(s)$ and $C_\lambda(s)$, where $s \in \mathbb{R}$ is a position on the specified (by a context) ray (i.e the ray is parameterized and s is the parameter)
- $\mu(\mathbf{v})$ and $C_\lambda(\mathbf{v})$, where $\mathbf{v} \in \mathbb{R}^3$ is a position in 3D-space.

If and only if for s on the specified ray its actual position in 3D-space is equal to $\mathbf{v}$ **then**
$\mu(s) = \mu(\mathbf{v})$ and $C_\lambda(s) = C_\lambda(\mathbf{v})$.

Note: One can assume such function definitions as definitions of overloaded functions in the C++ programming language (functions which have the same name but different input parameters).

Most of practical volume rendering algorithms discretize the above integral into series of sequential intervals i of width Δs:

$$I_\lambda(\mathbf{x}, r) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\mu(s_i)\Delta s \cdot \prod_{j=0}^{i-1} e^{-\mu(s_j)\Delta s}. \tag{2.5}$$

According to [139], if $\mu(s_j)$ is a constant value inside a voxel with side Δs at the ray position s_j, then the opacity α of that voxel is $\alpha = 1 - e^{-\mu(s_j)\Delta s}$.

Taking into account the above representation of opacity, using the Taylor series approximation of the exponential term and dropping all but the first two terms, we get the equation

$$I_\lambda(\mathbf{x}, r) = \sum_{i=0}^{L/\Delta s} C_\lambda(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1}(1 - \alpha(s_j)). \tag{2.6}$$

C_λ and μ (and therefore α) could be approximated in different ways depending on the concrete rendering pipeline and method.

Remarks:
In the above formulas the starting position of the ray (0) is at the border of the volume, which is the most distant from the "eye" and the end position of the ray (L) is at the "eye" (i.e. at $\mathbf{x}$ on the image plane). Using the formulas one can implement either a back-to-front or a front-to-back compositing algorithm for computation of the rendering integral.

In case a transfer function f is used to find C_λ and μ, formulas for them look as follows:

$$C_\lambda(s_i) = f_{C_\lambda}\left(p(s_i), s(p(s_i))\right), \tag{2.7}$$

where
s_i – a position on the ray;
f_{C_λ} – a "part" of the transfer function f returning a value of C_λ for the given parameters;
p – a function which returns a position in 3D space for the given position on the ray;
s – an intensity for the given 3D position in the volume V.
For μ the formula is similar.

Generally, two ways of applying a transfer function in order to find C_λ and μ at an arbitrary position in the volume (i.e. not only at discrete positions of source voxels) are distinguished: pre-classification and post-classification [66].

- **Pre-classification** – the following computation order is used: (1) Use the transfer function in order to determine necessary colors and opacities of the source voxels for the next step → (2) interpolation of the colors and opacities in order to find a color and opacity at the given ray position.
- **Post-classification** – the following computation order is used: (1) Interpolate intensities of the source voxels in order to find the necessary intensity at the given ray position → (2) use the transfer function in order to determine the color and opacity.

I.e. the difference is in the time of application of the transfer function: before or after the interpolation. These classification approaches are analogous to Gouraud shading – interpolating a shaded color (i.e. colors and opacities for pre-classification), and Phong shading – interpolating a normal (i.e. intensities for the post-classification).

Post-classification gives much better visual results ([66, 38, 139]).

Additionally, in case where shading is supported (i.e. "(Single) scattering and shading" or more sophisticated optical model is in use), pre-shading and post-shading techniques are distinguished:

- **pre-shading** – the illumination model is evaluated at the source voxels
- **post-shading** – the illumination model is evaluated for the interpolated data.

Post-shading gives better visual results ([66, 38]).

The shading could also be applied before or after the classification, which gives four different combinations of these techniques shown in figure 2.19 , where "reconstruction" represents an interpolation (of colors and opacities or intensities depending on the case). In the case when the shading is performed before the classification, intensities are used

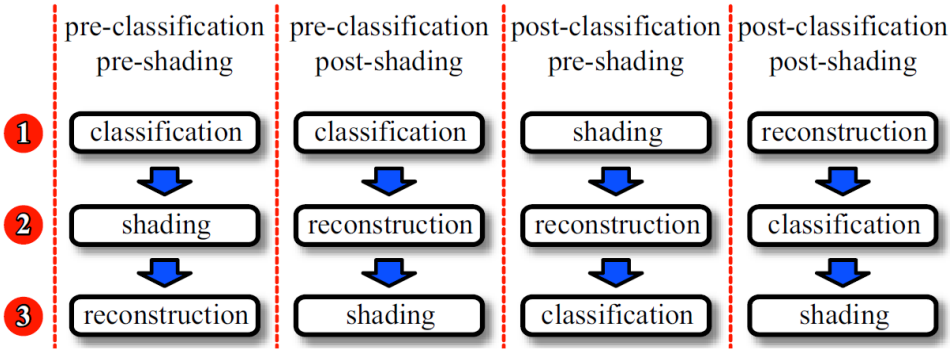


Figure 2.19: Combinations of the classification and shading techniques (source: modified from [38])

for shading computations. According to [66, 38], "post-classification and post-shading" combination gives the best visual results.

Chapter 3

Literature Overview

In this chapter we give an extensive overview and classification of existing visualization and haptic rendering methods and their advantages and disadvantages. The chapter is divided in two sections – visualization (DVR) and haptic rendering respectively.

3.1 Visualization by Direct Volume Rendering

Below an overview of commonly used DVR algorithms is given.

The authors of [143] distinguish two types of methods: fast but low quality methods *and* slow but high quality methods. We will also use this classification and group the methods as follows:

- *fast but low quality* – Rendering with 2D Textures, 2D Multi-Textures Rendering, Shear-Warp Algorithm, Rendering with 3D Textures
- *slow but high quality* – Splatting, Ray Casting, Ray Tracing.

Nowadays some of generally slow but high quality methods perform at interactive frame rates thanks to a new hardware, and we will mark this out in detailed descriptions of the methods.

The methods are described in order of increasing visual quality of the final image.

3.1.1 Rendering with 2D Textures

There are several works fully devoted to this technique and its enhancements, e.g. [66, 190, 67, 45, 242, 191]. The idea is to create three axis-aligned stacks of 2D textures of initial volume data at initialization, and then render each axis-aligned stack as a set of flat textured polygons ("slices") in back-to-front order. An alternative rendering strategy is to render only the stack, which is the most perpendicular to the viewing ray. When the slices are drawn, a transfer function for mapping of $< coordinates, intensity value >$ pairs to colors and opacities is used.

The optical model here is "Absorption and emission" with a restriction that only a directional light parallel to the viewing direction is allowed (see section 2.2.4).

"+":

- Simplicity
- Fast rendering speed (on both CPU and GPU) compared to high-quality methods mentioned earlier at the beginning of section 3.1
- Bilinear interpolation for 2D textures if graphics hardware is used [190]

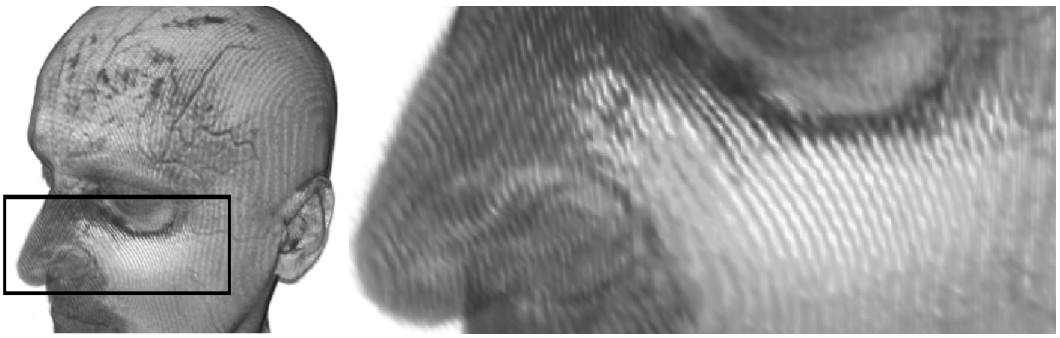


Figure 3.1: Aliasing artifacts become visible at edges of slice polygons (source: [190])

- Interactive on standard graphics hardware.

"-":

- Three times more memory is needed than required just to store volume data
- Sampling rate depends on the viewing axis
- Aliasing artifacts become visible at edges of slices because of a low sampling rate, i.e. because of fixed number of slices (see figure 3.1). Algorithm should be significantly changed in order to remove these artifacts [190]

  Remarks: This disadvantage has been somehow neglected by a method proposed in [67]: add polygons perpendicular to the viewing axis and connect borders of neighbouring slices
- If graphics hardware is used and all textures are stored in the video memory, then it takes a long time to update the volume data
- Low visual quality of the final image compared to the high-quality methods [143]
- Fixed number of slices causes visual artifacts ("jerks") for "fly-through" applications, i.e. when the camera goes through the volume
- Transparency artifacts when the slices, which are impossible to sort by distance to the view point because of their overlaps, are rendered. This could happen e.g. if the angles between the viewing direction and all two axes are 45 degrees.

In order to make the sampling rate independent of the viewing axis, to make it adjustable (unfixed) and to get triliniar interpolation, the authors of [191] proposed **the 2D Multi-**

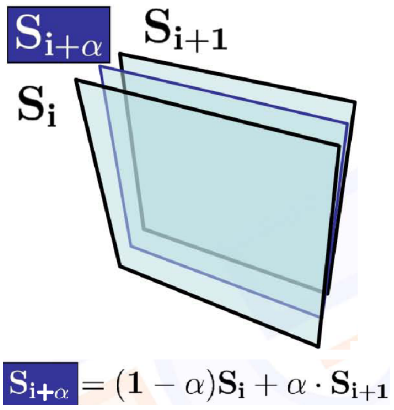


Figure 3.2: Fractional positions of slices (source: presentation for [65])

Texture Rendering method. They suggested to make the following changes to the source method:

- Axis aligned polygon slices now could not only have a fixed position but an arbitrary one on the axis. This means, that it is allowable to specify fractional slice positions, where integers correspond to slices existing in the source slice stack, and the fractional part determines the position between two adjacent slices. The number of rendered slices is now independent of the number of slices contained in the volume and can be adjusted arbitrarily
- For each polygonal slice its texture is computed via a texture blending between two textures corresponding to two neighbouring original slices from the source slice stack. The blending is performed with weights proportional to the distances between the given slice and the original slices along the main stack's axis (see figure 3.2).

The performance of the method decreases if additional stacks are added.

3.1.2 Shear-Warp Algorithm

There are several works fully devoted to this technique and its enhancements, including [25, 178, 122, 123]. The method was proposed by the authors of [122, 123], and its implementation on GPU was presented in [25]. In the shear-warp algorithm the volume is projected onto the image plane slice by slice. The idea in case of parallel projection is shown in figure 3.3. The projection takes a place not directly on the final image plane, but on an intermediate plane named a *base plane* ([65]), which is aligned not with the

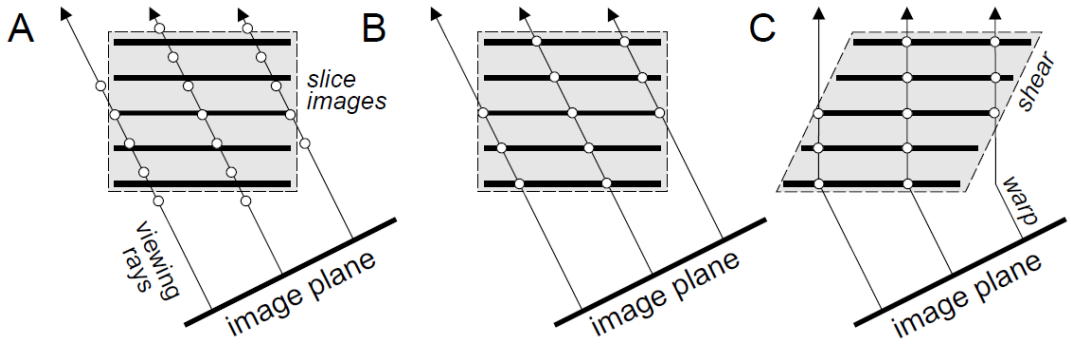


Figure 3.3: Principles of the shear-warp-algorithm for the parallel projection (source: [190])

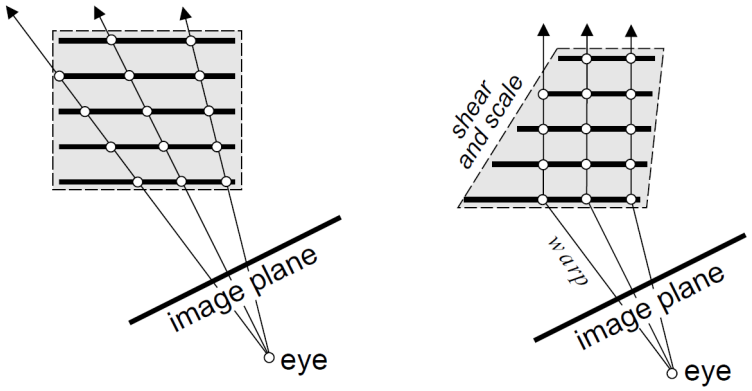


Figure 3.4: Principles of the shear-warp algorithm for the perspective projection (source: [190])

viewport but with the volume. The volume itself is sheared in order to turn the direction of oblique projection into the direction that is perpendicular to the base plane. This allows to make a fast implementation of the projection procedure: the entire slice can be projected by simple two-dimensional image resampling. After the projection of all the slices to the base plane has been finished, the base plane image is warped to the final image plane. A perspective projection can be accommodated similarly by scaling the volume slices in addition to shearing (see figure 3.4). When the slices are drawn, a transfer function for mapping of $< coordinates, intensity\ value >$ pairs to colors and opacities is used.

There are three axis-aligned slice stacks, and during rendering the slice stack whose axis is mostly parallel to the viewing direction is used in order to avoid a situation when viewing rays may pass between two slices without intersecting one of them [190].

The optical model here is “Absorption and emission” with a restriction that only a di-

rectional light parallel to the viewing direction is allowed (see section 2.2.4).

"+":

- Fast rendering speed (on both CPU and GPU; according to [65] this is the fastest software volume rendering method) compared to the high-quality methods mentioned earlier at the beginning of section 3.1
- Bilinear interpolation for 2D textures if graphics hardware is used [190]
- Sampling rate is constant
- Interactive on standard graphics hardware [25].

"-":

- "Switching" effect because of change of a slice stack when the "main" axis (the axis being mostly parallel to the viewing direction) is being changed [190]
- Three times more memory is needed than required just to store the volumetric data
- The constant sampling rate causes visual artifacts ("jerks") for "fly-through" applications, because it is not sufficient in such cases
- Low visual quality of the final image compared to the high-quality methods [143]
- If graphics hardware is used and all textures are stored in video memory, then it takes a long time to update the volumetric data.

3.1.3 Rendering with 3D Textures

There are several works fully devoted to this technique and its enhancements and applications [243, 79, 66, 190, 132, 249, 58, 127, 195]). It was first proposed in [243]. In this method the volume data is stored as a 3D texture. Since graphics hardware can only draw polygons, every time when we want to render the volumetric data we create viewport-aligned flat polygon slices of the volume (see figure 3.5) and draw them in back-to-front order. When the slices are drawn, a transfer function for mapping of $< coordinates, intensity\ value >$ pairs to colors and opacities is used.

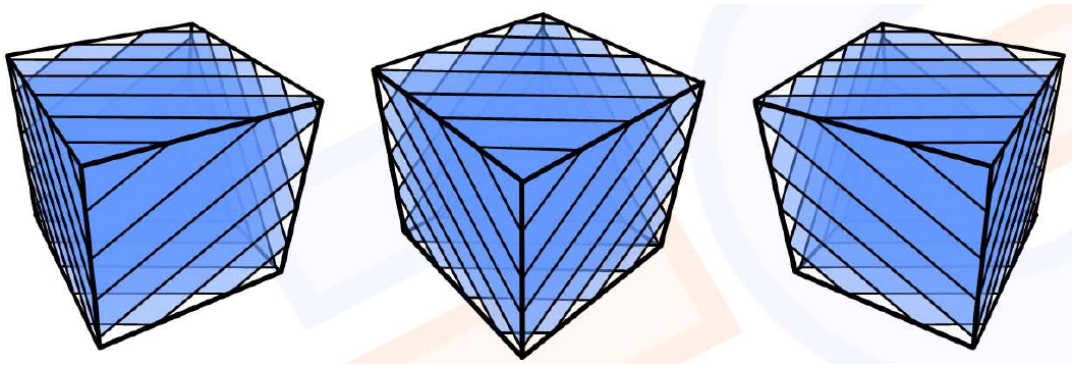


Figure 3.5: Slices are parallel to the image plane (source: presentation for [65])

The optical model here is "Absorption and emission" with the restriction that only a directional light parallel to the viewing direction is allowed (see section 2.2.4).

"+":

- Simplicity
- Relatively fast rendering speed (both on CPU and GPU; this method is a trade-off between quality of the final image and the rendering speed) compared to the high-quality methods mentioned earlier at the beginning of section 3.1
- Sampling rate is constant
- Trilinear interpolation if graphics hardware is used [190]
- Interactive on standard graphics hardware.

"-":

- For graphics hardware: method is not suitable in the presented form if the 3D texture does not fit into the video memory
- The constant sampling rate causes visual artifacts ("jerks") for "fly-through" applications, because it is not sufficient for such cases

  Remarks: In order to somehow neglect this disadvantage a non-constant sample rate could be used [143]
- Relatively low visual quality of the final image compared to the high-quality methods [143]

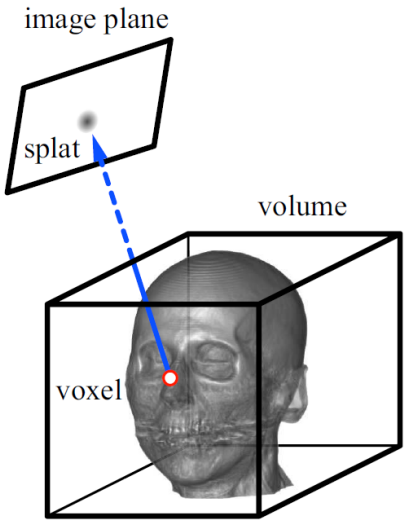


Figure 3.6: Idea of Splatting (source: [38])

- If graphics hardware is used and all textures are stored in the video memory, then it takes a long time to update the volumetric data.

3.1.4 Splatting

There are several works devoted to this DVR technique [251, 44, 189, 237, 238, 148, 149, 147]. The method was initially proposed by Westover [237]. The intensity values of volumetric data are resampled according to the specified parameters (regular or irregular grid), and the samples are projected onto the image plane in front-to-back order. Each voxel is represented as a radially symmetric interpolation kernel equivalent to the sphere with a fuzzy boundary [190]. Projecting such a structure generates a so-called footprint or splat on the image plane. The values of pixels of the image plane are accumulated, while the voxels are being projected. This process is shown in figure 3.6. Splatting classifies (finds colors and opacities for $< coordinates, intensity\ value >$ pairs via a transfer function) and shades the samples prior to projection. Although the authors of [148] also proposed a method allowing classification and shading to be performed after the projection.

The optical model here is either "(Single) scattering and shading" or "Absorption and emission" with a restriction that only a directional light parallel to the viewing direction is allowed (see section 2.2.4). It depends on whether a shading of samples is performed or not. If the shading is performed then normals for the samples will be computed.

"+":

- Adaptive resampling of volumetric data [238]
- Works well for "fly-through" applications
- High quality of the final image [38]
- Antialiasing effect [143].

"-":

- Computationally expensive compared to the low-quality methods [143]
- Inaccuracies on the final image because of the averaging effect of the interpolation kernel [143].

3.1.5 Ray Casting

There are a lot of works devoted to this DVR technique and its enhancements and special applications [124, 86, 132, 38, 39, 249, 192, 87, 196, 133, 125, 89, 65, 88, 144, 197, 145, 117, 13, 236, 55, 108, 247]. In fact, this is the most popular technique nowadays [196, 89, 88]. Additionally, already in 2003 the authors of [117] showed that interactive ray casting is possible on standard graphics hardware. The idea of ray casting in visualization is to numerically evaluate the volume rendering integral (see section 2.2.4) in a straightforward manner. The optical model is "Absorption and emission" with a restriction that only a directional light parallel to the viewing direction is allowed. For each pixel of the image a ray is cast into the scene (see figure 3.7). Along the cast ray the intensity values of the volumetric data (s in our definition of volumetric data in section 2.2.2) are resampled at equidistant intervals, usually using trilinear interpolation [65]. After the resampling an approximation of the volume rendering integral along the ray in either back-to-front or front-to-back order is computed. In this process the mapping of $< coordinates, intensity\ value >$ pairs for the resampled points to colors and opacities according to a previously selected transfer function is used.

Remarks [139]: It is important that the intensity values instead of colors and opacities are interpolated, because fine details would be missing otherwise. I.e. it is important that the transfer function is applied not before the resampling (pre-classification) but after it (post-classification) [66]. See section 2.2.4 for details.

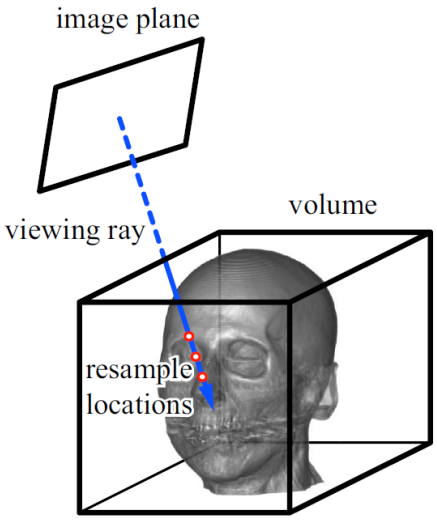


Figure 3.7: Idea of ray casting (source: [38])

The commonly used improvement for ray casting is to use the "(Single) scattering and shading" optical model. In order to use this model, gradients or normals for object surfaces are computed and then used for lighting computations of the sampled points of the volumetric data.

Additionally, in order to imitate "true" reflections, a precomputed environment mapping was proposed. This technique is well-known in polygonal rendering and came from there, especially from computer games. For a more detailed overview over the ray casting technique see [65].

Other techniques for imitation of the "Shadowing" (e.g. [87, 196]) and even "Multiple scattering" [113] optical models were proposed. Such enhancements of the original method are possible at the cost of longer computation time. Additionally, other variations, enhancements and applications of ray casting, e.g. interactive ray casting of large medical data [38], CPU-based ray casting of large data [86], illustrative context-preserving volume rendering [39], an opacity peeling approach [192], multi-volume rendering [195] and both multi-volume and multi-geometry rendering [108] (implemented using NVidia CUDA [163]) were presented by different authors. For a more detailed overview of these techniques we refer the interested reader to [89, 65, 88, 196]. Additionally, recently Crassin et al. [55] proposed a method for interactive rendering of very large data (about 8200^3 voxels). The authors used an adaptive view dependent data structures and made the assumption that details are mostly concentrated on the interface (i.e. the surface) between free space and clusters of density. Therefore this method could give bad results for medical data, because not only the interfaces but the

whole data set should be rendered in high quality.

"+":

- Trilinear interpolation
- Works well for "fly-through" applications
- High quality of the final image [38] (even for the "Absorption and Emission" optical model)
- Interactive on standard graphics hardware (shown in [117]).

"-":

- More computationally expensive compared to the low-quality methods
- Aliasing artifacts if the sample rate is not selected appropriatly [143]
- Misses some details due to point sampling [143].

An important particular case of ray casting is a **Maximum Intensity Projection** (MIP) approach. A detailed description could be found e.g. in [190] and [65]. The idea is to use the maximum intensity value of all resampled points along the cast ray instead of using the approximation of the volume rendering integral. This maximum intensity value together with the coordinates of corresponding resampled point is then mapped to colors and opacities according to the previously selected transfer function.

MIP is quite fast and simple and could be used in some special cases, e.g. visualization of vascular structures [190, 65], but its major drawback is that the depth information is completely lost [190].

3.1.6 Ray Tracing

There are a lot of works devoted to this method and its enhancements and applications [114, 180, 232, 28, 219, 214, 231, 11, 71, 239, 35, 230, 181, 24, 229, 228, 52, 59, 211, 212, 70, 60, 92, 6, 131, 210, 207, 109, 248]. In the modern form ray tracing was introduced in [239] in 1980. Ray tracing is not only a DVR technique but a general approach

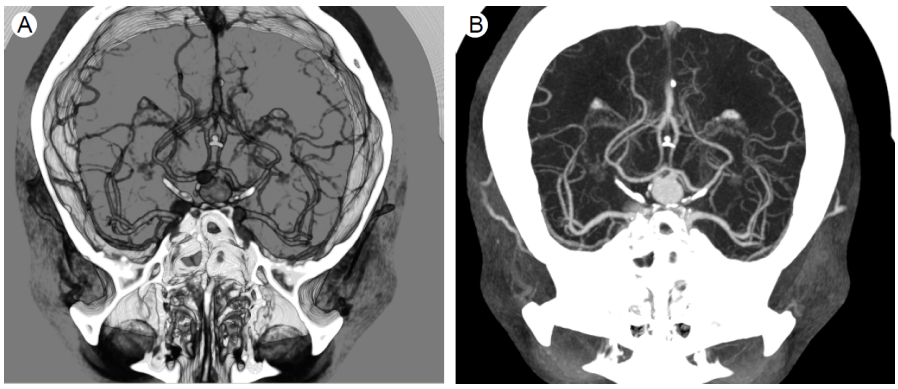


Figure 3.8: A comparison between alpha blending (A) and maximum intensity projection (B) (source: [190])

for high quality rendering of different kinds of data. This method was successfully applied for rendering of polygonal models, NURBS, volumetric data and other object representations. Additionally, ray tracing is used in the movie industry in order to generate photorealistic scenes.

Note: There exists another kind of ray tracing called *Distributed Ray Tracing*. It was introduced in [52] and is presented in some works, e.g. [35]. The difference is that the camera is represented not by a point but by a lens. In order to achieve that, multiple samples per pixel are taken, so that each sample is associated with a different position on the camera lens.

The idea of ray tracing is as follows (see figure 3.9 for an illustration). For each pixel of the final image a "primary" ray is thrown into the scene. When the ray reaches a scene object, it could produce new rays (depending on material properties) – reflection, refraction and shadow rays. These rays are traced according to the following rules:

- Shadow rays are cast from the current hit point cast to all light sources. If they can reach a light source then it means that the reached light source makes a contribution into the lighting of the hit point, and the color is computed and returned. If a shadow ray hits another object then the ray is traced recursevely as the primary ray and then returns the color of the intersection point
- Reflected rays are traced recursevely as primary rays
- Refracted rays are traced recursevely as primary rays.

The depth of recursion is limited. After all rays returned color values, the values are

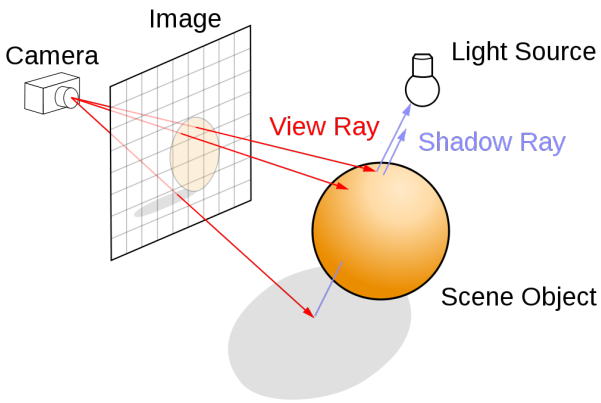


Figure 3.9: Ray tracing (source: Wikipedia article "Ray tracing")

combined according to the weights of the new rays and material properties at the current hit point. The result is then returned as the result of ray tracing for the current primary ray.

A transfer function is used to obtain material properties for a hit point.

Remarks:
A transfer function here could have more output parameters than in the definition in section 2.2.2, because not only colors and opacities could be used for description of materials.

The optical model here is "Multiple scattering and refraction".

To our best knowledge, no fully functional ray tracing system for medical visualization of real size volumetric data (128^3, 256^3 voxels or more) at interactive frame rates (8-30 Hz or more for a viewport of 1024x768 or 1024x1024 pixels) for high-end consumer PCs was presented. It is partially because the ray tracing algorithm itself is hardly appliable to the current GPU architecture. Recently presented works [6, 131, 207, 109] show that interactive ray tracing on GPU is already possible for scenes up to 1M triangles and 256^2 voxels, but not for richer scenes. Ludvigsen and Elster [131] guess that future GPU hardware together with the NVidia OptiX [164] ray tracing API could be promising.

"+":

- High quality of the final image. The quality is much higher than for other methods being considered in our overview

“-”:

- Much more computationally expensive than other methods being considered in our overview
- Hardly applicable to the current GPU architecture
- Aliasing artifacts

  Remarks: The last disadvantage could be eliminated by tracing additional rays for pixels, where the artifacts have appeared.

Figure 3.10: Visual subtraction of the haptic device (source: [54])

3.2 Haptic Interaction

Below we overview a variaty of haptic rendering methods. We should mention that there are generally two kinds of works devoted either to general haptic rendering methods or to more specific approaches for surgical simulation. We mostly write here about methods of the first kind, because we present our contributions more in this area (we considered important particular use cases in our own method, but without loss of generality).

We should also note, that not all works of the second kind describe haptic rendering methods generally or fully enough, or could be generalized from particular use cases, and we consider only those of them which meet the aforementioned requirements. Among the papers of the second kind, one can mark out e.g. Kühnapfel et al. [118], Kuroda et al. [119], Nakao et al. [156], Basdogan et al. [19], De et al. [56], Maciel et al. [135]. A good overview of most popular methods in surgical simulation can be found in the overview paper of Basdogan et al. [20].

Additionally, there are some extra works being related or useful in the field of haptic rendering, e.g. Bickel et al. [27] (devoted to capturing and modeling of a non-linear heterogeneous soft tissue; could be useful, because some of the haptic rendering methods use captured tissue properties), Cosco et al. [54] (devoted to a visual subtraction of the haptic device for mixed (augmented) reality, see figure 3.10), Palmerius et al. [179] (the authors have shown how subdivision of proxy movements can improve precision of volume haptic rendering), Nealen et al. [157] and Otaduy et al. [169] (the last two papers present overviews being devoted mostly to non-real-time physical models for deformable objects; these models could be used for haptic rendering in a reduced form, as well as in simple or special cases (e.g. Garre and Otaduy [77]), or become interactive in the near future because of increasing computer performance).

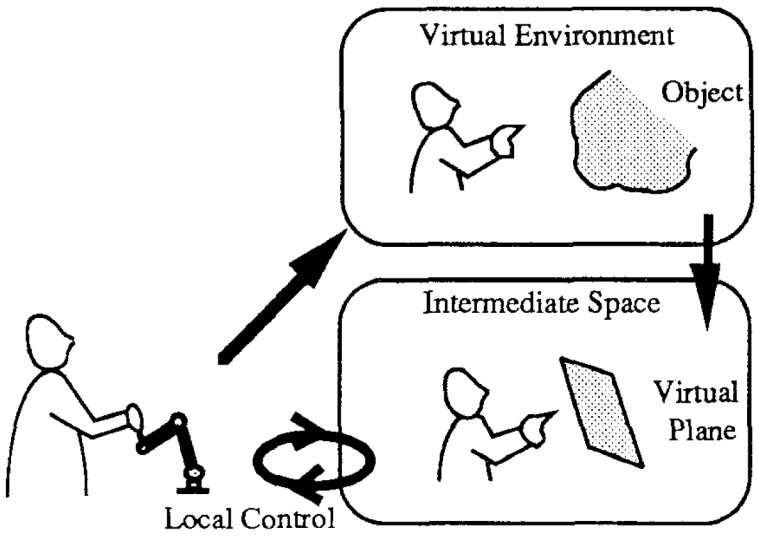


Figure 3.11: Idea of [3] (source: [3])

3.2.1 Rigid-Rigid Methods

For methods from this group the tool and all objects in the virtual world are rigid. Therefore there are only rigid-rigid interactions.

- **Adachi et al. [3] and Mark et al. [137]** – **Adachi et al.** were the first who proposed an intermediate representation of the virtual environment.

  As far as collision detection with complex objects worked too slowly for haptic rendering at that time (1995-1996), the authors proposed to use two threads, so that in the slow thread a tangential plane on the virtual surface at the nearest point from the position of the probe is transmitted to the fast force-feedback thread and serves there as a unilateral constraint (a virtual plane). A spring-damper penetration-based collision response was used for collision detection for high frequency computations. The authors developed their own "SPICE" haptic display, which required a haptic update rate of 500 Hz or more.

  A drawback of the method is that force discontinuities could arise if the new plane equation, which is transmitted from the thread with low frequency computations, causes the tool in the fast thread to be embedded in the new surface. This could happen if the tool in the fast thread is on one side of the virtual plane (out of the object) during the current iteration of the slow thread, and on the opposite side (inside the object) at the new iteration, i.e. after the new equation has been transmitted.

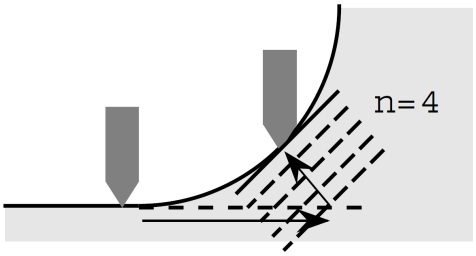


Figure 3.12: When a new plane equation causes the tool to be embedded in the surface, the algorithm will artificially lower the plane to the tool position and then raise it linearly to the correct position afte n force loop cycles (source: [137])

Later on, **Mark et al. [137]** adapted and extended the method. They used two intermediate representations: *tool-plane* and *point-to-point springs* (where one of the points is controlled by the probe). The idea of these representations altogether is to apply a penetration-based spring-based force to the probe. Additionally, the authors proposed an interpolation between two successive intermediate representations in order to eliminate strong force discontinuities arising in [3]. The idea is shown in figure 3.12. The authors reported about an update rate of approximately 1 kHz for the force-feedback thread.

- **Salisbury, Zilles et al. [203]** – the authors proposed a 3-DoFs haptic rendering method for polygonal objects. They used the virtual coupling and several object representations: initially a vector fields representation was employed (drawbacks could be found in [250]), but later on a god-object representation was proposed. The main idea is the following:
  The method is used for collision detection, collision response and force feedback generation. It is constraint-based and stops the virtual contact point (the **god-object**) from penetrating into other objects. The method tracks the god-object so that it remains on the surface when a virtual object is probed. In more detail, knowing the positions of the god-object at the current and previous frames, a set of surfaces constraining an inter-frame motion are identified, and then the new position is computed using Lagrange multipliers as a constrained optimization problem.

  Note: One should mention that no fast "high-level" collision detection was considered, since [250] was fully devoted to the idea of the god-object method.

Additionally, the authors pointed out, that in opposite to graphics rendering, a small part of the data is used for haptic rendering, because generally only local

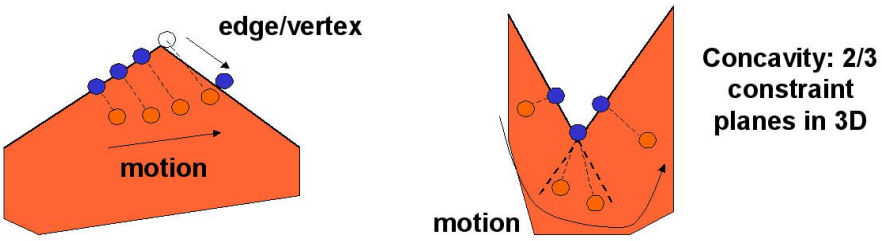


Figure 3.13: God-object method (source: [82])

interactions appear. Therefore the authors suggested to decouple these local interaction computations from global object dynamics and use a "local window" for haptic rendering.

The authors presented a support for surface friction and "texture rendering" for their system. They also mentioned, that it could be a good idea to represent a non-homogeneous material via a potential field, although no results were presented.

Remarks: The term "texture rendering" in [203] is actually not haptic rendering of texture producing a tactile perception, but height-field-based kinesthetic rendering. This means, that only kinesthetic sensations are produced and only a kinesthetic display was used.

In [250] the authors reported about objects consisting of about 600 triangles and an update rate of approximately 1 kHz for haptic rendering.

- **Ruspini et al. [198]** – the authors proposed a "Virtual proxy" 3-DoFs haptic rendering method for polygonal objects, which is an extension of the god-object method [203].

  Ruspini et al. suggested to model the tool (the **virtual proxy**) as a sphere and to solve the optimization problem in the configuration space (see figure 3.14).

  At each frame, the position of the probe in the virtual environment is set as a goal for the tool. Then possible constraint surfaces are identified using the ray between the old position of the virtual proxy (the tool) and the goal position. After that a quadratic optimization problem is solved and a subgoal position is found. This process is repeated until the subgoal position could not be closer to the goal.

  The authors incorporated a **force shading** technique allowing smooth haptic rendering of a surface, similar to the Phong shading in computer graphics. This is done by interpolation of object normals. Additionally, the authors included a

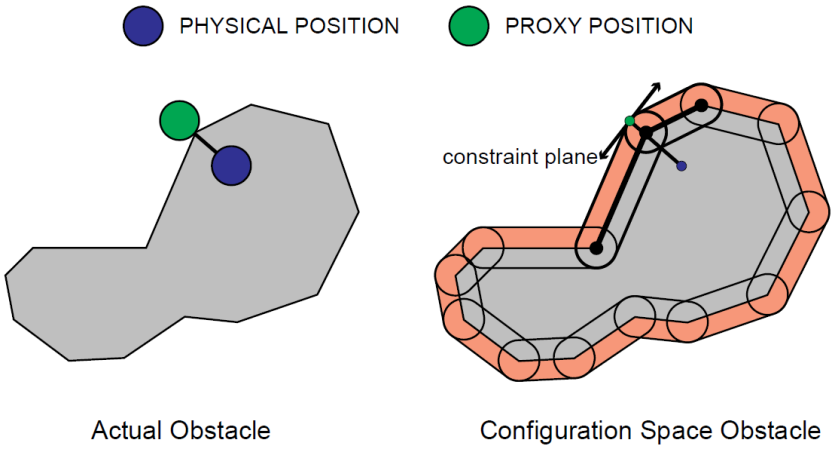


Figure 3.14: Actual and configuration space obstacles (source: [198])

support for the static, viscous and dynamic friction being realized by adding a dynamic behaviour to the tool.

For faster computations, a bounding spheres hierarchy representation was used for scene objects (see figure 3.15).

The authors wrote that a haptic rendering update rate is "typically greater than 1 kHz". Additionally, they mentioned that object translations and rotations are allowed and are performed at an update rate of only 30 Hz because of the client-server application architecture (the server – haptic rendering, the client – the remaining computations).

It was pointed out that an arbitrary number of objects could be presented in the scene, and the objects could be interactively added and removed. Among the presented examples, the maximum number of polygons per object was "more than 24000". It was not reported about the maximum number of polygons in the scene.

Because of the difference in the server and the client update rate, environmental objects move "jerky" from the point of view of the haptic server. Therefore "jerky" forces could appear in case of a contact between the tool and such a moving object. Additionally, the authors mentioned that in bad cases the tool could lie outside of an object at one haptic iteration and within it at the next haptic iteration.

- **McNeely et al. [141]** – 6-DoFs haptic rendering using voxmaps and pointshells was firstly proposed in this paper and then improved in later works. The idea is the following:
  Dynamic objects are represented by a set of surface point samples plus associated inward pointing surface normals, collectively called a **point shell**. Actually, there

Figure 3.15: Cut of the bounding spheres hierarchy (source: [198])

is only one dynamic object – the tool. The environment consisting of static objects is collectively represented by a single spatial occupancy map called a **voxmap** (see figure 3.16). The voxmap will be sampled when a contact with the point shell appears.

The authors introduced four types of voxels and used a voxel tree (based on octree) for faster collision detection calculations. Pre-contact penalty forces (also called a "force layer") were used in order to avoid objects interpenetration. The virtual coupling and numerical integration of Newton-Euler equation were used. No multirate force computations were employed. According to this work and [233], although haptic rendering works at 1 kHz, the fixed 1 ms timestep force calculations leads to the same instability problems as those generally mentioned in section 2.1.14.

The authors reported about the scene with static geometry voxelized from 593409 polygons and the tool represented by the pointshell consisting of up to 600 points.

- **Wan and McNeely [233]** – this is an evolution of the previous work of McNeely et al. [141]. In order to eliminate instability problems, the authors replaced Newtonian dynamics with a quasi-static approximation (QSA) approach. The idea is to ignore any dynamic properties and solve the static equilibrium at each haptic timestep. The equilibrium is found for the system consisting of collision penalty "spring-like" forces and spring-based virtual coupling forces. The exclusive

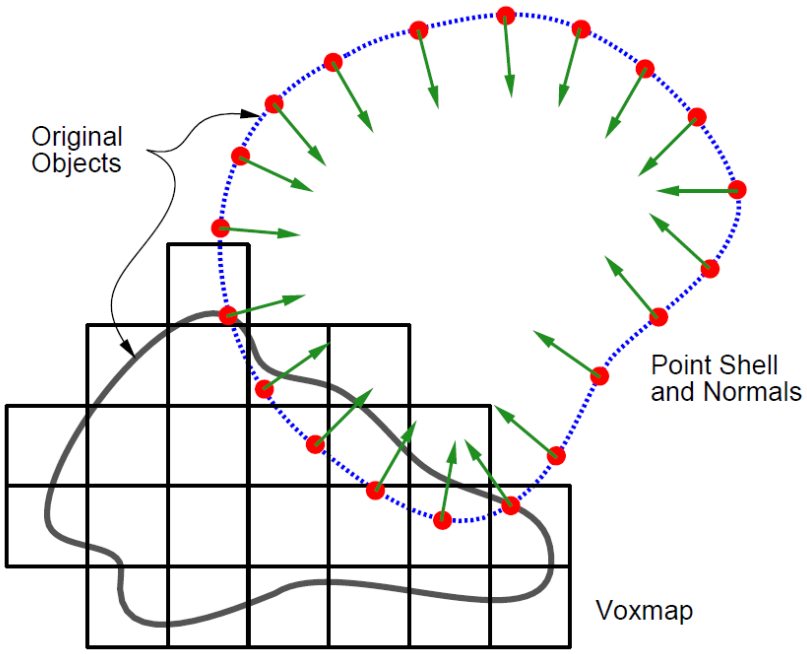


Figure 3.16: Point shell and voxmap (source: [141])

use of spring forces ensures that QSA yields a linear problem.

The authors wrote that the QSA approach increased stability of haptic rendering compared to [141], but sacrifies the dynamic realism [233, 142].

There is only one dynamic object – the tool – allowed in the system.

The authors reported about the virtual scenario with static geometry voxelized from 25700 polygons and contained 176220 surface voxels and the tool represented by 4754 points obtained from the model consisting of 4600 polygons.

- **McNeely et al. [142]** – this is also an evolution of [141], but not an evolution of [233]. The approach realizes an order-of-magnitude improvement in the spatial accuracy at the cost of reduced haptic fidelity, which is proved to be acceptable [142].

  Compared to [141], the authors extended the object voxelization beyond its surface to some degree and introduced distance fields on voxelized data. Additionally, if the volumetric data is obtained from polygonal data then surface-voxel, edge-voxel and vertex-voxel distance fields will be used in order to speed up collision detection computations. The authors also made optimizations to the original method (1) by introducing the dynamically adjustable $MaxTravel$ value (the maximum allowed movement per frame) for points representing the tool, (2) by exploiting

the hierarchical temporal coherence for the voxel tree and (3) by introducing point drifting.

Remarks: As far as the maximum movement per frame of the points representing the tool ($MaxTravel$) is limited by authors to $\frac{1}{2}$ voxel, this drastically increases haptic rendering stability of the system, because problems with "too fast" movement dissapeared, e.g. going through thin objects or appearing of force discontinuities because of crossing of boundaries of internal Voronoi cells.

Additionally, the authors made a dynamic pre-fetching of voxel data being too big to fit into the RAM, and introduced a collaborate multi-user haptic rendering approach. Although the latter approach has a drawback that there could be a divergence of multiple tool instances occuring when an object, e.g. a thin wall, is trapped between them.

Among the presented examples, the largest scene contains 1,78x10^9 voxels obtained from 2,76x10^6 triangles for the environment and 1,14x10^6 points obtained from 40476 triangles for the tool.

- **Sagardia et al. [201]** – in this work the authors proposed improvements of the voxmap-pointshell algorithm from [141]: fast algorithms to generate voxmaps and pointshells.

  Given the polygonal model, the most important part of the voxmap generation algorithm is to find surface-voxels. This is done in the following way. Initially the polygonal model is placed in the empty voxmap. Further, for each triangle of the polygonal model, the candidate surface-voxels within the triangle's bounding box are found and checked for collision against the triangle. In case they collide, they are marked as surface-voxels. The authors proposed several optimizations for fast navigation through the triangle's bounding box to find these candidate surface-voxels, as well as a separation axis theorem based approach for fast collision detection between the triangle and the found candidate surface-voxels.

  In order to obtain a pointshell of the polygonal model, its voxmap is generated first. Further, the pointshell is obtained from the voxmap by projecting the surface-voxel centers onto the corresponding triangles. The projection is based on a nonlinear optimization method that finds the closest points on the triangle to the voxel centers.

  The authors reported about the increased "quality" of generated voxmaps and pointshells compared to previous generation approaches. Further on, the genera-

Figure 3.17: The DLR's bi-manual haptic interface used in the VR simulator for telerobotic on-orbit servicing (source: [200])

tion process is 2 to 52 times faster for voxmaps and is 1.8 to 19 times faster for pointshells.

The proposed approach was later used in the DLR's Virtual Reality simulator for telerobotic on-orbit servicing with visual and haptic feedback [200] (see figure 3.17).

- **Gregory, Lin et al. [85]** – in this paper the authors proposed a sophisticated 6-DoFs haptic rendering system for polygonal objects. Additionally, environment is not fully static – the system can interactively handle a few moving environmental objects.

  At the preprocessing step all objects are decomposed to a set of convex primitives, and the whole collision detection technique generally consists of two steps:

  1. AABB collision detection
  2. exact collision detection between pairs of convex primitives in expected constant time per pair (the authors proposed an incremental algorithm, which uses information from the previous haptic frame).

  In more detail, Gregory, Lin et al. used an extended technique of the virtual proxy [198] and a time coherence based prioritization for the sweep-and-prune algorithm. In order to minimize penetration depth computations, pre-contact collisions were used – surface borders were incremented by a small $delta = current\_velocity \cdot force\_update\_period$.

Spring-based penalty-based collision response and a spring-based force feedback approach were employed in the system. Additionally, the authors used the adapted force shading technique from [198] (interpolation of force normals in order to achieve smooth surface haptic rendering) and linear smoothing for forces and torques in order to minimize haptic rendering discountinuities between successive frames and to achieve higher stability of haptic interaction. In order to further increase the stability, the maximum allowed force difference between successive frames was introduced.

There are no multirate force computations in the system. Haptic rendering works at 1 kHz or more. The system worked well with objects decomposited into 10-30 convex primitives.

The authors mentioned that instabilities could arise in some cases, e.g. in the peg-in-the-hole scenario or if the user exerts too much force.

Examples with up to 13 environmental objects were presented. Objects could move, rotate and interact with each other. Each object consisted of a few hundreds polygons.

- **Otaduy, Lin et al. [172]** and **Otaduy and Lin [173]** – the authors proposed 6-DoFs haptic kinesthetic rendering of interactions between triangulated models with haptic textures (i.e. objects "with fine surface details" in terms of haptics rendering). The papers are based on the PhD thesis of Otaduy [177]. Collision detection between low-resolution meshes is based on the contact levels of detail (see descriptions of Otaduy and Lin [175], [176] for more details).

  In order to represent textured objects, the authors associated a height field with each of them. Basing on results of perceptual studies from Klatzky et al. [112] and others, Otaduy, Lin et al. proposed a rendering technique which uses a gradient of directional penetration depth into the height fields for computing adopted penalty-based collision response and force feedback. In order to speed up haptic rendering, computations of the aforementioned penetration depth and gradient at every contact were implemented on GPU.

  In opposite to [177], a multirate architecture (a 1 kHz haptic thread and lower frequency simulation thread) and the virtual coupling were used.

  The authors mentioned that discontinuities in collision detection between low-resolution objects is a potential source of instabilities. Additionally, stability problems from [175, 176] take place. Additionally, in [177] it was pointed out that the high gradient of penetration depth produces high contact stiffness, which

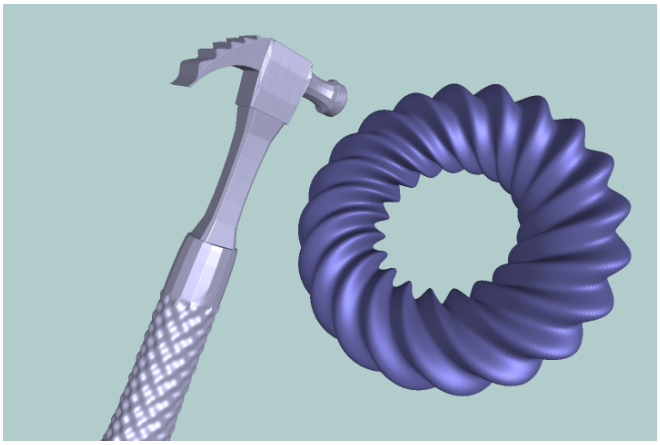


Figure 3.18: Haptically textured Hammer and textured Helicoidal Torus (source: [177])

can also induce instabilities. Another important issue of the presented method is that in contact scenarios with large contact areas, e.g. for the problem of screw insertion or for a contact between interlocking features, the definitions of a local and directional penetration depth are not applicable. This could lead to incorrect haptic rendering. One more issue is that the algorithm is susceptable to aliasing problems (as other sample-based techniques).

Due to the limitations of the method all objects in the scene are static, except for the tool.

Examples with the tool and up to two environmental objects were presented. The largest scene contains 433152 triangles for the tool (represented by 518 triangles model with haptic texture) and 658432 triangles for the environmental object (represented by 720 triangles model with haptic texture) .

- **Otaduy and Lin [175], [176]** – the authors proposed a 6-DoFs haptic rendering approach for polygonal objects represented by triangle meshes. These papers are based on the PhD thesis of Otaduy [177]. All objects in the scene are static, except for the tool.

  Note: For illustrations of these and some later works of the authors, one can take a look at Otaduy [170].

  Collision detection is based on the sensation-preserving object simplification technique in order to make computations faster. The idea of the technique is to introduce some metrics of sensation of haptic rendering, build multi-resolution levels of details (LODs) and then select appropriate LODs of collided objects according

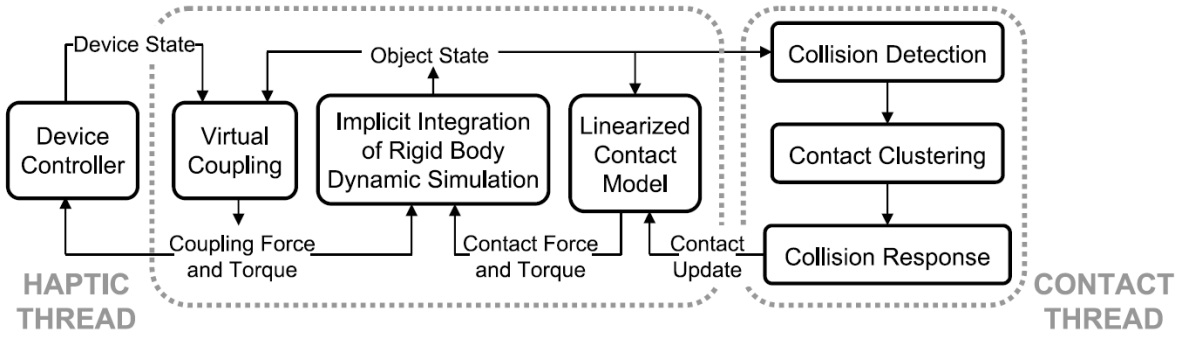


Figure 3.19: The haptic thread runs at force update rates of 1 kHz simulating the dynamics of the grasped object and computing force feedback, while the contact thread runs asynchronously and updates contact forces (source: [175])

to the introduced metrics at contact areas during haptic rendering. See [177] and [82] for more details.

Used haptic rendering scheme is shown in figure 3.19.

The authors employed viscoelastic penalty-based collision response. Additionally, they used the virtual non-linear viscoelastic coupling between tool coordinates and orientation in the contact thread and the haptic thread.

As stated in [175, 176, 177], there are some limitations of the system, e.g. geometric discontinuities in contacts between collided objects in the output of collision queries. Additionally, there will be haptic rendering instabilites e.g. if the tool passed through scene objects. The authors mentioned that instabilites arise due to discrete timestep computations and use of penalty based methods.

Examples with the tool and one environmental object were presented. The largest scene contains 47339 triangles for the tool and 40180 triangles for the environmental object.

- **Kim et al. [111]** – the authors presented an "implicit-based" haptic rendering technique for volumetric data with an additional pre-processing step.

  The implicit surface S of the object is described by the implicit function f, also called *potential*:

  $$S = \left\{(x, y, z) \in \mathbb{R}^3 | f(x, y, z) = 0\right\}. \tag{3.1}$$

  The set of points for which $f = 0$ defines the implicit surface. Additionally, if $f > 0$ then the point is outside the object. If $f < 0$ then inside.

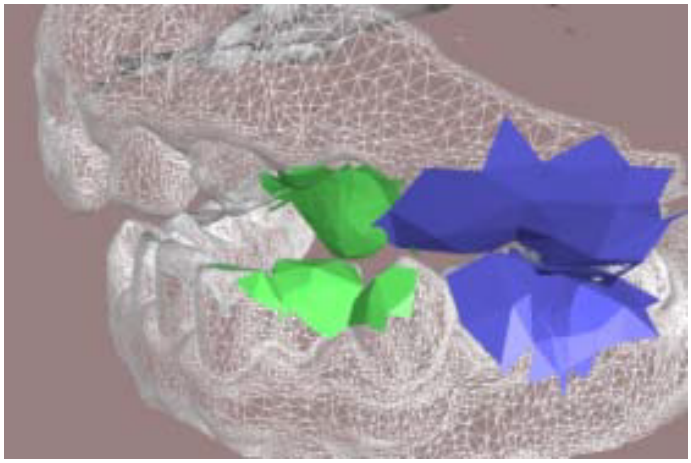


Figure 3.20: (*Wireframe*) the finest resolution of the objects; (*in color*) adaptively selected resolution for haptic rendering of the contact areas (source: [82])

In the algorithm of the authors the volumetric representation is defined using a discrete potential stored on a 3D regular grid. The potential value of each point indicates the proximity to the surface and is generated at the pre-processing step using the closest point transform (CPT). CPT converts the explicit representation of geometric surface of the input triangulated model into an implicite one.

The proximity to the surface is determined by sampling the potential value at the position of the device manipulator in the scene. If the distance becomes zero or changes sign, then a collision appeared. The virtual interaction point is constrained by the surface and moves along it. A spring-damper model between the position of the manipulator and the interaction point is used.

Additionally, a friction force and haptic texturing are supported. The friction force takes into account the friction coefficient and the depth of penetration. The haptic texturing is simulated by applying Gaussian noise and texture patterns directly to the potential value of each point in the 3D grid.

Additionally, a virtual sculpting prototype was proposed: when the position of the device manipulator is applied at a region close to the actual surface for a period of time, a force field is created. This forcee field propagates through time to the neighboring nodes, changing their potential values.

The authors reported about a constant haptic update rate of 1 kHz. There were presented examples with the interaction point and one object in the scene. The biggest input model consisted of 11820 triangles transformed into the 150x150x150 grid. For the sculpting prototype it was reported about 7468 triangles and the

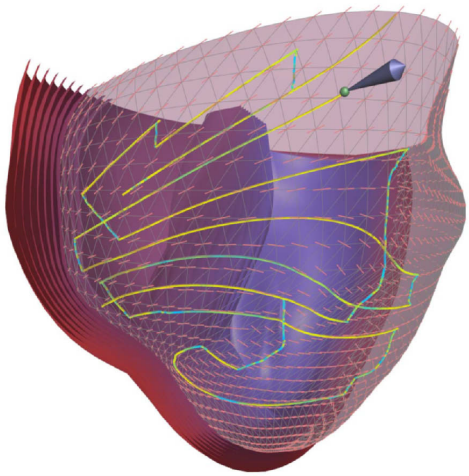


Figure 3.21: Constraint-based 3-DoFs haptic rendering of muscle fibers (source: [98])

70x70x70 grid respectively.

- **Ikits et al. [98]** – this work is devoted to a constraint-based technique for haptic volume exploration, and the authors showed example applications of their method for tracing heart muscle fibers (see figure 3.21) and exploration of diffusion tensor fields.

  Ikits et al. represented the tool as a point (called "proxy") and introduced constraint-based motion rules and haptic transfer functions in order to achieve an effect of constrained 3-DoFs haptic rendering. The virtual coupling technique was used. Due to the specificity of the method there are no collisions in the system. No information about an exact haptic rendering rate was given.

  Note: A similar task of force fields and 3D-functions exploration using 6-DoFs haptic rendering was mentioned in Lin et al. [128].

  Examples with one object for exploration were presented. The two biggest datasets consisted of 200000 tetrahedral elements and 148190 pixels (a DT-MRI slice) respectively.

- **Johnson and Willemsen [104]** – the authors proposed a 6-DoFs haptic rendering method for polygonal objects. They used spatialized normal cone hierarchies (SNCH, see later paper Johnson et al. [106]) for fast collision detection between the tool and an environmental object:
  The distance between two parametric surfaces $F(u, v)$ (e.g. the tool or its part)

and $G(s,t)$ (e.g. an environmental object or its part) can be described as

$$D^2(u,v,s,t) = \left(F(u,v) - F(s,t)\right)^2 . \tag{3.2}$$

The extrema of this distance can be found by differentiating and finding the roots of the resulting set of equations. The idea of the proposed SNCH search technique is to approximately find the aforementioned extrema, i.e. local minimum distances between the tool and the environmental object. For that, the SNC hierarchy is firstly created. The leaf-level of the hierarchy is built on object triangles, and each leaf consists of:

- a cone, represented by a cone axis vector colliniear to the triangle normal
- a cone semi-angle of maximum deviation of contained normal
- a sphere represented by a center and a radius, which bounds the representing geometry.

The next level of hierarchy is built by merging the preceding level. Using this hierarchy, *all* potential contacts could be found.

The authors used a cutoff (offset) from environmental objects, so that if the tool is closer than the cutoff-distance then a collision will be assumed and a spring-based collision response will be performed. The direct haptic rendering was used for the force-feedback generation.

Elements of the scene could be added and deleted without preprocessing, but precomputing of spatialized cone hierarchy for each object is necessary.

The authors reported about a haptic rendering rate of "hundreds of Hz". Examples of interaction between the tool and one environmental object were presented in the paper (maximum about 5650 triangles for the tool and 23600 triangles for the environment in one scene).

- **Johnson and Willemsen [103]** – this paper is devoted to an acceleration of the system being proposed in [104] by introducing a multirate architecture.

  The algorithm firstly computes all local minimum distances (LMDs) within the cutoff distance using the global SNCH search. Then these LMDs are put into the fast local update thread performing a local gradient descent on these LMDs according to the new positions of scene objects and updating these LMDs (i.e. by tracking the LMDs according to objects movement). The updated LMDs are used for collision response computations. In parallel to this fast thread, the slow

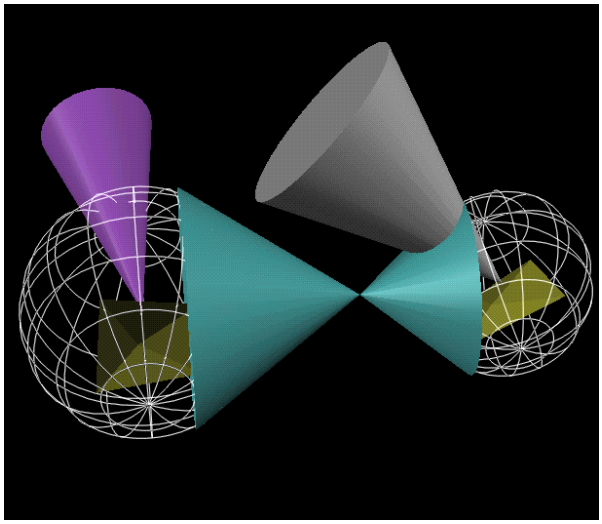


Figure 3.22: Normal cones (source: [106])

but exact thread is executed performing the global SNCH search to compute new *exact* LMDs. When it finishes an iteration, it will notify the fast thread that the new exact LMDs are available, and the fast thread updates its set of LMDs accordingly.

As a prerequisite to the algorithm, a topological connectivity precomputations for each object are required.

Additionally, the authors presented an interesting use case of their system – the training of a collision-free path finding (figure 3.23), which is described in **Johnson, Willemsen and Cohen [105]**.

The authors reported that their haptic rendering system works at about 1 kHz. Examples of interactions between the tool and maximum two environmental objects were presented in [106] and [105]. For those examples, the maximum total amount of triangles in the scene was 153000 (40000 for the tool and 113000 for the environment).

- **Johnson, Willemsen and Cohen [106]** – this work is based on and generally includes [103]. Here the authors described the SNCH search in more detail and showed that it could be adapted to estimation of model-model penetration depth in order to get preciser collision response and force-feedback generation. This was done by searching the maximum distance (instead of the minimum one) between the given objects. In order to speed up the approach, the authors introduced adaptive cutoff distances.

  The drawback of the penetration depth estimation approach is that LMDs are not

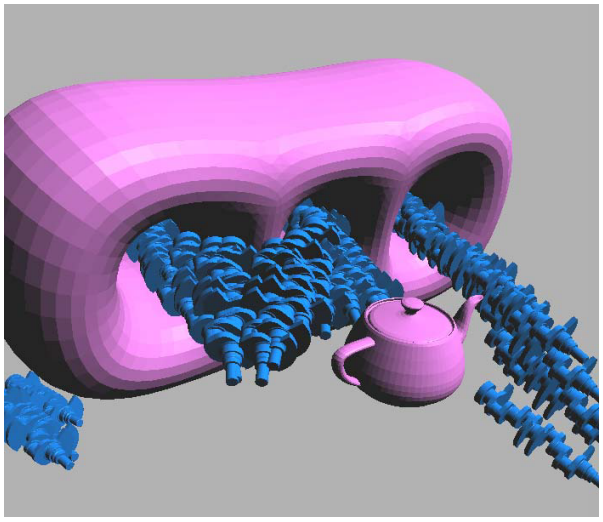


Figure 3.23: Collision-free path finding using 6-DoFs haptic rendering (source: [105])

used together with it. Therefore the approach cannot handle as high-resolution models as the approach with LMDs could. The authors showed the example with about 8200 triangles for the tool and 11800 triangles for the environment and reported about a haptic update rate of about a few hundreds of Hz.

- **Ortega et al. [168]** – the authors proposed a generalization of the 3-DoFs god-object method for haptic interaction between rigid bodies [203] to 6-DoFs.

  A multirate system architecture was used. In the slow thread the motion of the god-object is computed, and the computations roughly consist of the following steps:

  1. compute the unconstrained accelleration, based on the previous tool configuration and the currect probe configuration using the spring virtual coupling
  2. compute the constrained acceleration, based on the current contact information and the unconstained accelleration using constraint-based quasi-statics and Gauss' least constraint principle
  3. determine the target god-object configuration and perform a continuous collision detection from Redon et al. [188]. If there are no collisions then the new god-object configuration is equal to the target configuration, otherwise the continuous collision detection returns the new configuration.

  In the fast thread force feedback is generated. The calculations are similar to the constraint-based quasi-static computations in the slow thread, but position and orientation of the god-object are assumed to be fixed (therefore no collision

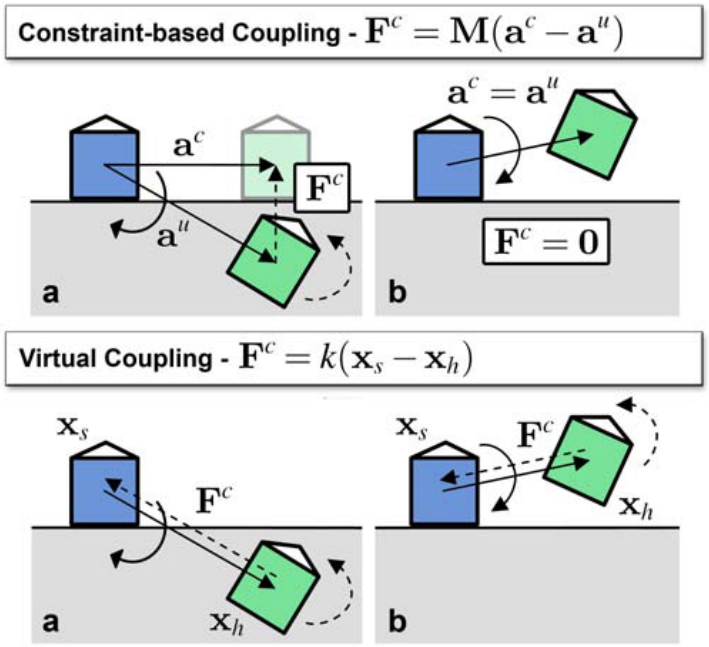


Figure 3.24: The constraint-based approach allows to remove force artifacts typically found in virtual coupling approaches (the handle is shown in green) (source: [168])

detection is needed) and the matrices computed in the slow thread are used. In order to transmit the computed 6-D force (linear force + torque) to a user, the spring-based **constraint-based** virtual coupling is used. Figure 3.24 shows the difference between the constraint-based virtual coupling and the virtual coupling.

In order to smooth force feedback in a case when new constrains from the slow thread appears in the fast thread after the update, a constrains adaptation technique was proposed – see figure 3.25.

Remarks: This technique generalizes the technique from **Mark et al.** [**137**].

Additionally, the authors put an accent that no objects interpenetration could occure during the simulation and that the god-object can slide over other objects.

Examples with the tool and one other object in the scene were presented. The maximum total number of triangles is 54000 (27000 for the tool and 27000 for the second object). It was reported about 70-300 Hz for the slow thread and "above 80 kHz" for the fast thread.

Among the drawbacks of the method, the authors mentioned that the non-penetration constraints linearization might reduce quality of the force applied to the user, when a large discrepancy between configurations of the god-object and a

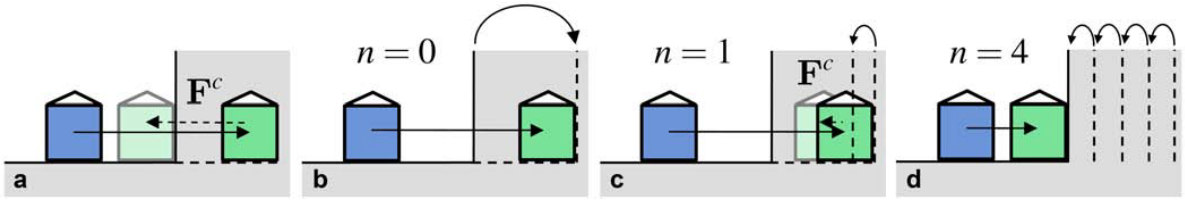


Figure 3.25: *Constrains adaptation technique*: when a new constraint (here the vertical plane), which would create too strong constraint force, appears (a), it is first translated so that it is satisfied by the current haptic device configuration (b), and then step-by-step returned to its initial position (c-d) (source: [168])

haptic device occurs.

- **Vidal et al. [224]** – the authors made a 6-DoFs simulation of ultrasound (US) guided needle puncture and proposed proxy-based surface/volume haptic rendering for that. It is used as a training tool for interventional radiology (IR) using actual patient data.

  Representation of an object is voxel-based, but its surface is represented as a triangular polygonal mesh.

  Two haptic devices are used. The first acts like a US transducer, and the second haptic device like a needle, which can puncture through the virtual skin and tissues.

  For the US transducer, the haptic rendering algorithm is proxy-based and uses the polygon mesh of the skin surface.

  For the needle, the haptic rendering has been done as follows. When the the skin surface is explored, the proxy-based algorithm with additional friction parameters is used. When the force applied by the needle at a given point of the skin exceeds some threshold, the haptic rendering is switched to a second mode allowing to penetrate internal tissues, where tissue properties are extracted from a look-up table. Additionally, a haptic volume rendering approach is used in this mode in order to prevent the needle penetrating bones. In more detail, if any voxel belonging to a bone is detected along the straight line from the proxy to the actual position of the device, then the proxy is moved to the entry point of the line into the found bone (see figure 3.26).

  The authors used a pre-measured forces of real tissue in order to make simulations more physically realistic. More specifically, they used them in the algrorithm generating the patient-specific anatomical model (the look-up table) basing on the input CT scan of the patient.

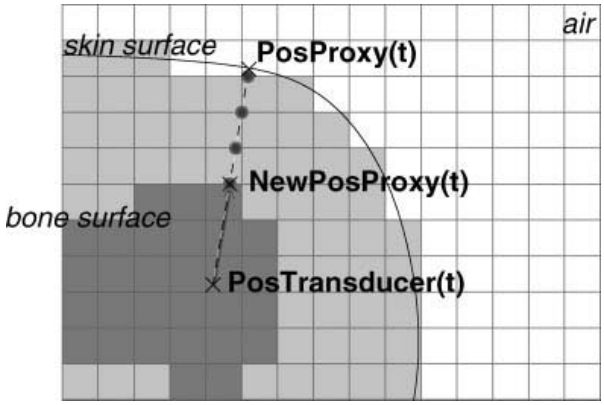


Figure 3.26: Detecting a contact with bone in [224] (source: [224])

The authors mentioned that the minimal haptic update rate for any haptic rendering approach must be 1 kHz, although no concrete numbers about the update rate and size of virtual objects were given.

- **Weller and Zachmann [235]** – the authors proposed a 6-DoFs haptic rendering method based on their new geometric data structure: inner sphere trees (ISTs).

  The main idea is to bound ("pack") an object from inside with a set of non-overlapping bounding volumes. Figure 3.27 illustrates the stages of the hierarchy building process:

  1. voxelize the object (left-top)
  2. compute distance from each voxel to the closest triangle (right-top; transparency = distance) and remember it together with the corresponding polygon (needed further for the determination of normal)
  3. pick a voxel with the largest distance and put a sphere at its center (left-bottom)
  4. proceed incrementally and obtain a dense sphere packing of the object at the end (right-bottom).

  ISTs answers proximity queries similarly to classical recursive schemes which simultaneously traverse two given hierarchies. additionally, ISTs supports a penetration query by returning a penetration volume (the volume of intersection of the two given objects). The authors proposed variations of the both proximity and penetration volume queries guaranteeing a predefined query time budget but returning an average result.

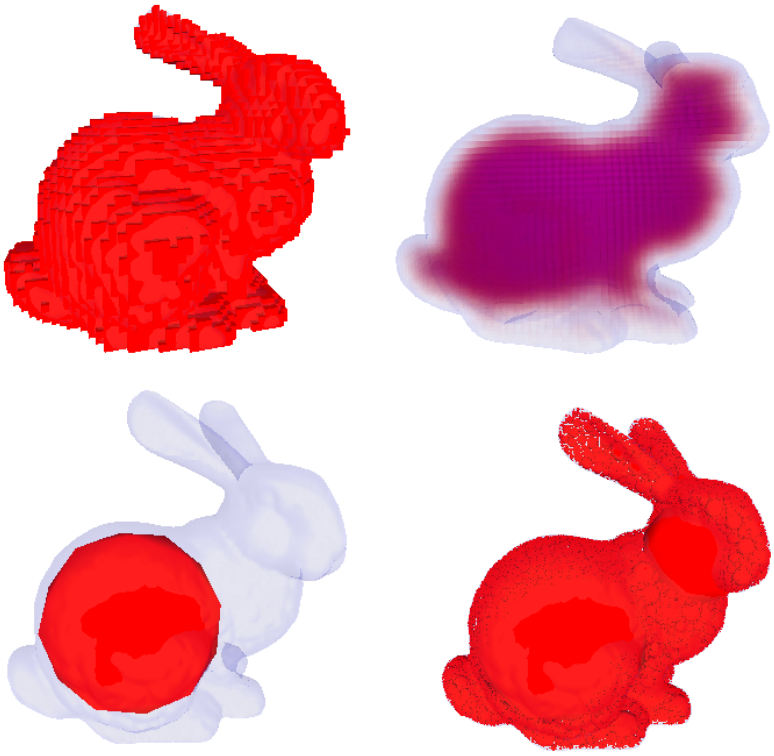


Figure 3.27: Stages of the sphere packing algorithm (source: [235])

A penalty-based approach is used for collision response and force feedback computations. In more detail, let us assume that we have an object represented by a set of spheres $R = \{R_i\}$ colliding with an object represented by $S = \{S_i\}$. After the penetration query has returned the set of overlapping spheres (potentially overlapping spheres in case of the time-budget version), the amount of repulsion force for each pair (R_i, S_j) is computed as:

$$f(R_i) = k_c \, Vol(R_i \cap S_j) \, \mathbf{n}_{R_i}, \tag{3.3}$$

where k_c is the contact stiffness; $Vol()$ is the volume; $\mathbf{n}_{R_i}$ is the contact normal. Then, the resulting linear penalty force for the object represented by R will be:

$$f(R) = \sum_{R_i \cap S_j \neq \varnothing} f(R_i). \tag{3.4}$$

Similar to equation 3.3, a torque for each pair of overlapping spheres is:

$$\tau(R_i) = \left(P_{(R_i,S_j)} - C_m\right) \times f(R_i), \tag{3.5}$$

where C_m is the center of mass of object represented by S; $P_{(R_i,S_j)}$ is the point of collision of spheres R_i and S_j, defined as a center of overlapping volume.

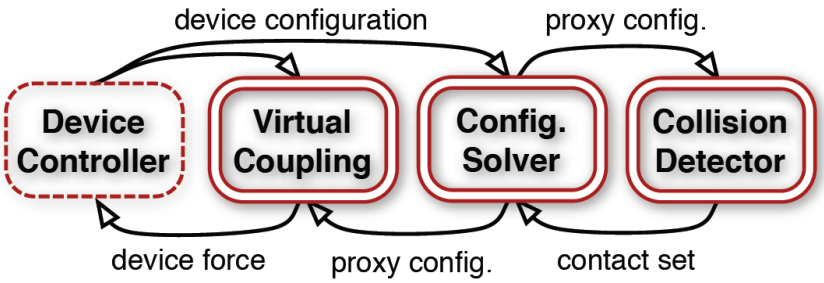


Figure 3.28: System architecture for [41] (source: [41])

The resulting torque is computed similar to equation 3.4.

The authors reported about maximum 700k triangles in the scene and a haptic update rate of "at least 200 Hz on average". Only examples with the tool and one other object in the scene were presented.

Additionally, it was mentioned that the approach is restricted to watertight objects.

- **Chan et al. [41]** – the authors presented a method of 6-DoFs haptic rendering of isosurfaces embedded within volumetric data. The virtual tool is represented as a point-sampled surface (a point shell) and is massless.

  The algorithm is based on a quasi-static formulation of motion, and multiple contacts are allowed. The authors used a configuration solver to compute an unconstrained and constrained motion of the tool. The architecture of the system is shown in figure 3.28.

  The collision detection is designed to work at haptic rates, and it finds the earliest time at which the collision of the tool with a volumetric isosuface occurs. The motion of the tool is constrained (the tool cannot go besides the isosurface), and a spring-based virtual coupling between the tool and the actual position and orientation of the device manipulator is used.

  All components of the algorithm run at a haptic update rate of 1 kHz. The largest reported resolution of the volume was 512x512x361 voxels.

- **Corenthy et al. [53]** – the authors proposed a 3-DoFs haptic rendering approach to feel isosurfaces in volumetric data.

  Isosurfaces are defined on tetrahedral meshes created from the volumetric data (24 tetrahedra per voxel). An isosurface is extracted dynamically in a proximity of the interaction point according to the desired isosurface value being interactively

defined by the user. The authors pointed out that the use of tetrahedral meshes provides continuity and watertightness of the isosurface.

A constrained movement of the tool is based on the constraint-based haptic rendering algorithm by Ruspini et al. [198]. A spring-based virtual coupling between the interaction point and the actual position of the device manipulator was used.

It is also allowed to penetrate besides the isosurface. In more detail, if the user presses against the isosurface over the specified *force* threshold, this is interpreted as if the user wants to go deeper into the data, and the isovalue will be changed accordingly. The authors mentioned, that in practice they just check that the *distance* between the position of the device manipulator and the interaction point does not exceed the user defined threshold.

The authors wrote that the haptic update rate is 1 kHz. The two largest reported datasets consisted of 128x256x256 voxels and 1024x1024x39 voxels respectively.

3.2.2 Methods with Allowed Data Modification

For methods of this group the tool is rigid and the environment could be modified. The following methods could be marked out:

- **Avila and Sobierajski [12]** – the authors proposed a 3-DoFs haptic rendering method for volumetric data. They used the direct haptic rendering and haptic transfer functions (analogous to transfer functions for volume visualization). For haptic rendering of iso-surfaces penalty-based force computations were used. Forces were calculated at an update rate of 1-5 kHz. Additionally, it was possible to modify the data with simple "tools" (see figure 3.29), and such modifications were performed at a lower update rate.

  The maximum reported volumetric data set consisted of 256x256x225 voxels.

- **Foskey et al. [69]** – this paper is devoted to the system "ArtNova" for 3D model design with a haptic device. The system supports 3-DoFs haptic rendering, dynamic viewing techniques (see the paper for details) and allows a user to put colors and textures onto objects, and to deform them. An object is represented by several triangulated mesh levels at different resolutions, and it is allowed to edit any triangles of any level (triangles of other levels will be changed accordingly to the applied deformation). A simple spring-based force model was used for force feedback generation during mesh editing, and the generation of forces was

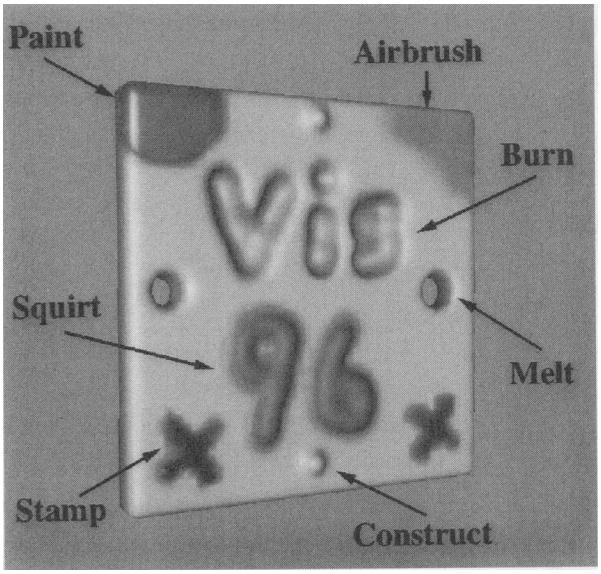


Figure 3.29: Effects of the "tools" for data modification in [12] (source: [12])

decoupled from the mesh editing. The force update rate was 1 kHz. The H-Collide library (see e.g. [82]) was used for collision detection.

Examples with one modelled object in the scene were presented in the paper.

- **Kim et al. [111]** – this method is described in section 3.2.1. It belongs to the current group of methods, because the authors additionally proposed a virtual sculpting prototype.

- **Cani and Angelidis [40]** – this work is devoted to virtual sculpting using a 6-DoFs haptic display. The authors proposed two modelling approaches:

  1. The representation of object's shape is a discrete field function stored in an adaptive hierarchiral 3D grid (i.e. an adaptive hierarchical voxel representation), and the surface of the sculpture is represented as an iso-surface in this grid. The data structure was implemented using hash-tables. An adaptive subdivision and undivision of objects is supported. For force feedback, the authors used a viscosity force model for object editing and a spring-damper force model for an exploration of object's surface. In order to increase the stability of haptic rendering, a device position filtering was introduced. Additionally, the authors proposed and realized number of modeling tools. The haptic rendering rate was not less than 1 kHz
  2. In order to support large deformations the authors proposed another approach, because the one above is limited to only local object modifications.The

authors used a volumetric data structure and a volumetric clay model, and the modelling approach has three "layers": "large scale deformations" (e.g. global bending and twisting), "volume conservation" (pushing the clay from the cells with a density above some threshold to their neighbours) and "surface tension" (moving the clay from the cells with a density value below some threshold towards the surface of the sculptured object). Additionally, a number of virtual tools for deformation modeling were realized, like swirling and aforementioned twisting and bending.

Note: The authors pointed out that the aim of their work was not to make a physically accurate modeling but to create a convenient system for artists. Therefore, all deformation computations for both approaches are not supposed to be physically accurate.

Among the presented examples for the second method, the maximum number of voxels was 40495. No exact quantative characteristics of objects for the first approach were given.

- **De et al. [56]** – this method is described in section 3.2.3. It belongs to the current group of methods, because topological changes for objects are allowed.

- **Vashisth and Mudur [222]** – this work is devoted to deformation of point-based models using an electronic force-feedback glove. The glove has 15-DoFs (3 for each fingertip).

  As far as objects are point-based, a meshless deformation technique based on the solution to the Kelvin problem from analytical physical mathematics was used for deformation and force feedback computations. Additionally, as a restriction, the tool should always be outside an object. No multirate approach for force calculations was employed.

  Remarks: Among drawbacks of the method, the authors mentioned that exaggerated deformations could lead to highly non-uniform density of points.

  The direct rendering technique was used. There is no information about performance of the system. Examples with one deformable object in the scene were presented. The authors reported about objects consisting of up to 60000 points.

- **Maciel et al. [135]** – This method is described in section 3.2.4. It belongs to the current group of methods, because topological changes for "cloth-like" objects

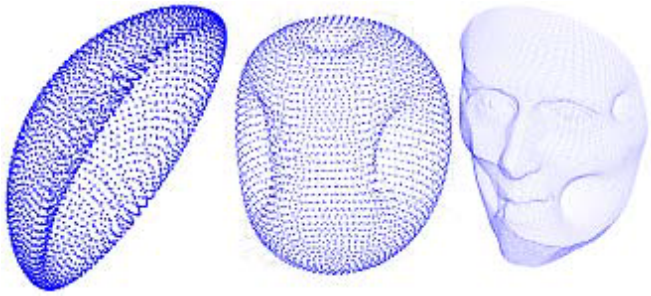


Figure 3.30: Deformation experiments on point-based models (source: [222])

are allowed.

3.2.3 Rigid-Defo Methods

Here we assume that the tool is rigid, but the environment is deformable. Up to now, there exist methods allowing only a certain degree of deformations, so that large deformations and cuts are mostly disallowed.

One can mark out the following methods belonging to the current group:

- **Debunne et al.** [57] – the authors presented a method for animating dynamic deformations of a visco-elastic object with a guaranteed frame-rate, built into a 6-DoFs haptic rendering framework.

  An object is represented via a tetrahedral mesh, and the proposed physical simulation approach belongs to physics-based continuous models. It is solved via an explicit finite element method and employs the Green deformation tensor in order to allow very large displacements.

  In order to achieve an adjustable fixed frame-rate, the authors presented an adaptive space and time resolution technique for tetrahedral object representation. They introduced a quality criteria that indicates where and when the resolution for qualitative deformation modeling is too coarse. See figure 3.31.

  Collision detection was made using graphics hardware, so that the tool was modeled as a viewing frustum intersecting a surface of the deformable object, and was performed only before each iteration of the visualization loop.

  Collision response consists of computing an object deformation based on the surface displacements imposed by the motion of the tool: affected surface points

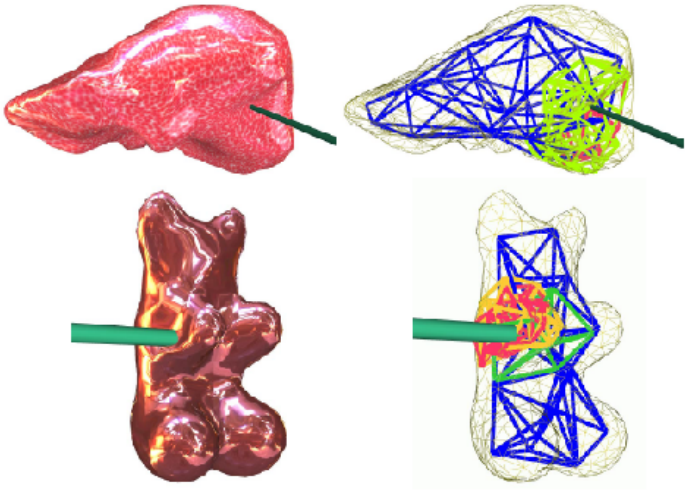


Figure 3.31: Use of local refinement technique in order to ensure the physical fidelity while bounding the global computation load in order to guarantee animations with the desired frame-rate (source: [57])

are moved away from the surface along their normal directions. These computations are performed at each haptic frame, and the motion of the tool is calculated gradually between the collision detections.

The force feedback generation was done as following:
At each haptic frame, nodes of the active mesh (the mesh used in the computations or affected by them), which are linked to the affected surface points, are moved. Then each moved node transmits the accumulated force to the linked surface point. Each such point sums these forces, and at the end the forces weighted by area are again summed up and returned to the tool.

The authors reported about a haptic update rate of approximately 1 kHz for a few hundreds surface points being animated. Interactions with only one object in the scene were presented in the paper.

Additionally, the authors mentioned that the method could theoretically be adapted for handling of topological changes.

- **Kuroda et al. [119]** – the authors proposed an interaction model between the interaction point and *two* physically-based deformable objects in the scene for 3-DoFs haptic rendering (therefore this is actually a *rigid-defo-defo* method). The model is applicable for simulations, where the interaction point pushes a deformable object being in contact with another deformanble object. The model allows to feel fine differences resulting from the physical behavior of neighboring

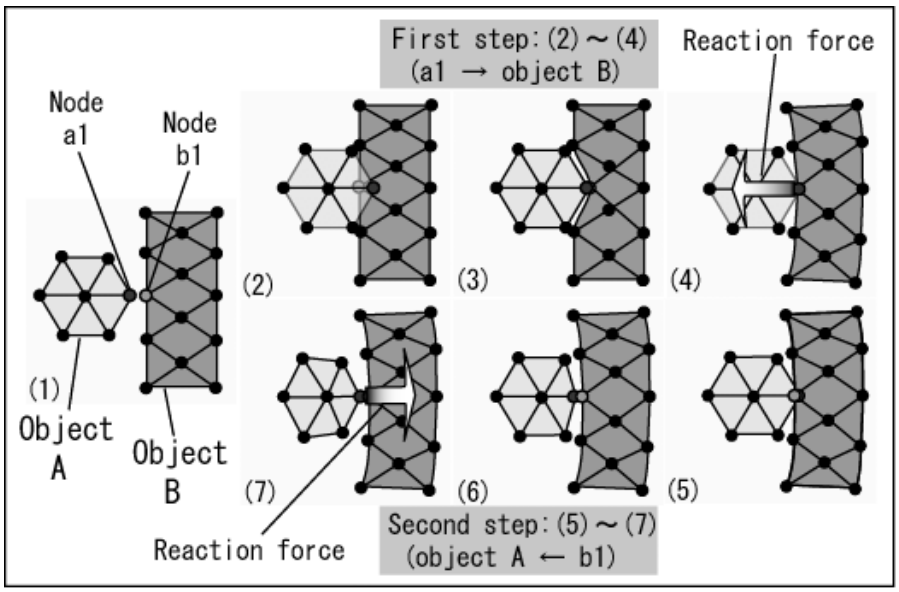


Figure 3.32: The first two series of example interaction: (2)-(4) – calculation of deformations of object B and the force conveyance, (5)-(7) – calculation of deformations of object A and the force conveyance (source: [119])

objects.

In the examples presented in the paper, the first model consisted of tetrahedrons and the second one – of hexahedrons.

The interaction between two models is represented by series of the following procedures: update of pairs of nearest nodes, collision detection, calculation of deformations, conveyance of the force. All procedures are carried out on pairs of nearest nodes, where each such pair consists of two nodes belonging to different objects (see the example in figure 3.32).

The collision detection looks as follows. It checks, whether there is a collision between the first node in the pair and the polygon including the second node of the pair. If so, then the polygon is displaced forcibly.

For the deformation computations, the authors applied the mass spring model to the "front object" (being directly pushed by the interaction point) and a simplified Finite Element Method (FEM) to the "behind object" (located behind the front one). The mass spring model is calculated by applying the Euler's method to the Newton's movement equation.

For the simplified FEM, the size of the stiffness matrix was reduced and a linear elasticity was used. Surface and interior nodes are distinguished. Surface nodes

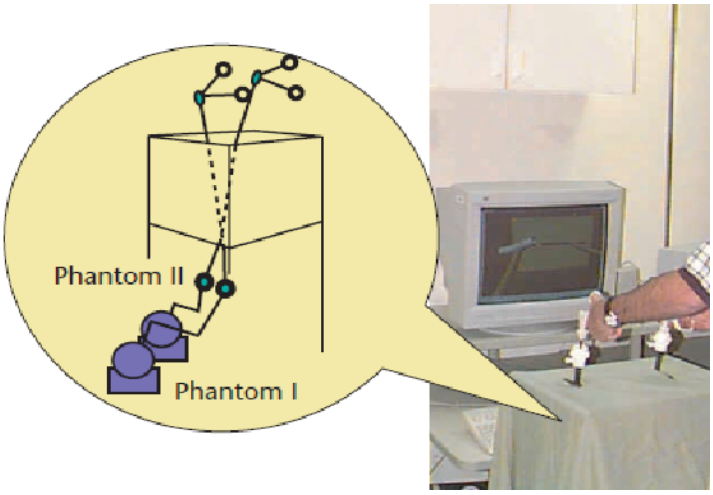


Figure 3.33: Laparoscopic training system from [19] (source: modified from [19])

are classified into fixed surface nodes and free surface nodes. The force of fixed nodes is assumed to be zero. Free nodes are classified into contact nodes and other nodes, and the solution for contact forces is obtained by solving a small system of equations, where the number of equations is proportional to the number of contacts.

The authors reported about haptic update rates from 285 Hz to above 1 kHz depending on the number of displaced nodes. The largest scene contains 158 tetharedrons for the first object and 1448 hexahedrons for the second one.

- **Basdogan et al. [19]** – this paper is devoted to (6-DoFs) haptics in minimally invasive surgical simulation and training. This method is more a specific case and a case study than most of other methods considered in our work.

  The authors proposed to represent surgical tools as points and lines, and therefore the point-object and line-object collision detection was used. In order to simulate soft tissues of organs, the authors developed a mesh-based finite-element model with assumed modeling simplification that high-frequency deformation modes contribute little to the overall computation of deformations and forces, and therefore dynamic equalibrium equations were transformed into a more effective form.

  In order to obtain material and geometrical properties of organs, the authors proposed to use a haptic device for recording a force and displacement response of soft tissues. Additionally, they presented special user interaction techniques based on force feedback for guiding a user during a training session.

  It was reported about a haptic update rate of several hundreds of Hz.

- **Sedef et al.** [**204**] – the authors proposed a numerical scheme for simulating linear viscoelastic tissue behavior using FEM, which is integrated with 6-DoFs haptic rendering.

  An object is represented as a tetrahedral mesh.

  For collision detection, the authors used the following two-step scheme:

  1. **if** the tool is outside of the given undeformed object **then** there are no collisions **else** step 2
  2. perform collision detection using a displacement history of nodes.

  In order to interactively calculate nodal displacements and interaction forces, i.e. in order to calculate collision response and generate force feedback, the authors took advantage of the linearity and superposition principle:
  Before interactive simulations, a response of each surface node to unit step force and unit step displacement are pre-recorded separately. During the real-time interactions the pre-recorded forces are scaled by a penetration amount in order to calculate the reaction (interaction) forces, and the pre-recorded displacements are superimposed in order to calculate the nodal displacements.

  In order to obtain material properties of simulated object, the authors developed a robotic indenter for minimally invasive measurement in living body of tissue properties during the laparoscopic surgery. Measurements of the pig's liver were presented in the paper.

  The authors reported about 100 Hz for deformation computations and 1 kHz for force feedback. The sample object consisting of 136 tetrahedrons was used for the system validation.

  The authors used constant force feedback between iterations of the deformation computations, what decreases a haptic rendering realism.

  The deformation model was validated with ANSYS (see [204] for details), and the validation tests "matched perfectly". The authors mentioned that a user can even feel a relaxation behaviour of a simulated object via a haptic device when he/she penetrates into it with the tool and stays at a certain depth for a while.

- **De et al.** [**56**] – the authors proposed a method for "physically realistic" virtual surgery, which uses a Point-Associated Finite Field (PAFF) approach. Deformations and topology modifications of objects (except for the tool) are allowed. This method is more a specific case and a case study than most of other methods considered in our work.

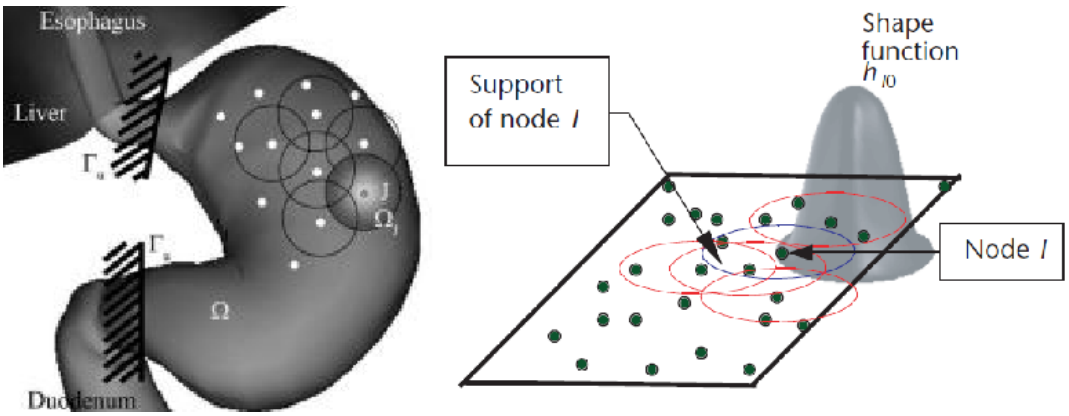


Figure 3.34: (*left*) Discretization of the stomach. (*right*) Support (influence zone) and shape function of the node I (sources: [56, 19])

The idea of the proposed mesh-free solution is to discretize a computational domain (an organ) using a scattered set of points ("nodes") with spherical influence zone with defined nodal shape function.

The approach is a combination of mass-spring and FEM-based techniques: it is meshless like a mass-spring technique and it solves governing partial differential equations as a finite element technique. The approach supports the linear elastic tissue response. In order to obtain a "real-time performance" of deformation computations, the authors used key assumptions that any interaction between the surgical tool and a soft tissue is local, and that a deformation field fades rapidly with increase in distance from the tool tip. Based on these assumptions, the authors presented two techniques:

- *Real-Time Global PAFF (GPAFF)* – here it is assumed that a prescribed boundary condition changes on a very small portion of boundary, where the surgical tool interacts with a virtual organ. Therefore it is possible to make incremental corrections to the previously computed solution. In order to obtain a real-time performance, precomputations of global linear stiffness matrix, its inversion and application of fixed boundary conditions are made in the pre-processing step
- *Real-Time Local PAFF (LPAFF)* – a local discretization is performed using only a group of nodal points traveling with the tool tip.

Additionally, a smoke generation using PAFF was proposed. It can be used e.g. for smoke simulation during cauterization.

A multirate architecture being used in the system is shown in figure 3.35.

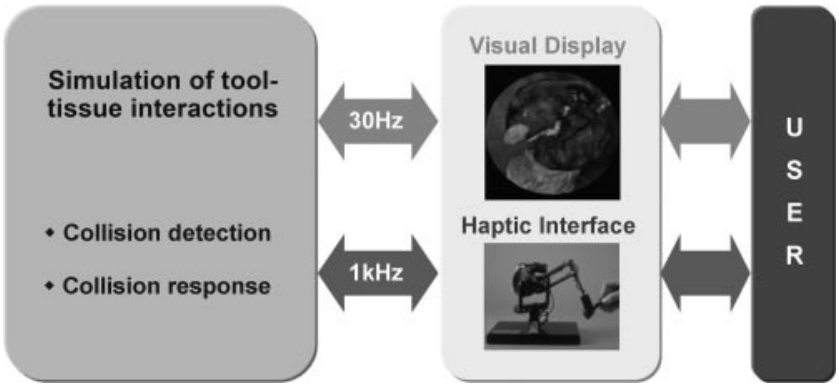


Figure 3.35: Multirate architecture for [56] (source: [56])

For collision detection and collision response, the authors used a point-based representation of the surgical tool (only one point at the end interacts with an object, therefore 3-DoFs haptic rendering is actually supported) and a bounding box hierarchy with the local neighborhood search algorithm. The force field is calculated from the deformable field and is used for the generation of force feedback.

Examples with the tool and one other object in the scene were presented in the paper. The authors reported about maximum 1026 nodes and 9 ms for the GPAFF approach and 28 nodes and 14 ms for the LPAFF approach. The timings were given for one simulation iteration. It was mentioned that maximum 1080 polygons per simulated object were used for the GPAFF and 1364 for the LPAFF.

Since collision detection and collision response (both in the fast thread) are decoupled from the simulation (the slow thread) and no interpolation between consequent slow simulation iterations is used in the fast thread, force feedback discontinuities could arise at the time when new results come to the fast thread from the slow one. This issue is generally equal to the one in [3] (see section 3.2.1).

- **Otaduy and Gross [171]** – the authors proposed a 6-DoFs haptic rendering method supporting the rigid tool and deformable environmental objects. This work is an evolution of the rigid-rigid system from [175, 176] (see section 3.2.1).

  Deformable objects are represented by tetrahedral meshes.

  The authors adopted continuum mechanics in order to simulate deformable models and opted for corotational FEM methods with linear elasticity for modeling deformable objects from [153].

  The multirate architecture being used in the system is shown in figure 3.35.

  The authors used a continuous collision detection from Redon et al. [188], collision

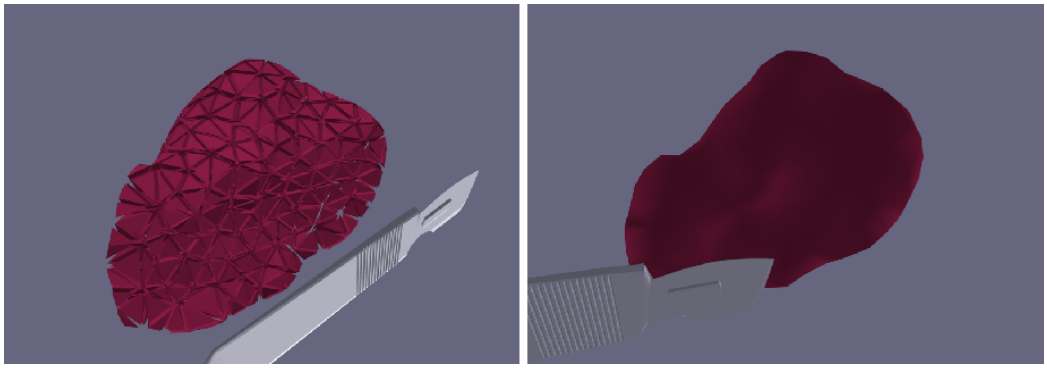


Figure 3.36: (*left*) Tetrahedral decomposition of the liver model and (*right*) the liver model being deformed (source: [171])

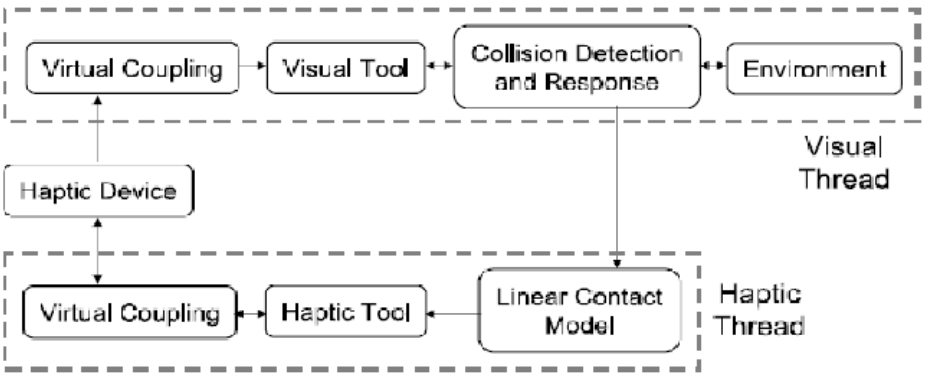


Figure 3.37: Multirate system architecture for [171] (source: modified from [171])

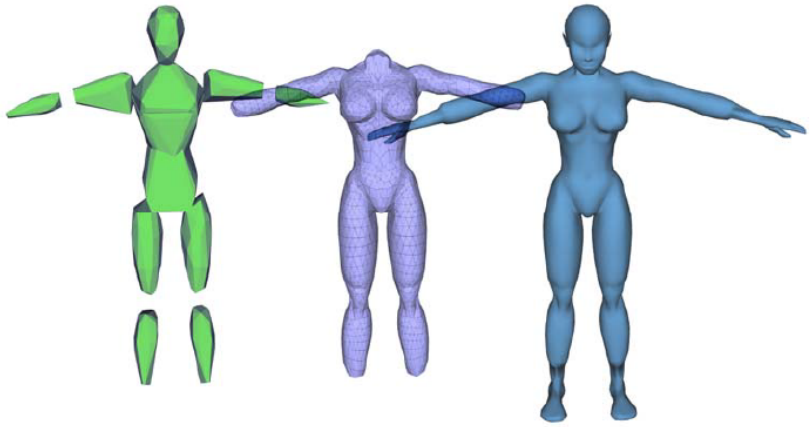


Figure 3.38: Layered representation of an object in [75]: (*left*) low-resolution proxies (meshes) used for collision detection and haptic interaction; (*middle*) deformable tetrahedral mesh; (*right*) highly detailed surface mesh for the deformable skin simulation (source: [75])

response through velocity constrains and a linear contact model for force feedback in the haptic thread. The haptic thread works at 1 kHz and the visual thread at several tens of Hz.

Examples with the tool and one other object in the scene were presented. Number of polygons/tetrahedrons were given for only one scene: 160 triangles for the tool and 2560 tetrahedrons for the deformable object.

- **Galoppo, Tekin, Otaduy, Gross and Lin [75]** – the authors proposed a 6-DoFs haptic rendering method supporting the rigid tool and deformable environmental objects.

  Remarks: Examples with only one deformable object in the scene were presented in the paper.

  As [171], this work is an evolution of the rigid-rigid system presented in [175, 176]. Additionally, it shares some ideas from [171].

  The object representation is shown in figure 3.38.

  The multirate system architecture is an evolution of the one from [175, 176] – see figure 3.39.

  The authors proposed an image-based three-step algorithm for collision queries:

  1. Identify potentially colliding contact patches using low-resolution proxies

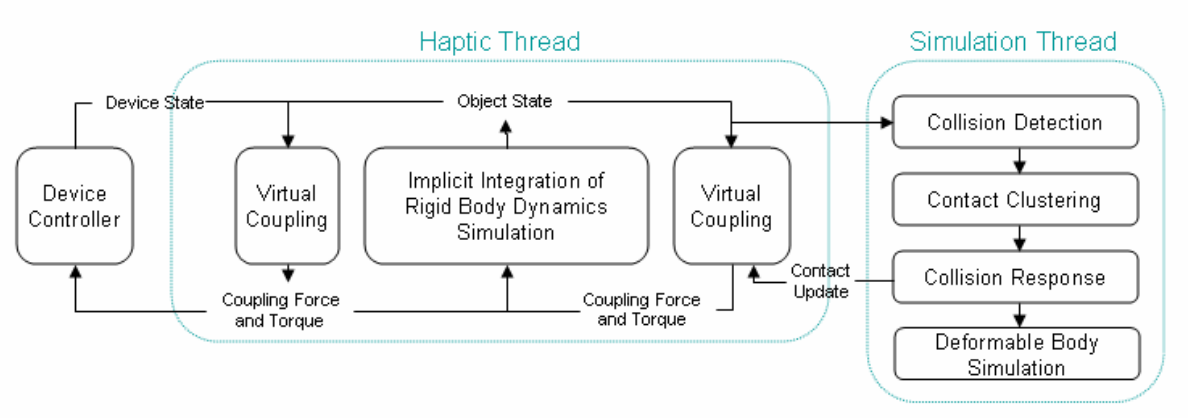


Figure 3.39: Multirate system architecture for [75] (source: [75])

2. Compute localized penetration depth fields
3. Get high-resolution skin surface collisions and directional penetration depths.

The last two steps are performed using image-space techniques with the aid of graphics hardware, and the GPU approach itself is based on the one from [172] and [177] (see a brief description of the last paper in section 3.2.1).

The authors simulated a deformable material with a rotationally invariant FEM simulator with implicit integration guaranteeing stability and proposed penalty-based collision response using the spring-damper model between vertices of deformable tetrahedral mesh and the penetration depth.

It was mentioned that the layered representation poses some limitations on type of deformations, which can be modeled: an object could deform only up to 30-40% of its radius.

In presented examples the authors used objects consisted of a low-resolution proxy of few hundreds triangles and high-resolution tetrahedral and surface meshes with up to 44k deformable vertices.

- **Barbic and James [16]** – the authors proposed a CPU based approach for 6-DoFs haptic rendering supporting a contact between rigid and reduced deformable objects, both with complex geometry. A distributed multi-point contact between objects is allowed, i.e. an interaction with potentially several simultaneous contact sites each distributed over a non-zero surface area. This work is based on the PhD thesis of Barbic [15].

  Remarks: Each mesh vertex of a *general 3D deformable object* has 3-DoFs. *Reduced deformable objects* are obtained by substituting these general DoFs for a

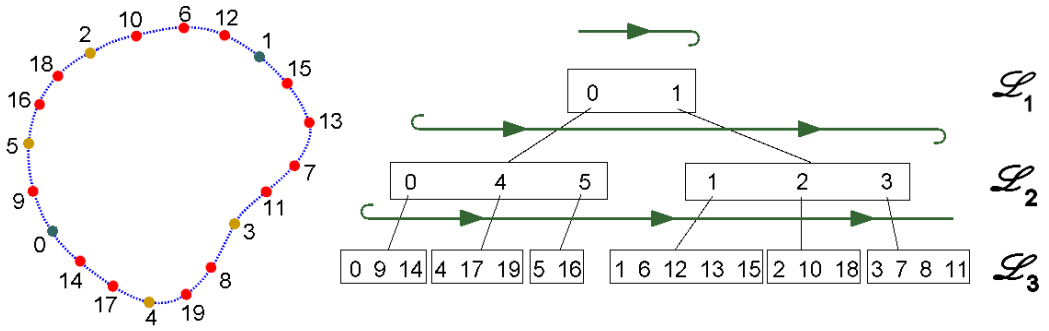


Figure 3.40: Nested point-tree: (*left*) the multi-resolution pointshell and (*right*) the hierarchy, the traversal order and tree levels. Particle-repulsion levels are 0-1, 2-5, and 6-19 in this case (source: [16])

much smaller appropriately defined set of reduced DoFs. This allows to perform faster computations.

The approach was designed to work with a variety of reduced deformable models, which support the two-step simulation process: (1) the fast timestep of reduced deformable dynamics and (2) the fast evaluation of individual deformed surface point positions and normals in order to adaptively resolve the contact (for the time-critical force estimation). The authors used techniques for reduced geometrically nonlinear models from Barbic and James [17] being suitable for large deformations with large rotations but small local strain. The techniques are based on the formal reductions of large-deformation FEM models.

The authors proposed to use the point-based representation (pointshell) for the first object (deformable one) and the signed-distance field for the second one. In order to support haptic rendering of geometrically detailed models (1M points), the pointshell is organized into a nested multi-resolution hierarchy (see figure 3.40) by sampling points and normals from the given closed manifold oriented surface of the object (i.e. the pointshell actually samples the surface). Point positions are being generated by fitting a set of particles onto an offset surface (i.e onto one being "larger" than the original one) employing ideas of particle repulsion.

The pointshell resolution should be equal or finer than the one for the distance field. Deformed point locations are approximated by linear superposition of precomputed displacement matrices. The authors proposed to use the precomputed sphere-tree hierarchy in order to bound pointshell points (sphere centers are located at the centers of the points). In fact, all together gives a point-based modification of the Bounded Deformation Tree (BD-Tree) from [102]. Pointshells and distance fields

are computed in the pre-processing stage.

Additionally, a spatial adaptive approach, called "graceful degradation" of contact, was introduced: if there is not enough computation time for fully completing the tree traversal then the algorithm will still return a reasonable answer with accuracy dependent on the contact-configuration difficulty and the available processing power. The graceful degradation is achieved by traversing the nested hierarchy in the breadth-first order and rendering deeper and deeper tree levels until out of computation time. Two separate activation thresholds were employed in order to avoid a "gap" in the rendered depth for consecutive haptic frames. Furthermore, a temporal coherence technique for timesampling of individual points at update rates depending on distance to the contact, was proposed.

The authors employed penalty-based (i.e. penetration-based) spring-based contact force computations. Contact penalty forces are determined by querying points of the pointshell object against the one represented by the signed distance field (the last one is manipulated).

The haptic cycle looks as follows: firstly read the position and orientation of the haptic device, then compute contact penalty forces and torques by traversing the nested point-tree and compute virtual coupling forces and torques between the tool and the probe, and then calculate gradients for all of them with respect to the simulation position. Further, in order to find the displacement of the simulated object, the system of equations for the condition that the sum of all forces and torques vanishes is solved. Finally, force-feedback is computed using the result of the previous step.

In order to increase the haptic rendering stability, the authors introduced the maximum velocity and the maximum angular velocity, which can occur in a simulation. Additionally, the maximum contact stiffness, which is defined as the largest increase in coupling force per the given tool's displacement so that it is linear only up to the certain displacement and saturates to the certain value after that, was introduced. Similar maximum values and behaviour were suggested for torques. Furthemore, the authors mentioned that thre lack of dissipation in the standard virtual coupling model can lead to slight instabilities, e.g. during a fast sliding contact. In order to partially neglect them, the authors augmented the virtual coupling by introducing a quasi-static damping: a tool's displacement is applied with the damping factor.

Examples with the tool and one environmental object were presented in the paper. The authors reported about a haptic update rate of more than 1 kHz. For rigid-

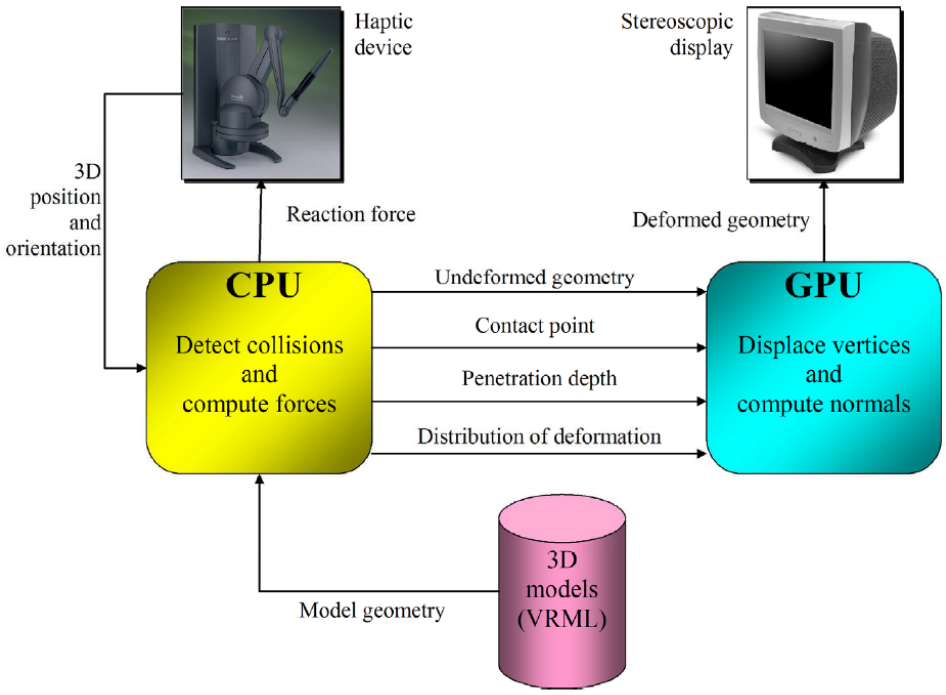


Figure 3.41: Architecture of the system from [130] (source: [130])

rigid contacts, the maximum characteristics of the scene are 1.02M points for the environmental object and the distance field resolution of 256 for the tool. For rigid-defo contacts, that is 256k points and the resolution of 256 respectively.

Among drawbacks of the method, the authors mentioned that self-collisions were unadressed in the system.

- **Luciano et al. [130]** – the authors proposed an approach for haptic rendering of elastic deformable objects using GPU.

  The scheme of the system is shown in figure 3.41.

  The authors pointed out that the paper is focused on the point-based local deformation around the contact point, therefore a global deformation and volume preservation were beyond the scope of research. Additionally, they mentioned, that the algorithm can be thought of as a trade-off between real-time interaction and sophisticated physics-based realism.

  Since it was necessary to compute a deformation at the contact point and its neighborhood, the authors proposed to employ a GPU in order to displace each vertex in parallel. In more detail, vertices are moved along their normals in order to deform the object's surface: the maximum displacement is found at the contact

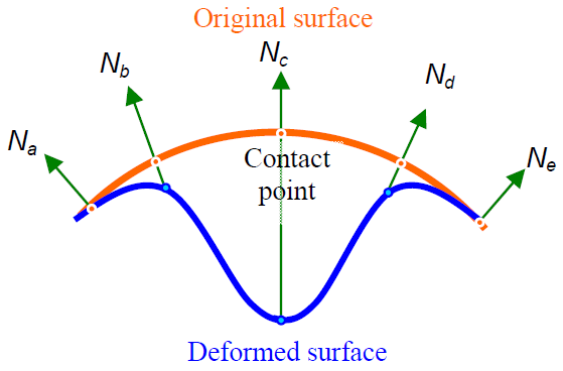


Figure 3.42: Vertex displacements along their normals (done by the vertex shader) (source: [130])

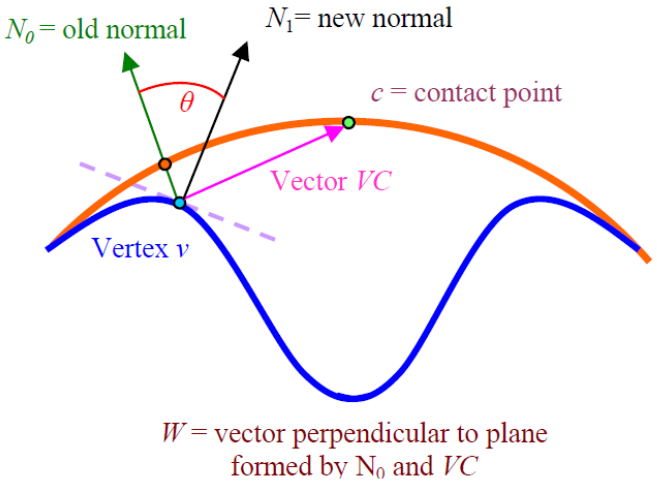


Figure 3.43: Normal calculation (done by the fragment shader) (source: [130])

point, and then it decreases non-linearly as vertices are located farther away (see figure 3.42). The computations are performed in the vertex shader.

In order to achieve realistic graphics rendering of the deformation, it is also necessary to re-compute the normals of displaced surface vertices, because lighting depends on them. The authors proposed to perturb old normals in the fragment shader in order to reflect changes of the deformed surface. The idea is to rotate the original normal N_0 at every displaced vertex v towards the contact point c by a certain angle Θ around the rotation axis W (see figure 3.43).

The authors reported about haptic rendering of polygonal iso-surfaces previously extracted from the 3D volume, created from the CT scan data of the real patient. The largest scene consists of an object with 59063 vertices and 53021 faces. Haptic rendering for all examples was performed at about 1 kHz.

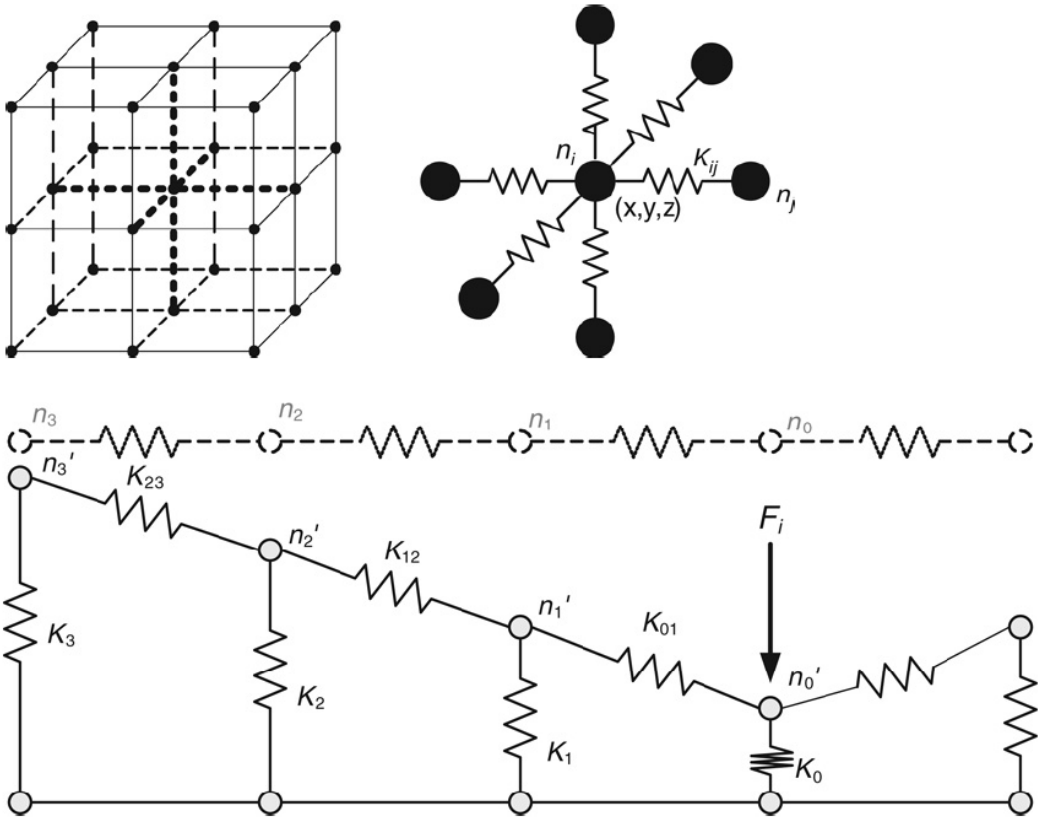


Figure 3.44: Mass-spring model (source: [42])

Among drawbacks of the method, authors pointed out that since collision detection and the computation of forces are done with original (undeformed) geometry, the approach cannot be extended to plastic (i.e. permanent) deformations.

- **Chang et al. [42]** – the authors proposed a 6-DoFs haptic rendering approach using the mass-spring simulation model and stated about two features of concern in their work. First, it was advantageous to enable an economical and simple implementation with a generic customer computing environment and a standardized haptic device. Second, a balance between the computation complexity and the level of realism had to be maintained.

  The mass-spring scheme was used as a dynamic model of virtual objects. A deformable soft tissue is composed by discrete nodes connected via springs, i.e. a virtual object is considered as a collection of spring-connected point masses in a grid mesh structure (see figure 3.44). A dynamic motion of node is described by Newton's and Hooke's laws.

  The mass-spring representation of objects is built from the given source 3D volume

data.

The penetration-based collision detection and the penetration-based spring-damper-based force-feedback calculations were employed in the system.

The authors showed an application of their work for the brain surgery simulation and reported about a haptic update rate of 1 kHz.

- **Böttcher et al. [32], [30]** – this work is based on the PhD thesis of Böttcher [31] and was a part of the HAPTEX EU-project - Haptic Sensing of Virtual Textiles. The work is devoted to a kinesthetic haptic rendering of virtual fabrics grasped by two fingers. The fingers were represented via spherical tools manipulated by two 3-DoFs probes, while the simulation of tactile perception was proposed by Allerkamp et al. [10], [8].

  An elongation of body was expressed in strains, and a motion in terms of displacements with regard to the initial state. The Kawabata evaluation system was used in order to obtain physical properties of fabrics.

  For the physical simulation, the numerical solution includes:

  – discretization of a textile into nodes and elements (triangles). I.e. the triangle mesh representation was used
  – representation of strains and stresses in elements
  – condensation of element mass into nodes.

  The second order ordinary differentional equation was obtained, and the numerical integration yielded a sparse matrix. Then the problem was iteratively solved by the CG (conjugate gradient) method. A nonlinear anisotropic tensile and bending behaviour was modelled by linear elements, where the bending elements were associated to the particles.

  For collision queries, AABB bounding volume hierarchy was used.

  As far as the proposed global physics simulation was not able to function at a haptic update rate of at least 1 kHz, a multi-resolution computation model with a finer mesh resolution near the contacts with the tools was proposed, and the multirate system architecture was used – see figure 3.45. The local simulation model includes:

  – generation of the refined mesh at the contact
  – provision of a contact geometry for the two-finger contact model (see details below)

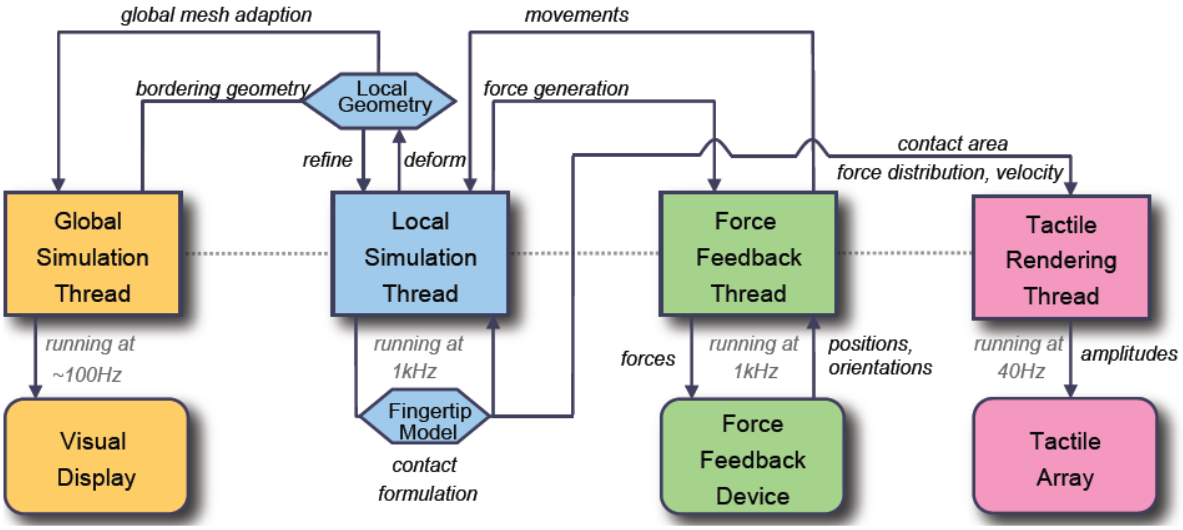


Figure 3.45: Multirate multithread architecture used in [30, 31] (source: [31])

- simulation at haptic real-time (not longer than 1 ms per time step)
- constraining of the system at borders
- use of the same physical model as in the global simulation.

The idea of the two-finger model (the ability to feel fingers to be in contact with each other) was to use evenly distributed springs each one allowing a stick-slip transition for modelling of the contact pressure (see figure 3.46). Additionally, the ends of the springs provided the contact information for the tactile feedback.

For the force feedback transmition, a special device – GRAB Force-Feedback Device – was used. The device has two 3-DoFs probes, each of them supporting tactile feedback.

Further on, the authors proposed a run-time control technique for local simulation, which allows to effectively use free CPU time for calculations and which ensures the response at haptic times (less or equal to 1 ms). In more detail, as far as an iteration of the force feedback thread always takes 1 ms, there could be some "unspent" CPU time in case the last iterations took less than 1 ms each one. The idea of the run-time control technique is to use this free CPU time by adapting simulation parameters in order to make more iterations per ms or perform higher quality (therefore longer) simulation. On the other hand, if the last iterations took longer than 1 ms each then the control algorithm will change simulation parameters in order to speed up the computations. The duration of the time step is calculated using spent time prediction capabilities based on the number of

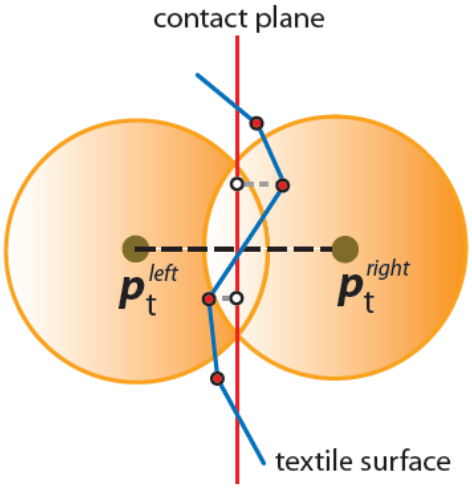


Figure 3.46: Two-finger contact model in [30, 31] (source: [31])

particles, number of faces, number of bend elements and maximum CG steps, and then the parameters are changed accordingly. In [33] Boettcher et al. generalized this multi-rate coupling scheme of physical simulations for haptic interaction with deformable objects.

As far as the run-time control technique was used, a haptic update rate was always about 1 kHz. 480 triangles were used for the global simulation of fabrics, and mostly about 128-160 triangles and 10-12 CG iteration steps for the local thread (due to the run-time control algorithm, the last numbers were dynamically changed).

3.2.4 Defo-Defo Methods

Methods in this group are similar to the methods from section 3.2.3, but a deformable tool is allowed. Up to now, the tool generally has the same deformation restrictions as those for the environment.

One can mark out the following methods belonging to the current group:

- **Duriez et al. [63]** – this work is devoted to the Signorini's contact model for deformable objects in haptic simulations and is focused on contact response. The approach belongs to approaches with non-penetration constraints. It is an extension of Signorini's theory [209] (1933) on rigid-deformable contacts to contacts between deformable bodies.

The approach is time-stepped (constant timestep integration), and the formulation is made to be independent from a collision detection technique in order to be as generic as possible. Only two data are needed:

1. directions of the contacts
2. the spots of the contacts.

Remarks: The authors mentioned that a proximity detection method is more suitable than just a simple collision detection. They used a proximity distance algorithm for their experiments.

The authors considered that objects are perfectly elastic and isotropic and that there are only frictionless contacts in the system. Under these conditions, the normal surface stress is an unknown for the Signorini's problem, and the shearing surface stress is zero because of no friction.

In Signorini's formulation, for every point in the defined proximity to another object, two states may be distinguished:

- the point is actually a contact point
- the point is not yet a contact point.

(This is why a proximity detection approach is more suitable than a simple collision detection.)

The authors used a finite element discretization to solve the problem and employed linear interpolation functions using tetrahedrons with four nodes. Contact points are necessarily on the surface of the objects. To linearize the problem, each contact's direction is frozen during the current time step. But with linear elements, only one point of contact allows integrating the pressure force on the surface. Therefore, in order to be able to distribute the maximum of collision tests between the elements, the authors used an algorithm close to the Gauss-Seidel method for contact resolution, whose principle is to visit every contact considering that the states of all others are frozen.

In order to reach a desirable speed of calculations, coarse tetrahedral meshes were often used for FEM deformable models and special surface meshes with more triangles were used for collision detection. Interpolation of each vertex of the collision mesh within its corresponding element of the FEM mesh was computed off-line.

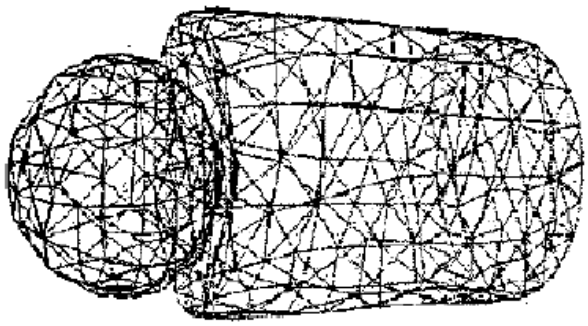


Figure 3.47: Example from [63]: a deformable ball inleracting with a deformable cylinder (source: [63])

Examples with a deformable tool and one deformable object were presented. A haptic update rate was about 160-200 Hz for 70 simultaneous contacts.

Among drawbacks of the method, the authors mentioned that different LODs for tetrahedral and surface meshes for the same object lead to non-regular contact surface between models and perturbs haptic feedback.

- **Duriez et al. [64]** – this work is an evolution of [63].

  The authors incorporated the dry friction based on Coulomb's law into their algorithm. This nonlinear law describes two states on the tangential contact space: stick and slip. The law is difficult to solve correctly in the considered multicontact context, because it is not possible to separate its computations from contact calculations: each contact's force can modify the state of other contact spots through the tangent space, which is also coupled, locally, to the normal one. (Though fast and precise solutions for a single contact case were proposed by different authors.) Computation of the contact and friction force also takes into account user defined material and structural properties of the contacts, and each contact may also be linked to others.

  The presented method is still independent from a collision/proximity detection. The detection algorithm should only identify potential contacts between a pair of triangulated bodies and provide:

  1. two contact points
  2. their positions within the contacting triangles
  3. the contact normal. If it is not provided then the normalized vector from the second point to the first one is used.

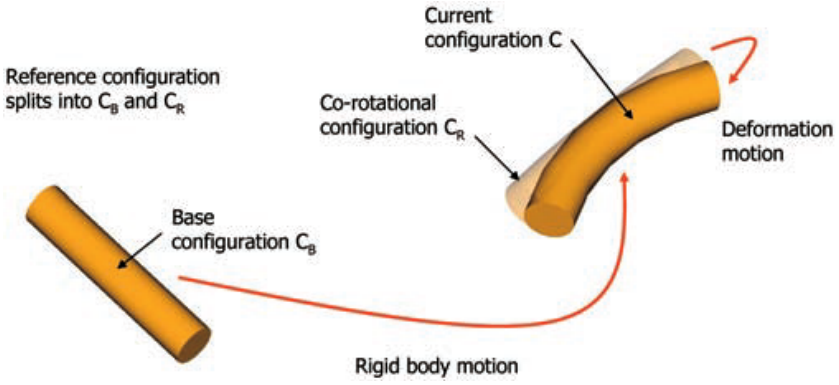


Figure 3.48: The motion of a deformable object split in two parts: a deformable motion in its current configuration and a rigid motion in the world coordinate system (source: [64])

(The authors used stochastic proximity detection algorithm for their experiments.)

Deformation computations in case of no friction are as in [63]. In case of friction, the authors incorporated the Coulomb's law into their Gauss-Seidel-like algorithm: for each contact the contact and friction laws are solved by considering a contribution of other contacts to be frozen. Additionally, the computation of the contact and friction force takes into account user defined material and structural properties of contacts, and each contact may be linked to others.

For force feedback, the authors used a global corotational approach that decouples a rigid global motion from a deformable one. It splits the global transformation (driven by a rigid model) from the local relative displacement (driven by the linear deformable model). See figure 3.48. The virtual coupling technique between the probe of the haptic device and the rigid part of the corotational model for the manipulated virtual object is then used for the calculation of the forces returned to the user.

The authors mentioned that the method was developed considering several deformable objects moving randomly and coming into contact with each other. Two examples, with defo-rigid and defo-defo contacts respectively, were presented. In both examples there are two objects in the scene, and a user manipulates the deformable one or both of them using 6-DoFs haptic displays (one device per object was used in the last case).

A reported average haptic frame rate is about 330 Hz for 30 simultaneous contacts without friction and about 250 Hz for 20 simultaneous contacts with friction.

- **Barbic and James [18]** – as [16], this work is based on [15] (see section 3.2.3).

Figure 3.49: The interactive snap-in and snap-out task on deformable pipes from [64] (source: modified from [64])

The authors added a support for the reduced deformable tool by employing parametrically deformed distance fields. Deformations are assumed to be "reasonably coarse" (low frequency).

Specifics for the distance query computations for a deformable distance field compared to those for underformable one are the following:
During the pre-processing, a small pointshell (typically about 40 points) is fitted onto the surface of the distance field object. The authors called this pointshell *proxyshell*, and the points *proxies*. Further the proxies deform together with the object.

In order to evaluate the deformed distance field at some query point location $\mathbf{x}$ for the pointshell-based object, the following steps are performed (see figure 3.50):

1. At first, the k-nearest neighbor search is performed in order to locate k current closest proxies (typically k=5)
2. Then local first-order deformation model at each of the k found proxies is used in order to generate k approximations to the deformed distance field at $\mathbf{x}$. This is done by using precomputed deformation gradients and k unknown points in the vicinity of the undeformed positions of the found proxies (one point per proxy)
3. Then, for each of the found approximations from the previous step, their approximation equations are inverted in order to find approximations of $\mathbf{x}$ in

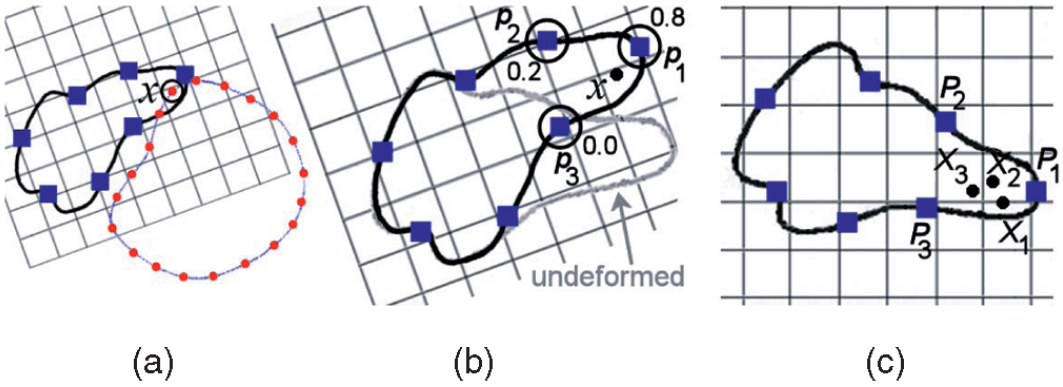


Figure 3.50: Approximation of deformed distance field for k=3. (a) Proxies (squares) and the query pointshell point at $\mathbf{x}$. (b) Three-nearest neighbors. (c) k approximations of $\mathbf{x}$ in the undeformable distance field (source: [18])

the undeformed distance field. (Informally, it is like pulling points from the deformed distance field to the undeformed one)

4. Then k distances for the found approximations are obtained by looking up the undeformed field
5. The final result, i.e. the value for the distance query for the *deformable* distance field for the point $\mathbf{x}$, is computed as the weighted sum of the k distances calculated in the previous step.

The authors reported about a haptic update rate of "more than 1 kHz". Examples with the tool and one environmental object were presented in the paper. The largest "defo-defo" scene consists of 256k points for the deformable environmental object and the deformable tool with a distance field resolution of 256.

It was mentioned that self-collisions were unadressed in the system.

- **Garre and Otaduy [77]** – the authors proposed a 6-DoFs haptic rendering method, where both the tool and environmental objects could be deformable. The work is an evolution of the previous works of Otaduy and others [175, 176, 177, 171, 75], employs some ideas and concepts from them and has similar drawbacks.

  Remarks: Examples with a deformable tool and one deformable object in [77] and up to three deformable objects in [78] were presented.

  Objects are represented by deformable tetrahedral meshes for simulation and polygonal meshes for visualization and collision detection. A handle of the tool is assumed to be rigid. Any area of an object could be selected as the handle.

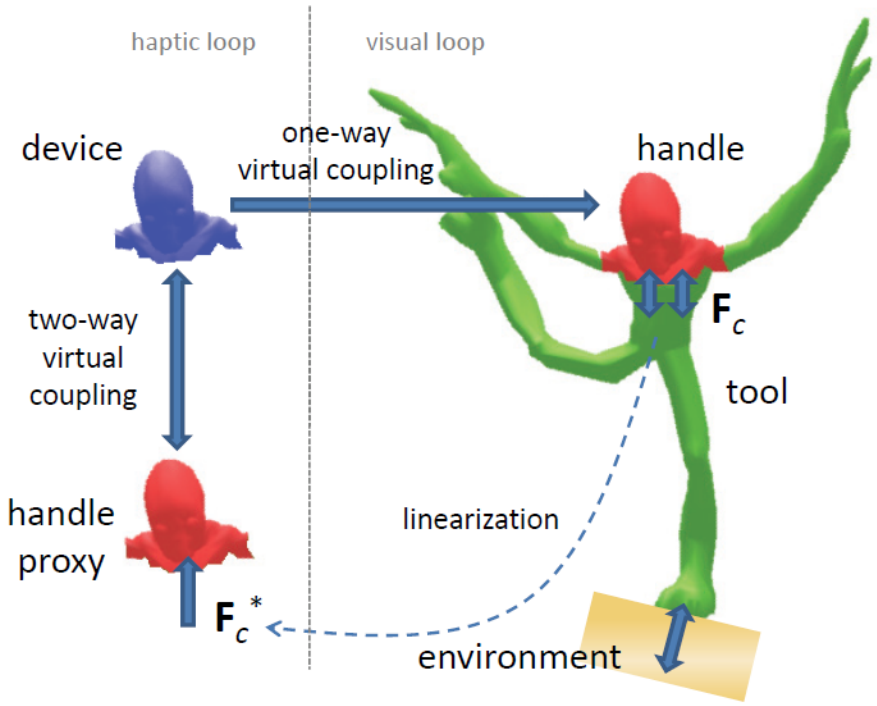


Figure 3.51: In each simulation step in the visual loop, a linear approximation F_c^* of the coupling force F_c between handle and the rest of the tool is computed, that encapsulates the constrained dynamics of the tool (source: [77])

The proposed multirate haptic rendering architecture is shown in figure 3.51.

The authors suggested to use one-way probe-handle virtual coupling, linearized "handle–handle proxy" coupling (*handle proxy* means "virtual proxy" in terms of [198]) and virtual coupling between the handle proxy and the probe (see figure 3.51). The constrained dynamics model from Otaduy et al. [169] was used for collision response and a co-rotational finite elements model was used for simulation of dynamic deformations (as in [171]).

An interesting use case of the proposed system is haptic rendering of hand touch in **Garre and Otaduy [78]**. The idea is that a user can manipulate the virtual hand via the rigid handle being a part of it (see figure 3.52).

Among the examples presented in [77], the tool was represented by maximum 281 tetrahedra, and no concrete information about the tool's surface mesh was given. For [78], the hand was represented by maximum 1700 tetrahedra for deformation computations and 1733 triangles for collision detection, and the environment by maximum 271 tetrahedra and 4000 triangles respectively.

- **Maciel et al. [135]** – the authors proposed 6-DoFs haptic rendering for physics-

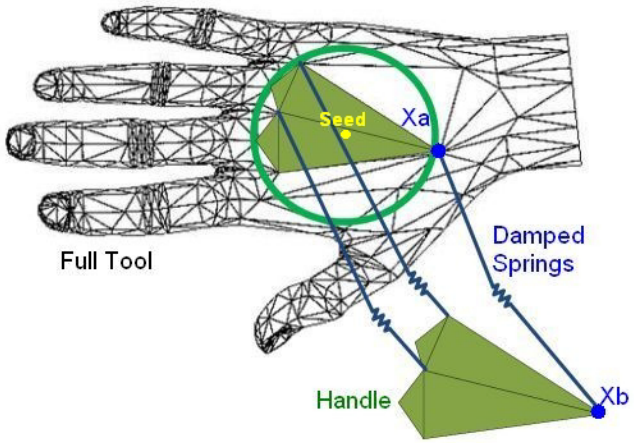


Figure 3.52: The rigid handle (in green) is selected as a part of the hand. Connections between the probe, handle, tool and handle proxy are equal to those in [77] (source: modified from [78])

based virtual surgery using NVIDIA's PhysX physics liblary [165], which is GPU accelerated. The proposed haptic rendering system supports rigid tool and deformable and rigid environmental objects, which could interact with each other, move and rotate. Additionally, "cloth-like" objects, for which it is allowed to change topological structure, and objects with joints are supported. Fluid objects are theoretically supported – no examples were presented in the paper. There are no limitations to make the tool deformable, "cloth-like" or with joints.

Deformable objects are assumed to be isotropic and homogeneous. Actually, the capabilities of the system are the capabilities of PhysX for the time when the paper was written.

The source data for PhysX is a surface mesh, and a tetrahedral one is generated based on it. Deformation computations are performed on the tetrahedral mesh, and the surface mesh is updated accordingly.

The proposed system architecture employs an extended Model-View-Control design pattern [76] and is shown in figure 3.53.

The PhysX's collision detection was used for all cases except for the "soft body-soft body" one. The last case was not efficient in PhysX, and therefore a method from Maciel et al. [134] was employed for it. Force feedback was calculated based on collision detection and collision response and updated at a rate of about 500 Hz.

The authors reported about an application of their system for the laparoscopic adjustable gastric banding (LAGB) case study and presented an example with

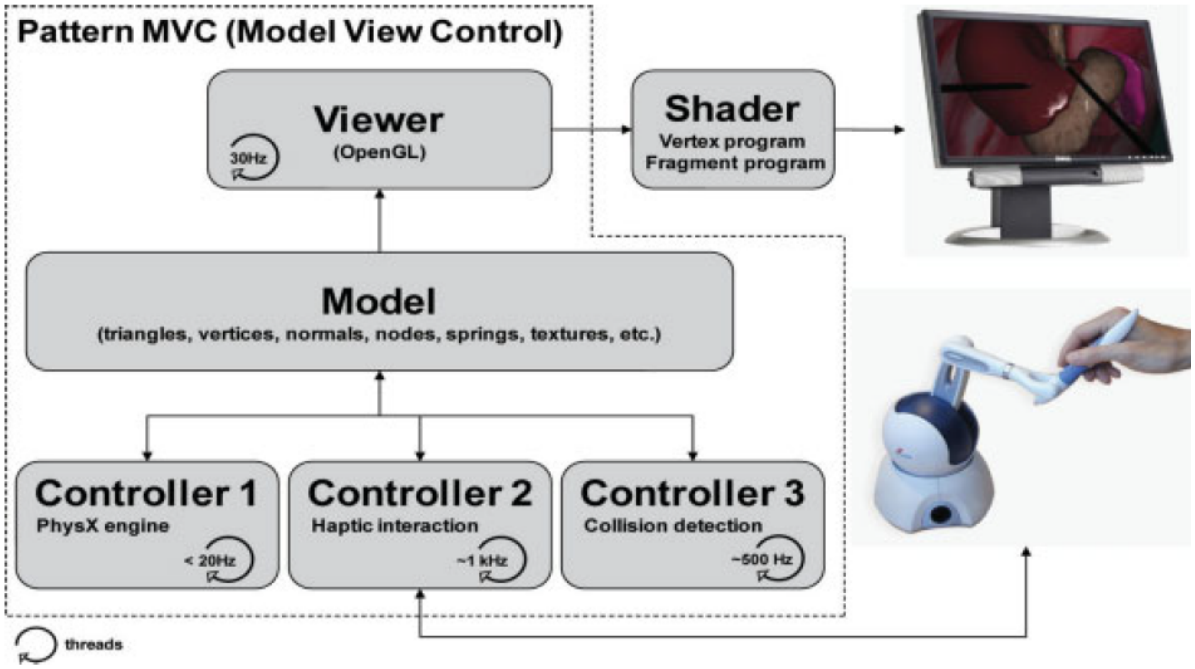


Figure 3.53: System architecture used in [135] (source: [135])

the rigid tool and a few rigid, deformable, "cloth-like" and jointed objects in one scene. The total number of triangles for all objects except for the tool was about 5800. The number of triangles for the user controlled surgical tool was not given. The liver and the stomach were represented by 3000 tetrahedrons each one (for PhysX deformation simulations).

Number of tetrahedrons for object in another example – a comparison with FEM – was 3901.

The authors mentioned that one of the drawbacks of their system is that the PhysX source code is closed and the API does not allow an integration of custom algorithms. We would also like to draw attention to other drawbacks. As far as the feedback force is updated at about 500 Hz but not at 1 kHz, force feedback discontinuities could arise in case of a fast probe motion. Additionally, the physics simulation is performed at only approximately 20 Hz, and no interpolation between consequent iterations of this thread is used for collision detection. This means that strong force feedback discontinuities could arise at the moment when results of the iteration from the slow physics simulation thread are transmitted to the fast collision detection thread. This problem and its cause are generally equal to those in [3] (see section 3.2.1).

3.2.5 Summary

There exist many different surface-based and voxel-based haptic rendering methods, and almost all of them:

- give no collision detection guarantees and/or
- require a special topology of objects and/or
- need generation of precalculated structures or an explicit surface representation.

These limitations may make it difficult to use a method in practice and may not be acceptable for such precise procedures as pre-operation planning in surgery. Additionally, in practice the real medical data we work with can have any structure if segmentation has been done automatically.

In order not to have the aforementioned issues, we propose our haptic rendering approach and its improvements being published in [225, 227, 226] and being presented in chapter 4. Our approach employs ray casting for the collision detection and a "sliding along a surface" model together with a local path finding approach for rigid collision response. Additionally, the method operates directly on voxel data and does not use any precalculated structures, but uses an implicit surface representation being generated on the fly. This means that a virtual scene may be both dynamic or static. Our method was implemented and tested within the framework provided by the YaDiV platform [73] – a powerful virtual system for working with 3D volume data. This allows us to combine novel haptic rendering methods for exploration of medical data with high-quality visualization. Our approach has nearly constant time complexity independent of data resolution and is very fast – for a moderate end-user PC, up to 750 points could be simulated at about 1 kHz for collision detection without collision response, and up to 145 points for the collision detection and collision response.

Chapter 4

Our Haptic Rendering Approach

In this chapter we present our haptic rendering approach for volumetric data being published in our works [227, 225], and its improvements being published in our work [226]. We also discuss implementation details, and give the results of tests with real volumetric data.

Our approach addresses a recurring flaw in almost all related approaches, where the manipulated object, when moved too quickly, can go through or inside an obstacle. Additionally, either a specific topological structure for the collision objects is needed, or extra speed-up data structures should be prepared. These issues could make it difficult to use a method in practice. Our approach was designed to be free of such drawbacks. The method operates directly on voxel data and does not use any precalculated structures, but uses an implicit surface representation being generated on the fly.

4.1 Data Representation

Generally, 3D data could be in different representations (triangulated surface, hexahedrons, volumetric, ...). Here we focus on a volumetric one, since it is a direct output from the scanning devices. Other data types could be transformed to this one, if necessary. Furthermore, we assume that 3D data is already segmented, i.e. that a set of segments (a set of scene objects) is provided (see section 2.2.1 for definition of segmentation). We use a bit cube representation of segments, though other representations are possible.

In case of the bit cube representation, for each object (segment) a 3D bit array (a bit cube) of size of the volumetric data is created. Elements of the bit cube corresponding

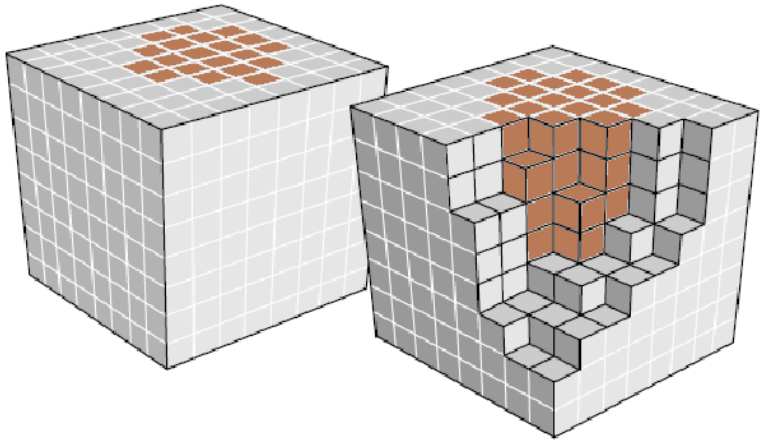


Figure 4.1: A segment as the bit cube. Brown – 1, gray – 0 (source: modified from [89])

to the object's voxels are then set to 1, and the rest are set to 0 – see figure 4.1 for illustration. For further details, we suggest the reader to look into [72].

Below we present each step of our haptic rendering algorithm. Firstly, we present in detail the method being published in our works [227, 225], and further on its improvements being presented in [226].

4.2 Collision Detection using Ray Casting

The collision detection in our haptic rendering pipeline employs the ray casting technique, which has its roots in computer graphics (see section 3.1.5 for details).

For the collision detection of the interaction point (IP) following the position of the manipulator, we perform ray casting from its last position $\mathbf{p}_1$ to the current one $\mathbf{p}_2$ – figure 4.2(a). In more detail, we are going along the ray with 1-voxel steps – figure 4.2(b). If the value of any bit cube representing an obstacle at the sampled point is *true* – figure 4.2(c), – then a collision information and *true* is returned by the collision detection procedure – figure 4.2(d). *False* is returned otherwise. We use 1-voxel steps, because a minimum possible thickness of an object is also one voxel. By performing the ray casting we can always find the exact collision, if it happened between the haptic rendering updates, and react to it accordingly.

In order to have even higher precision for collision detection, ray casting at sub-voxel resolution or sampling once between each pair of consecutive intersections of the ray

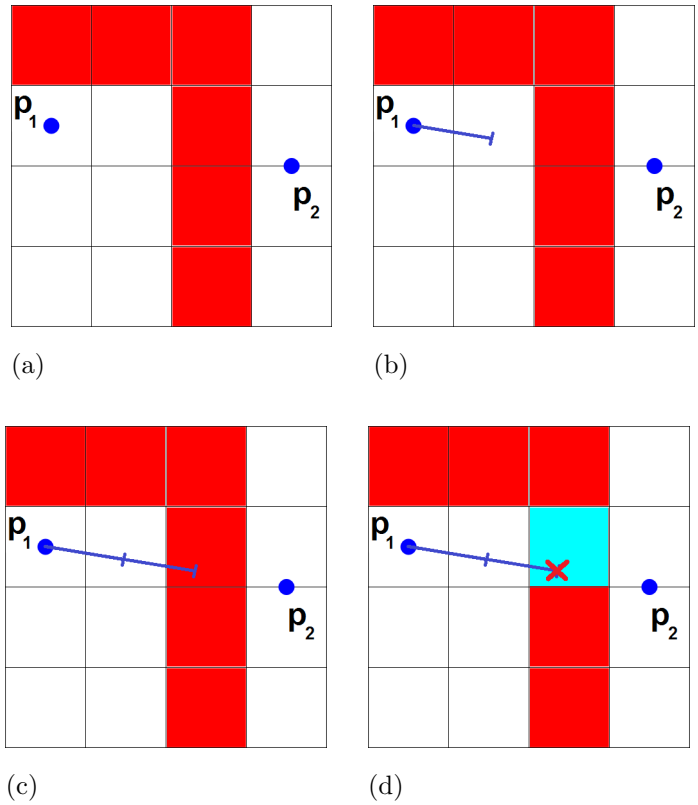


Figure 4.2: The ray from the previous position $\mathbf{p}_1$ to the current one $\mathbf{p}_2$ is cast with 1-voxel steps until an obstacle is found or $\mathbf{p}_2$ is reached (source: our work [226])

and a grid plane could be used. Though we found that a 1-voxel step is quite enough for our experimental data.

To further speed up the computations, we firstly create a list of objects that are determined as *collision candidates*. For that, we check if the ray from the last position to the current one collides with the Axis-Aligned-Bounding-Box (AABB) of each object. If so, then the object is a candidate. The detailed collision detection is performed for these candidates only.

Additionally, we impose a reasonable upper limit on the maximal movement of the IP between two haptic frames. This allows us to perform localized and therefore faster ray casting using the cached information from the previous frame and avoid possible haptic rendering instabilities (the last is also done in [15]).

If all data has been already loaded then the time complexity of the method is
$O\left(N_{obj} \cdot \frac{w_{max}}{step}\right)$, where
N_{obj} – number of objects in the scene;
w_{max} – maximum path length of the IP per frame, in voxels (introduced to ensure the stability of haptic rendering);
step – the sampling step of ray casting (chosen as 1).

Indeed, in the worst case all objects in the scene could become the collision candidates and be checked all the way from the previous position of the IP to the current one.

Since w_{max} and *step* are either constants or have a small reasonable upper limit, we can rewrite the time complexity as $O\left(N_{obj}\right)$. Furthermore, the resulting time complexity is independent of data resolution.

The space complexity of the method is $O\left(N_{obj}\right)$.

Indeed, in the worst case all objects in the scene could become the collision candidates and therefore need to be stored. Furthermore, the resulting space complexity is independent of data resolution.

4.3 Collision Response

The collision detection method described above is used in our joint collision detection and response stage of the haptic rendering pipeline. The method is based on the god object/proxy paradigm. It works directly with volumetric data and has no limitations.

In this section we present the method being published in our work [227]. The improved method being published in our work [226] is presented in section 4.8.

Because of the collision detection and non-penetration guarantees the IP should not go inside any object or pass through it. Therefore we made it slide over the surface. The surface is calculated locally "on the fly". The IP can encounter multiple surfaces on its way. It is connected with the actual position of the device's manipulator via a virtual spring.

The position of the IP from the last haptic frame is denoted as $\mathbf{p}_1$, and the position to be calculated as $\mathbf{p}_2$. For the device's manipulator, we denote its last position as $\mathbf{d}_1$ and the current one as $\mathbf{d}_2$. The IP always moves in the direction of $\mathbf{d}_2$. *Empty-space border voxels* below are the voxels which are empty but have at least one non-empty N_{26}-neighbour (two voxels are N_{26}-neighbours if the first one is orthogonally or diagonally adjacent to the second one, also see [72]).

The algorithm deals with different obstacles at the same time and looks as follows:

1. $\mathbf{p}_2 := \mathbf{p}_1$
2. Do the collision test from $\mathbf{p}_2$ to $\mathbf{d}_2$. If there is no collision then $\mathbf{p}_2 := \mathbf{d}_2$ and exit. Else move $\mathbf{p}_2$ towards the collision point $\mathbf{p}_{col}$, so that the distance between $\mathbf{p}_2$ and $\mathbf{p}_{col}$ is less than the predefined $\epsilon < 1$
3. While $\mathbf{p}_2 \neq \mathbf{d}_2$ *and* the total path length of the IP at this haptic frame has not exceeded w_{max} (see section 4.2) *and* it is not shorter just to move directly from $\mathbf{p}_2$ to $\mathbf{d}_2$ do:
   (a) Locate empty-space border voxels neighbouring to $\mathbf{p}_2$
   (b) Select a voxel with the maximal dot product (voxel-$\mathbf{p}_2$, $\mathbf{d}_2$-$\mathbf{p}_2$) > 0. If there is no such voxel then go to step 4
   (c) Move $\mathbf{p}_2$ to this voxel. If $\mathbf{p}_2$ is inside another object after this movement then cancel the movement and go to step 4
   (d) go to step 3
4. If the path length of the IP at this haptic frame $> w_{max}$ *or* $\mathbf{p}_2 = \mathbf{d}_2$ *or* $\mathbf{p}_2 =$ the value of $\mathbf{p}_2$ at the beginning of step 2, then exit. Else go to step 2.

Remarks: There are some additional checks and details, which we omitted in the above description for clarity. We will give a complete listing of the algorithm later in this sec-

tion.

An example of how the method works is shown in figure 4.3. After the initialization at step 1, figure 4.3(a), the collision test is performed at step 2, figure 4.3(b). There is a collision, so the "sliding along the surface" part of the algorithm – step 3 – is executed, figure 4.3(c). Then the conditions for the outer loop (steps 2–4) are checked at step 4. As long as they are fulfilled, step 2, figure 4.3(d), and step 3, figure 4.3(e), are executed again. At step 4 these conditions are met again, therefore the method starts the third iteration of the outer loop. But the IP can not come closer to $\mathbf{d}_2$ this time, so nothing is changed, and the algorithm stops at step 4.

The complete listing of algorithm is following:

1: Get $\mathbf{p}_1$, $\mathbf{d}_1$, $\mathbf{d}_2$
2: $\mathbf{p}_2 := \mathbf{p}_1$ // Initialize $\mathbf{p}_2$
3: $\mathbf{p}_{2last} := \mathbf{p}_2$ - (1,1,1) // make it unequal to $\mathbf{p}_2$
4: $w := 0$ // Path length travelled by the IP at this frame
5: **while** ($\mathbf{p}_2 \neq \mathbf{d}_2$ **and** $w < w_{max}$ **and** $\mathbf{p}_{2last} \neq \mathbf{p}_2$) **do**
6: $\quad \mathbf{p}_{2last} := \mathbf{p}_2$
7: $\quad$ Make the collision test from $\mathbf{p}_2$ to $\mathbf{d}_2$
8: $\quad$ **if** (no collision) **then**
9: $\quad\quad$ Move $\mathbf{p}_2$ towards $\mathbf{d}_2$ for the distance $\min(||\mathbf{d}_2\text{-}\mathbf{p}_2||_2, w_{max} - w)$
10: $\quad\quad$ $w := w +$ (the above movement of $\mathbf{p}_2$)
11: $\quad\quad$ **break**
12: $\quad$ **else**
13: $\quad\quad$ Move $\mathbf{p}_2$ towards the collision point $\mathbf{p}_{col}$ so that it is at the given $\epsilon < 1$ before $\mathbf{p}_{col}$, or for the distance $(w_{max} - w)$ from $\mathbf{p}_2$ in case the last is shorter
14: $\quad\quad$ $w := w +$ (the above movement of $\mathbf{p}_2$)
15: $\quad\quad$ // Slide over the obstacle in the direction of $\mathbf{d}_2$:
16: $\quad\quad$ **while** $w < w_{max}$ **and** $\mathbf{p}_2 \neq \mathbf{d}_2$ **do**
17: $\quad\quad\quad$ // Is it shorter just to move from $\mathbf{p}_2$ towards $\mathbf{d}_2$
18: $\quad\quad\quad$ // without following the surface?
19: $\quad\quad\quad$ **if** ($\mathbf{p}_2$ will not be inside any obstacle if moved by 1 voxel towards $\mathbf{d}_2$) **then**
20: $\quad\quad\quad\quad$ // We will move directly to $\mathbf{d}_2$ at the beginning
21: $\quad\quad\quad\quad$ // of the next iteration of the outer loop
22: $\quad\quad\quad\quad$ **break**
23: $\quad\quad\quad$ **end if**

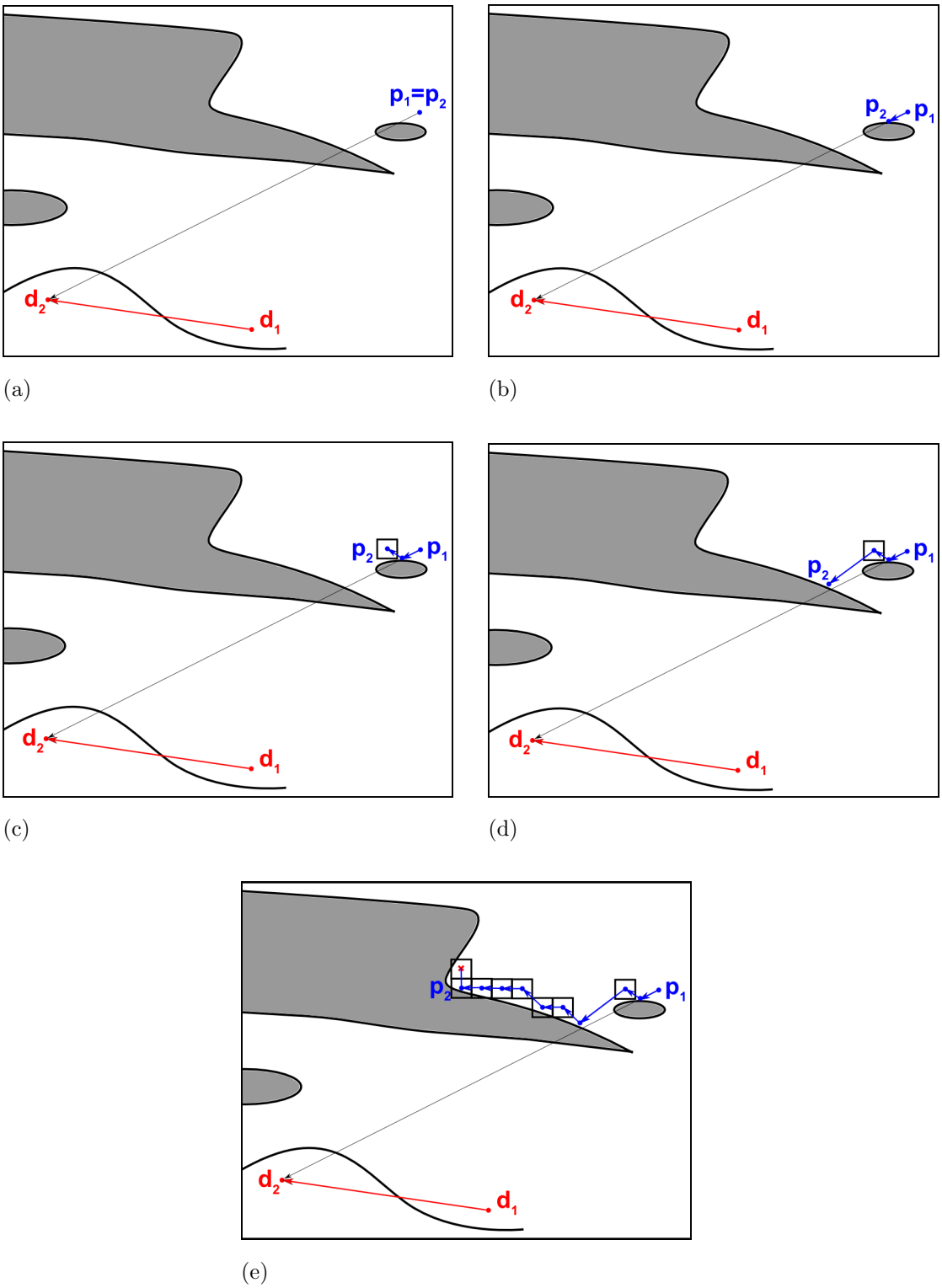


Figure 4.3: Example of execution of the original "sliding along the surface" approach (source: our work [226])

24: Locate neighbour empty-space border voxels for $\mathbf{p}_2$
25: Select a voxel with the biggest dot product of (voxel-$\mathbf{p}_2$) and ($\mathbf{d}_2$-$\mathbf{p}_1$)
26: **if** (the biggest dot product $\leq$0) **then**
27: **break**
28: **end if**
29: Move $\mathbf{p}_2$ towards the selected voxel for the distance min($||$voxel-$\mathbf{p}_2||_2$, $w_{max} - w$)
30: **if** ($\mathbf{p}_2$ is inside another obstacle) **then**
31: Cancel the above movement of $\mathbf{p}_2$
32: **break**
33: **end if**
34: $w := w +$ (the above movement of $\mathbf{p}_2$)
35: **end while**
36: **end if**
37: **end while**

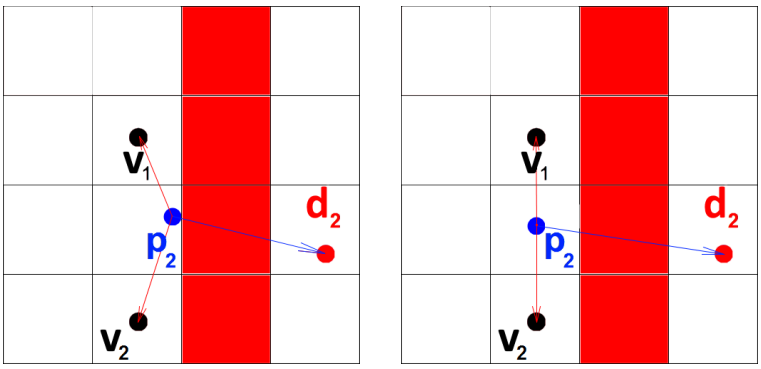
Note: If the empty-space border voxels are precalculated for each segment at the preprocessing step then it gives 25% speed-up. All the frame rates in sections below are given for the case without preprocessing.

4.4 Additional Remarks on Collision Response

The "sliding along a surface" method described in the above section needs to take into account certain additional issues:

1. Since integral arithmetic is used for N_{26}-neighbour voxel coordinates whereas real arithmetic is used for coordinates of $\mathbf{p}_2$ at step 3b in the algorithm given in section 4.3, angles between ($\mathbf{d}_2$-$\mathbf{p}_2$) and (voxel-$\mathbf{p}_2$) for some empty-space border voxels could be more than 90 degrees in case the collision has just appeared and therefore $\mathbf{p}_2 = \mathbf{p}_{col}$. In this case such empty-space border voxels will not be considered at step 3b (figure 4.4(a)) although some of them could be good if $\mathbf{p}_2$ was in the middle of the voxel but not on the border between voxels (figure 4.4(b)). Further on, let us denote all N_{26}-neighbour empty-space border voxels of $\mathbf{p}_2$ as $V_{\mathbf{p}_2}$.

   In order to deal with this issue, we additionally create a new point $\mathbf{p}_3$ and move it from $\mathbf{p}_2$ along the ray ($\mathbf{d}_2$,$\mathbf{p}_1$) with a small step < 1 until value of at least one of



(a) (b)

Figure 4.4: $\mathbf{v}_1$ and $\mathbf{v}_2$ are N_{26}-neighbour empty-space border voxels for $\mathbf{p}_2$. (a) $\mathbf{p}_2$ is on the border between empty and filled voxels, and both $\mathbf{v}_1$ and $\mathbf{v}_2$ are not considered because $(\mathbf{v}_1 - \mathbf{p}_2, \mathbf{d}_2 - \mathbf{p}_2) < 0$ and $(\mathbf{v}_2 - \mathbf{p}_2, \mathbf{d}_2 - \mathbf{p}_2) < 0$. (b) $\mathbf{p}_2$ is in the middle of border empty-space voxel, and $\mathbf{v}_2$ is considered because $(\mathbf{v}_2 - \mathbf{p}_2, \mathbf{d}_2 - \mathbf{p}_2) > 0$

its coordinates differs from the corresponding value of $\mathbf{p}_2$ by at least 1 unit (length of a side of voxel), or until w_{max} is exceeded. Once one of the above conditions are met, we find N_{26}-neighbour empty-space border voxels for $\mathbf{p}_3$. Let us denote them as $V_{\mathbf{p}_3}$. Then, we additionally consider the dot products $(\text{voxel} - \mathbf{p}_3, \mathbf{d}_2 - \mathbf{p}_2) > 0$, where $\text{voxel} \in V_{\mathbf{p}_3}$, for selection of a voxel to move $\mathbf{p}_2$ to at step 3b. See figures 4.5 and 4.6 for illustration. Additionally, we would like to note that $V_{\mathbf{p}_3}$ is not necessarily equal to $V_{\mathbf{p}_2}$. Such an example is shown on the screen shot of our prototype system in figure 4.7.

2. Since segments can be moved/rotated, their reverse transformations should be applied to the positions being used for N_{26}-neighbour search, and their direct transformations should be applied to the coordinates of the found empty-space border voxels.

3. We can not simply limit the movement of the IP by the plane perpendicular to ($\mathbf{d}_1$,$\mathbf{d}_2$) and going through $\mathbf{d}_2$ (see figure 4.8), and use this limitation as a stop-condition in line 5 of the listing of algorithm in section 4.3. This is because the movement shown in figure 4.9 would not be possible in such a case.

4. Similar to the above remark, we can not limit the movement of the IP by the plane going through $\mathbf{d}_1$ and $\mathbf{d}_2$ and perpendicular to the plane going through $\mathbf{d}_1$, $\mathbf{d}_2$ and the current position of $\mathbf{p}_2$ by the end of the outer while-loop of the same listing (see figure 4.10), and use this limitation as a stop-condition in line 5. The reason

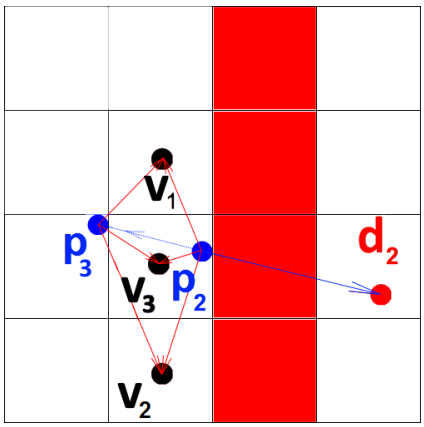


Figure 4.5: Since $(\mathbf{v}_1 - \mathbf{p}_3, \mathbf{d}_2 - \mathbf{p}_2) > 0$, $(\mathbf{v}_2 - \mathbf{p}_3, \mathbf{d}_2 - \mathbf{p}_2) > 0$ and $(\mathbf{v}_3 - \mathbf{p}_3, \mathbf{d}_2 - \mathbf{p}_2) > 0$ (where $\mathbf{v}_1$,$\mathbf{v}_2$,$\mathbf{v}_3 \in V_{\mathbf{p}_3}$), $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ will be taken into consideration

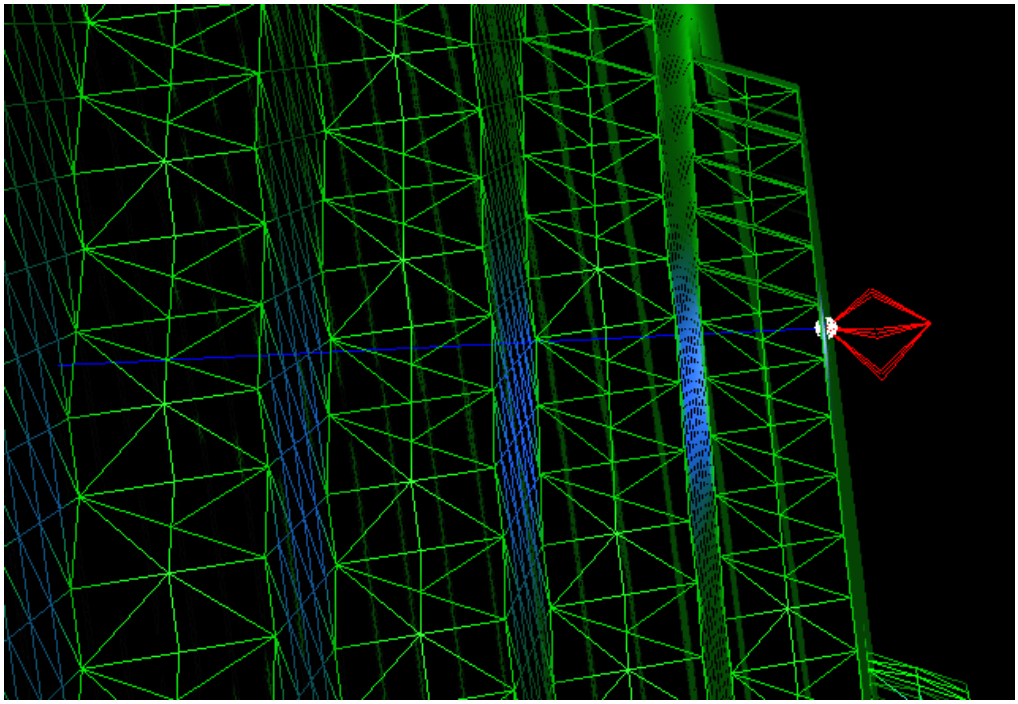


Figure 4.6: N_{26}-neighbour empty-space border voxels shown in our system (the segment is triangulated using the modified marching cubes algorithm – see section 5.2) for details

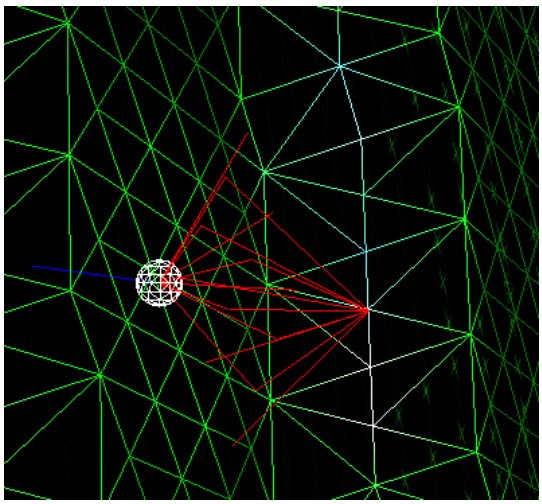


Figure 4.7: The case when $V_{\mathbf{p}_3} \neq V_{\mathbf{p}_2}$, shown in our prototype system

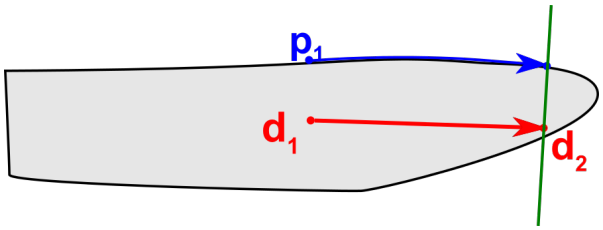


Figure 4.8: The positive scenario of limiting the movement of the IP (blue) by the plane perpendicular to ($\mathbf{d}_1$,$\mathbf{d}_2$) and going through $\mathbf{d}_2$ (green)

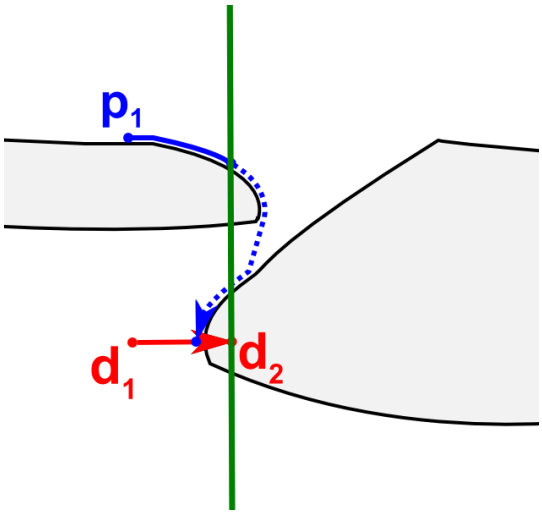


Figure 4.9: The negative scenario of limiting the movement of the IP (blue) by the plane perpendicular to ($\mathbf{d}_1$,$\mathbf{d}_2$) and going through $\mathbf{d}_2$ (green). The IP should follow the full path shown in blue, but can only follow the part of it drawn in solid

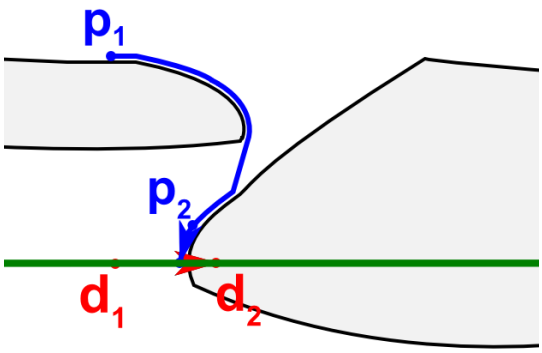


Figure 4.10: The positive scenario of limiting the movement of the IP (blue) by the plane (green) in the way described in point 4 in section 4.4

for this is that the movement shown in figure 4.11 would not be possible in this case.

4.5 Time and Space Complexities of Collision Response

Here we discuss the time and space complexities of our joint collision detection and response stage of the haptic rendering pipeline.

Let us follow the listing of algorithm in section 4.3. The main work is done inside the outer while-loop (lines 5-37). In line 7 the collision test is performed, therefore the time complexity of this line is $O\left(N_{obj} \cdot \frac{w_{max}}{step}\right)$ (see section 4.2). For the inner while-loop (lines 16–35), lines 17–29 take $O(1)$ and lines 30–34 take $O(N_{obj})$. In the worst case it can be $O(w_{max})$ iterations of the innner while-loop. The outer while-loop can also be executed maximum $O(w_{max})$ times. Additionally, we should note that $\mathbf{p}_2$ can move in total only $O(w_{max})$ times during the run of the algorithm. Therefore the total time complexity being equal to the time complexity of the outer while-loop is $O\left(w_{max} \cdot N_{obj} \cdot \frac{w_{max}}{step}\right) = O\left(N_{obj} \cdot \frac{w_{max}^2}{step}\right)$.

Since w_{max} and $step$ are constants or have a small reasonable upper limit, we can rewrite the above equation as $O(N_{obj})$. Furthermore, the resulting time complexity is

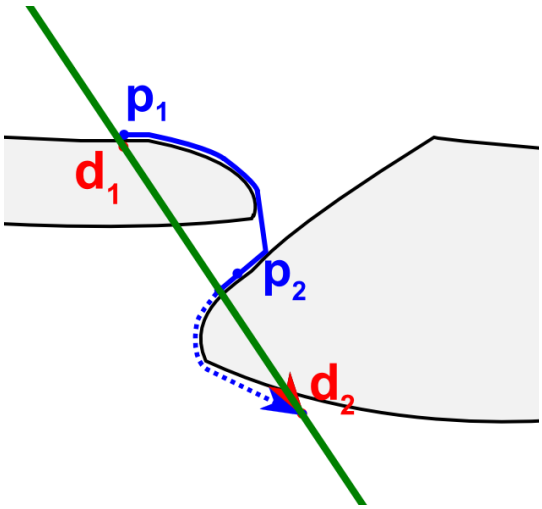


Figure 4.11: The negative scenario of limiting the movement of the IP (blue) by the plane (green) in the way described in point 4 in section 4.4. The IP should follow the full path shown in blue, but can only follow the part of it drawn in solid

independent of data resolution.

The space complexity of the method is $O(N_{obj})$.

Indeed, in the worst case all objects in the scene could become the collision candidates and therefore need to be stored for line 7. Since the algorithm works "on the fly", no other additional structures are required. Furthermore, the resulting space complexity is independent of data resolution.

4.6 Force-Feedback

In this section we present the method being published in our work [227]. The improved method being published in our work [226] is presented in section 4.9.

The specificity of our force feedback generation is that we do not use surface normals, because we do not employ an explicit surface representation.

The total force transferred to a user via the haptic manipulator is

$$\boldsymbol{F} = \boldsymbol{F}_c + \boldsymbol{F}_{fr}, \tag{4.1}$$

where
$\boldsymbol{F}_c$ – a coupling force;
$\boldsymbol{F}_{fr}$ – a friction force.

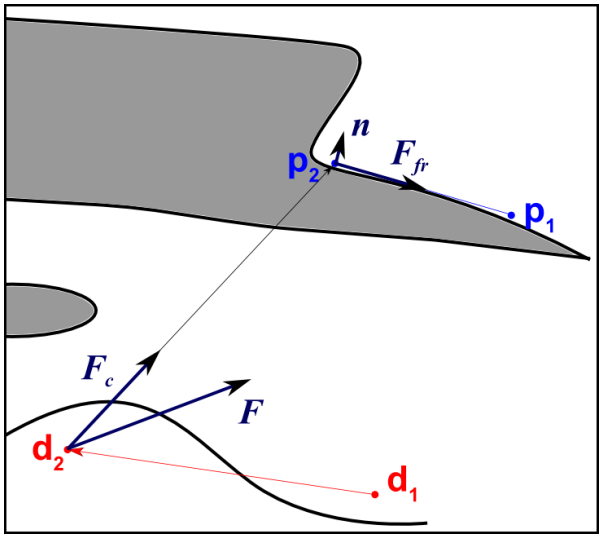


Figure 4.12: Forces involved in the computation of force-feedback in section 4.6

If $\boldsymbol{F}$ exceeds a maximum for a given haptic device then we scale it as to fit to the device limitations.

A calculation of $\boldsymbol{F}_c$ yields

$$\boldsymbol{F}_c = -\frac{\mathbf{d}_2 - \mathbf{p}_2}{\|\mathbf{d}_2 - \mathbf{p}_2\|_2} \cdot (\|\mathbf{d}_2 - \mathbf{p}_2\|_2 \cdot k) = (\mathbf{p}_2 - \mathbf{d}_2) \cdot k, \tag{4.2}$$

while for $\boldsymbol{F}_{fr}$ we obtain

$$\boldsymbol{F}_{fr} = -\frac{\mathbf{p}_1 - \mathbf{p}_2}{\|\mathbf{p}_1 - \mathbf{p}_2\|_2} \cdot |\boldsymbol{F}_c \cdot \boldsymbol{n}| \cdot \mu \cdot \frac{N_{bv}}{w}, \tag{4.3}$$

where
k – the coefficient of the spring;
$\boldsymbol{n}$ – a normalized vector, which is perpendicular to $\mathbf{p}_2$-$\mathbf{p}_1$ and lies on the plane defined by vectors $\mathbf{p}_2$-$\mathbf{p}_1$ and $\mathbf{d}_2$-$\mathbf{p}_2$;
μ – the friction coefficient;
N_{bv} – number of the border empty-space voxels, which the IP moved through in the algorithm above at this haptic frame;
w – the total path length at this frame, also from the algorithm above.
See figure 4.12 for illustration.

We would like to note that for easier calculations $|\boldsymbol{F}_c \cdot \boldsymbol{n}|$ could be rewritten as

$$|\boldsymbol{F}_c \cdot \boldsymbol{n}| = \|\boldsymbol{F}_c\|_2 - \left| \boldsymbol{F}_c \cdot \frac{\mathbf{p}_1 - \mathbf{p}_2}{\|\mathbf{p}_1 - \mathbf{p}_2\|_2} \right|. \tag{4.4}$$

We use the given expression for $\boldsymbol{F}_{fr}$ because at the end of a haptic frame the IP is moved from $\mathbf{p}_1$ to $\mathbf{p}_2$, and therefore we turn the friction force to the opposite direction (the improved approach being published in our work [226] is presented in section 4.9). Also we make it proportional to the part of $\boldsymbol{F}_c$, which is perpendicular to $\mathbf{p}_2$-$\mathbf{p}_1$ in analogy to the normal force for a dry friction. Finally, we ensure it to be proportional to N_{bv}, i.e. the path length that the interaction point actually slid over a surface. We would like to note that making the forces related to physical properties of certain materials was not our goal on that stage of research.

Both the time and space complexities for this stage of the haptic rendering pipeline can be written as $O(1)$.

4.7 Workspaces and Movement/Rotation of Objects

Our system maintains several workspaces (coordinate systems):

- The first is the scene workspace (SWS), where all virtual objects are positioned. The IP belongs to this workspace, too.
- The second is the view workspace (VWS). It defines how the user looks at the SWS.
- The third workspace is the real workspace of the haptic device (HWS). It is measured in millimeters.
- Since it is more intuitive for the user when axes of the HWS are parallel (X and Y) and perpendicular (Z) to the viewing plane (in other words, to the display), one more workspace is needed – we call it the workspace of movement (MWS). This workspace makes the axes of the HWS be always parallel to the corresponding axes of the VWS. The MWS is needed, because in a general case the axes of the HWS will not be parallel to the corresponding axes of the VWS after camera rotations. Actually, the MWS differs from the SWS only in the orientation. This means that the IP, while being positioned in the SWS, moves along the coordinate axes, which belong to the MWS, and the right position/orientation of the IP in the SWS is maintained all the time.

Similarly to maintaining the right position/orientation of the IP, we added a support of movement/rotation of any object in the scene according to manipulations with the

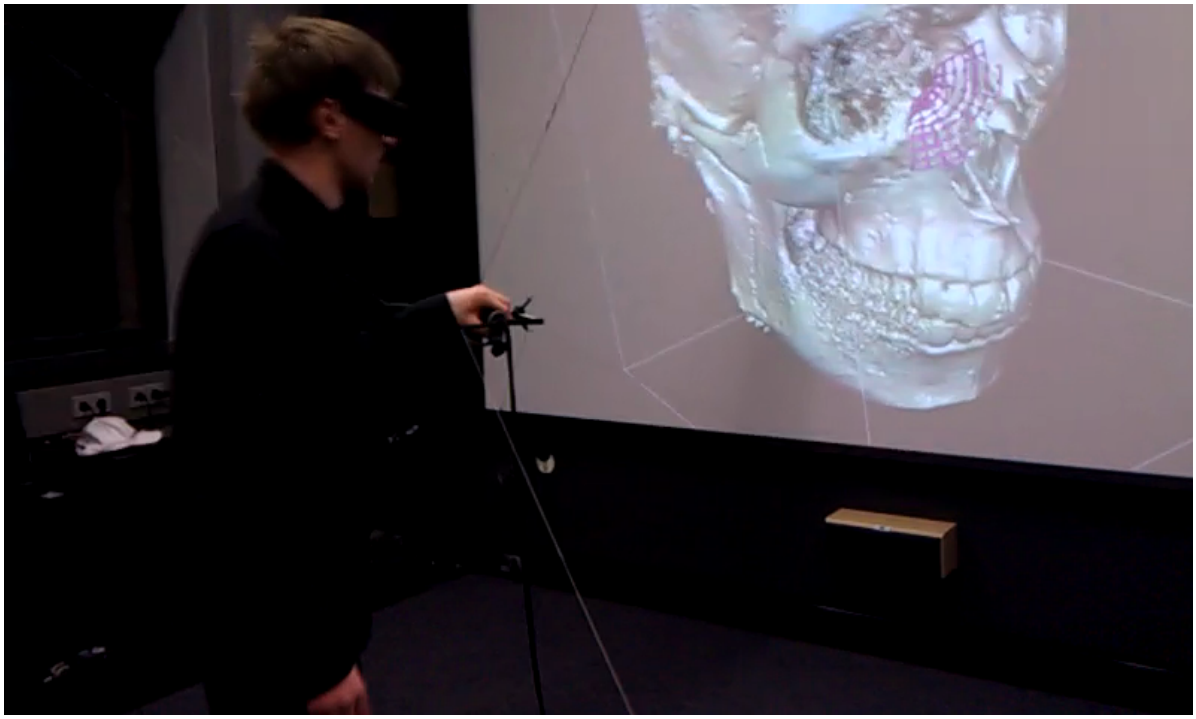


Figure 4.13: Adjusting the metal mesh (in purple) to the eyeball using the INCA 6D haptic device (source: our work [225])

device's probe. To accomplish this, the system goes into the special no-collision mode, and when the object is selected, a movement and rotation of the IP are *directly* applied to the object.

An example application of the described approach is verification of the metal mesh for the surgical operation on correction of position of the eyeball in case of a complex skull fracture as shown in figure 4.13. Please note that the way to find the form of the mesh was developed in the bounds of the joint research project between physicians from the Hanover Medical School, Germany, and our Institute. See [22] for details.

4.8 Improved Collision Response

The original version of our joint collision detection and response stage of the haptic rendering pipeline was proposed in our work [227] and was discussed in the above sections. In this section we present its improved version being published in [226], which uses the path-finding approach combined with the god object/proxy paradigm.

We have found out that the use of the dot product of the vectors at step 3b of our

original approach (section 4.3) in order to find the next voxel to move to sometimes leads to an issue, namely that the IP oscillates around the point being locally the closest surface point to $\mathbf{d}_2$ (let us denote it as $\mathbf{p}'_2$). This oscillation could happen because of the following. If there is always a next voxel on the surface, to where the IP can move in the direction of $\mathbf{d}_2$-$\mathbf{p}_1$ according to the conditions at step 3b, the IP may pass $\mathbf{p}'_2$ and go further. This could happen because the IP will move until its total path length at this haptic frame is less than w_{max} and because w_{max} may be not exceeded at $\mathbf{p}'_2$. If $\mathbf{d}_2$ remains unchanged at the next haptic frame then the IP will go the way back and will also pass $\mathbf{p}'_2$ backwards direction and go further because of the same reason. At the next haptic frame the IP will go in the same direction as at the first haptic frame and will pass $\mathbf{p}'_2$ again. These oscillations may continue until the position of the probe is changed.

In order to eliminate this drawback, we suggest to replace the use of the dot product at step 3b with the search for the voxel with the smallest distance to $\mathbf{d}_2$. In other words, we suggested to use a path finding algorithm looking for a locally optimal path to $\mathbf{d}_2$ for the given metric and limitations. Our improved method still deals with different obstacles at the same time and looks as follows:

1: Get $\mathbf{p}_1$, $\mathbf{d}_1$, $\mathbf{d}_2$
2: $\mathbf{p}_2 := \mathbf{p}_1$ // Initialize $\mathbf{p}_2$
3: Set $\mathbf{p}_{2last}$ to be unequal to $\mathbf{p}_2$
4: $w := 0$ // Path length travelled by the IP at this frame
5: **while** ($\mathbf{p}_2 \neq \mathbf{d}_2$ **and** $w < w_{max}$ **and** $\mathbf{p}_{2last} \neq \mathbf{p}_2$) **do**
6: $\quad \mathbf{p}_{2last} := \mathbf{p}_2$
7: $\quad$ Make the collision test from $\mathbf{p}_2$ to $\mathbf{d}_2$
8: $\quad$ **if** (no collision) **then**
9: $\quad\quad$ Move $\mathbf{p}_2$ towards $\mathbf{d}_2$ for the distance $\min(||\mathbf{d}_2\text{-}\mathbf{p}_2||_2, w_{max} - w)$
10: $\quad\quad$ $w := w +$ (the above movement of $\mathbf{p}_2$)
11: $\quad\quad$ **break**
12: $\quad$ **else**
13: $\quad\quad$ Move $\mathbf{p}_2$ towards the collision point $\mathbf{p}_{col}$ so that it is at the given $\epsilon < 1$ before $\mathbf{p}_{col}$, or for the distance $(w_{max} - w)$ from $\mathbf{p}_2$ in case the last is shorter
14: $\quad\quad$ $w := w +$ (the above movement of $\mathbf{p}_2$)
15: $\quad\quad$ // Find a path to $\mathbf{d}_2$ along the obstacle's surface, so that
16: $\quad\quad$ // the path is locally optimal at each step:
17: $\quad\quad$ **while** $w < w_{max}$ **and** $\mathbf{p}_2 \neq \mathbf{d}_2$ **do**
18: $\quad\quad\quad$ // Is it shorter just to move from $\mathbf{p}_2$ towards $\mathbf{d}_2$
19: $\quad\quad\quad$ // without following the surface?

20: **if** ($\mathbf{p}_2$ will not be inside any obstacle if moved by 1 voxel towards $\mathbf{d}_2$) **then**
21: // We will move directly to $\mathbf{d}_2$ at the beginning
22: // of the next iteration of the outer loop
23: **break**
24: **end if**
25: Locate neighbour empty-sp. border voxels for $\mathbf{p}_2$
26: $dist\_sq := \infty$
27: Select a voxel with the smallest square distance to $\mathbf{d}_2$, and remember this distance as $dist\_sq$
28: **if** ($dist\_sq = \infty$) **then**
29: **break**
30: **end if**
31: Move $\mathbf{p}_2$ towards the selected voxel for the distance $\min(||\text{voxel-}\mathbf{p}_2||_2, w_{max} - w)$
32: **if** ($\mathbf{p}_2$ is inside another obstacle) **then**
33: Cancel the above movement of $\mathbf{p}_2$
34: **break**
35: **end if**
36: $w := w +$ (the above movement of $\mathbf{p}_2$)
37: **end while**
38: **end if**
39: **end while**

The time and space complexities for the improved collision response remain the same as in the original approach. Indeed, looking for a voxel with the smallest square distance to $\mathbf{d}_2$ takes $O(1)$ because the maximum number of empty-space border voxels is limited, and it requires $O(1)$ space.

4.9 Improved Force-Feedback

The original version of our force-feedback generation stage of the haptic rendering pipeline was proposed in our work [227] and was discussed in section 4.6. In this section we present its improved version being published in [226]. The improvements were necessary because the direction of the friction force $\boldsymbol{F}_{fr}$ could be wrong in the case of multiple obstacles or a complex surface, since we used the direction of $\mathbf{p}_2$-$\mathbf{p}_1$. Additionally, in the new expression for $\boldsymbol{F}_{fr}$ (expression 4.5) we used w_{bv}, *the path length* which the IP trav-

elled through empty-space border voxels, instead of *the number* of those empty-space border voxels in the original expression 4.3. This was done since the IP could move less than one voxel in the inner loop of the algorithm above.

For $\boldsymbol{F}_{fr}$ the updated expression can be written as

$$\boldsymbol{F}_{fr} = -\mu \cdot \frac{\mathbf{v}_{bv}}{\|\mathbf{v}_{bv}\|_2} \cdot |\boldsymbol{F}_c \cdot \boldsymbol{n}| \cdot \frac{w_{bv}}{w}, \tag{4.5}$$

where
μ is the friction coefficient;
$\mathbf{v}_{bv} = \sum_i \mathbf{v}_i$, where
$\mathbf{v}_i$ are linear path segments being travelled by the IP through the empty-space border voxels at this haptic frame;
$\boldsymbol{n}$ – a normalized vector being perpendicular to v_{bv} and located on the plane spanned by v_{bv} and $\mathbf{d}_2$-$\mathbf{p}_2$;
w_{bv} – the length of the path where (during this haptic frame) the IP travelled through the empty-space border voxels in the algorithm described above;
w – the total of the path covered by the IP during this frame according to the algorithm described above.

For easier calculations $|\boldsymbol{F}_c \cdot \boldsymbol{n}|$ could be rewritten as $\|\boldsymbol{F}_c\|_2 - \left|\boldsymbol{F}_c \cdot \frac{\mathbf{v}_{bv}}{\|\mathbf{v}_{bv}\|_2}\right|$.

We suggest the new formula for $\boldsymbol{F}_{fr}$ in [226] as opposed to [227] because at the end of a haptic frame the IP is moved from $\mathbf{p}_1$ to $\mathbf{p}_2$, so it is logical to turn $\boldsymbol{F}_{fr}$ into the direction of the normalized vector given by the average obtained (via their sum) from all path segments, where the IP travelled along a surface. Additionally, we ensure $\boldsymbol{F}_{fr}$ to be proportional to the part of $\boldsymbol{F}_c$ which is perpendicular to $\mathbf{v}_{bv}$ in analogy to the normal force for a dry friction, Finally, we make it proportional to w_{bv}, i.e. the path length that the IP actually slid over a surface.

The time and space complexities for the improved force feedback generation remain the same as in the original approach. Indeed, the number of $\mathbf{v}_i$ is limited by $O(w_{max}) = O(1)$. Furthermore, in practice $\mathbf{v}_{bv}$ is updated during the run of the improved joint collision detection and response stage of our approach.

4.10 Prototype System

As it was already mentioned before, our interactive VR system is based on the YaDiV Open-Source platform [73]. The main features of the YaDiV include reading of input

data in the DICOM format and offering modules for the volumetric data processing pipeline (the volumetric data processing pipeline is discussed in detail in section 2.2.1):

- 2D Visualization;
- 3D Volume Visualization (2D-Texturing and 3D-Texturing – see section 2.2.4 for the basics and sections 3.1.1, 3.1.3 for an overview of the methods);
- 3D Segmentation;
- 3D Segment Visualization and Registration.

The YaDiV platform was successfully employed for teaching and educational purposes and extended by many student projects. It is also currently used by physicians at Hanover Medical School, Germany, in various research projects.

Our prototype system is structurally a plug-in for YaDiV. In order to allow absolute platform independence, YaDiV was developed using the Java platform. This is the case for our system, too. Only the device dependent part was developed using C++, because there are no device APIs on Java being supported by the devices manufacturers. The system is independent from a haptic display, so that a wide range of devices can be used, including:

- Phantom Omni (figure 4.14);
- High-end Phantom Premium 1.5 6-DOF (figure 4.15);
- INCA 6D with a very large workspace of approx. 2*1*1.6m (figure 4.16).

The size of the virtual workspace can be scaled and varies from case to case.

For communication between Java and C++ we used the JNA (Java Native Access) in the DirectMapping mode, because communication delays are less than 1 µs.

As an option, we also considered communication via the TCP-IP, but our tests showed that it is not fast enough. In more detail, there are often (several times per second) delays of about 1 ms, as well as delays of up to 20 ms from time to time. In order to verify that the delays are because of communication via the TCP-IP and not because of Java-specific issues, both the test client and the test server were also implemented in C++ and showed the same delay of 1 ms several times per second.

Figure 4.14: Phantom Omni haptic device

Figure 4.15: Phantom Premium 1.5 6-DOF haptic device

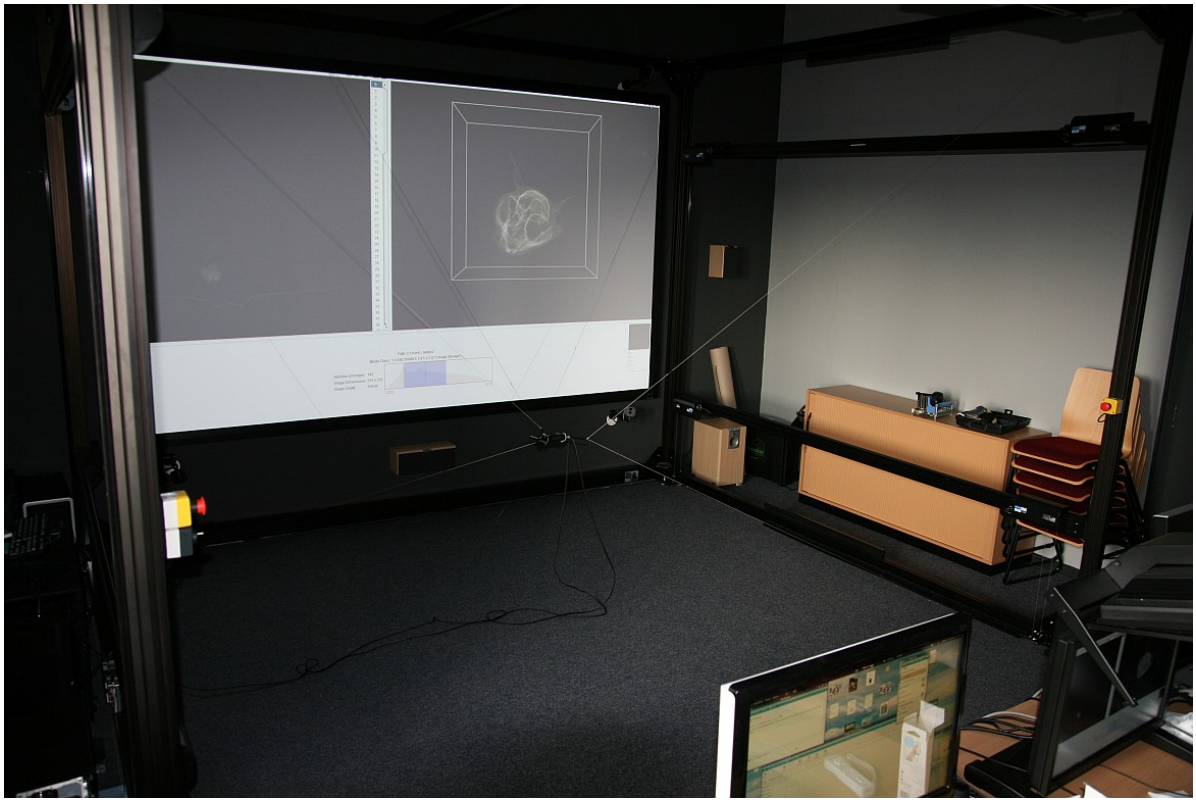


Figure 4.16: INCA 6D haptic device

4.11 Dealing with Synchronization Issues

The graphical representation of objects in YaDiV is re-rendered upon request. That is, when properties (color, position and orientation, ...) of a scene object are changed, the scene is redrawn. In more detail, when object properties are changed, a rendering thread per object is created and executed. Together with haptic interaction, this rendering scheme leads to synchronization problems. If we change graphics properties directly in the haptic thread, then every change in the properties would cause a new redraw event, creating unacceptable delays of tens of ms during the execution of the haptic thread. Indeed, if we change graphics properties directly in the haptic thread then for every object's change in the haptic thread, a separate rendering thread would be created. The creation of this thread would take a few tens of milliseconds. Besides that, thousands of rendering threads would be running even in the case of one moving object. It would be so because:

- the update rate of the haptic thread is at least 1000 frames/second;
- the rendering thread for the moving object would be created at each frame;
- each rendering thread would run longer than 1 ms.

In order to deal with the aforementioned issues, we proposed to use special objects in the haptic thread, which accumulate changes of the graphics properties, and apply them to the corresponding YaDiV entities in a dedicated synchronization thread – see figure 4.17 (see section 4.12 for details of our prototype system). In other words, these accumulating objects wrap all object properties which could cause re-rendering. An access to them is made using synchronized Java-statements. In case a wrapped property was changed, a corresponding accumulating object is added to the list of objects which should be synchronized. The synchronization thread performs a synchronization with the corresponding entities of the graphics thread at about 30 Hz by going through this list.

We would like to mention that there were other synchronization issues being addressed in our prototype system, such as correct work of the system while scene objects are being added/deleted or modified.

4.12 Scheme of the Prototype System

Our prototype system is multi-rate and multi-threaded, as it is shown in figure 4.17. Thus, the device thread and the haptic thread run at not less than 1 kHz, while the haptics to graphics synchronization thread runs at 30 Hz, because more frequent updates for graphics representation are not necessary. Here, the device thread is device-specific and is being run in the C++ part of our system. The haptics to graphics synchronization thread is necessary because of the synchronization issues discussed in section 4.11.

We should note that in YaDiV we additionally modified polygonal rendering of segments, as well as 2D/3D texture rendering of volumetric data in order to support rendering of segments being moved and/or rotated.

As was mentioned above, our prototype system is structurally a plug-in for YaDiV. The prototype system has many components. Its structure overview diagram is hence split into two parts and is shown in figures 4.18, 4.19. Below we discuss it in more detail.

Figure 4.18 shows that our system was designed in such a way that it allows easy addition of new haptic devices, as well as collision detection/response and force-feedback algorithms. Below we describe responsibilities of important classes shown in the diagram:

- "YHapticUIPlugin"
  - the main class (the entry point) of the plug-in;
  - does initial loading/deleting of scene objects;
  - responsible for the GUI;
- "HapticRenderer"
  - responsible for all haptic rendering, does initialization/finalization etc.;
  - maintains and modifies the list of scene objects (shown in figure 4.19);
- "HapticLoop" (thread) is the main haptic rendering loop. From here, the device state is read, different haptic rendering algorithms are called and the force feedback is sent back to the device.

Different haptic devices and haptic rendering algorithms could be easily added, since we used abstract classes in the haptic rendering loop. Derived classes of "Abstract Device Listener" can be not only collision detection and response algorithms, but also perform

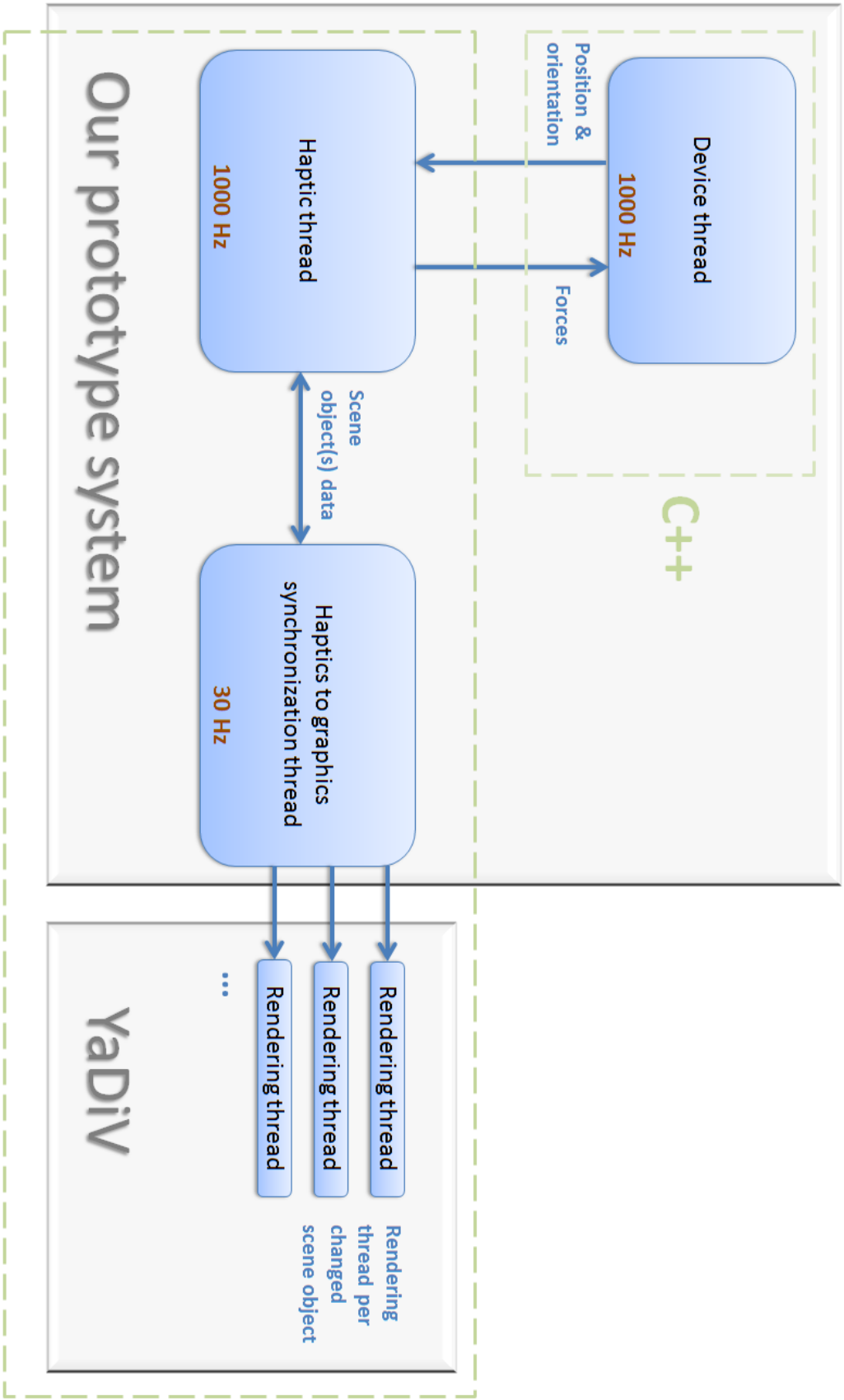


Figure 4.17: Our approach is multi-rate and multi-threaded

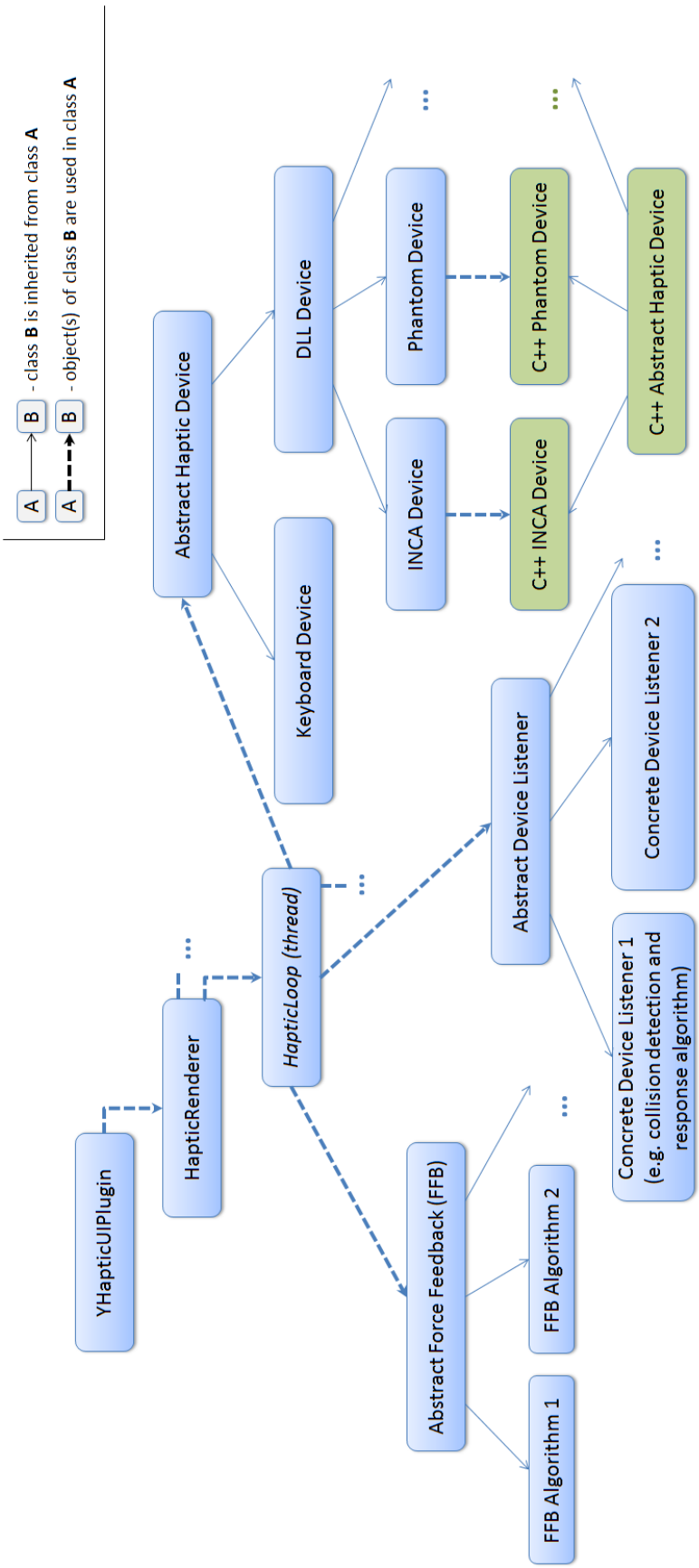


Figure 4.18: Part 1 of the structure diagram of our approach: devices and algorithms

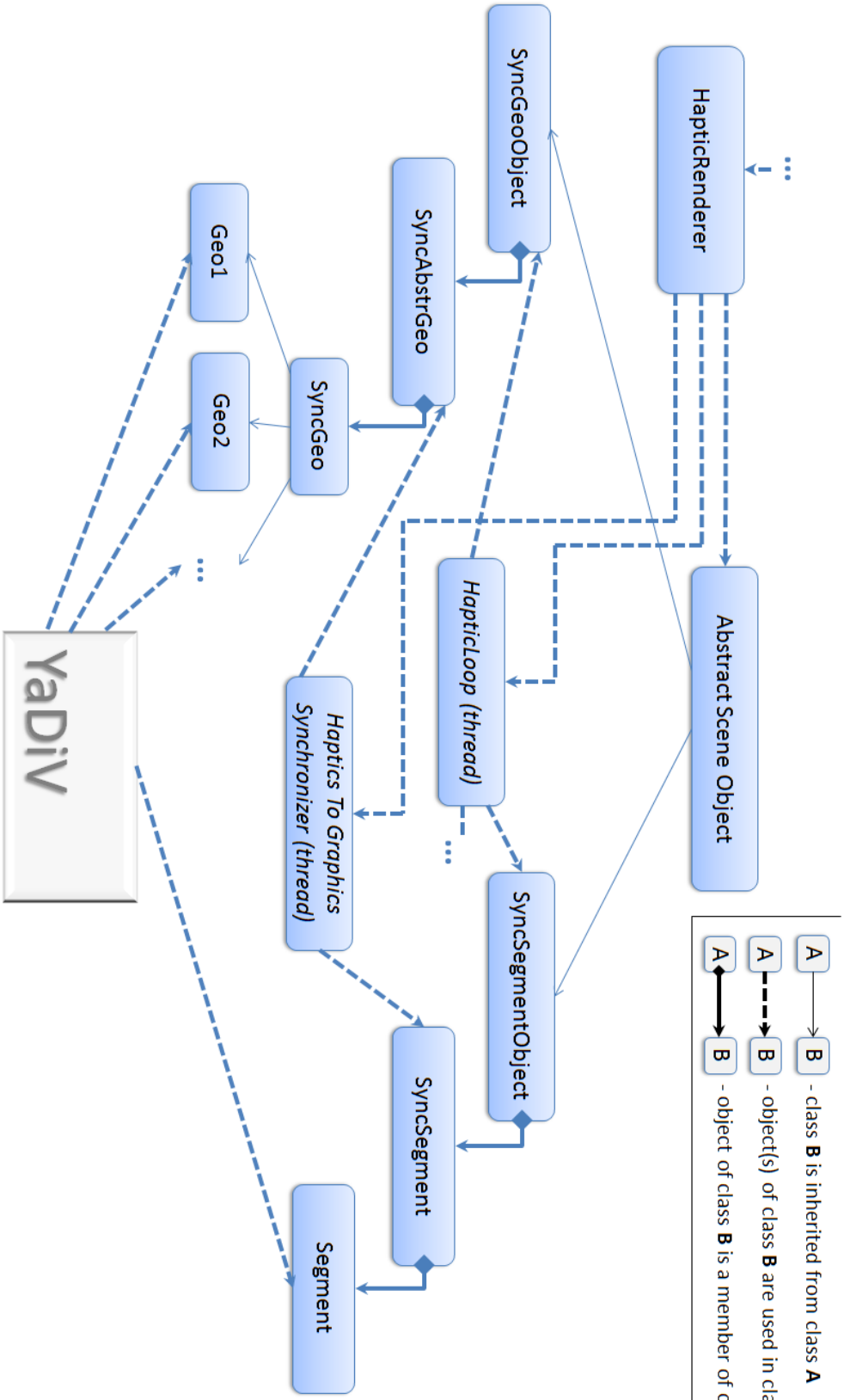


Figure 4.19: Part 2 of the structure diagram of our approach: scene objects and their usage by threads

some special actions in response to user input, since any number of device listeners at the same time is supported.

In figure 4.19 we show the class structure for scene objects, as well as the threads that use them. "AbstractSceneObject" and its derivative classes contain specific object data for haptic rendering and are used in the haptic loop. Derivative classes of "AbstractSceneObject" contain references to the synchronized versions of corresponding YaDiV object entities ("Geo1", "Geo2", "Segment" etc.). In more detail, "SyncAbstrGeo" and "SyncSegment" are the wrappers over the YaDiV entities. These wrappers allow safe reading of object data and remember required changes of properties for the wrapped entities. These wrappers are used for synchronization of properties with YaDiV's entities in the specially designated "Haptics To Graphics Synchronizer" (thread) – see section 4.11 for details.

4.13 Dealing with Java Virtual Machine Issues

Since Java is executed on a Virtual Machine (VM) with garbage collection, we experienced indeterministic delays from a few milliseconds to tens of milliseconds from time to time during the run of the haptic system. This a is serious drawback, since the haptic update rate should constantly be at least 1 kHz.

In more detail, when we tried to run our system on a standard VM, delays were about 0.5-4 ms (mostly 1-2.5 ms). An additional delay of 20–50 ms occurred when the memory was full and the garbage collector did the cleaning. The delays occurred even with the finest tuning of parameters of the Java VM and with the simplest Java application. Additionally, we observed a strange behaviour on Linux only: when a "simple loop" test was run on two different VMs simultaneously, both instances of the test had delays almost at the same time. This did not happen when we ran only one "simple loop" test at a time or ran one "simple loop" test on the VM and another one an executable written in C++.

Because of the aforementioned drawbacks we looked for a solution and came across the works [216], [167] which say that a *real-time* VM can provide a deterministic execution time, i.e. it can eliminate the aforementioned issues. Therefore we conducted experiments with two common real-time VMs: Sun JavaRTS [167] and IBM Web Sphere Real Time [216]. We followed all recommendations of the developers, like installation of Linux with a real-time core and fine tuning of the VM. For Sun JavaRTS we also "downgraded" our code to Java 1.5 because that time Sun JavaRTS VM supported Java

1.5 only. As a result, we found out that there are still delays of 1-3 ms.

In more detail, when our application was run without any special real-time Java features, the delay was in the range 300 µs–3 ms (mostly 1–1.8 ms). When we used special real-time threads and timing, the delay was 300 µs–1 ms (mostly 600 µs). The upper limit for the delay was not good enough because the haptic thread should be updated every millisecond, which is not possible when there is already a delay of 1 ms in addition to the 1 ms being spent for computation of haptics logic. Additionally, there were "freezings" of graphics rendering from time to time for about a minute, which ruined the interactivity of the application. Furthermore, the time required for the garbage collector to perform cleaning for both the real-time Java machines was still about 11–20 ms. We would like to note that we were unable to compile the code with real-time threads using the evaluation version of SDK from IBM, because there were no real-time libraries included, and we could not buy it before obtaining satisfactory test results. For Sun JavaRTS VM, we ran our tests in both precompiled and not-precompiled modes.

In summary, we would like to point out that the observed results differ from the information stated in [216] and [167], which was officially presented by IBM and Sun respectively.

Since Java does not provide a stable update rate even with a real-time VM, we used a standard VM and added virtual coupling into our C++ part having nearly constant update rate of at least 1 kHz. Using this approach, a sufficient and stable haptic update rate is always provided to the user.

4.14 Results

For tests real medical tomography data sets were used, including Torso (520x512x512, fig. 4.20), Head_{big} (464x532x532, fig. 4.21) and Head_{small} (113x256x256, fig. 4.22).

For the point-object collisions only, the haptic update rate during the *peak load* is about 750 kHz on our moderate high-end user PC (8 x Intel Xeon CPU W5580 @ 3.20 GHz, 24 GB RAM, NVIDIA Quadro FX 5800). For the joint collision detection and response approach the value is about 160-170 kHz. Both values exceed the minimum requirement for real-time haptics by orders of magnitude. This efficiency and the conceptual clarity of our approach contrasts most triangle-based approaches, where millions of triangles would be generated and complex speeding-up traversing structures are required for the fast and precise collision detection. The values were obtained for the virtual haptic device, which

Table 4.1: Resulting update rates

Data	**Size**	**Triangles**	**Update Rate**
Head_{small}	113x256x256	690k	152 kHz
Torso	520x512x512	2,222 Mi	138 kHz
Head_{big}	464x532x532	6,136 Mi	146 kHz

is simulated in Java. For real devices, Java-C++ communication (transferring of the device transformations and forces) since the haptic device dependent part was developed using C++ (see section 4.10). We have measured the timings and found out that because of these communication costs the resulting update rate is a little lower – about 150 kHz. The values for the data sets for the joint collision detection and response approach are shown in table 4.1. **Triangles** denotes number of triangles in the scene for the graphics rendering as a reference. Triangulation was extracted from the volumetric data using a modified marching cubes algorithm. **Update Rate** is given for real devices and during the peak load.

Our prototype system was tested under Microsoft Windows, as well as under Linux. Under Linux it was also run using the stereo graphics mode. The users found the last one especially useful for an intuitive interaction with 3D data compared to the normal graphics mode.

4.15 Results for the Improved Approach

Using the improved method, we repeated the tests as stated in [227] and in section 4.14. We used the same real tomography data sets, including Torso, Head_{big} and Head_{small}. The results could be also found in our work [226].

The point-object collisions mode with no collision response remained unchanged, therefore the haptic update rate did not change and is about 750 kHz during the *peak load* on our moderate high-end user PC (see section 4.14 for specifications). For our improved joint collision detection and response approach the value is about 140-150 kHz. Both values still exceed the minimum requirement for real-time haptics by orders of magnitude. The values were obtained for the virtual haptic device, which is simulated in Java. For real devices, the resulting update rate is a little lower – about 135 kHz. As before, the update rate is lower because Java-C++ communication and transferring of the de-

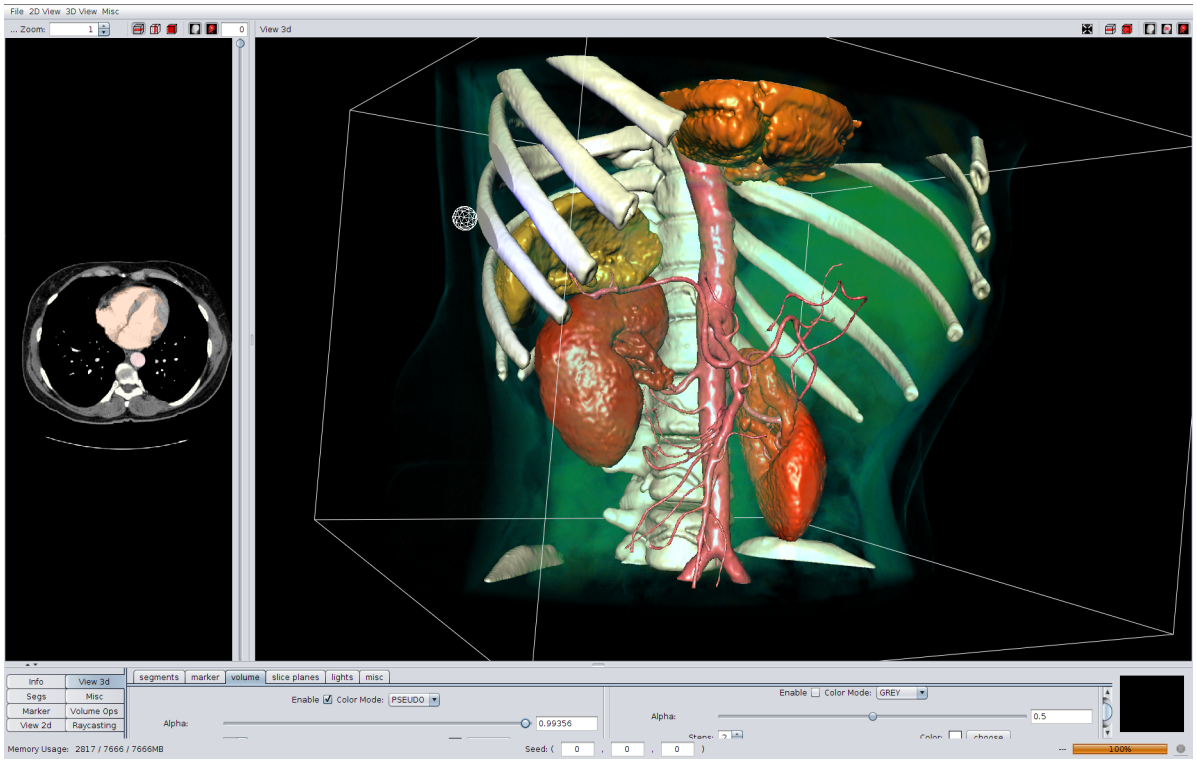


Figure 4.20: The Torso data set (source: our work [226])

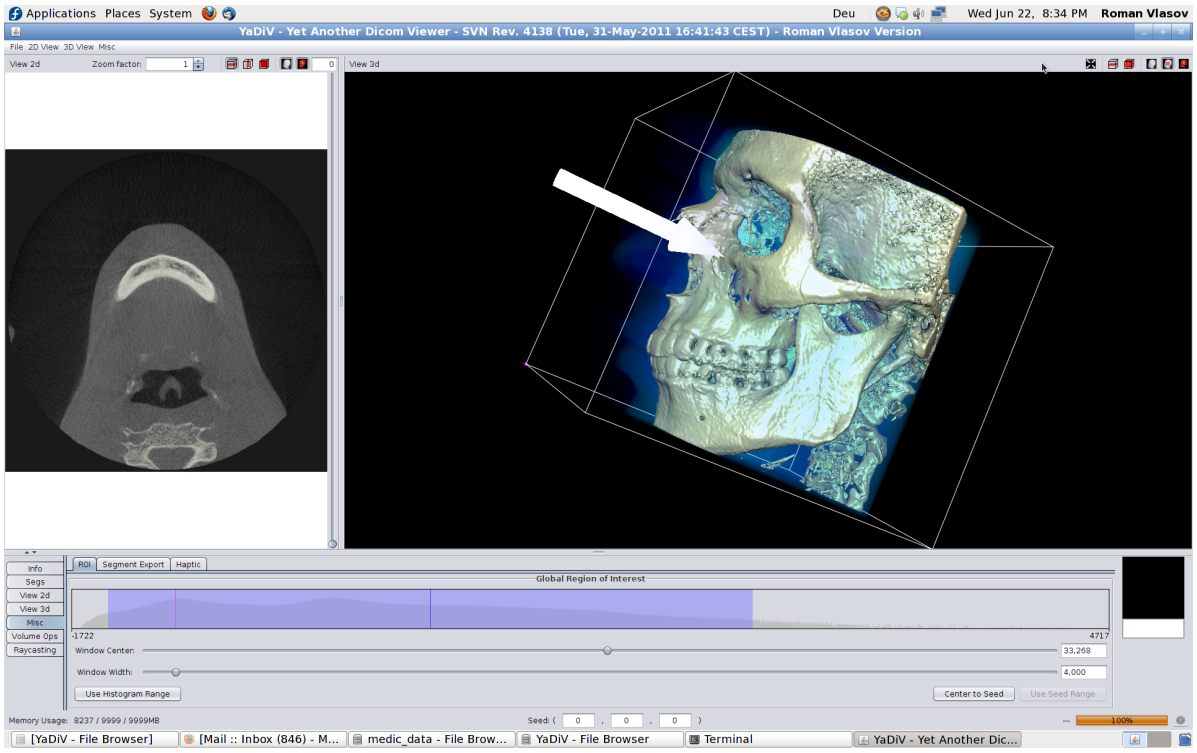


Figure 4.21: The data set Head_{big} (source: our work ([227])

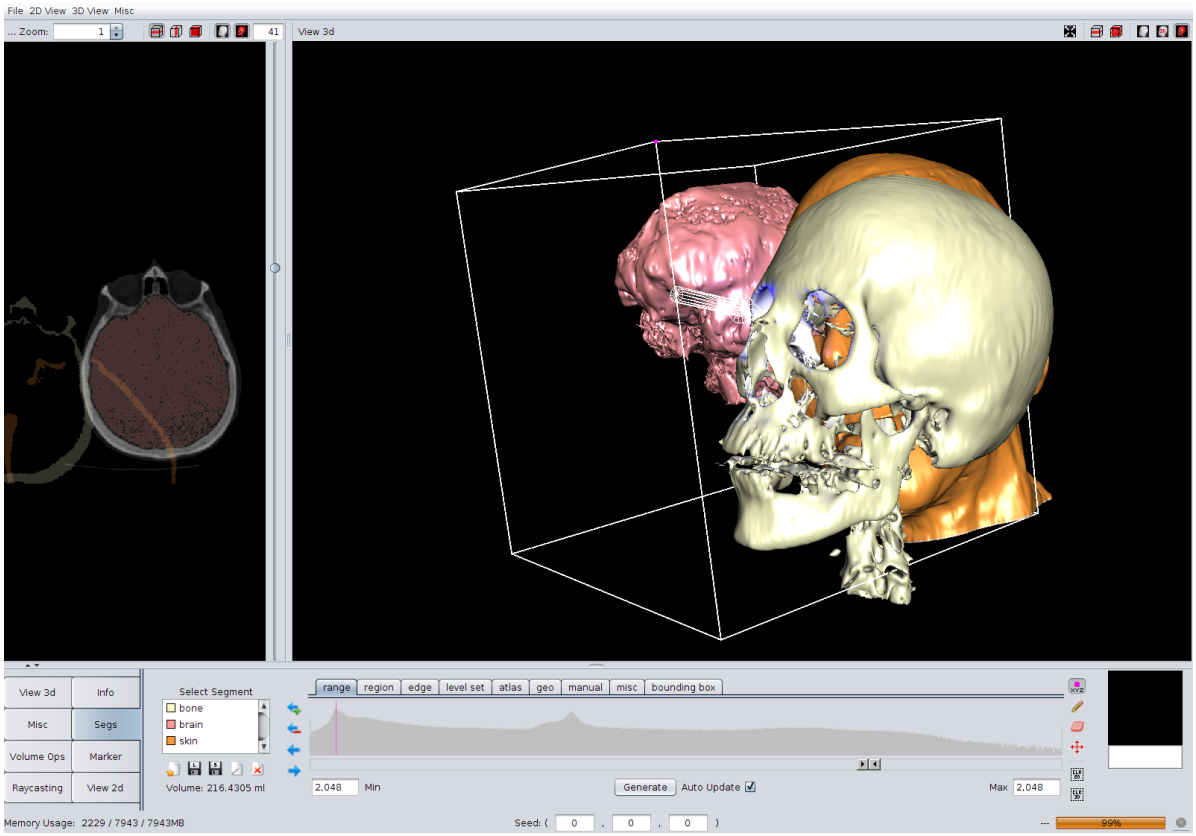


Figure 4.22: The data set Head$_{small}$ (source: our work [227])

Table 4.2: Resulting update rates for the Improved Approach

Data	**Size**	**Triangles**	**Update Rate**
Head_{small}	113x256x256	690k	146 kHz
Torso	520x512x512	2,222 Mi	134 kHz
Head_{big}	464x532x532	6,136 Mi	141 kHz

vice transformations and forces is required. All values for the data sets for the joint collision detection and response approach are shown in table 4.2. As in section 4.14, *triangles* denotes the number of triangles in the scene for the graphics rendering as a reference, and *Update Rate* is given for real devices and during the peak load.

Additionally, we would like to mention that the users of our prototype system with the improved haptic component reported about a better and more natural haptic experience.

4.16 Discussion and Future Outlook

We presented a new haptic rendering approach employing a novel collision detection technique based on ray casting concepts known from computer graphics. The approach was published in [227, 225]. The method gives collision detection guarantees that a manipulated object does not pass through "thin" obstacles and is never inside any of them while not requiring any special topological object structure. The collision detection was extensively tested with a new "slide along a surface" approach using an implicit surface representation "on the fly". The results confirm our approach to be a viable alternative to existing techniques avoiding most common drawbacks. The prototype was implemented as a plug-in of the YaDiV VR system and supports different haptic devices and operation systems.

Additionally, we presented an improved version of our haptic rendering approach. The improved approach has all properties of the original method (including an implicit surface representation "on the fly") and does not have the drawbacks described in section 4.9. It was published in [226]. The method employs local path finding and ray casting concepts and gives collision detection guarantees that a manipulated object does not pass through "thin" obstacles and is never inside any of them while not requiring any special topological object structure. Further on, we presented an improved force feedback generation scheme, which does not suffer issues of the original scheme given

in [227] and in section 4.6. The results show that our approach is a good alternative to existing techniques, while avoiding most common drawbacks. Furthermore, it contrasts most triangle-based approaches, where millions of triangles would be generated and complex speeding-up traversing structures are required for the collision detection with the same guarantees.

Our work shows that the path finding paradigm could be successfully employed in other research areas, such as haptic rendering in our case.

As an ongoing research, object-object interactions could be introduced, where the controlled object is represented as a set of points, and the collision detection stage could be implemented on GPUs. As was shown e.g. in [117], [196], ray casting could be efficiently parallelized using GPUs and/or multi-processor systems. This will allow making computations faster and therefore representing the controlled object with more points and/or performing a more sophisticated collision response. We plan to conduct the tests on the hardware which we already have at our Institute. It includes the high-end Tesla cluster granted by NVIDIA in the context of a Professor Partnership Program, modern graphics hardware including NVIDIA Fermi (GF 480), multi-core processor systems and an IBM Cell Cluster.

The practical use cases of our VR system could be assembling a fractured bone being an important step for pre-operation planning in facial surgery, putting landmarks for automatic segmentation and registration methods and correction of the results of automatic approaches (see section 5.28.1).

The next chapter is devoted to the advanced contact resolution. There we focus on a flexible framework which allows us to use our improved approach of haptic rendering of volume data presented in this chapter together with deformation models. We focus as well on the modified and improved method of potential fields.

Chapter 5

Our Deformation Framework and Deformation Approaches

In this chapter we present our flexible framework allowing us to use our improved approach of haptic rendering of volume data with collision detection guarantee which has been presented in chapter 4 together with deformation models. We show that it is feasible to use our previously developed haptic rendering approach together with a deformation model, since our approach adds its properties including collision detection guarantee and non-penetration guarantee to the selected deformation model. This is especially important for such delicate procedures as pre-operation planning. Furthermore, we present our graphics approach which we use to keep the graphics representation of segments up-to-date during the deformation simulation. The challenge here is to reflect deformations of objects interactively.

In order to validate our framework, we propose our local deformation simulation approach based on the method of potential fields (see remarks regarding the generalization of definition of deformation in section 5.4). As stated in section 5.3, potential fields can be considered as specific finite elements, i.e. discrete carriers of properties of the medium.

Furthermore, we introduce our novel cuboid potential fields (see remarks in section 5.18) and propose how to use them for the local deformation simulation. We demonstrate that cuboid potential fields are better suited to haptic rendering of volumetric data. Further on, we show how to establish the correspondence of parameters of our proposed deformation simulation models to parameters of real materials, and propose a way to take the heterogeneity of the simulated material into account. Furthermore, we extend

the classical potential fields approach in other aspects, such as adding additional forces and parameters to the model. Additionally, we introduce the prototype of the global potential fields based deformation approach.

The potential field based deformation simulation approaches are a good "illustration", because they initially do not have the "nice" properties of our haptic rendering approach presented in chapter 4. Additionally, the resulting combined haptic rendering approach with our proposed deformation simulation approaches within our deformation framework does not require any pre-calculated structure and works "on the fly". Furthermore, in this chapter we give the results of tests of our deformation framework and our deformation simulation approaches with real volumetric data.

The rest of the chapter is organized as follows. Firstly, we present the structure of our deformation framework and how our proposed deformation model fits there. Next, we present our graphics approach. Further on, we introduce the local potential fields approach and show how we improved it. After this, we show the correspondence of real world parameters to the parameters of the simulation model. Further on, we present our local cuboid potential fields approach. Further on, we present our prototype of the global deformation potential fields based simulation approach, which is later used for validation of how our deformation framework works with a global deformation simulation. And lastly, we show and discuss the results.

5.1 Our Deformation Framework

The scheme of our original multi-rate and multi-threaded prototype system is described in section 4.12. In order to simulate deformations, a new deformation simulation thread has been added, which works at about 30 Hz – see figure 5.1.

The structure overview diagram of our original prototype system being split in two parts is presented in section 4.12 and is shown in figures 4.18, 4.19. In order to simulate deformations, we updated the second part of the diagram and added new components to the new part of the diagram – part 3. See figures 5.2, 5.3.

Figure 5.3 shows that the part of our system devoted to simulation of deformations was designed in such a way that allows to add new deformation algorithms easily. Thus, we used "Abstract Defo Algorithm" class in the deformation loop and "Abstract Data for Defo Algorithm" class in the "DefoObject" class. In case of potential fields based approaches, every concrete deformation simulation algorithm uses the corresponding concrete "data for deformation algorithm" class and the concrete potential field class being inherited from the aforementioned abstract classes. Additionally, the "DefoObject" class has a member called "Defo Potential Fields Renderer" which is responsible for updates of its graphics representation of potential fields. This graphics representation is very useful for checking how do potential fields behave for the particular deformation algorithm. "Defo Potential Fields Renderer" is inherited from "CustomSyncObj" and updates the graphics representation of potential fields every iteration of the "Haptics To Graphics Synchronizer" thread. "Defo Potential Fields Renderer" takes the parameters for potential fields by accessing the list of "BasePotentialField" being stored at "DefoObject" and being the base class for concrete potential fields classes for concrete deformation simulation algorithms.

5.2 Update of Graphics Representation

An update of the graphics representation of a simulated object within our deformation framework is necessary for showing the results of the deformation simulation for the chosen deformation model within our prototype system. The challenge here is to reflect deformations of objects interactively.

If the chosen deformation model has its own internal representation of an object and it does not update the volumetric data, as it is for our potential fields approaches, then an additional step depending on the chosen deformation model is needed to update the

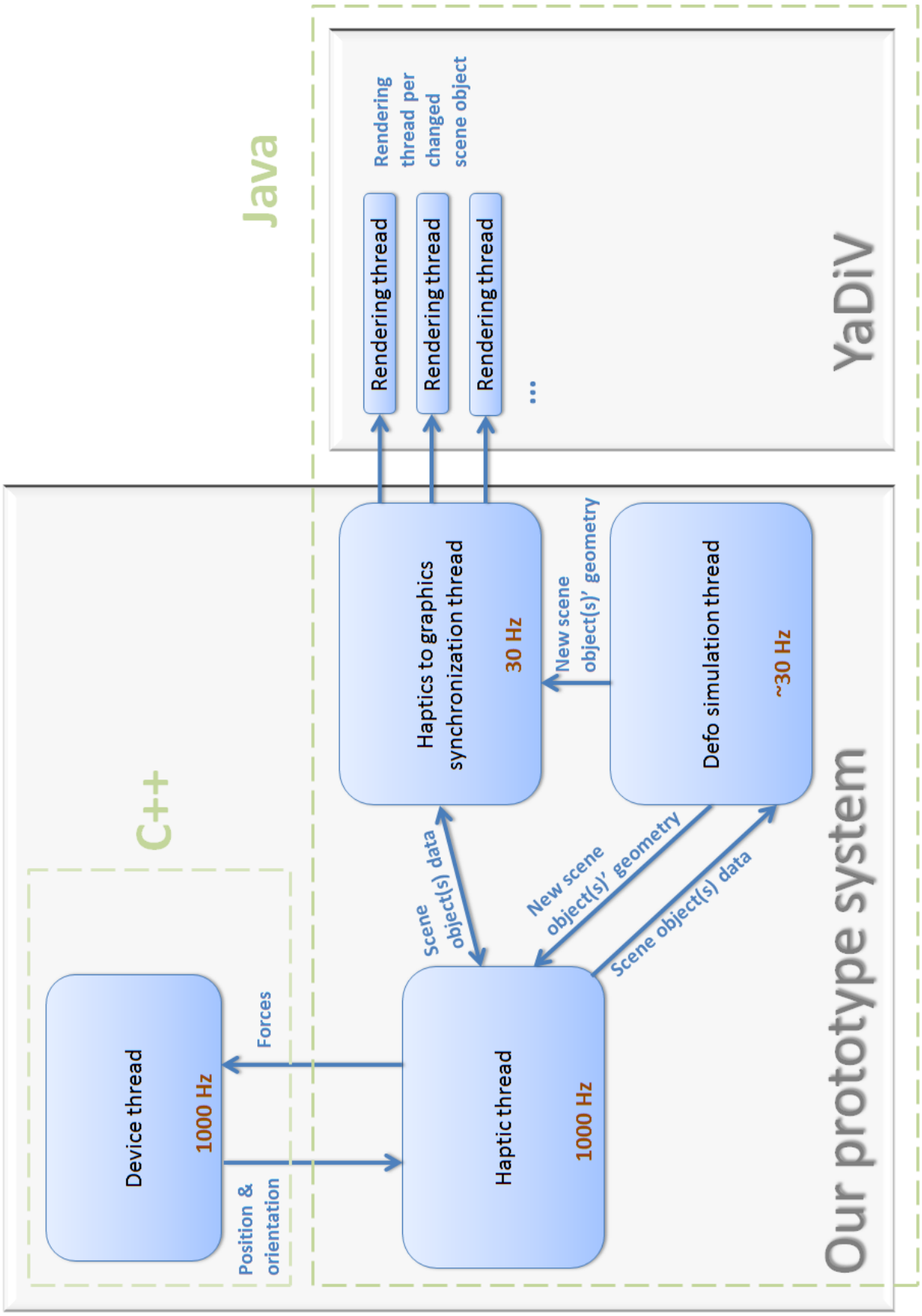


Figure 5.1: Our multi-rate and multi-threaded approach with added simulation of deformations

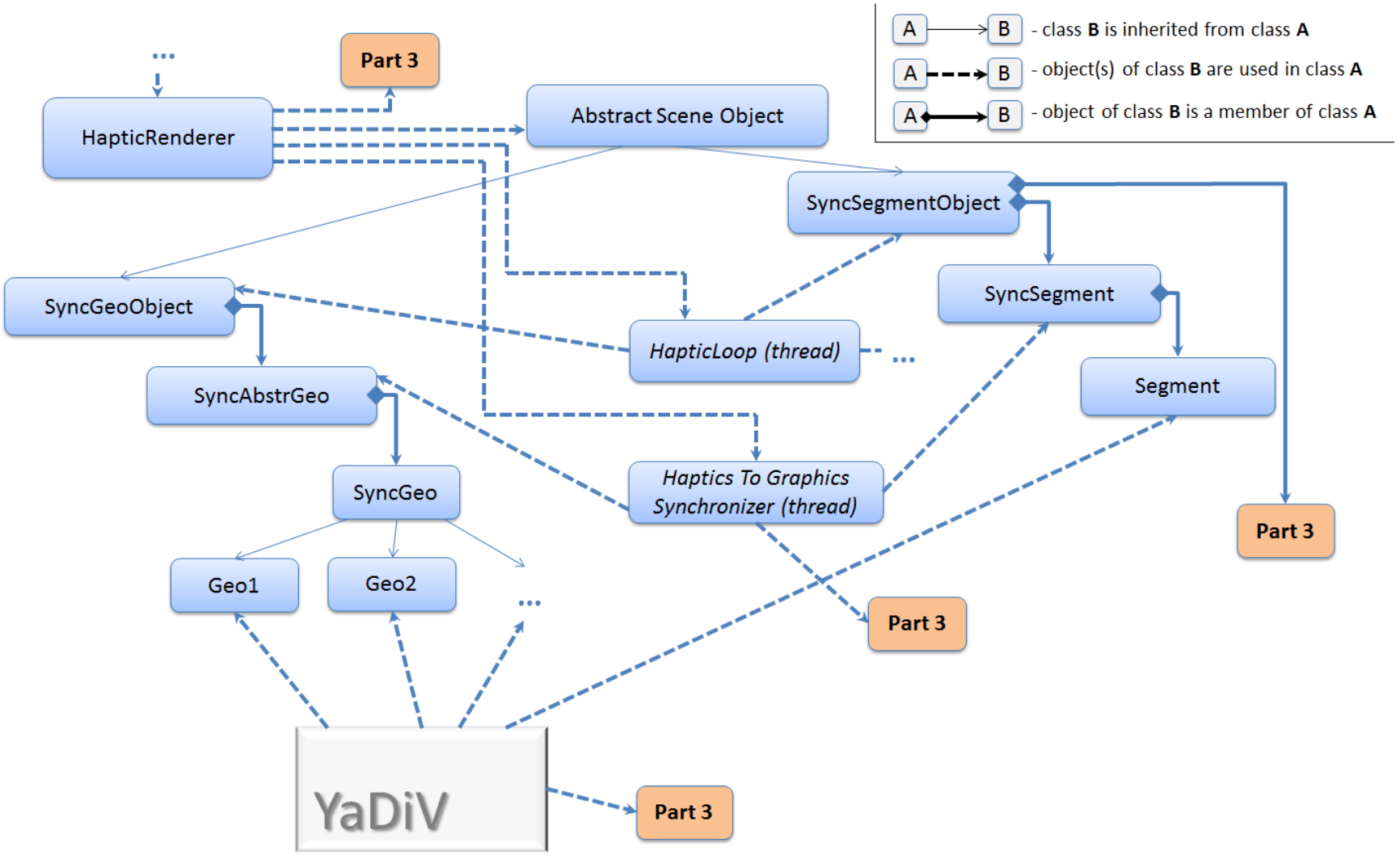


Figure 5.2: Part 2 of the structure diagram of our approach with added simulation of deformations: scene objects and their usage by threads

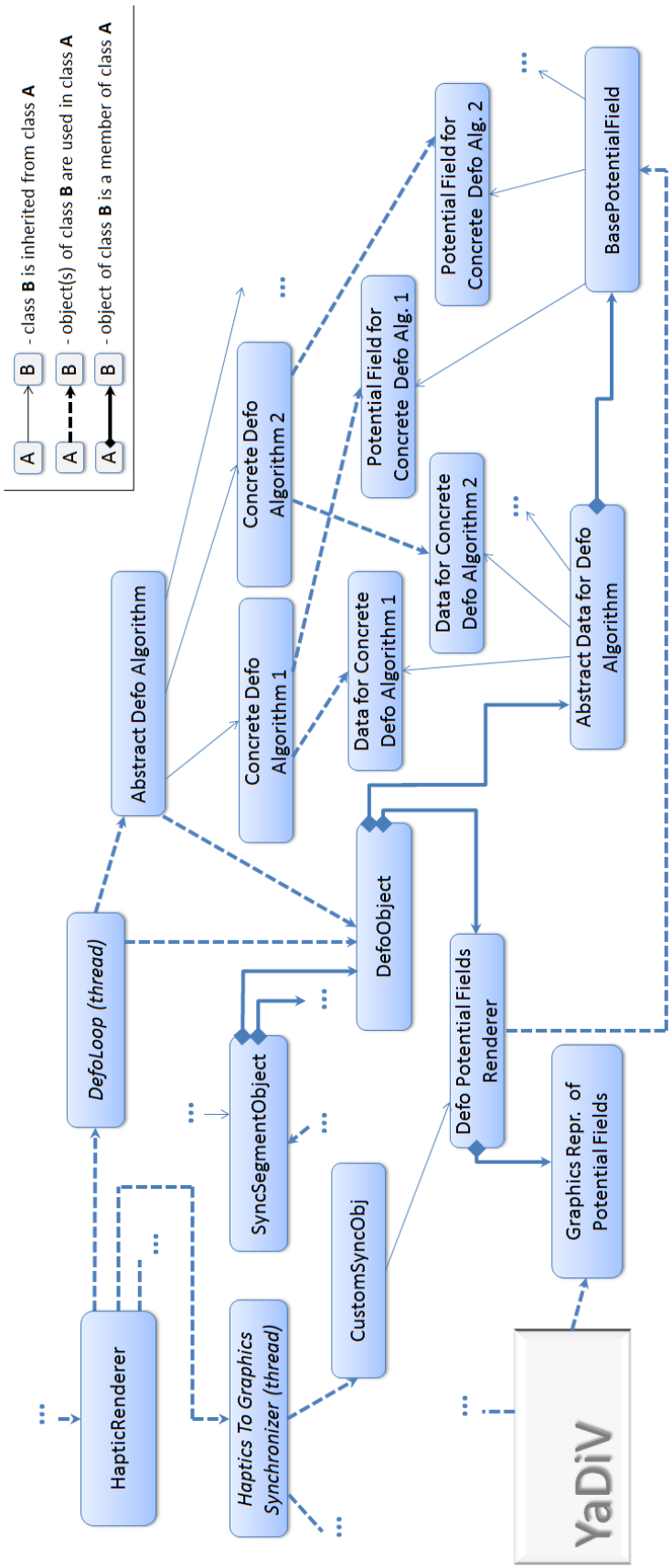


Figure 5.3: Part 3 of the structure diagram of our approach with added simulation of deformations: scene objects and their usage by threads, and deformation simulation algorithms

volumetric data from the internal representation of the chosen deformable model. For our potential fields approach such step is described in section 5.25.

In case of graphics direct volume rendering (see section 2.2.4), the changes in the areas of volumetric data being affected by the deformation simulation will be immediately reflected during the next rerendering.

In case of graphics surface rendering (see section 2.2.3), the graphics surface representation of the volumetric data should be updated after the volumetric data has been changed. Therefore an approach to update the surface representation from the volumetric representation is needed. The following sections are devoted to this issue.

5.2.1 Possible Solutions

As it was already mentioned in section 4.10, we use the YaDiV Open-Source platform [73]. The fastest graphics rendering in YaDiV is triangle-based, where triangles are obtained using a modified marching cubes algorithm (MMCA) (see [73] for details). In case of deformations the triangulation must be changed very fast. There are several approaches for that, including the following:

1. The resulting triangulation of any segment is currently represented in YaDiV using an indexed triangle list L. In order to do fast retriangulation with the MMCA at an area of the deformation it is necessary to use an additional data structure containing a map that links marching cube positions to the resulting triangle positions in L. In more detail, when there is a deformation, we need to do the following:

   1) Localize the area of the triangulation, which is affected by the deformation
   2) Remove all triangles affected by the deformation from L
   3) Rerun the MMCA for the deformed segment within the affected area
   4) Add the new triangles to L.

   L is represented using two arrays (triangle array and vertex array), so when the affected triangles are removed, the new triangles should firstly be put at their place. In case the number of the new triangles is greater than the number of those removed, the rest of the triangles are put at the end of the array. For that purpose, the array is initially created bigger than necessary by a specified factor

2. Firstly divide each segment into smaller sub-segments and do the triangulation using the MMCA for each of them so that triangles match on borders of sub-segments. Then, when there is a deformation of a segment, rerun the MMCA with the above boundary conditions for those sub-segments which were affected by the deformation. A disadvantage of this approach is that it can lead to mismatch of triangles at the edges of the sub-segment patches in case of smoothing.

In order not to create an additional data structure, to keep the core of the MMCA unchanged and to take advantage of parallelization of retriangulation of affected sub-segments on multi-processor machines, the second approach has been chosen. Furthermore, we eliminated the disadvantage of mismatch of triangles at the edges of sub-segment patches. The approach is described in detail in the following section.

5.2.2 Update for Marching Cubes

When the segment is loaded and visualized for the first time, we split it into sub-segment patches. The number of patches can vary depending on the resolution of the volumetric data. There are different ways to split the segment into the patches. We use a regular grid along each coordinate axis, and there is one patch per grid cell or no patches if the MMCA did not generate any triangles within the given grid cell.

During the deformation simulation, if the volumetric representation of segment is changed then the following will be done:

1. The corresponding area of the volumetric data will be invalidated (i.e. marked as needed rerendering)

2. After this, the rerender request will be set for the “SyncSegment” object of the “SyncSegmentObject”. The latter “SyncSegmentObject” is the one having the current “DefoObject” being responsible for deformations of the given segment as a member. See section 5.1 and figures 5.2 and 5.3 for details of relationship between the aforementioned objects

3. After the rerender request is set for the “SyncSegment” object, it will be processed during the next iteration of the haptics to graphics synchronization thread and the invalidated area will be retriangulated. Additionally, other necessary operations such as update of the axis-aligned bounding box of the segment will be performed. On slow machines often retriangulation of invalidated areas only can be too slow

to be interactive. In this case the update is done every k-th iteration of the haptics to graphics synchronization thread, and all the necessary data is cached between the iterations.

For the very first visualization of the segment we use the same algorithm as for the later updates of the graphics representation of invalidated areas during the deformation simulation. The difference is that for the very first visualization we invalidate the whole segment, so that all sub-segment patches are recomputed.

Further on, YaDiV uses a smoothing algorithm in order to smooth the triangulation generated by the MMCA. The difference between graphics rendering with and without smoothing is shown in figures 5.4 and 5.5. We should note here that the smoothing algorithm works with the surface representation, but not with the volumetric representation. Therefore for the purposes of haptic rendering it could be reasonable to turn off smoothing in order to see the real volumetric data. Additionally, the smoothing can be turned off in order to have faster rerendering on slow machines. In contrast, the smoothing can be useful e.g. in order to have better looking visuals or in case a haptic rendering approach incorporates a smoothing algorithm working on volumetric data.

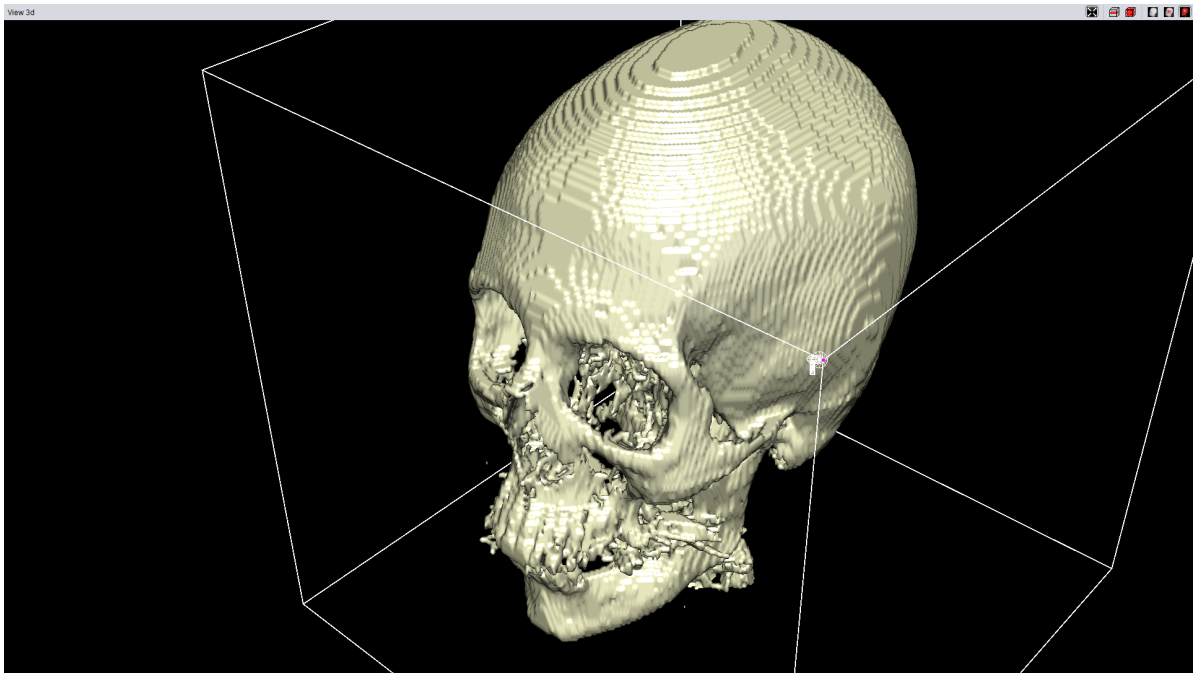
Details about the smoothing algorithm used in YaDiV can be found in [73]. The algorithm smoothes vertex positions and recalculates vertex normals. The algorithm has a degree of smoothing d, where $d = 0$ means no smoothing, and $d > 0$ indicates how many smoothing cycles will be done. Due to the specifics of the smoothing algorithm, d is also used to find the distance from the current vertex to other vertices being used for smoothing. See the listing of the algorithm presented in this section for more details.

We use two approaches for the smoothing of the retriangulated area:

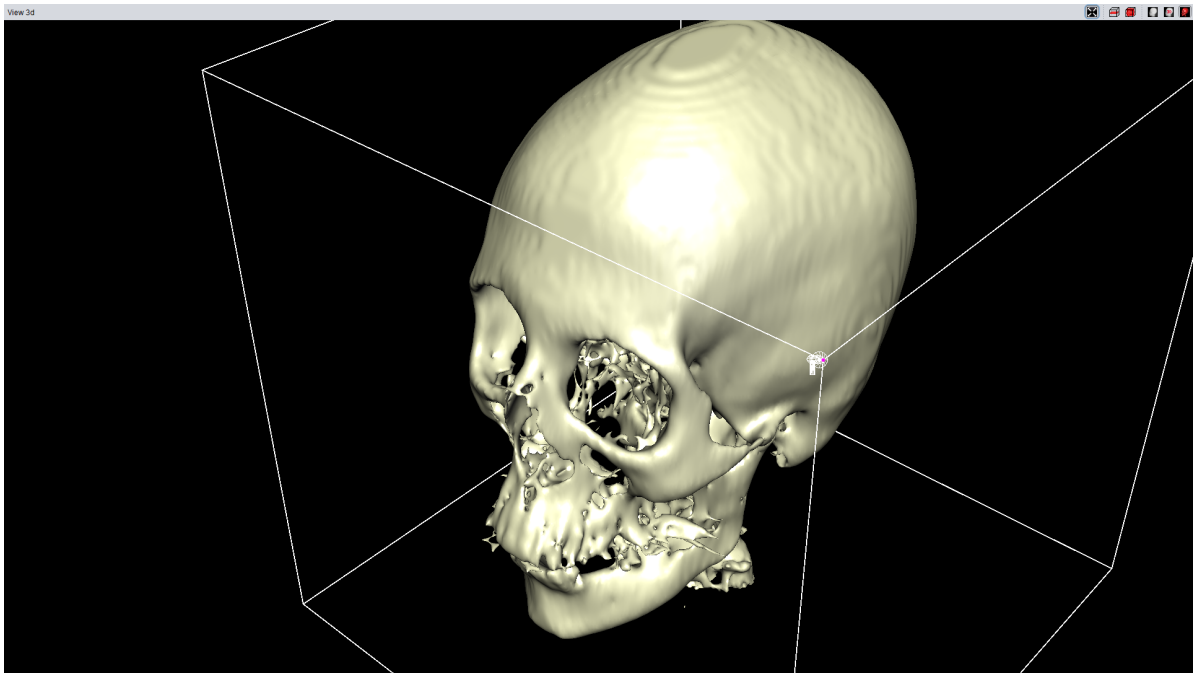
- The global smoothing – take into account surrounding sub-segment patches of each patch requiring smoothing
- The local smoothing – the smoothing is done for each sub-segment patch separately, in isolation from others.

The local smoothing works faster then the global smoothing, but can lead to mismatch of triangles at the edges of sub-segment patches. The global smoothing does not have this drawback but takes more time to compute. The visual difference between the local and the global smoothing algorithms is shown in figure 5.6.

The complete listing of our algorithm for fast segment retriangulation using the MMCA and for the smoothing of affected areas of the given segment during the deformation

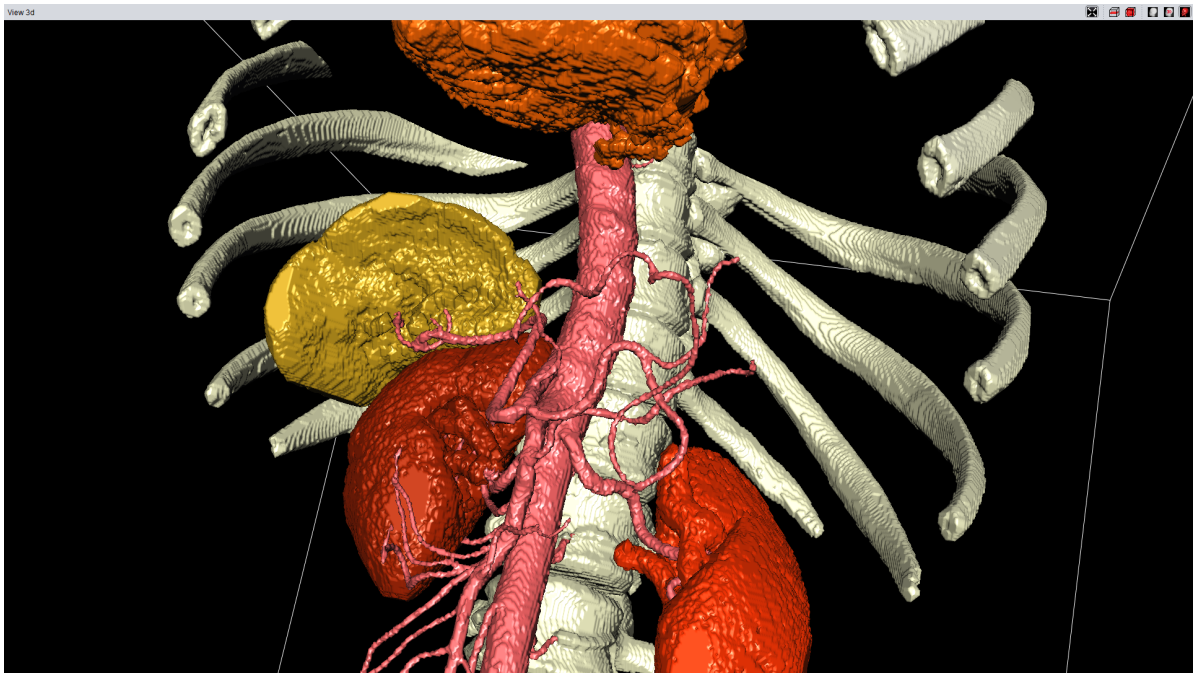


(a)

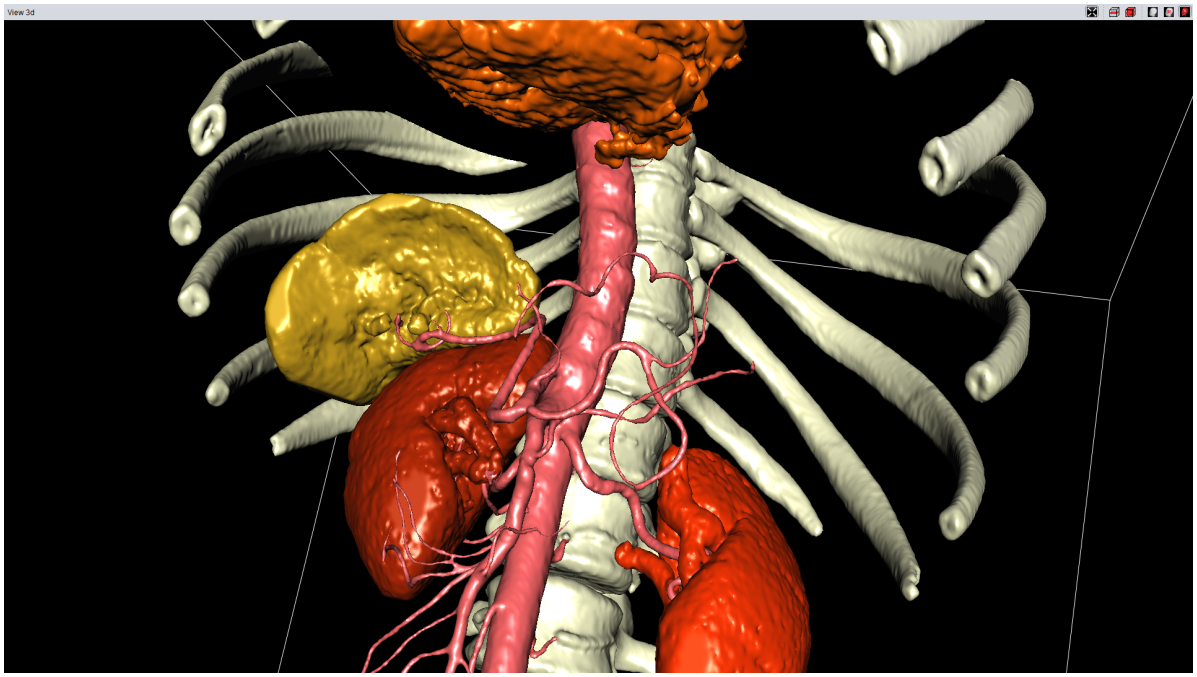


(b)

Figure 5.4: The difference between graphics rendering without (a) and with (b) smoothing. The data set Head_{small} is visualized

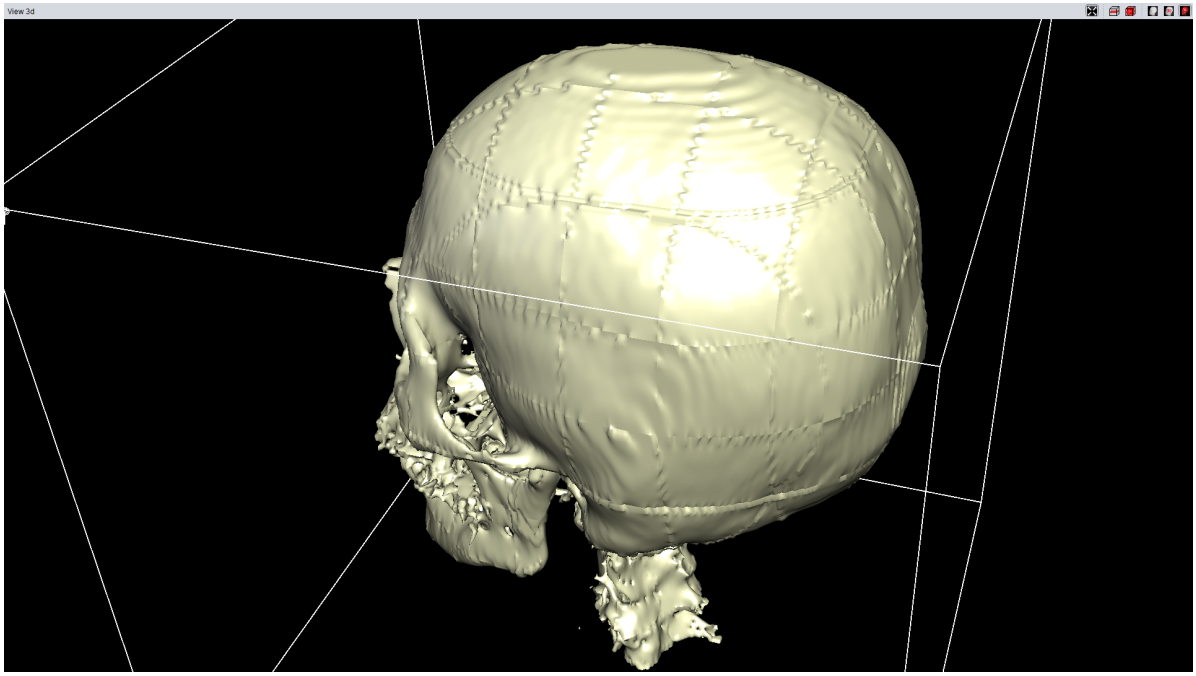


(a)

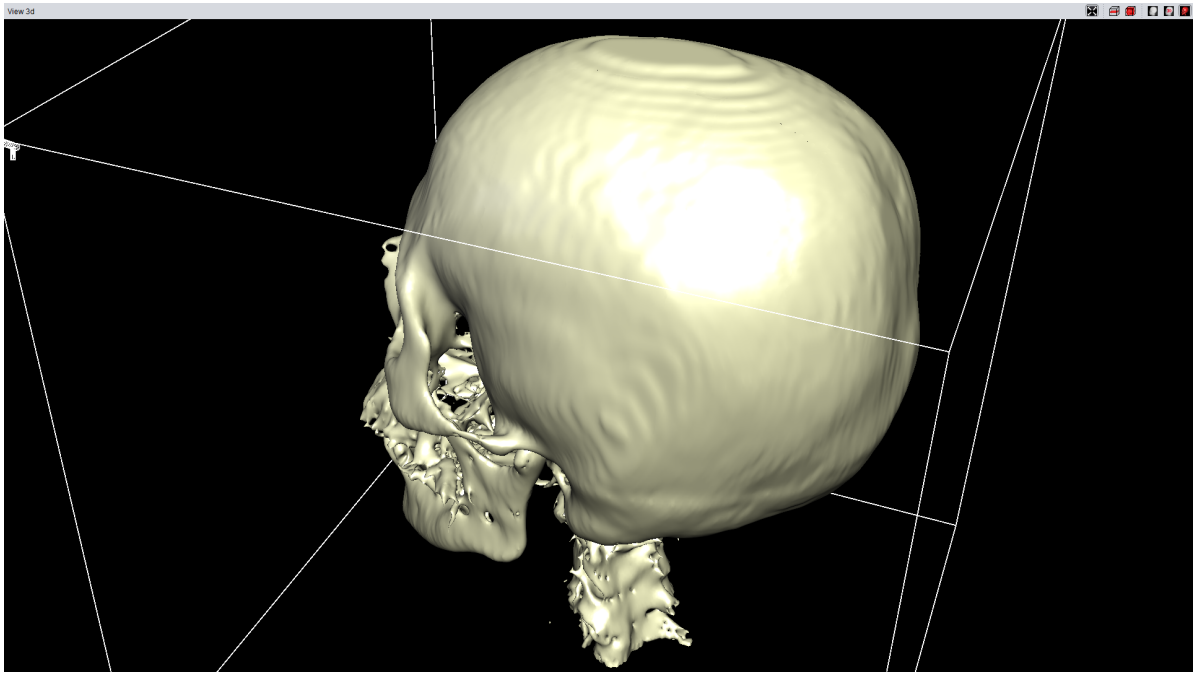


(b)

Figure 5.5: The difference between graphics rendering without (a) and with (b) smoothing. The Torso data set is visualized



(a)



(b)

Figure 5.6: The difference between the local smoothing algorithm (a) and the global smoothing algorithm (b)

simulation is presented below. The following denotations are used:
$dmc \in \mathbb{I}^+$ – marching cubes distance, that is the step (in voxels) used for the MMCA;
$isGlobalSmoothing$ – **true** if global smoothing is used, **false** if local smoothing is used;
A – the invalidated area of the given segment;
d – the above mentioned degree of smoothing for the smoothing algorithm. Due to specifics of the smoothing algorithm, it is equal to the minimum possible width of the zone affected by smoothing (measured from the original borders of the given sub-segment patch), divided by dmc.

1: // I. Find the affected patches $patchesToRecalc$ for the given A:
2: $patchesToRecalc := \emptyset$
3: **for** (each patch P within (A extended by the size of the patch along each coordinate axis)) **do**
4: // 1.
5: **if** ($isGlobalSmoothing =$ **true**) **then**
6: Extend the borders of P by $(d + 2) \cdot dmc$. Since this is done for every patch, it means that all patches will overlap for $(d + 2) \cdot dmc$. This ensures that the patches will be smoothed correctly using all the required neighbor information. We added "+2" to d in order to have a sufficient overlapping even in case of significant movements of vertices during smoothing and even in case of wrong rounding and computation errors. After the smoothing we will cut the patches to their original sizes (will be done in the later steps of the algorithm)
7: **end if**
8: // 2. Check whether the axis aligned bounding box (AABB) of P intersects with the AABB of A or not:
9: // 2.1
10: Calculate the first approximation of the AABB of P by using the dimensions of the grid cell corresponding to P
11: // 2.2
12: (AABB of P) := (AABB of P) $\cap$ (AABB of the whole segment)
13: // 2.3
14: **if** ((AABB of P) = $\emptyset$) **then**
15: // skip P because it is actually empty (has no non-empty voxels)
16: **continue**
17: **end if**
18: // 2.4
19: **if** ((AABB of P) $\cap$ (AABB of A) $\neq \emptyset$) **then**

20: *patchesToRecalc.add*(P)
21: **end if**
22: **end for**
23:
24: // II. Retriangulate the affected patches *patchesToRecalc*
25: **for** (each patch $P \in$ *patchesToRecalc*) **do**
26: // 1.
27: Create a geometry object for the current patch to keep the triangulation, if it was not created before
28: // 2.
29: Run the MMCA for P and store its results into the above created geometry object
30: **end for**
31:
32: // III. Do smoothing of all retriangulated sub-segment patches
33: **for** (each patch P within (A extended by the size of the patch along each coordinate axis)) **do**
34: // 1.
35: Do steps I.2.1–I.2.3
36: // We do the above steps because some patches could have become empty
37: // after retriangulation
38: // 2.
39: Run the smoothing algorithm for P
40: // 3.
41: **if** (*isGlobalSmoothing* = **true**) **then**
42: Cut (the new triangulation of P) to fit into original dimensions of (the grid cell corresponding to P), but with additional +/-*dmc* along each coordinate axis, because the points could have been moved a little during the smoothing process
43: **end if**
44: **end for**

The worst time complexity of the above algorithm in case of the local smoothing is the same as for the original MMCA with the original smoothing algorithm being initially used in YaDiV. Indeed, in the worst case, that is if the whole segment is invalidated, the same number of triangles will be created and smoothed, as in the original algorithm. The only additional work to be done is to find which patches should be recalculated (step I), which takes $O(N_{ap})$ time, where N_{ap} is the number of patches within the invalidated area of the given segment. Since N_{ap} is usually much less than the number triangles created by the original MMCA, we can omit the time complexity for step I when estimating the

time complexity of the whole algorithm.

The average time complexity of the above algorithm in case of the local smoothing, that is when only a certain area of the segment is invalidated, will be $O(N_{ap}/N_{patches})$ times the time complexity of the original algorithm. Here, $N_{patches}$ is the number of patches for the given segment. For instance, for our potential fields based deformation approach, only a few patches are getting invalidated during the interaction.

The time complexity of the above algorithm in case of the global smoothing is the same as for the local smoothing. Indeed, on step I.1 we extend the borders of P for the fixed constant number of voxels. This increases the execution time on steps II and III constant number of times. Therefore the time complexity of the algorithm will remain the same.

Let us find the space complexity of our algorithm. Instead of one patch for the whole segment in the original algorithm used in YaDiV, we need to store $N_{patches}$ smaller patches, but with the same amount of triangles in total and with the constant overhead for storing data structures per patch. Therefore the extra space complexity compared to the original algorithm is $O(N_{patches})$. Further on, we need to store the list of affected patches, which requires another $O(N_{patches})$ in the worst case. In case of the global smoothing, borders of patches are extended for the fixed constant number of voxels on step I.1 (and will be cut at the end of the algorithm, on step III.3). But since the borders are extended for the fixed constant number of voxels, the space complexity of the whole algorithm will increase for a constant which can be ignored. Therefore the extra space complexity of the above algorithm compared to the original algorithm used in YaDiV is $O(N_{patches})$. Since $N_{patches}$ is usually much lesser than the number triangles created by the original MMCA, we can consider the space complexity of our algorithm to be the same as for the original algorithm.

In the description of the above algorithm we omitted some technical details for clarity, e.g. synchronization issues. Thus, the execution of step I and the execution of steps II and III should be synchronized using $patchesToRecalc$, because in practice step I is performed in another thread.

A possible improvement to the presented algorithm can be an adaptive sub-division of the given segment into sub-segment patches: the more non-empty voxels there are in the particular region of the segment, the more patches should be used for this region.

5.3 Introduction to Potential Fields Approach

There are several groups of simulation approaches discretizing an object or its area as a set of material points and performing computations with them. One of such simulation paradigms being popular nowadays is Smoothed Particle Hydrodynamics (SPH) based on Navier–Stokes equations and being described/used e.g. in [61, 159, 151, 155, 48, 36, 213]. A disadvantage of these approaches arise from this paradigm – they are mostly suitable for simulations of gases, liquids and highly deformable bodies, but may be not so good for rigid or deformable objects, although some nice approaches were presented in [46, 47].

There are also other particle-based methods employing various techniques. Tonnesen [218] used dynamically coupled particle systems employing the Lennard-Jones potential for geometric modeling, melting, tearing and surface reconstruction, as well as surfaces represented using oriented particles. Neither empty space inside an object nor "small" cuts are allowed. Baudet et al. [21] used the Lennard-Jones potential based simulation of a particle system to track changes in the object's shape from some partial information provided by an ultrasound sensor. The authors of [154] presented a shape matching approach being applied to an Euler integration scheme. Later on, in [152] the authors extended the approach to oriented particles. De et al. [56] proposed a haptic rendering method, which uses a point-associated finite field approach. The idea is to discretize a computational domain (an organ) using a scattered set of points ("nodes") with a spherical influence zone with defined nodal shape function. The approach is a combination of mass-spring and FEM-based techniques, although it is "is vastly simplified compared to the FEM" [56]. Wicke et al. [240] presented a method for computing elastic strain without storing rest states or a connectivity, and the strain state of each particle is computed by comparing the actual positions of the neighboring particles to their assigned lattice positions. Harada et al. [90] presented a particle-based simulation using GPU. They showed the simulation of fluids using SPH and the simulation of rigid bodies approximated by a set of spheres.

We address the simulation problem from the point of view of theoretical mechanics, and therefore use the paradigm of energy equations and potential fields. For the potential fields approach, an object (or its area) is discretized using the set of potential fields with associated material points, which interact with each other and are under the influence of external forces and constraints - see e.g. [116] for details.

We would like to note that the potential fields method differs from SPH. As stated in [116], for SPH particles are used as a numeric approach to integrate continuous equations

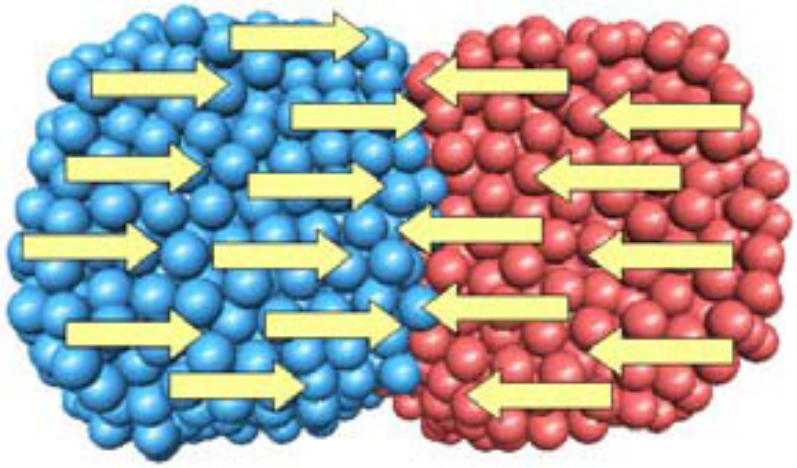


Figure 5.7: The potential fields based simulation of powder grains used in printing (source: [80])

of dynamics of continuous media. For the potential fields method, equations of motion of centers of potential fields, being defined by the balance of momentum and by the interaction potential between the material points, are taken as the basis. That is, the method is "truly" discrete. Additionally, Kuzkin, Krivtsov et al. [120, 121] showed that SPH and the potential fields method are different in the respect that for SPH viscosity is specified explicitly. Furthermore, the authors wrote that it is not rare that computational artifacts appear for the SPH.

The potential fields method is used in a wide variety of simulations, and one can mark out the following related works.

Krivtsov et al. [116] used the potential fields approach for calculation of mechanics of deformable solids and for finding the relationship between micro– and macroscopic parameters. Amrani et al. [26] proposed a 3D reconstruction methodology using the method based on multilayer (bigger–smaller) potential fields systems using the Lennard-Jones potential function. Its key idea is to put small potential fields in the areas where details are required, and big ones everywhere else. Further on, as shown in papers described below, Krivtsov and others scientifically proved that many properties of objects can be simulated with a good agreement to real experimental results and with high precision using methods of potential fields without any extra parameters (the methods of potential fields are sometimes called methods of particle dynamics, where particles are potential fields with associated material points). Such, Gilabert, Krivtsov et al. [80] presented results for simulation of polymer particles (powder grains) used in xerographic and printing industries (see figure 5.7). Use of the Lennard-Jones potential showed a

good agreement with the real compression tests using specimens of polystyrene. This work was continued in [81], where the authors additionally used different levels of potential "granularity" to model adhesive interaction force between two cohesive polymer grains, and a good agreement with the elastic contact theories has been obtained. Further on, the authors of [74] presented the hypothesis of origin of the Earth-Moon system being simulated with the potential fields method. Indeitsev et al. [99] presented an analysis of the relation between the spall strength and strain rate for solids using the potential fields approach. As mentioned by the authors, in classical molecular dynamics particles are atoms, whereas for the potential fields approach they can also be associated with other structural elements such as grains of the material, or be used as specific finite elements, i.e. discrete carriers of properties of the medium. The authors used Lennard-Jones potentials. They reported that the computer material considered under spall fracture showed properties close to the properties of real materials. They mentioned that the results ensure satisfactory agreement with the experimental data. Hou et al. [95] employed the Lennard-Jones potential and haptic rendering for biomolecular docking. Podolskaya, Krivtsov et al. [185] did an analysis of stability and structural transition in the FCC lattice under large deformations. They used the Morse potential for the computer modeling using the potential fields approach. Kuzkin, Krivtsov et al. [120, 121] presented a computer simulation of effective viscosity of fluid-proppant mixture used in hydraulic fracturing (see figure 5.8). Both SPH and the potential fields approach were used in order to have more reliable results. For the potential fields approach, the spline potential was used. The authors wrote that the results are in a good agreement with the experimental study, are reliable and can be used for setting effective viscosity of the mixture for practical tasks related to hydraulic fracturing.

To our best knowledge, there are no works using the paradigm of potential fields being discussed above for haptic rendering of deformable objects, especially for the local deformation simulation. Additionally, our local simulation approach works "on-the-fly" and does not put any requirements on topology as e.g. [218].

5.4 Characteristics of Potential Fields Approach

As mentioned in [116], the potential fields approach requires less apriori assumptions about material properties compared to continuous methods, such as the FEM. It allows to model complex properties using even simple potential fields, and many effects such as plasticity and fractures can be gotten "automatically".

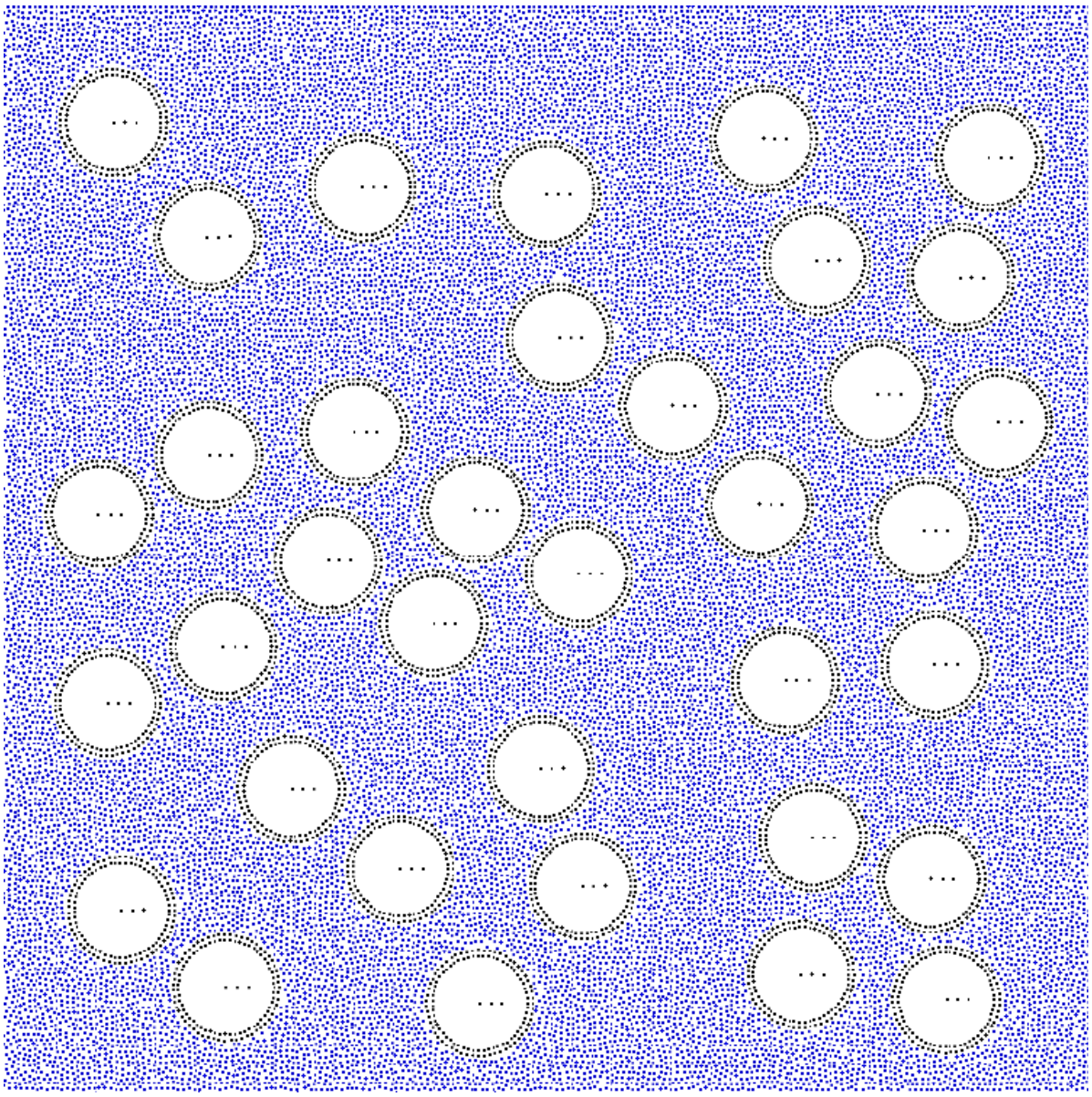


Figure 5.8: The distribution of fluid and proppant particles – simulated using potential fields approach (source: [120])

In contrast to the FEM, the potential fields approach can easily model a discontinuous surface and topological changes, as well as breaks and fractures. Additionally, the potential fields approach allows to handle self-collisions. Further on, during the deformation simulation object topology can be modified by the tool manupulated by the user, which can have different forms by changing the potential field associated with the tool.

Remarks:
Following the above discussion, we generalize the definition of deformation to be not necessarily a diffeomorphism. That is, the deformed solid should not necessarily be diffeomorphic to the original one. The deformed solid can have topological changes, such as it can have new holes or it can be split into several solids.

Moreover, the potential fields approach is well-scalable just by adding more potential fields. Further on, it can be parallelized well – see section 5.22 for details.

In the following sections we give the explanation of the classical method of potential fields, as well as present our deformation simulation approaches. In the explanation of the classical method of potential fields, we follow the works [116] and [115].

5.5 Equations of Motion

In the below sections we will use the bold font (e.g. $\mathbf{r}$ and $\boldsymbol{\chi}$) for vectors and the non-bold font (e.g. r and Π) for scalar values.

Following [116], let us consider N potential fields with associated material points. These potential fields have pairwise interactions with each other. In case of a closed (conservative) system, it follows from the energy balance equation that for each potential field i the force acting on other potential fields equals the sum of forces acting from the other potential fields on the potential field i:

$$m_i \ddot{\mathbf{r}}_i = \sum_{j=1,\neq i}^{N} f\left(\|\mathbf{r}_{ij}\|_2\right) \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|_2}, \tag{5.1}$$

where
m_i – mass of material point associated with the potential field i;
$\mathbf{r}_i$ – vector of position of center of potential field i;
$\mathbf{r}_{ij}$ – vector from the center of potential field i to the center of potential field j;

$f(r)$ – scalar value of interaction force $\mathbf{f}$ between the potential fields, which is defined as

$$f(r) \stackrel{\text{def}}{=} -\Pi'(r), \tag{5.2}$$

where
$r \stackrel{\text{def}}{=} \|\mathbf{r}\|_2$ – distance between the centers of potential fields;
$\Pi(r)$ – **interaction potential** (scalar value). We will consider various interaction potentials in the next sections.

If an external conservative force field $\boldsymbol{\chi}(\mathbf{r}_i)$ (vector), such as gravity, has to be added to the system, then equation 5.1 will transform into the following:

$$m_i\ddot{\mathbf{r}}_i = \sum_{j=1,\neq i}^{N} f\left(\|\mathbf{r}_{ij}\|_2\right) \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|_2} + \boldsymbol{\chi}(\mathbf{r}_i). \tag{5.3}$$

In case of a nonconservative system, that is if modeling of dissipation and/or modeling of energy supply/removal is needed, an external nonconservative force field $\boldsymbol{\psi}\left(\mathbf{r}_i, \mathbf{v}_i\right)$ (vector) and nonconservative components $\Psi\left(\|\mathbf{r}_{ij}\|_2, \|\mathbf{v}_{ij}\|_2\right)$ of pairwise interaction (scalars) are added to the right part of equation 5.3:

$$m_i\ddot{\mathbf{r}}_i = \sum_{j=1,\neq i}^{N} f\left(\|\mathbf{r}_{ij}\|_2\right) \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|_2} + \sum_{j=1,\neq i}^{N} \Psi\left(\|\mathbf{r}_{ij}\|_2, \|\mathbf{v}_{ij}\|_2\right) \mathbf{r}_{ij} + \boldsymbol{\chi}(\mathbf{r}_i) + \boldsymbol{\psi}\left(\mathbf{r}_i, \mathbf{v}_i\right), \tag{5.4}$$

where
$\mathbf{v}_i \stackrel{\text{def}}{=} \dot{\mathbf{r}}_i$ – velocity of potential field i;
$\mathbf{v}_{ij} \stackrel{\text{def}}{=} \mathbf{v}_j - \mathbf{v}_i$.

Remarks:
Further on we will use term *position/velocity of potential field* meaning position/velocity of position/velocity of center of the potential field.

An external nonconservative force field $\boldsymbol{\psi}\left(\mathbf{r}_i, \mathbf{v}_i\right)$ is usually used for adding force in vicinity of specified surfaces, as well as for energy removal using dissipation.

From the mathematical point of view, modeling of interactions for the described system is a solution of the Cauchy problem for equations 5.4. Initial positions and velocities are set according to the given task.

There are different ways of numerical integration of the equations of motion 5.4. As stated in [115], for the method of potential fields it is necessary to integrate a lot of

equations putting some requirements on memory consumption and computation time. Further on, most of computation time goes for calculation of the force acting on the material point of the current potential field (right-hand side of equations 5.4). The reason for this is that the force is significantly non-linear and that there is a big number of summands (mainly interaction forces with neighbor potential fields). This reduces the effectiveness of methods requiring repeated calculation of the right-hand side of equations 5.4. This is one of the reasons why the Runge-Kutta method is rarely used in the method of potential fields.

Since we want to model structures of large volume or with high level of detail, one should choose numerical integration methods taking less computation time, such as Verlet integration [223] or finite difference method [93]. Further on, one can use rectangle methods or trapezoidal rule [107] requiring less computational resources.

For more details we refer an interested reader to the comprehensive overview of the numerical integration of equations of motion for the method of potential fields which has been presented in [115].

5.6 Interaction Potentials

Let us denote a pairwise interaction potential as $\Pi(r)$. According to equation 5.2, the force corresponding to this potential equals to $f(r) \stackrel{\text{def}}{=} -\Pi'(r)$.

Let us define σ, a and b as the distances between centers of two potential fields for which:

$$\Pi(\sigma) \equiv 0, \quad \Pi'(a) \equiv -f(a) \equiv 0, \quad \Pi''(b) \equiv -f'(b) \equiv 0. \tag{5.5}$$

As in [115], *further we consider only interaction potentials which have only one solution for 5.5 and for which* $\sigma < a < b$. That is, further we consider potentials having the following properties:

- if two potential fields (their centers) get closer to each other ($r < a$) then they repulse;
- if two potential fields get further from each other ($r > a$) then they gravitate (attract);
- if $a < r < b$ then the attraction force increases. Starting from $r = b$ and further the attraction force gets weaker, so that for larger r both the interaction potential

and the interaction force converge to 0, and for $r > 2a$ they are already small.

Remark:
The statement that for $r > 2a$ both the interaction potential and the interaction force converge to 0 is correct only for interaction potentials being symmetric in all directions.

An example of such interaction potential and the corresponding force is shown in figure 5.9.

Definition:
Distance a is called an **equilibrium distance**. It is also called **length of linkage**.

Definition:
Distance b is called a **critical distance**, because it is the distance between the centers of potential fields when the linkage breaks.

Remark:
We should note that in the system with more than two interacting potential fields the equilibrium and critical distances are insignificantly different. We cover this topic in more detail in the sections devoted to the correspondence between parameters of the simulation model and real parameters of materials.

Let us introduce additional useful characteristics of interaction potentials.

Definition:
D is an **energy of linkage**. It is defined by the following equation:

$$D \stackrel{\text{def}}{=} |\Pi(a)|. \tag{5.6}$$

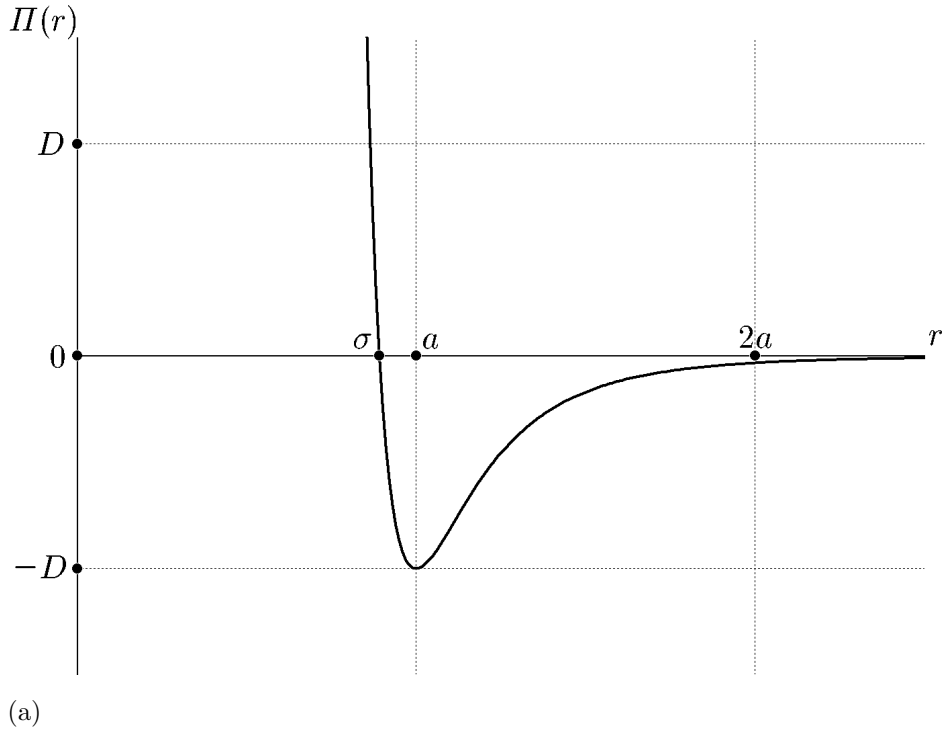
Definition:
f_* is a **strength of linkage**, that is the maximum absolute value of the interaction force f. It is defined as follows:

$$f_* \stackrel{\text{def}}{=} |f(b)|. \tag{5.7}$$

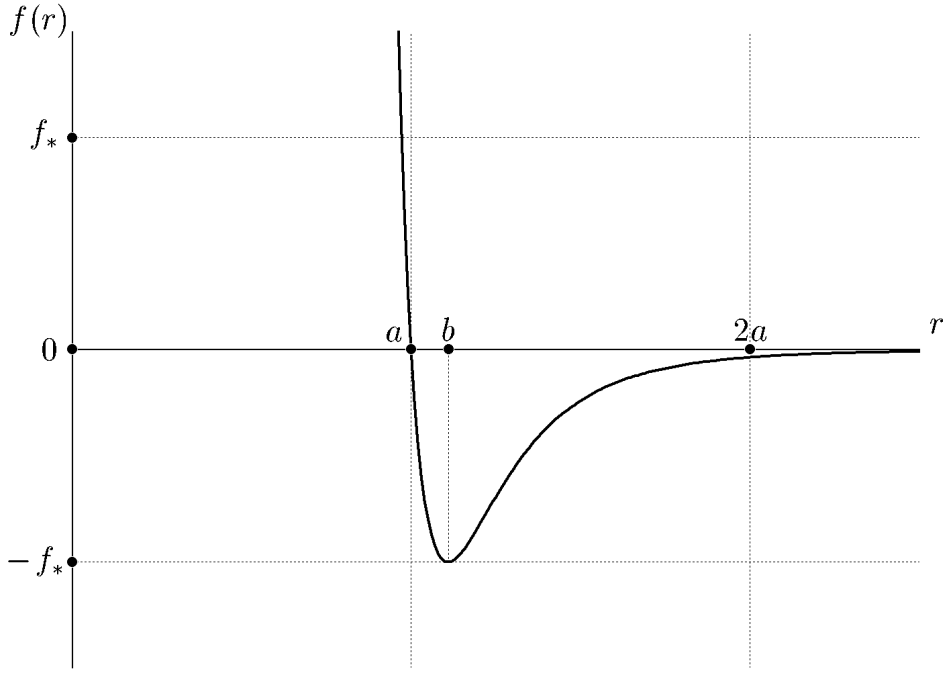
Definition:
C is a **stiffness of linkage** in the equilibrium position. It is defined as

$$C \stackrel{\text{def}}{=} \Pi''(a) \equiv -f'(a). \tag{5.8}$$

Definition:
ϵ_* is a **percentage elongation of linkage** when it breaks. It is also called an **ultimate**



(a)



(b)

Figure 5.9: General interaction potential and the corresponding interaction force for $\sigma < a < b$ (source: [116])

strain. It is defined by the following equation:

$$\epsilon_* \stackrel{\text{def}}{=} \frac{b-a}{a}. \tag{5.9}$$

Let us denote a linearized interaction force f_L as

$$f_L(r) = C(a-r). \tag{5.10}$$

Definition:
k_* is a **nonlinearity factor of linkage**. It is defined as follows:

$$k_* \stackrel{\text{def}}{=} \frac{f_L(b)}{f(b)} = \frac{C(b-a)}{f_*}. \tag{5.11}$$

Definition:
k_v is a **dynamics factor**. It is expressed as follows:

$$k_v = \sqrt{-\frac{a^2 \Pi''(a)}{2\Pi(a)}}. \tag{5.12}$$

The dynamics factor characterizes how fast the perturbation in the material consisting of potential fields is propagated compared to the critical propagation speed causing distruction of the material (such as the speed of dissociation). The higher is k_v, the higher is the fragility of the material. See [115] for more details.

In order to speed-up computations of interactions, the interaction potential is usually cut at the **cut distance** a_{cut}. That is, if centers of potential fields are further than a_{cut} then the interaction force is considered to be 0. Usually $a_{cut} = 2.1a$ because for $r > 2a$ both the interaction potential and the interaction force converge to 0 (see the beginning of this section).

5.7 Commonly Used Interaction Potentials

There is a number of commonly used interaction potentials (see [115]), including those being discussed below.

5.7.1 Lennard-Jones Potential

The equation for this interaction potential is

$$\Pi(r) = D\left(\left(\frac{a}{r}\right)^{12} - 2\left(\frac{a}{r}\right)^{6}\right), \tag{5.13}$$

where
D is the energy of linkage;
a is the equilibrium distance.

Therefore the corresponding interaction force is the following:

$$f(r) = \frac{12D}{a}\left(\left(\frac{a}{r}\right)^{13} - \left(\frac{a}{r}\right)^{7}\right). \tag{5.14}$$

Additional characteristics of the potential are as follows.

From 5.6 it follows that D is an energy of linkage. According to the equation 5.7, the strength of linkage is:

$$f_* = \frac{504}{159}\sqrt[6]{\frac{7}{13}}\frac{D}{a}. \tag{5.15}$$

Following 5.8, the stiffness of linkage is equal to

$$C = 72\frac{D}{a^2}. \tag{5.16}$$

According to 5.9 the percentage elongation of linkage when it breaks is:

$$\epsilon_* = \sqrt[6]{\frac{7}{13}} - 1 \approx 0.109. \tag{5.17}$$

Similarly, according to 5.11 the nonlinearity factor of linkage is:

$$k_* = \frac{169}{7}\sqrt[6]{\frac{7}{13}} \approx 2.910. \tag{5.18}$$

Following 5.12, the dynamics factor is equal to

$$k_v = 6. \tag{5.19}$$

5.7.2 Mi Potential

This is a generalization of the Lennard-Jones potential. The equation for this interaction potential is

$$\mathit{\Pi}(r) = \frac{D}{n-m}\left(m\left(\frac{a}{r}\right)^{n} - n\left(\frac{a}{r}\right)^{m}\right), \tag{5.20}$$

where m and n are additional dimensionless parameters, and $m < n$.

Therefore the corresponding interaction force is the following:

$$f(r) = \frac{D}{a}\frac{nm}{n-m}\left(\left(\frac{a}{r}\right)^{n+1} - \left(\frac{a}{r}\right)^{m+1}\right). \tag{5.21}$$

Additional characteristics of the potential are as follows.

According to the equation 5.7, the strength of linkage is:

$$f_* = mn\frac{D}{a}\sqrt[n-m]{\frac{(m+1)^{m+1}}{(n+1)^{n+1}}}. \tag{5.22}$$

Following 5.8, the stiffness of linkage is equal to

$$C = mn\frac{D}{a^2}. \tag{5.23}$$

According to 5.9 the percentage elongation of linkage when it breaks is:

$$\epsilon_* = \sqrt[n-m]{\frac{n+1}{m+1}} - 1. \tag{5.24}$$

Similarly, according to 5.11 the nonlinearity factor of linkage is:

$$k_* = \epsilon_*\sqrt[n-m]{\frac{(m+1)^{m+1}}{(n+1)^{n+1}}}. \tag{5.25}$$

Following 5.12, the dynamics factor is equal to

$$k_v = \sqrt{\frac{mn}{2}}. \tag{5.26}$$

5.7.3 Morse Potential

The equation for this interaction potential is

$$\Pi(r) = D\left(e^{-2\alpha(r-a)} - 2e^{-\alpha(r-a)}\right), \tag{5.27}$$

where α is an additional dimensionless parameter.

Therefore the corresponding interaction force is the following:

$$f(r) = 2\alpha D\left(e^{-2\alpha(r-a)} - e^{-\alpha(r-a)}\right). \tag{5.28}$$

Additional characteristics of the potential are as follows.

According to the equation 5.7, the strength of linkage is:

$$f_* = \frac{\alpha D}{2}. \tag{5.29}$$

Following 5.8, the stiffness of linkage is equal to

$$C = 2\alpha^2 D. \tag{5.30}$$

According to 5.9 the percentage elongation of linkage when it breaks is:

$$\epsilon_* = \frac{1}{\alpha a} \ln 2. \tag{5.31}$$

Similarly, according to 5.11 the nonlinearity factor of linkage is:

$$k_* = 4 \ln 2. \tag{5.32}$$

Following 5.12, the dynamics factor is equal to

$$k_v = \alpha a. \tag{5.33}$$

5.7.4 Composite potentials

Composite potentials can be used in order to have faster computations or in order to have some specific properties of the interaction potential, such as the continuity for the second derivative at $r = b$. The solution for the latter case would be a potential having different expressions for different ranges of r. The expressions must be constructed in a way to fulfill the given requirements.

Another example of a composite potential is the modified potential, which has a different action range but preserves major properties of the original potential. Such modified potential is shown below:

$$\hat{\Pi}(r) \stackrel{\text{def}}{=} \Pi(k(r-a)+a), \tag{5.34}$$

where
Π is the given original potential;
k is the range modifier. For $k < 1$ the range of $\hat{\Pi}(r)$ is greater than the range of $\Pi(r)$, for $k = 1$ they are equal, and for $k > 1$ the range is smaller.

5.8 Simulation Setup

We choose the Lennard-Jones interaction potential (see section 5.7.1), since it is proved to be suitable for simulations (see section 5.3), and there is a clear way to find correspondence between parameters of the potential and real parameters of the simulated material (see sections 5.14, 5.15 and 5.19 for details).

Furthermore, in the following sections we present our novel approach extending the original Lennard-Jones potential by using cuboid potential fields, as well as other improvements.

In order to have more precise simulation, we focus on the simulation of the smallest elements of 3D volumetric objects – on voxels. Since an average segment has tens of thousands to hundreds of thousands of voxels, for our deformation simulation prototype we limit the area of simulation by a moving local simulation area described in section 5.10. The sizes of potential fields are chosen to be not bigger than voxels (kindly see section 5.9 for details).

5.9 Initial Positions and Velocities of Potential Fields

Setting initial positions and velocities of potential fields is in general a non-trivial task. It may change behaviour of simulated material [116, 115].

Since we work with complicated heterogeneous structures such as bone or muscle, we must reflect this in our simulation. The author of [115] mentioned that one of the ways of modeling such structures is to create "mono-grains" using only one kind of potential fields and then press them together. Although this approach gives good results, it requires hundreds thousands of potential fields and will be not interactive as required by our prototype system.

We propose to use the dense Face-Centered Cubic (FFC) lattice (see figure 5.10) for initial positions of potential fields and to set initial velocities to zero. We use the FFC lattice because:

1. we would like to have at least one potential field per voxel;
2. initial system of potential fields should be stable, that is there should be no significant "compression" or "tearing". In other words, the system should be in the state of a local minimum of the energy balance. This is achieved quite well by the

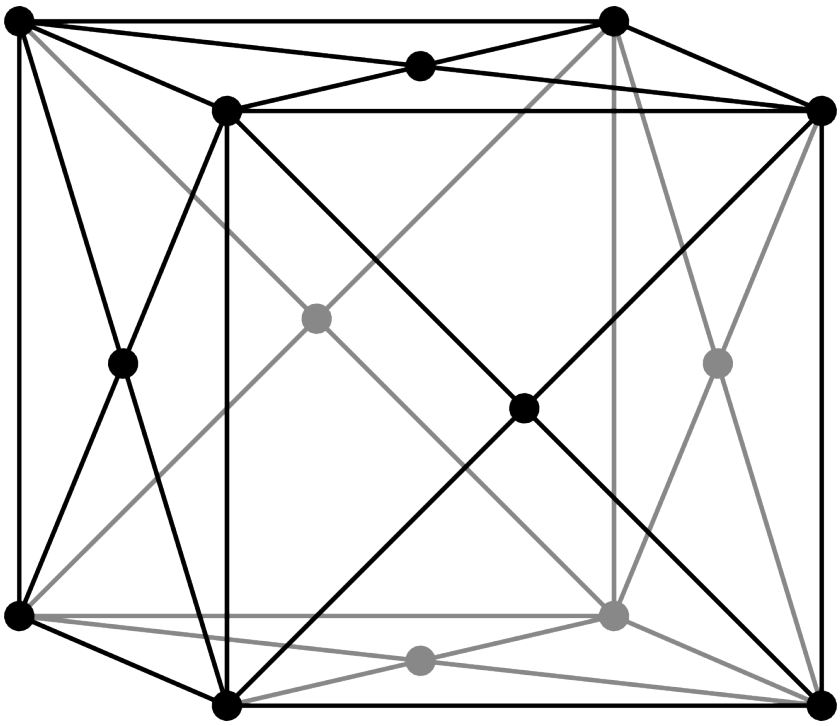


Figure 5.10: The Face-Centered Cubic (FCC) lattice. Black spheres correspond to the centers of potential fields for the FCC packing (source: modified from Wikipedia article "Cubic crystal system")

> FCC packing because distances between centers of every two potential fields on the first coordination sphere is a – the equilibrium distance.

Initial velocities are set to zero because the material should be in the rest state until a user starts to haptically interact with it. Heterogeneous behaviour of the simulated material is reached by adjusting parameters of the interaction potential individually for each pair of potential fields depending on intensities of voxels corresponding to their initial positions. This is discussed in detail in section 5.15.

In more detail, the voxel is taken as a *unit cell* (not as a *primitive unit cell* – see Remark below) for the FCC packing. Since we use the FCC packing, each voxel contains 4 lattice points (potential fields) in total. This is illustrated by spheres as potential fields in figure 5.11. Indeed, the FCC cube has 6 centers of potential fields on the faces of the cube, each giving half of potential field contribution. This results in $6 \times \frac{1}{2} = 3$ contribution. Additionally, the FCC has 8 centers of potential fields on the corners giving $\frac{1}{8}$ contribution each. This results in additional $8 \times \frac{1}{8} = 1$ contribution.

Remark:

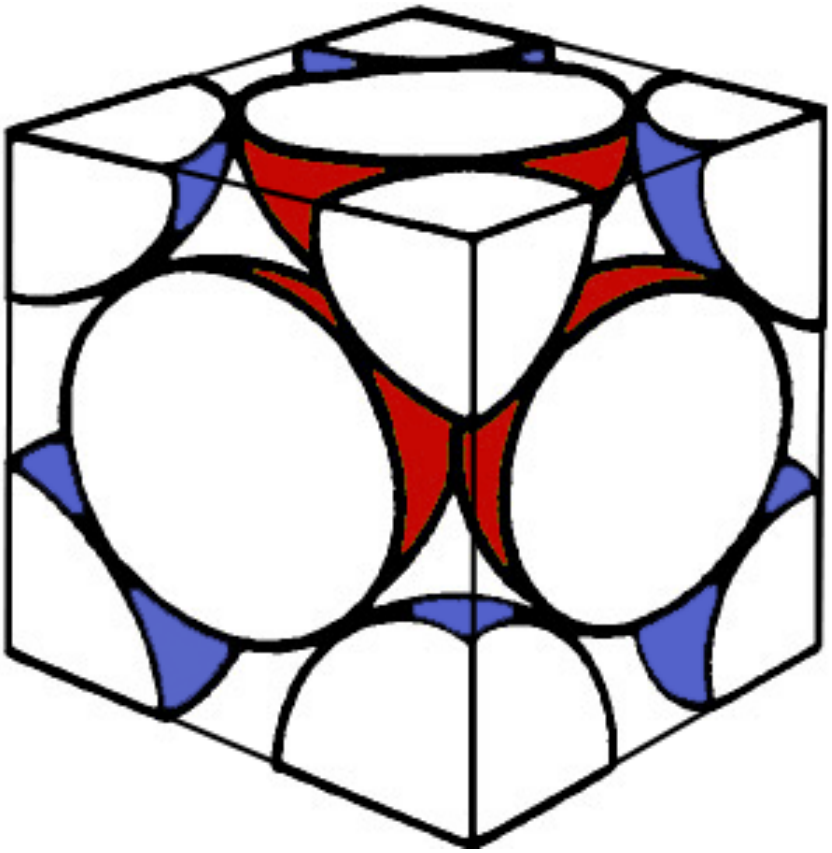


Figure 5.11: A unit cell of the FCC packing contains 4 lattice points (potential fields) in total. Potential fields are illustrated as spheres

A primitive unit cell is constructed in such a way that it contains only one lattice point (center of potential field) in total. That is, each vertex of the cell "sits" on a lattice point being shared with the surrounding cells. It is considered that each lattice point contributes $\frac{1}{n}$ to the total number of lattice points in the cell, where n is the number of cells sharing this lattice point.

Since each voxel contains 4 potential fields, we need $N_{sv} \times 4$ potential fields for the simulation, where N_{sv} is number of voxels in the simulation area.

Since a voxel is used as a unit cell for FCC packing, the equilibrium distance a for the interaction potential is defined as

$$a = \frac{\sqrt{2}}{2}\, a_{voxel}, \tag{5.35}$$

where
a_{voxel} is the length of the side of a voxel.

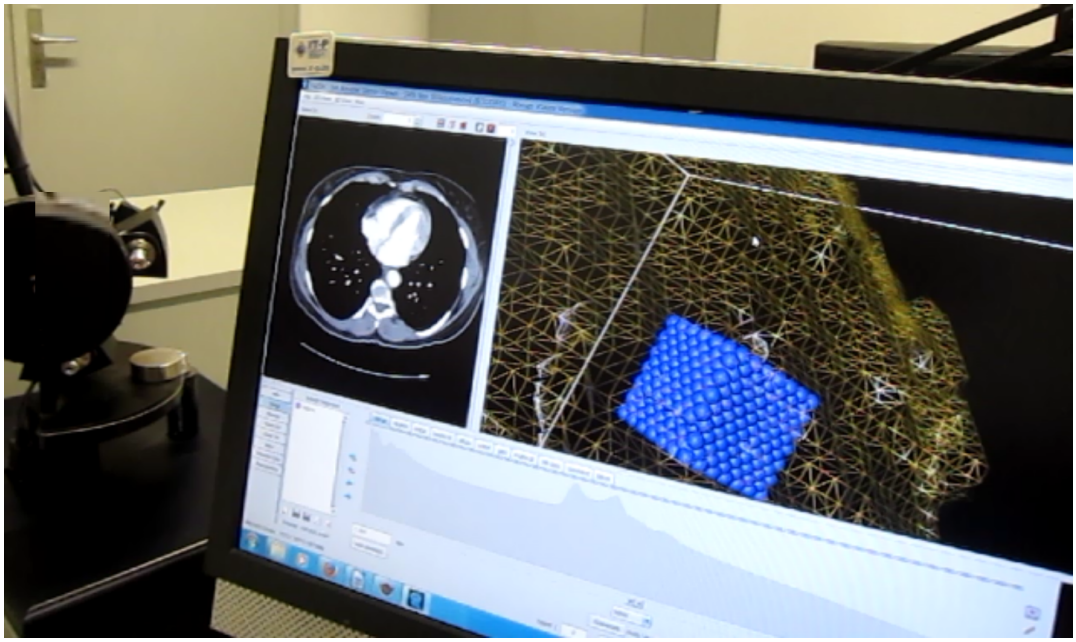


Figure 5.12: The FCC packing within the simulation area in our simulation. Potential fields are illustrated as spheres

5.10 Moving Local Simulation Area

As stated in section 5.8, we limit the area of our simulation, and this area can be moved. In more detail, we set our simulation area as a cuboid with the center being at the position of the IP. This form of the simulation area will be especially convenient later for our potential fields approach using our novel cuboid potential fields, which is presented in the sections below. The simulation area can also be viewed as a grid with cells of voxel size.

Since we work on voxels, the dimensions of the simulation area are odd integer numbers of voxels. The position of the IP is rounded to the closest integer value.

Since we:

1. know the dimensions of the simulation area, that is the maximum possible number of voxels N_{sv} within the simulation "window";

2. know that there should be $N_{sv} \times 4$ potential fields for the simulation;

we create all the required potential field objects in advance. Each potential field object has an *isUsed* flag being initially set to **false**.

Each iteration of the deformation loop (the loop designated for calculations of deformations – see section 5.1 for details regarding the structure of our deformation framework) the position of the IP is updated. Therefore we need to update the simulation area, too, so that the IP is still in the center of the simulation "window". Potential fields being outside the simulation area are disabled. For the voxels of the segment being "new" to the simulated area new potential fields are added. This process is described in more detail in section 5.11.

5.11 Reuse of Potential Field Objects

As mentioned in section 5.10, we create all potential field objects which can be possibly used during the simulation in advance. This is possible because we know the maximum possible number of voxels N_{sv} within the simulation "window". On initialization, every potential field has the *isUsed* flag being set to **false**. In order to effectively re-use potential fields which are out of the simulation area due to the movement of the IP, each iteration of the defo loop we do the steps outlined below. The following denotations are used:
A_{prev} – the simulation area after the end of the previous iteration of the defo loop;
A – the simulation area being updated by the position of the IP in the beginning of the current iteration of the defo loop (see details regarding this update in section 5.10);
L – the list of unused potential fields;
BC – the bit cube for the segment being processed.

1: **for** ((each potential field P) $\in$ (potential fields with *isUsed*=**true**)) **do**
2: **if** (initial position of center of $P \notin A$) **then**
3: $P.isUsed :=$ **false**
4: add P to L // Add to the list of unused potential fields
5: **end if**
6: **end for**
7: // Fill grid cells of the simulation area which are empty but should be
8: // filled by potential fields
9: **for** (each cell $C \in A \setminus A_{prev}$) **do**
10: **if** ($BC.get(C.pos)$=**false**) **then**
11: // No voxel at this position for the segment
12: **continue**
13: **end if**
14: **while** (need potential fields to fill the FCC packing for C) **do**

15: $P := L.removeLast()$
16: Set initial position of P according to the dense FCC packing for C
17: **end while**
18: **end for**

Remarks:

1. In line 2 we check not the current position of the center of the potential field but its *initial position*, that is the position which was assigned to the center of the potential field when it started to be in use

2. In practice we do more effective filling of the grid cells by potential fields than in the second **for**–loop, and this more effective approach has $O(N_{sv})$ time complexity. But this approach would be less illustrative than the presented one. Additionally, we do not put potential fields within $\frac{a}{2}$ from the position of the IP in order to avoid suddenly appearing repulsive forces causing an unstable behaviour of the simulation system.

5.12 Binding to Initial Positions

Since we use the local simulation area, an approach to keep the potential fields inside the simulation area is needed. Otherwise the user might be able to "press" the complete simulation area away from its original location, because it just "hangs in the air" without being attached to anything. Another issue would be the following. For the system of potential fields being symmetric in all directions the configuration of the potential fields corresponding to the minimum energy for the whole system is a sphere. Therefore the configuration of potential fields would become a sphere after the simulation is run for a while, i.e. the potential fields will move away from their initial positions even without any interaction with the user. This behaviour will be presented especially if the local simulation area is not fully packed by potential fields but has e.g. empty spaces inside or curved surfaces.

Our first approach to keep potential fields in place is to bind their centers to their initial positions using spring forces (see section 5.11 for details about how the initial position of potential field is defined). The binding spring forces should be stronger than the interaction forces between the potential fields in the initial configuration. Furthermore, the binding forces help to keep the original structure of the object within the local

simulation area. Additionally, it may be reasonable to make the binding forces so strong, that the potential fields always return to their initial positions.

We should note here that although the above approach works fine and fits for the purpose of validation of our deformation framework, our experiments showed that introducing the binding forces is less realistic and makes finding the correspondence between the simulation parameters and parameters of the real material a non-trivial task. Therefore we do not go in more detail for the current approach, but instead present another approach being described in section 5.13.

5.13 Interaction with Borders of the Simulation Area and with Empty Space

Another approach to solve the issues described in section 5.12 is to introduce interactions with borders of the simulation area and with empty space. We define the empty space as the space where voxels of the segment's bit cube are set to zero and as the space outside the bit cube. Since no additional explicit parameters are added into the simulation system, it is easier to find the correspondence between simulation parameters and parameters of the real material in this case, compared to the approach proposed in section 5.12.

Describing our approach in more detail, it is not allowed for potential fields to leave the simulation area or enter an empty space. In order to achieve this, the following is done every simulation step for every active potential field (i.e. every potential field with *isUsed* flag set to **true**) after update of forces, velocity and position:

1. Let us denote the movement of the center of the potential field from the previous position $\mathbf{p}_1$ to the current position $\mathbf{p}_2$ as $\mathbf{p}_{12}$, and denote $-\mathbf{p}_{12}$ as $\mathbf{p}_{21}$. Additionally, let us denote normalized vectors of $\mathbf{p}_{12}$ and $\mathbf{p}_{21}$ as $\mathbf{n}_{12}$ and $\mathbf{n}_{21}$, respectively

2. Since (1) the length of linkage for pairwise interaction of potential fields is a and (2) borders of the simulation area and the empty space are absolutely rigid objects, the equilibrium distance from the center of the potential field to any border of the simulation area or to the empty space is $\frac{a}{2}$. Therefore if the point

$$\mathbf{p}_{2_{offset}} \stackrel{\text{def}}{=} \mathbf{p}_2 + \frac{a}{2}\,\mathbf{n}_{12} \tag{5.36}$$

   is outside the simulation area or inside the empty space then go to step 3, else exit

3. Mirror the component of velocity of potential field which is perpendicular to the tangent plane of the border/empty space at the hit point. We should note here that voxels forming the border of the local moving simulation area are axis aligned. Further on, the empty space is represented by empty-space voxels, and each side of the voxel is axis-aligned, too

4. "Cancel" the last update of the position of the center of the potential field:

$$\mathbf{p}_2 := \mathbf{p}_1 \tag{5.37}$$

5.14 Correspondence to Parameters of Real Materials

Here we present the way to find the correspondence between simulation parameters and parameters of real materials in the case of a homogeneous isotropic material. Our approach to work with heterogeneous materials is presented in section 5.15.

An extensive and detailed description of properties of different packings corresponding to initial positions of potential fields and a detailed description of how to find correspondence between simulation parameters and physical properties of real materials can be found in [115]. Here we present the main equations and how they are applied to the local simulation deformation approach.

In order to define parameters of our simulation model, we need three basic parameters (basic units): the mass, the distance and the time. Other parameters could be represented via these parameters and dimensionless coefficients.

Let us take the mass m of the material point associated with a potential field as the basic parameter of mass. Next, let us take the equilibrium distance a between the centers of two potential fields in one-dimensional space as the basic parameter of distance. Further on, let us take T_0 as the basic parameter of time, where T_0 is the period of small oscillations of the first potential field around the equilibrium distance in case the position of the second potential field is fixed, all in one-dimensional space.

Generally, the mass m of the material point associated with the potential field can be found from the following equation:

$$m = \frac{M}{N}, \tag{5.38}$$

where
M – the mass of the simulated object;
N – the number of potential fields being used to simulate it.

Since we do not simulate the entire object but simulate its part within the local simulation area only, the equation 5.38 should be adjusted accordingly. Furthermore, since we take into account voxel intensities, we actually use another equation to find masses of material points associated with potential fields. This is discussed in detail in section 5.15.

In order to find the basic simulation parameter of distance a, let us first write down the equation for the volume V of the simulated object:

$$V = p\mathcal{V}_0(\chi a)^3 N, \tag{5.39}$$

where
p – the density of the packing of initial positions of potential fields compared to the dense packing;
$\mathcal{V}_0$ – the dimensionless volume of the primitive unit cell of the dense packing being calculated for the unit distance between the centers of the closest potential fields; its value is a constant depending on the concrete packing, and it can be found in [115];
χ – the coefficient characterizing the change of the equilibrium distance in the dense packing when the potential fields forming the packing interact with the potential fields from the next coordination spheres. Its value is also a constant depending on the concrete packing, and it can be found in [115].
It should be noted here that the values of $\mathcal{V}_0$ and χ in [115] are given for the case of interaction potentials being symmetric in all directions.

From equation 5.39, the following expression for a can be written:

$$a = \frac{1}{\chi}\sqrt[3]{\frac{V}{p\mathcal{V}_0 N}}. \tag{5.40}$$

In our case, since the simulated object is represented using voxels and since we use the FCC packing for the initial positions of potential fields in a way it is described in section 5.9, we know the proportion between a and the size of a voxel. From this, we can write down the equation for a – see equation 5.35.

In order to find the basic parameter of time T_0, first let us write down the expression for the stiffness of linkage C using the scalar velocity of propagation of longitudinal sound waves in the medium v_l:

$$C = m\left(\frac{v_l}{\lambda a}\right)^2, \tag{5.41}$$

where

$$\lambda \stackrel{\text{def}}{=} \frac{v_l}{v_0}, \tag{5.42}$$

where v_0 is a scalar velocity of propagation of longitudinal sound waves in 1-dimensional chain of potential fields. The value of λ is a constant depending on the concrete packing, and it can be found in [115].

From equation 5.41 and from the below expression for T_0

$$T_0 = 2\pi\sqrt{\frac{m}{C}}, \tag{5.43}$$

the following expression for T_0 can be written:

$$T_0 = 2\pi\lambda\frac{a}{v_l}, \tag{5.44}$$

The scalar velocity of propagation of longitudinal sound waves in the medium v_l being used in expression 5.44 can be found from the following expression (see [215] for details):

$$v_l = \sqrt{\frac{l_1 + 2l_2}{\rho}}, \tag{5.45}$$

where
ρ – the density of the material;
l_1 and l_2 are the first and the second Lame's parameters, which can be written as:

$$l_1 = \frac{E\nu}{(1-2\nu)(1+\nu)}, \tag{5.46}$$

$$l_2 = \frac{E}{2(1+\nu)}, \tag{5.47}$$

where
E – Young's modulus of the material;
ν – Poisson's ratio of the material.

As mentioned in [115], the time step for the integration of the equations of motion is defined as

$$\Delta t = k_t T_0, \tag{5.48}$$

where k_t is a dimensionless coefficient being normally chosen as 0.01–0.05 depending on the required accuracy of computations.

The parameters of the interaction potential can be found using its stiffness of linkage C, its strength of linkage f_* and its energy of linkage D. These characteristics can be expressed via m, a and T_0 as follows:

$$C = 4\pi^2 \frac{m}{T_0^2}, \tag{5.49}$$

$$f_* = \frac{\epsilon_*}{k_*} Ca, \tag{5.50}$$

$$D = \frac{1}{2k_v^2} Ca^2, \tag{5.51}$$

where ϵ, k_* and k_v are the characteristics of the interaction potential defined by expressions 5.9, 5.11 and 5.12. Values of ϵ, k_* and k_v are constants for the concrete potential.

For the Lennard-Jones interaction potential $k_v = 6$ (see expression 5.19), therefore the expression for the energy of linkage D is as follows:

$$D = \frac{1}{72} Ca^2. \tag{5.52}$$

We would like to note that in our prototype system, in order to simplify the calculations, we store coordinates of centers of potential fields in the voxel space, and therefore a transformation to the coordinate system with the real spacing and back is done for every operation with coordinates.

5.15 Taking into Account Voxel Intensities

In order to take into account the heterogeneity of the simulated material, we use an important additional information stored in the volume data – voxel intensities. We consider X-Ray Computed Tomography (CT) scans only, but using similar approach it is possible to generalize it to MRI scans and other volume data with intensities. Of course, for such approaches the range of intensities for different materials and the correspondence to the material density could be different.

A good overview of how CT works can be found in [101] and [72]. In case of CT scans we use the Hounsfield unit (HU) scale, also called as CT numbers (see e.g. Friese [72] and [101]). The HU is a linear transformation of the measured original linear attenuation coefficient μ into one in which the radiodensity of air at standard temperature and pressure is defined as -1000 HU, and the radiodensity of distilled water at standard temperature and pressure is defined as 0 HU:

$$HU(\mu) = 1000 \times \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}}, \tag{5.53}$$

where
μ_{air} – the linear attenuation coefficient of air;
μ_{water} – the linear attenuation coefficient of water.

The above definition for the HU is generally used as a definition for calibration of CT scanners with reference to water: the change for 1 HU corresponds to the change of 0.1% of $(\mu_{water} - \mu_{air})$. Furthermore, μ_{air} is nearly 0. The use of the HU scale helps to compare original linear attenuation coefficients acquired from different CT scanning devices and with different X-ray beam energy spectra.

The HU value being assigned to the voxel is equal to $HU(\mu_v)$, where μ_v is the average of all attenuation coefficients contained within the voxel.

For each particular material or organ, there is a correspondent interval of the HU values – see section 5.28.3 for the concrete values. We assume that each segment in our simulation represents a specific organ or material, and therefore has an associated interval of HU values. That is, every voxel of the given segment has the HU value from the interval of HU values associated with this segment.

As shown e.g. in [150], there is a correspondence between the HU values and the density of the material. We use an approximation in a form of linear transformation from the interval of HU values to the interval of densities. In more detail, for the given HU value HU of the voxel, we use the following expression to approximate its density ρ:

$$\rho = \rho_{min} + \frac{HU - HU_{min}}{HU_{max} - HU_{min}}(\rho_{max} - \rho_{min}), \tag{5.54}$$

where
$[HU_{min}, HU_{max}]$ – an interval of HU values associated with the given material;
$[\rho_{min}, \rho_{max}]$ – an interval of densities associated with the given material.
If $HU < HU_{min}$ then we set it as $HU := HU_{min}$. If $HU > HU_{max}$ then we set it as $HU := HU_{max}$.

The mass m of the material point associated with the potential field is then approximated by the following expression:

$$m = \rho V_0, \tag{5.55}$$

where V_0 is the volume of the material, which (the volume) corresponds to the material point associated with the potential field. It is expressed as

$$V_0 = \frac{a^3}{4}, \tag{5.56}$$

since there are 4 potential fields per voxel.

Furthermore, for each pair of interacting potential fields we calculate their own parameters of the interaction potential. For the Lennard-Jones interaction potential such a parameter is the energy of linkage D. For the calculation of D (expression 5.53) we need to calculate the stiffness of linkage C. For that, we use expression 5.49 but replace m by m_{av}, where m_{av} is computed as

$$m_{av} = \frac{1}{2}(m_1 + m_2), \tag{5.57}$$

where m_1 and m_2 are the masses of the material points associated with the interacting potential fields, each computed using expression 5.55. Further on, in order to use expression 5.49 for calculating C, the period of small oscillations T_0 needs to be computed using expression 5.44. And for that, the scalar velocity of propagation of longitudinal sound waves in the medium v_l needs to be calculated. For that, we use expression 5.45, but replace ρ by ρ_{av}, where ρ_{av} is computed as

$$\rho_{av} = \frac{1}{2}(\rho_1 + \rho_2), \tag{5.58}$$

where ρ_1 and ρ_2 are the densities of the material points associated with the interacting potential fields. For calculation of the first and the second Lame's parameters l_1 and l_2 (expressions 5.46 and 5.48) required for calculation of v_l, one can either (a) use original values of Young's modulus E and Poisson's ratio ν of the simulated material, or (b) similar to expression 5.54 use an approximation to make E and μ proportional to the average of the HU values of the two material points associated with the interacting potential fields.

5.16 Interactions of the IP with Potential Fields

In order to simulate interactions of the haptic IP with potential fields, we consider the IP as a potential field, too, but do not perform the integration of equations of motion for it. Furthermore, for interactions between the IP and other potential fields we use the repulsive part of the Lennard-Jones interaction potential only, because the potential field associated with the IP does not belong to the same material as other potential fields, and therefore there should be no attraction forces.

Additionally, the interaction potential for interactions between the potential field associated with the IP and other potential fields can be varied, e.g. can have a different equilibrium distance to implement a finger which is larger than the other potential fields, can have a different expression (e.g. Mi potential or Morse potential) or can be anisotropic. This will change the interactions.

Of course, the "IP–potential field" interactions are carried within our framework allowing to use our improved approach of haptic rendering of volume data together with deformation models (see section 5.1).

5.17 Dissipation in Our Approach

A dissipation is needed to take away extra energy from the system. The easiest way to do it is to add the nonconservative force field $\boldsymbol{\psi}(\mathbf{r}, \mathbf{v})$ for every potential field (see expression 5.4), where $\mathbf{r}$ and $\mathbf{v}$ are position and speed of the center of the given potential field, respectively. As suggested in [115], we use viscous friction:

$$\boldsymbol{\psi}(\mathbf{r}, \mathbf{v}) = \boldsymbol{\psi}(\mathbf{v}) = -B\mathbf{v}, \tag{5.59}$$

where $B > 0$ – a coefficient of viscous friction.

As mentioned in [115], a more flexible control over dissipation can be achieved using thermostats [160, 161, 162, 208].

Following [115], let us find the correspondence of B to other parameters of the simulated system. Let us consider oscillations of the material point associated with the potential field under the dissipative force (expression 5.59) and the linearized elastic force (expression 5.10):

$$m\ddot{x} + B\dot{x} + Cx = 0, \tag{5.60}$$

where x is the displacement of the material point from the equilibrium position.

Let us define B_0 as

$$B_0 \stackrel{\text{def}}{=} 2\sqrt{mC} = 2m\omega_0 = \frac{2m}{T_0}. \tag{5.61}$$

According to [115], B_0 is the value of B turning the discriminant of the frequency equation corresponding to the equation to 0. That is, B_0 is the critical value of B:

- for $B < B_0$ there is an oscillative motion within the system;
- for $B \geq B_0$ there are no oscillations because of a high dissipation.

In order not to have oscillations in our simulation system, we choose B to be a bit greater than B_0. Furthermore, in order for all potential fields of the same material to have the same dissipative force, we use the same value of m, of ρ, of E and of ν (the

latter three are required for the calculation of C – see section 5.15 for details). We choose them as the maximum values for the current material.

Since the above equations are for one potential field in the system of two potential fields in one-dimentional case, in order to generalize it to the three-dimentional case with many potential fields, the maximum number of potential fields sitting on the first coordination sphere should be taken into account in order to ensure that $B_0 \geq B$ for this case. Therefore B_0 should be multiplied by 5 for cuboid potential fields and should be multiplied by 4 for regular potential fields. These numbers reflect the maximum factor by which the stiffness of linkage along each coordinate axis increases if potential fields are superimposed. Interaction forces with potential fields from the next coordination spheres are negligible compared to the interaction forces from the first coordination sphere, and therefore can be omitted.

5.18 Cuboid Potential Fields

Since the volume data consists of voxels being cuboids (or cubes in the voxel space, that is in the coordinate system where the unit along each axis is equal to the voxel length along this axis), it is more natural to represent a potential field as a cuboid of the size of a voxel. That is, we introduce a potential field with a varied equilibrium distance, so that potential fields with the associated material points, which are put in the center of each voxel, already form a dense packing, and the system in this configuration is already in the equilibrium state. In the following we consider cubic voxels, but all the expressions and algorithms in this chapter can be generalized to cuboid voxels.

Important Remarks:
While cuboid potential fields are inspired by classic potential fields, they do not match the original definition of potential field by 100%. The cuboid force field we associate with the potential is not a potential field because only the radial component of the gradient of the potential is later used for the calculation of the interaction force, and the work integral is not path-independent. However the cuboid field fulfils a role similar to the potential field because it has a set of equilibrium points which form a cuboid shape. Also, if one varies only the distance and not the direction, it yields the same forces as a spherically symmetric Lennard-Jones potential.

Our approach to use cuboid potential fields for the local deformation simulation is designed in such a way that minimum changes to our potential fields based local de-

formation simulation approach presented earlier in this chapter are required. Instead of using the original Lennard-Jones interaction potential, we use a potential, which we denote as Cuboid Lennard-Jones potential Π_{cube}. This interaction potential should keep imaginary "cubes" of potential fields being axis aligned and touching each other. That is, there should be repulsive forces if the "cube" of one potential field is (partially) inside the second one, and there should be attraction forces if the "cubes" of two potential fields do not touch each other by any side. In two-dimensional case this is shown in figure 5.13. Therefore, the equilibrium distance for the "cubes" touching each other but not being on the same axis should be adjusted (increased) in order not to cause unnecessary attraction forces. We do this as follows. Firstly, let us define the potential not as an expression but as an algorithm. The following denotations are used:
$\mathbf{p}_1$ and $\mathbf{p}_2$ – positions of centers of two interacting potential fields.

1. Choose one of the three coordinate axes, called **A** from now on, with the maximum difference between $\mathbf{p}_1$ and $\mathbf{p}_2$ along it, that is the coordinate axis with the maximum length of projection of vector $(\mathbf{p}_2 - \mathbf{p}_1)$ on it. There is always a coordinate axis with non-zero difference, unless $\mathbf{p}_1$ and $\mathbf{p}_2$ have the same coordinates. But this will not happen due to constraints of our simulation

2. Calculate the adjusted equilibrium distance a_{cube} answering the requirements described in the beginning of this section:

$$a_{cube} = \frac{a}{|\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2} \cdot \mathbf{n}_{\mathbf{A}}|}, \tag{5.62}$$

   where
   a - the side length of a voxel;
   $\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2}$ – a normalized vector $(\mathbf{p}_2 - \mathbf{p}_1)$;
   $\mathbf{n}_{\mathbf{A}}$ – a normalized vector of the coordinate axis **A**

3. Use a_{cube} as an equilibrium distance for the standard Lennard-Jones interaction potential (expression 5.13):

$$\Pi(r) = D\left(\left(\frac{a_{cube}}{r}\right)^{12} - 2\left(\frac{a_{cube}}{r}\right)^{6}\right), \tag{5.63}$$

The above algorithm for calculation of Π_{cube} can be rewritten as the following expression:

$$\Pi_{cube}(r, \mathbf{n}_{\mathbf{p}_1\mathbf{p}_2}) = D\left(\left(\frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{r}\right)^{12} - 2\left(\frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{r}\right)^{6}\right), \tag{5.64}$$

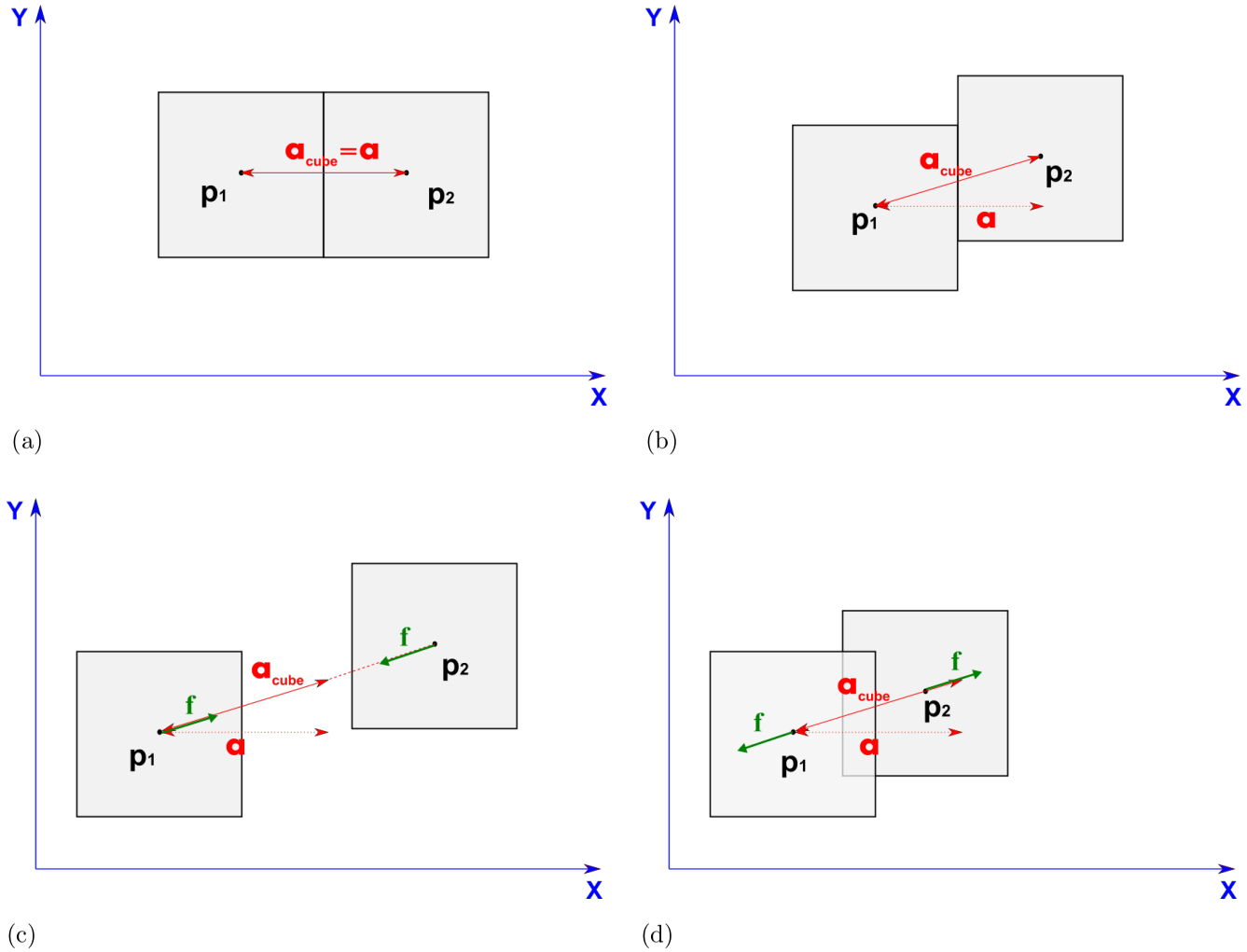


Figure 5.13: Our interaction forces in two-dimensional case: (a) and (b) – no forces, (c) – attraction forces, (d) – repulsive forces. Potential fields with varied equilibrium distance a_{cube} are represented as imaginary "cubes" with centers at $\mathbf{p}_1$ and $\mathbf{p}_2$. The side length of a voxel is a.

where

$$a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2}) = \frac{a}{max\left(|\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2} \cdot \mathbf{n}_{\mathbf{X}}|, |\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2} \cdot \mathbf{n}_{\mathbf{Y}}|, |\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2} \cdot \mathbf{n}_{\mathbf{Z}}|\right)}, \tag{5.65}$$

where $\mathbf{n}_{\mathbf{X}}, \mathbf{n}_{\mathbf{Y}}, \mathbf{n}_{\mathbf{Z}}$ – normalized vectors along coordinate axis $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, respectively.

Similar to section 5.6, in order to speed-up computations of interactions, the interaction potential is usually cut at a **cut distance** a_{cut}. For regular potential fields it is normally set to $a_{cut} = 2.1a$, because then it "captures" the first two coordination spheres and because for $r > 2a$ both the interaction potential and the interaction force converge to 0. Since we use cuboid potential fields, we should take into account the varied equilibrium distance. Therefore we can consider not coordination spheres but coordination cubes, so that we take into account all 26 neighbors for the first coordination cube (see section 5.19 for more details). "Converting" the coordination cube back to the coordination sphere, we will take into account all 26 neighbors if the first coordination sphere has such a radius that even centers of the most distant of N_{26} neighbors of the given potential field (those being on the diagonals of the coordination cube) are within the first coordination sphere. Therefore its radius should be $a\sqrt{3}$. Then a_{cut} should be set to $a_{cut} = 2.1a\sqrt{3} \approx 3.64a$. In practice we set it to $a_{cut} = 3.7a$.

5.19 Correspondence to Parameters of Real Materials for Cuboid Potential Fields

Let us find the correspondence between parameters of the simulated system and parameters of real materials when we use cuboid potential fields for our local deformation simulation.

The expression for the mass m of the material point associated with the potential field is the same as for the "regular" potential field (section 5.14) – see expression 5.38. Furthermore, in the same way as described in section 5.15, we take into account the Hounsfield unit value for each voxel, and therefore in practice we use expression 5.55 for calculation of m. We write it down one more time below:

$$m = \rho V_0. \tag{5.66}$$

But compared to "normal" potential fields, for cuboid potential fields the volume V_0 at the half of the varied equilibrium distance from the material point associated with the

potential field, that is the volume of a voxel, is calculated as

$$V_0 = a^3. \tag{5.67}$$

We can write down the above expression for V_0 because we know that for our local deformation simulation using cuboid potential fields the equilibrium distance a between the centers of two cuboid potential fields in one-dimensional space a should be equal to the side length of a voxel. Furthermore, a should be taken as the basic parameter of distance. Although the size of a voxel is usually explicitly given in the volumetric data file, it can also be expressed via the equation 5.40 (see section 5.14 for details). We write it down one more time below:

$$a = \frac{1}{\chi}\sqrt[3]{\frac{V}{p\mathcal{V}_0 N}}. \tag{5.68}$$

In order to make sure that a was chosen correctly, let us check that after finding all the constants for the above equation, it will become a well-known expression for the side of a cubic voxel:

$$a = \sqrt[3]{\frac{V}{N_{vxls}}}, \tag{5.69}$$

where
V – the volume of the simulated object or the volume of the simulation area;
N_{vxls} – number of voxels within V. Since we explicitly chose potential fields to be cuboids of the size of a voxel (see section 5.18), the number of potential fields N used for the simulation of the object within V equals to N_{vxls}:

$$N = N_{vxls}. \tag{5.70}$$

We start with $\mathcal{V}_0$ – the dimensionless volume of the primitive unit cell of a dense packing being calculated for the unit distance between the centers of the closest potential fields. In our case of cuboid potential fields, the primitive unit cell equals one voxel, therefore its volume equals to the volume of a voxel. Since $\mathcal{V}_0$ is dimensionless, the expression for it is the following:

$$\mathcal{V}_0 = 1^3 = 1. \tag{5.71}$$

Another constant in the equation 5.68 is p – the density of packing for initial positions of potential fields compared to the dense packing. In the case of cuboid potential fields we have the densest possible packing, therefore

$$p = 1. \tag{5.72}$$

One more constant in the equation 5.68 is χ – the coefficient characterizing the change of the equilibrium distance in the dense packing when the potential fields forming the packing interact with the potential fields from the next coordination spheres. For "regular" potential fields its value is a constant depending on the concrete packing, and it can be found in [115]. But for the case of cuboid potential fields we need to calculate its value ourselves. In order to do this, we need to understand how it is computed. According to [115], the expression for χ for "regular" potential fields is as follows:

$$\chi = \frac{R_1}{a}, \tag{5.73}$$

where the radius of the first coordination sphere R_1 is written as

$$R_1 = R\varrho_1, \tag{5.74}$$

where
ϱ_1 – the relative radius of the first coordination sphere (relative to the radius of the first coordination sphere for the one-dimensional chain of potential fields);
R – unknown value, which could be found from the expression 5.77 below.

Since we use cuboid potential fields, expression 5.74 should be rewritten as

$$\chi = \frac{R_1}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}. \tag{5.75}$$

Since we use cuboid potential fields, the coordination cube instead of the coordination sphere should be used in order to take into account all N_{26}–neighbours of the given potential field. It will be implicitly used if we incorporate $a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})$ into the expression for ϱ_1:

$$\varrho_1 = \frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a}. \tag{5.76}$$

For the standard Lennard-Jones interaction potential, the expression for R is the following (see [115] for details):

$$R = a\sqrt[6]{\frac{\sum_{k=1}^{n} N_k \varrho_k^{-12}}{\sum_{k=1}^{n} N_k \varrho_k^{-6}}}, \tag{5.77}$$

where
k – the number of the coordination sphere;
n – the number of considered coordination spheres;
N_k – the number of potential fields on the k–th coordination sphere;
ϱ_k – the relative radius of the k–th coordination sphere.

For the first coordination sphere only, i.e. for $n = 1$, expression 5.77 will become the following:

$$R = a\sqrt[6]{\frac{N_1\varrho_1^{-12}}{N_1\varrho_1^{-6}}}. \tag{5.78}$$

In order to find the value of R, we need first to understand how expression 5.77 is found. It is found from the following equation for R:

$$\sum_{k=1}^{n} N_k\varrho_k f(\varrho_k R) = 0, \tag{5.79}$$

where $f(r)$ is the interaction force for the given interaction potential.

For the first coordination sphere, i.e. for $n = 1$, and for the cuboid Lennard-Jones potential equation 5.79 will become the following:

$$N_1\varrho_1 f(\varrho_1 R) = 0. \tag{5.80}$$

After replacing ϱ_1 by the right side of expression 5.76 and after taking into account that because of the above expression for ϱ_1, N_1 equals to 26, equation 5.80 will become the following:

$$\begin{aligned} 26\ \frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a}\ f(\frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a}R) = 0 \Leftrightarrow \\ 26\ \frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a}\frac{12D}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}\left(\left(\frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})\ a}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})\ R}\right)^{13} - \left(\frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})\ a}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})\ R}\right)^{7}\right) = 0 \Leftrightarrow \\ \frac{312D}{a}\left(\left(\frac{a}{R}\right)^{13} - \left(\frac{a}{R}\right)^{7}\right) = 0 \end{aligned} \tag{5.81}$$

The only positive real solution for the above equation is:

$$R = a. \tag{5.82}$$

As could be seen from the above expression, the value of R does not depend on $a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})$.

Now, using expressions 5.74, 5.76 and 5.82, let us write the new expression for R_1:

$$R_1 = R\varrho_1 = a\ \frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a} = a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2}). \tag{5.83}$$

Further, using the above expression and expression 5.75, let us write the new expression for χ:

$$\chi = \frac{R_1}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})} = \frac{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})}{a_{cube}(\mathbf{n}_{\mathbf{p}_1\mathbf{p}_2})} = 1. \tag{5.84}$$

Now we know all the constants for expression 5.68, and therefore we can rewrite it as follows:

$$a = \frac{1}{\chi}\sqrt[3]{\frac{V}{p\mathcal{V}_0 N}} = \frac{1}{1}\sqrt[3]{\frac{V}{1 \cdot 1 \cdot N}} = \sqrt[3]{\frac{V}{N}}. \tag{5.85}$$

Since $N = N_{vxls}$ (expression 5.70), the above expression can be rewritten as:

$$a = \sqrt[3]{\frac{V}{N_{vxls}}}.$$

As expected, the resulting expression for a is the same as the well-known expression for the side of a cubic voxel 5.69. This is what we wanted to check. Therefore the basic parameter of distance a is chosen correctly.

Since we use cuboid potential fields, we also need to find a new value of λ (see expression 5.42). According to [115], for an isotropic material λ can be found as follows:

$$\lambda = \sqrt{\frac{3M}{2d(d+2)}}, \tag{5.86}$$

where
M – the coordination number, that is the number of the closest neighbors for each potential field;
d – the dimension of space.

In case of cuboid potential fields in three-dimensional space, $M = 26$ and $d = 3$. We insert this into the above expression, which works for potential fields, to estimate λ for our related cuboid potential fields:

$$\lambda = \sqrt{\frac{3 \cdot 26}{2 \cdot 3 \cdot 5}} = \sqrt{\frac{13}{5}}. \tag{5.87}$$

For the basic parameter of time T_0 we use the same expressions as in section 5.14, but with updated values for the parameters and constants found in the current section. After finding all basic parameters, parameters of interaction potentials can be found using the same expressions as is in section 5.14.

Similarly, in order to take into account the heterogeneity of the simulated material, we use the same expressions as in section 5.15, but with corrections and updated parameters and constants from the current section.

Similar to section 5.14, we would like to note that in our prototype system, in order to simplify the calculations, we store coordinates of centers of potential fields in the voxel space, and therefore a transformation to the coordinate system with real spacing and back is done for every operation with coordinates.

5.20 Limit Maximum Interaction Force

In order to prevent instability of the simulation system, we should limit the maximum velocity and limit the travel distance for potential fields per a haptic frame. Such instability can arise if centers of potential fields are at the distance $r \ll a$ from each other, causing a high repulsive force by the repulsive component of the Lennard-Jones interaction potential. In practice limiting the velocity or the travel distance is not effective, because the potential field will still "remember" high interaction force and will bring it to the next haptic frame. Instead, we limit the maximum interaction force f for each pairwise interaction of potential fields. That is, if $|f| > f_{max},\ f_{max} > 0$ then

$$f = sign(f)\ f_{max}. \tag{5.88}$$

5.21 "Multi-Layered" Simulation

In order to have faster simulation and higher precision of the simulation, we use a "multi-layered" simulation approach. Each simulation step, we do calculations in close vicinities of the position of the IP first, and then calculations of the whole simulation area. The outline of the approach is as follows:
Let us denote the passed time since the beginning of the last simulation step as Δt. Further on, let us split Δt into n parts, so that $\Delta t = \Delta t_1 + \Delta t_2 + ... + \Delta t_n$, where n is the number of sub-steps (that is, the number "layers"), which we want to have each simulation step. Then:

- Sub-step 1: for the 1st closest vicinity around the IP – we use the integration step Δt_1
- Sub-step 2: for the 2nd closest vicinity around the IP – we use the integration step Δt_2 for the potential fields being in the 1st vicinity, and $\Delta t_1 + \Delta t_2$ for the potential fields being only in the 2nd vicinity

- Sub-step 3: for the 3rd closest vicinity around the IP – we use Δt_3 for the potential fields being in the 1st and the 2nd vicinities, and $\Delta t_1 + \Delta t_2 + \Delta t_2$ for those being only in the 3rd vicinity
- ...
- Sub-step n: for the n-th closest vicinity around the IP – we use Δt_n for the potential fields being in the 1st to $n-1$-th vicinities, and $\Delta t_1 + \Delta t_2 + ... + \Delta t_n$ for those being only in the n-th vicinity.

There are additional technical issues, such as dealing with the following case: if a potential field A being in the i-th vicinity interacts with a potential field B which is out of the 1st to the i-th vicinities, then B should be considered to be a part of the i-th vicinity. Otherwise the interaction between the potential fields A and B will be ignored. Such issues have been solved during the implementation of the approach presented above, but their technical description lies out of focus of the current dissertation.

5.22 Speed-up Structure to Find Interactions

In order to speed-up finding of interactions between potential fields, a speed-up structure is used.

As mentioned in section 5.6, if centers of two potential fields are further than the cut distance a_{cut} then it is assumed that interaction forces between them are negligible, and therefore we do not need to consider such pairs. Therefore we need to find all pairs of potential fields for which the distance between their centers is not greater than a_{cut}.

A naive algorithm to find the interactions is to check for every potential field the distance between its center and the center of every other potential field. But this algorithm has time complexity of

$$O(N_{pf}^2), \tag{5.89}$$

where N_{pf} is the number of potential fields in the simulation system.

Much more effective approaches (with time complexity being proportional to N_{pf}) are described in [83] and in [116, 115]. Such the authors of two latter works divide all the space into a regular cubic grid with the side of cubic cell being equal to a_{cut}. For every cell only interactions between potential fields inside the cell and with the potential fields being in the N_{26}-neighboring cells are considered, and the integration of equations of

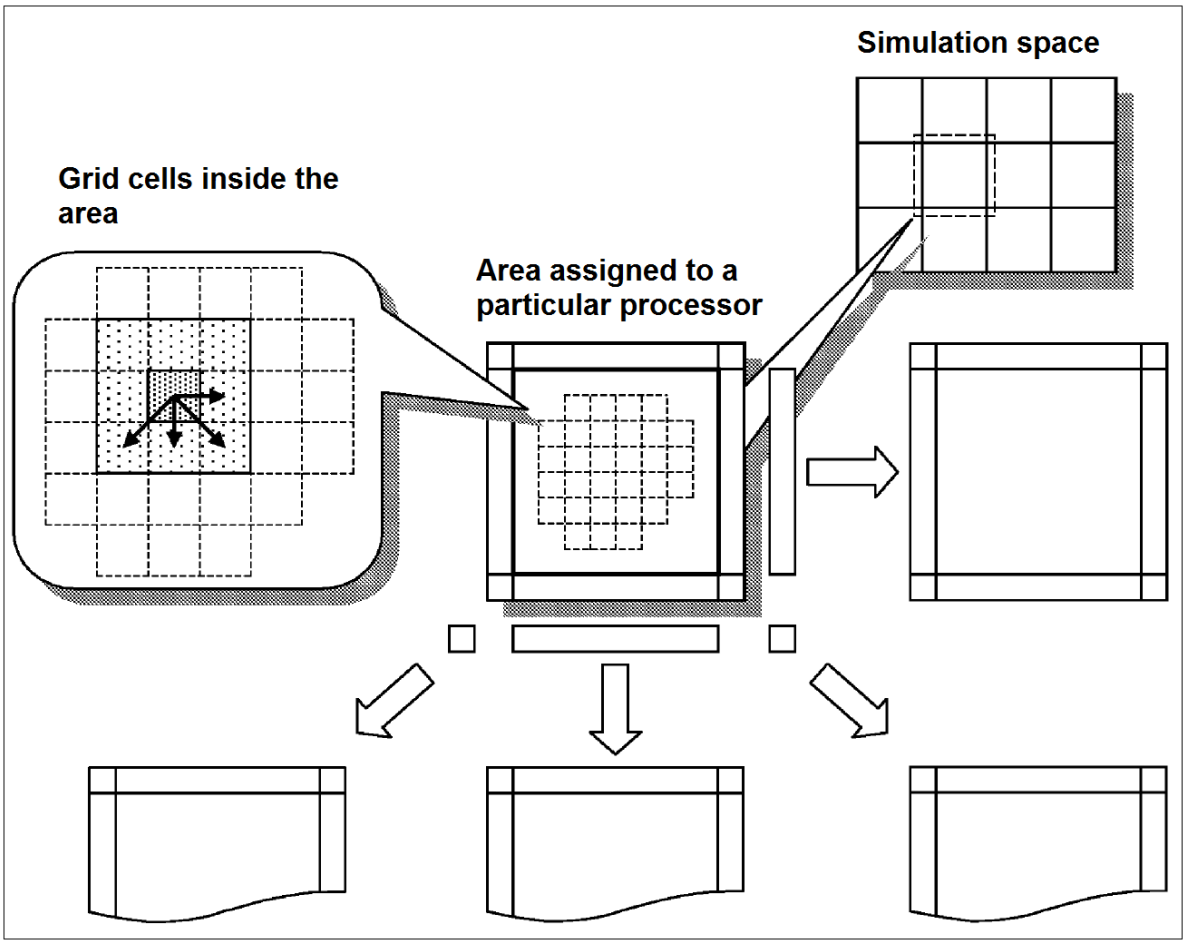


Figure 5.14: The space is divided into areas being assigned to different processors. Each such area contains cubic grid cells with side a_{cut} (source: modified from [115])

motion is being made for them only. The authors wrote that in case of multi-processor systems these calculations can be effectively parallelized. In order to achieve this, the simulation space is divided into bigger areas, and each area is assigned to a particular processor. Furthermore, the grid cells being on the borders of the bigger areas being assigned to different processors should be updated after the computations for all grid cells is completed in order to consider interactions and results of integration from all adjacent areas. See figure 5.14 for details.

Based on the aforementioned approach, we propose the algorithm presented below. The presented algorithm and time complexity analysis below are given for the case when everything is executed on one processor only. This can be seen as a theoretical analysis preparing the future extensions to the parallel computing with multi-processors, which will significantly speed up the computations. The proposed algorithm is as follows.

As in the aforementioned approach, the space is divided into a regular cubic grid. The side of the cube can be set differently, but we experimented with cubic cells with the side being equal to a_{cut}. Potential fields are assigned to corresponding grid cells depending on coordinates of their centers. In order to create and store this assignment, we use a multi-hash map (a multi-hash map is a hash map which can have more than one value for the same key). Each grid cell has its own hash key, and hash keys are calculated from coordinates of centers of potential fields in such a way that if and only if potential fields have centers inside the same grid cell then they have the same hash key. For instance, for the grid with cells with side being equal to a_{cut}, we create a hash key from coordinates as below. The following denotations are used:
$x_R, y_R, z_R \in \mathbb{R}$ – input coordinates;
$x, y, z \in \mathbb{I}$ – variables;
$k \in \mathbb{I}^+ \cup \{0\}$ – a hash key;
$x_{max}, y_{max}, z_{max} \in \mathbb{I}^+ \cup \{0\}$ – maximum allowed integer coordinates depending on how many bits are used in the representation of single (long) integer value inside the computer;
$>>$ – left bit shift operation (we assume that the lowest bit is in the right-most position);
$<<$ – right bit shift operation;
(<predicate>) ? <value1> : <value2> – a compact notation of **if–else** block being used in some programming languages, such as C++.

1: // Use the integer part of input coordinates only. By this we ensure that potential
2: // fields with centers within the same grid cell will have the same hash key
3: $x := \lfloor x_R / a_{cut} \rfloor$
4: $y := \lfloor y_R / a_{cut} \rfloor$
5: $z := \lfloor z_R / a_{cut} \rfloor$
6: **if** ($|x| > x_{max}$ **or** $|y| > y_{max}$ **or** $|z| > z_{max}$) **then**
7: // The key will not fit into long integer. Normally it means that the
8: // potential field is far away from the simulation area, so just ignore
9: // it and return the "bad key"
10: k := INVALID_KEY // Special value indicating that the key is invalid
11: **return**
12: **end if**
13: // create the hash key, which has the following bit representation:
14: // $|x|$ $|y|$ $|z|$ *<sign of x>* *<sign of y>* *<sign of z>*.
15: // Put absolute values of the coordinates into the hash key
16: $k := |z| + (|y| << bitShiftForY) + (|x| << bitShiftForX)$

17: // Put signs of the coordinates into the lowest three bits
18: $k := (k << 3) + ((z < 0)\ ?\ 1 : 0) + (((y < 0)\ ?\ 1 : 0) << 1) + (((x < 0)\ ?\ 1 : 0) << 2)$

For the hash key constructed in the above listing, the higher bits contain absolute integer values of the coordinates (with possibly equal number of bits per coordinate – depending on the size of (long) integer representation inside the computer), and lower three bits contain their signs: 1 for sign -1, and 0 for sign 0 or sign +1.

In order to find interactions of the potential fields having centers in the current grid cell with other potential fields using the above presented speed-up struture, the grid cells are iterated one by one. For every grid cell, only the interactions between potential fields inside this cell and the interactions between potential fields inside this cell and potential fields being in the N_{26}-neighboring cells are considered. The visited cells have the corresponding flag being set to true in order not to search for interactions with potential fields from already visited grid cells. Of course, the more there are potential fields in the system, the faster the described approach works compared to the naive $O(N_{pf}^2)$ algorithm.

Although in the worst case (when all potential fields are inside the same grid cell) the proposed algorithm still has $O(N_{pf}^2)$ time complexity, the average case time complexity (with potential fields being distributed nearly regularly among the grid cells) is

$$O\left(\frac{N_{pf}^2}{N_{cells}}\right), \tag{5.90}$$

where N_{cells} is the number of grid cells within the simulation area.
If $N_{cells} \sim N_{pf}$ then the time complexity can be rewritten as

$$O(N_{pf}). \tag{5.91}$$

As mentioned above, we experimented with cubic grid cells with the side being equal to a_{cut}. Therefore the factor in the proportion $N_{cells} \sim N_{pf}$ is reasonably small.

The space complexity is

$$O(N_{pf} + N_{cells}). \tag{5.92}$$

Indeed, we need an additional space to store all non-empty cells and to store the information about in which cell is each potential field. Since the number of cells within the simulation area N_{cells} is not greater than the total number of potential fields N_{pf}, the space complexity can be rewritten as

$$O(N_{pf}). \tag{5.93}$$

An additional improvement to the above algorithm can be an adaptive grid cell resolution, similar to the ideas from [83].

5.23 Force-feedback

We did not need to add an extra force-feedback, because the already existing force-feedback within our framework (see chapter 4 for details) gives good results, as expected. Indeed: our rigid-based collision detection and response does not allow the IP to go inside any object, and the corresponding force-feedback is delivered to the user. Further on, we brought this force-feedback into our haptic rendering framework with collision detection guarantee and with support for different deformation models (see section 5.1 for details about the framework). Since we use the potential fields–based deformation approaches described in this chapter together with this framework, the aforementioned collision detection guarantee and the force-feedback are still "in force". Furthermore, when the IP, which is considered as a repulsive potential field for our deformation approach (see section 5.16), interacts with other potential fields, it causes other potential fields to move, causing a deformation, which is then transfered to the object's bit cube. Once there is a deformation, the IP can be moved further "inside" the object until it hits its new border being computed using the updated bit cube of the object. This new border is considered at the moment as the rigid border of the object. Therefore the hit is computed and delivered to the user using the rigid-based approach. Once the object is deformed again, the IP can be moved further inside until it hits the new border of the object, which at that moment is again considered as the rigid border of the object. Then the process is repeated again. We can note here that the haptic thread works faster than the simulation (deformation) thread, and therefore for the most of the haptic frames the border of the object is considered unchanged, until there are new results from the simulation thread causing an update of the object's bit cube.

There is also an option to consider the forces caused by the interaction of the IP's potential field with other potential fields to be a part of the force-feedback. But according to our experiments and comparison of two approaches, reasonably good results are achieved without this option.

5.24 Time and Space Complexities of the Potential Fields Approach

First, let us estimate the time complexity. Each simulation step, the following is done:

1. The set of potential fields is being updated by reusing potential field objects (see section 5.11). This takes $O(N_{pf})$ time, where N_{pf} is the number of potential fields in the simulation system
2. Interacting potential fields are being searched. According to section 5.22, this takes $O(N_{pf})$ time in the average case and $O(N_{pf}^2)$ time in the worst case
3. Interaction forces are being computed for each interacting pair. Although in the worst case (when all potential fields are inside the same grid cell) there are $O(N_{pf}^2)$ interaction pairs, the number of pairs in the average case (when potential fields are distributed nearly regularly among the grid cells and the number of grid cells is proportional to N_{pf}) is proportional to N_{pf} (see section 5.22 for details). Therefore the average time complexity for this step will be $O(N_{pf})$, because computing forces for each pair takes $O(1)$ time
4. The integration of equations of motion is done for every active potential field. Since the integration takes $O(1)$ time per potential field, the time complexity for this step is $O(N_{pf})$.

As can be seen from the above summary, the overall worst case time complexity is

$$O(N_{pf}^2), \tag{5.94}$$

while the overall average time complexity is

$$O(N_{pf}). \tag{5.95}$$

The space complexity is

$$O(N_{pf}). \tag{5.96}$$

Indeed, we need to keep data for every potential field, such as the position, the velocity, the force and the mass. It requires $O(1)$ space. Therefore, all potential fields require $O(N_{pf})$ space. Further on, the speed-up structure to find interacting potentials requires $O(N_{pf})$. In the worst case there are $O(N_{pf}^2)$ interaction pairs, but it is possible to design the algorithm so that the pairs are not stored for later processing but are processed immediately. Furthermore, in the average case (see above for more details about what is called the average case) there are $O(N_{pf})$ interaction pairs only.

5.25 Update of Volumetric Data for the Potential Fields Approach

At the end of each simulation step, the segment's volumetric data should be updated according to the current configuration of potential fields. We do it as described below (all coordinates are rounded to integers where needed).

1. Reset array C_{vc}, which keeps the number of potential fields for each voxel within the simulation area A (where the position of the simulation area is defined by the position of the IP in the beginning of the simulation step). Array C_{vc} has the size of the simulation area and is reused every simulation step. We reset all elements of the array by setting them to -1 indicating that there is no potential field at the corresponding voxel

2. Fill C_{vc}. That is, for each potential field P with $P.isUsed$ = **true** we do the following:

   (a) If the *initial* position of the center of P is out of A then skip P (see remark 1 in section 5.11 for the definition of the initial position of the center of potential field)

   (b) If the value of the element of C_{vc} corresponding to the *initial* position of the center of P is -1 then set it to 0. This will indicate that there initially was a potential field at the current voxel

   (c) If the *current* position of the center of P is out of A then skip P

   (d) If the value of the element of C_{vc} corresponding to the *current* position of the center of P is -1 then set it to 0. This step will be followed by the next step where we increase the value of the element of C_{vc} by one, therefore the value of the current element will be greater than 0. The current step is needed to ensure that 0 value will be set only for those elements which initially had a potential field at the corresponding voxel

   (e) Increase the value of the element of C_{vc} corresponding to the *current* position of the center of P by 1

3. Update voxels within A. That is, for each element $C_{vc}[i]$ of C_{vc} do:

   (a) If $C_{vc}[i] = -1$ then skip the corresponding voxel, because there initially was no potential field at this voxel (i.e. it was empty space)

(b) If $C_{vc}[i] \geq 0$ is less then the threshold number (it is different for the original and cuboid potential fields based approaches) then the corresponsing voxel is set as empty, otherwise it is set as non-empty.

The time complexity of the above method is $O(N_{pf})$. Indeed, in order to fill C_{vc} we go through all the potenital fields once. Further on, in order to update voxels within A we go through each element of C_{vc} once. The number of elements in C_{vc} is the same as the number of voxels within A. Further on, the number of voxels within A is proportional to N_{pf} (see sections 5.10 and 5.18 for details).

The space complexity is $O(N_{pf})$, because we need to store the array C_{vc} having the size being proportional to N_{pf}.

5.26 The Global Simulation using Potential Fields

In addition to the local simulation approach, we propose the first prototype for the global simulation using the potential fields approach. The goal of this prototype is to show that our haptic rendering framework supporting different deformation models works well with a global deformation approach.

The outline of our global simulation algorithm is as follows.

1. Initialization (done before the start of the deformation simulation). Ideally, the union of all areas within the specified distance d_a being not less than $0.5a$ from the centers of potential fields should cover all non-empty voxels of the segment, and the configuration of potential fields should be in the equilibrium state (see figure 5.15). For the current global simulation prototype we do it as follows:

   (a) Set initial positions of potential fields within the segment. For our prototype, we put them at the regular interval from each other being lesser than a (this is needed for the next step), where the equilibrium distance a is chosen depending on the available computational power, so that the simulation is interactive. Furthermore, we put a potential field into the segment only if the number of non-empty voxels within $0.5a$ radius from its center is greater than the specified threshold

   (b) Run the potential fields based simulation until the potential fields system reaches an equilibrium state. During the simulation, interaction of potential fields with empty space is done as described in section 5.13

(c) Set the initial positions of centers of the potential fields to be equal to their current positions

(d) Bind the centers of the potential fields to their initial positions by the binding force as described section 5.12

(e) For each potential field P, associate voxels, which are within d_a radius from the center of P, with P. The voxels can be associated with several potential fields at the same time

2. Each iteration of the deformation simulation we do the following:

(a) Compute interactions between potential fields using similar equations as for potential fields for the local deformation simulation. The difference is that for the global deformation simulation we additionally compute binding to initial positions forces and do not compute interactions of potential fields with empty space and with borders of the simulation area because we make the simulation area for the global deformation simulation approach being unlimited. Additionally, parameters of the simulation model are different compared to the local simulation approach, e.g. a and m are typically larger

(b) For each potential field, if its center moved more than the specified threshold since the last iteration of the simulation loop then we update the positions of the associated voxels accordingly

(c) Do smoothing of the voxel positions depending on how many potential fields "own" the voxel, depending on translations of the "owning" potential fields and depending on other parameters. For the current version of the global simulation prototype, we move the associated non-empty voxels together with the centers of potential fields and invalidate the changed areas of the volumetric data.

As could be seen from the above algorithm, some voxels of the original segment may be omitted if the configuration of potential fields does not cover all non-empty voxels. It is so because the goal of the current global simulation prototype was to show that our haptic rendering framework supporting different deformation models works well with a global deformation simulation approach, while the global deformation approach itself is considered as a "black-box", and therefore its details are not important for the validation. In this sense, the proposed global simulation prototype using potential fields answers all the requirements. Improvement of the voxel coverage, as well as better initial positioning of potential fields and improved deformation simulation is planned for future

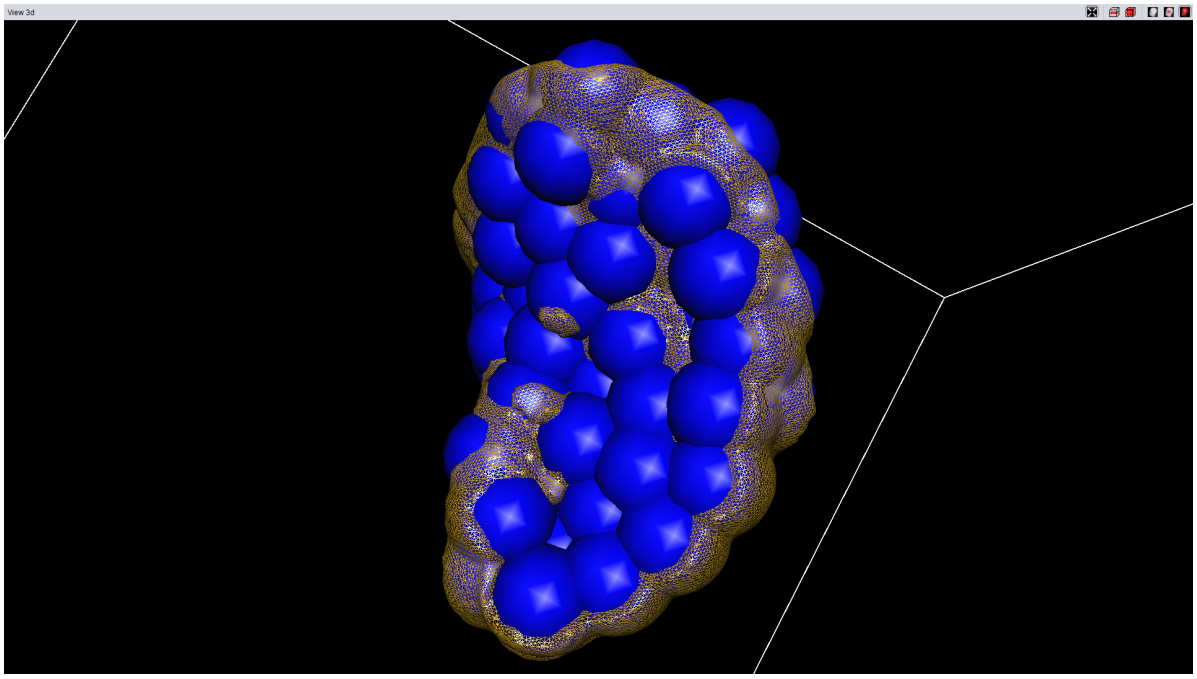


Figure 5.15: The prototype of the global deformation simulation using potential fields. Potential fields are illustrated as spheres of diameter a (the equilibrium distance). Each potential field "owns" voxels within d_a ($d_a \geq 0.5a$), which are associated with it

work. Thus, e.g. replacement of binding to initial position forces by the paradigm of interaction with empty space outside the segment (as for our local simulation approach) will allow to find the correspondence between parameters of the simulation model and parameters of real materials in the similar way as for the local simulation approach, but considering bigger "size" of potential fields. Additionally, the global simulation approach can be combined with our local simulation approaches.

5.27 Results

As mentioned in section 5.1, our deformation framework was designed for easy integration of different deformation simulation approaches into our prototype system. For each of our local and global potential field based deformation simulation approaches, a child class of the "Abstract Defo Algorithm" class is created. Within each of these classes, required abstract methods are overloaded with the actual logic of the concrete algorithm. Additionally, a child class of "Abstract Data for Defo Algorithm" class is created for each child class of "Abstract Defo Algorithm" in order to cache and/or keep the concrete deformation approach specific data between iterations of the deformation loop. The diagram showing the relations between all the classes is shown in figure 5.3. See section 5.1 for more details.

As mentioned in section 5.1, the deformation simulation is run in a separate deformation thread. This ensures that our prototype system works with the stable update rate being independent from the deformation simulation approach (if the PC has enough processor cores). To test this, we used the same real tomography data sets, including Torso (figure 5.16), Head_{big} (figure 5.17) and Head_{small} (figure 5.18), as in chapter 4 and in our works [225, 227, 226]. As was mentioned in the beginning of this chapter, we chose the method of potential fields for deformation simulation approaches used for validation of our prototype system. Based on it, we proposed local and global potential field based approaches and introduced novel cuboid potential fields. These methods were described in detail in the this chapter.

In more detail, we measured the haptic update rate for our improved joined collision detection and response approach described in chapter 4 and in our work [226]. We did the measurements without a deformation simulation, with the local potential fields based local deformation simulation, with the cuboid potential fields based local deformation simulation and with the global deformation simulation. The haptic update rate was measured for real haptic devices and during the maximum load for our joined collision detection and response approach, and during the maximum load for the selected deformation simulation. Under the "maximum load" a continuous interaction with scene objects is meant. The same haptic devices as in chapter 4 and in our works [225, 227, 226] were used (see section 4.10 for details). Additionally, compared to chapter 4 and our works [225, 227, 226], we used a less powerful moderate end-user PC (4 x AMD FX-4100 CPU, 8 GB RAM, NVIDIA GeForce GTS 450).The obtained haptic update rates for all the measurements are shown in tables 5.1 and 5.2, where:
Data – the name of data set;

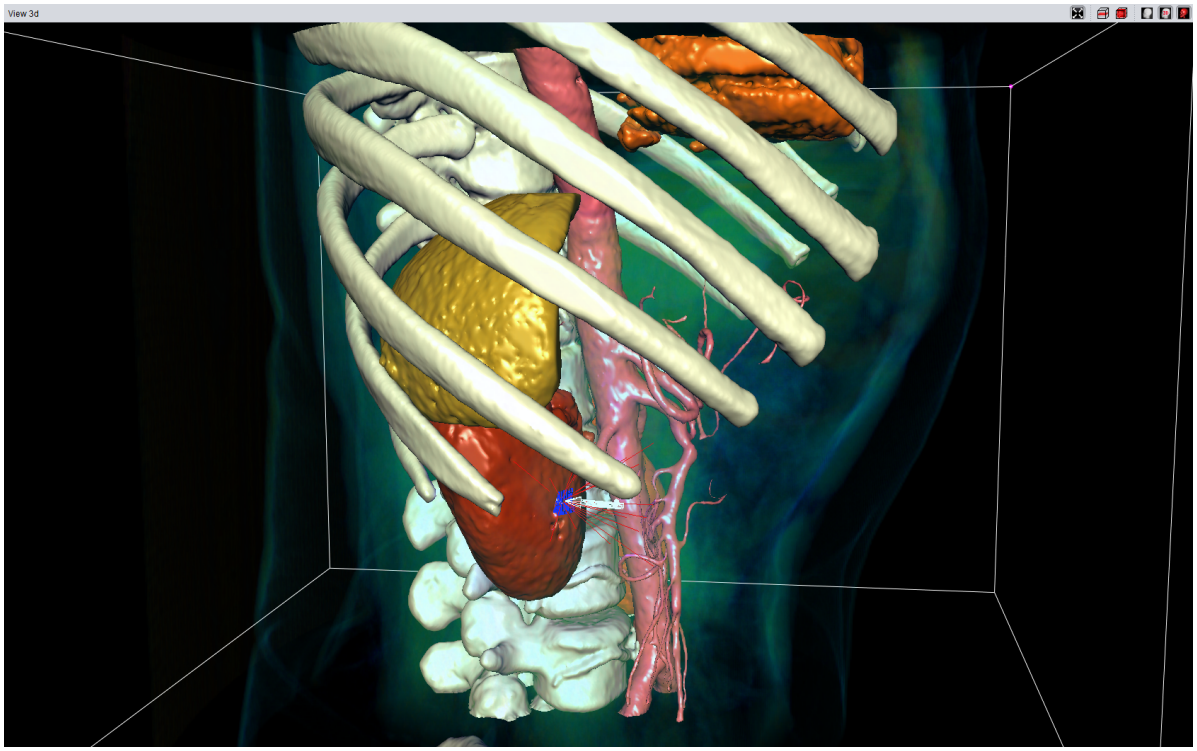


Figure 5.16: The Torso data set with visual debug information

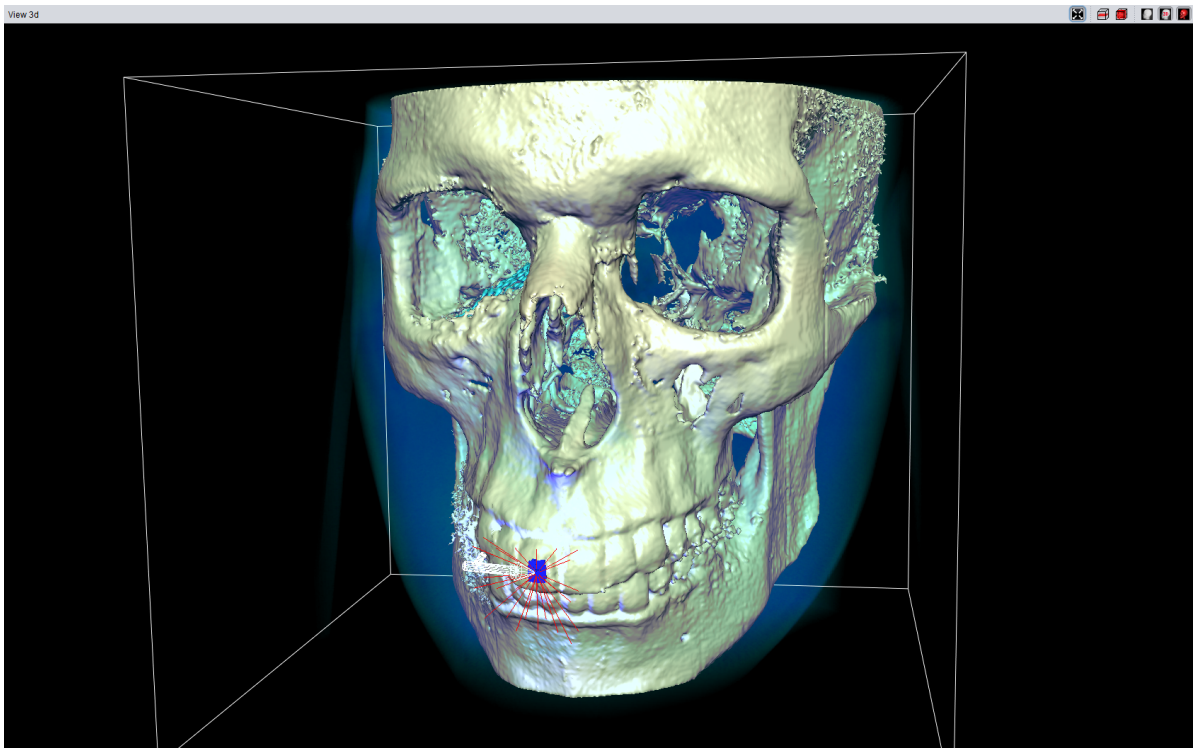


Figure 5.17: The data set Head_{big} with visual debug information

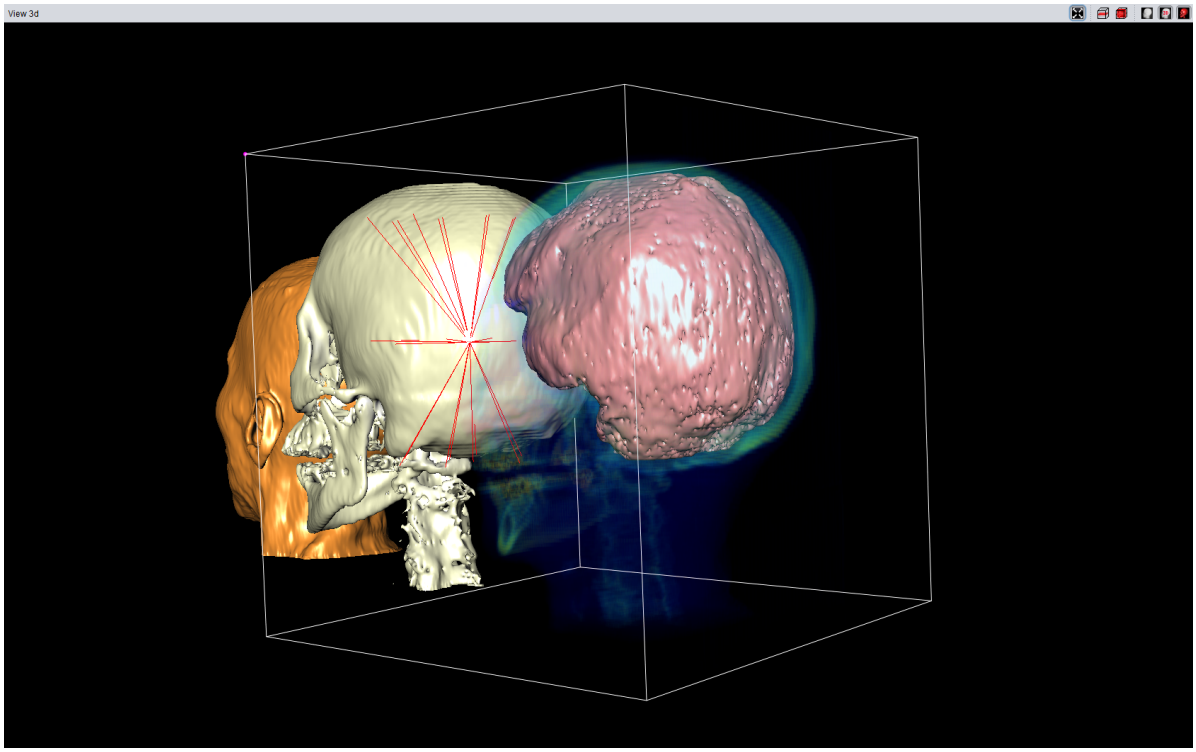


Figure 5.18: The data set Head_{small} with visual debug information

Size – the size of volumetric data of the given data set;
Triangles – the initial number of triangles in the scene (i.e. when all segments are not modified) for the graphics rendering as a reference;
No Defo Rate – the update rate of our prototype system (for our improved joined collision detection and response approach described in chapter 4 and in our work [226]) without a deformation simulation;
Local PFs Rate – the update rate of our prototype system with the local potential fields based local deformation simulation;
Local Cuboid PFs Rate – the update rate of our prototype system with the cuboid potential fields based local deformation simulation;
Global PFs Rate – the update rate of our prototype system with the global potential fields based deformation simulation.
The haptic rates were averaged over one second time intervals in order to minimize fluctuations within each time interval. Sixty one–second samples were collected per each experiment. The average resulting update rates presented in table 5.2 are the update rates found by averaging values for all 60 one-second samples for each experiment.

As expected, the results show that the haptic update rate of the prototype system remains stable when a deformation simulation is added. It does not decrease for both

Table 5.1: Resulting ranges of update rates for the deformation framework

Data	**Size**	**Triangles**	**No Defo Rate**	**Local PFs Rate**	**Local Cuboid PFs Rate**	**Global PFs Rate**
Head_{small}	113x256x256	690k	61–77 kHz	56–74 kHz	57–77 kHz	109–129 kHz
Torso	520x512x512	2,222 Mi	47–65 kHz	59–67 kHz	65–71 kHz	49–140 kHz
Head_{big}	464x532x532	6,136 Mi	54–74 kHz	67–74 kHz	64–72 kHz	55–127 kHz

Table 5.2: Resulting average update rates for the deformation framework

Data	**Size**	**Triangles**	**No Defo Rate**	**Local PFs Rate**	**Local Cuboid PFs Rate**	**Global PFs Rate**
Head_{small}	113x256x256	690k	66 kHz	68 kHz	64 kHz	120 kHz
Torso	520x512x512	2,222 Mi	57 kHz	63 kHz	68 kHz	96 kHz
Head_{big}	464x532x532	6,136 Mi	69 kHz	72 kHz	67 kHz	96 kHz

local and global simulation approaches. Furthermore, the haptic update rate is still an order of magnitude higher than the required 1 kHz. As in chapter 4 and in our work [226]), the update rate still does not depend on the size of the volume data because the algorithm works locally. The update rates are given as a range because segments have different shape and topology and therefore calculation of interactions takes slightly different time in different areas of segments. Another reason is that the user may not touch the segment's surface 100% of the time if the surface is uneven. E.g. the handle may "fly" over some small concavities if the user moves it fast. Additionally, for the deformation simulation, the IP never penetrates or goes through any object, but due to specific of the potential fields based approaches there will be some haptic frames when all potential fields are pushed away from the IP, and therefore the IP will be in the empty space. In such cases the IP may remain in the empty space for a number of haptic frames, what adds fluctuations into the averaged haptic update rate over one-second intervals. These cases happen more often for global potential fields based deformation simulation because of the bigger size of potential fields and because of the current approach used for the global simulation (see section 5.26 for details). This results in the higher update rates for the upper values of the range for the global simulation. Further on, the lower value of the range of update rates for the global simulation for Head_{small} data set is higher compared to other data sets because of the smaller size of the data set, which resulted in faster deformation simulation and smaller number of potential fields used during the simulation. Among other reasons for the resulting update rates to be presented as a range, there are fluctuations in the update rate in case of interactions of the IP with thin objects or objects with empty-space voxels inside (i.e. "holes"). E.g. if there is an interaction with a skin layer of the Head_{small} data set, the handle will penetrate through it during the deformation simulation, and then there will be some haptic frames with no interactions until the handle reaches the bone surface. The haptic frames with no interactions are run very fast and therefore will increase the resulting averaged update rate.

The update rates of the deformation loop for the same data sets for the local potential fields approach and for the cuboid potential fields approach are shown in table 5.3, where "av." means the average update rate of the deformation loop. The average update rate for the deformation loop was acquired in the same way as the average haptic update rate for our improved joined collision detection and response approach described in chapter 4 and in our work [226] (table 5.2). The measurements were conducted in the same way as for tables 5.1, 5.2. As could be seen from table 5.3, the cuboid potential fields approach has the same order-of-magnitude simulation time while providing a simpler and more natural simulation for volumetric data using less potential field objects. For

Table 5.3: The update rates for the deformation simulation

Data	**Size**	**Triangles**	**Local PFs Rate**	**Loc. Cub. PFs Rate**
Head_{small}	113x256x256	690k	48–92 (av. 60) Hz	29–45 (av. 30) Hz
Torso	520x512x512	2,222 Mi	49–54 (av. 51) Hz	26–30 (av. 28) Hz
Head_{big}	464x532x532	6,136 Mi	51–100 (av. 70) Hz	21–23 (av. 22) Hz

Table 5.4: The update rates for the deformation simulation, normalized by the number of voxels in the simulation area

Data	**Size**	**Triangles**	**Local PFs Rate**	**Loc. Cub. PFs Rate**
Head_{small}	113x256x256	690k	48–92 (av. 60) Hz	116–180 (av. 120) Hz
Torso	520x512x512	2,222 Mi	49–54 (av. 51) Hz	104–120 (av. 112) Hz
Head_{big}	464x532x532	6,136 Mi	51–100 (av. 70) Hz	84–92 (av. 88) Hz

the same number of potential fields, the simulation speed for the cuboid potenial fields approach is lower compared to the "classical" local potential fields approach because of the way the cuboid potential is computed, because we consider interaction with more neighbors resulting in more accurate simulation and because the same number of cuboid potential fields covers 4 times more voxels than "classical" potential fields. The cuboid potential fields approach has the same time complexity as the "classical" local potential fields approach, while still ensuring stability and smoothness of the force feedback. Furthermore, if normalized by the number of voxels in the simulation area, the simulation speed for the cuboid potential fields approach is faster than for the "classical' potential fields approach – see table 5.4.

5.28 Results – Use Cases

5.28.1 Adding Meta–Information

Riga in his Bachelor work [194] proposed a method based on our approach. The author of the current PhD thesis was a co-supervisor of his work. The motivation for [194] was to enable the user to add a meta–information to virtual surfaces simply and naturally

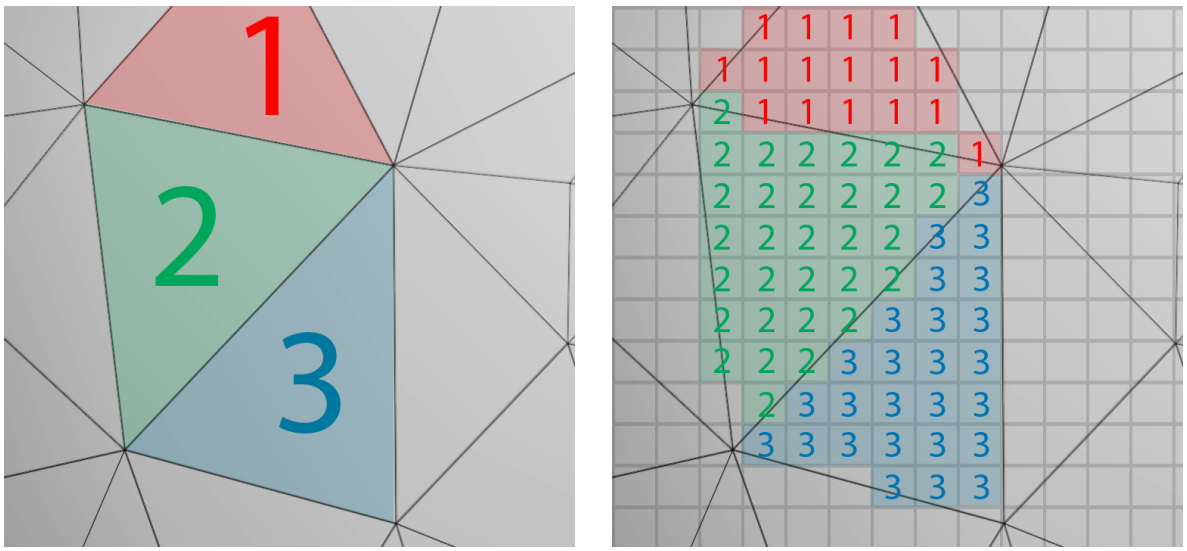


Figure 5.19: Triangles before (*left*) and after (*right*) discretization (source: [194])

using a haptic device. This is needed e.g. for marking anatomic regions and landmarks. The approach is used together with the YaDiV Deformable Model Framework (YDMF) developed at Welfenlab by Becker, Friese et al. [23]. The idea of the approach is as follows.

The YDMF uses a triangulated surface for the object representation. Since our approach works with volumetric data, the output of the YDMF is voxelized first. But instead of using a bit cube for the voxel representation of an object (a segment), the triangle index is used. That is, each voxel stores the value equal to the index of the triangle whose surface the voxel intersects, or the special index indicating no triangle. This is illustrated in figure 5.19.

During the haptic interaction, our approach is used with the voxelized data obtained as described above. In order to find the triangle with which the user is currently interacting, our ray casting approach is used, and then the triangle is obtained using the index being stored in the hit voxel – see figure 5.20. This takes $O(1)$ time. If the user presses the "mark it" button, the triangle is marked. In order to improve the force feedback, the triangle normal can be used instead of the normal from our approach. In order to further improve the quality of the force feedback, Phong shading of normals of adjacent triangles can be employed. This would take $O(1)$ because the adjacent triangles can be obtained in $O(1)$ time using the pre-computed adjacent triangles structure, which already exists in YaDiV.

The approach requires additional $O\left(N_{voxels}\right)$, where N_{voxels} is the number of voxels used

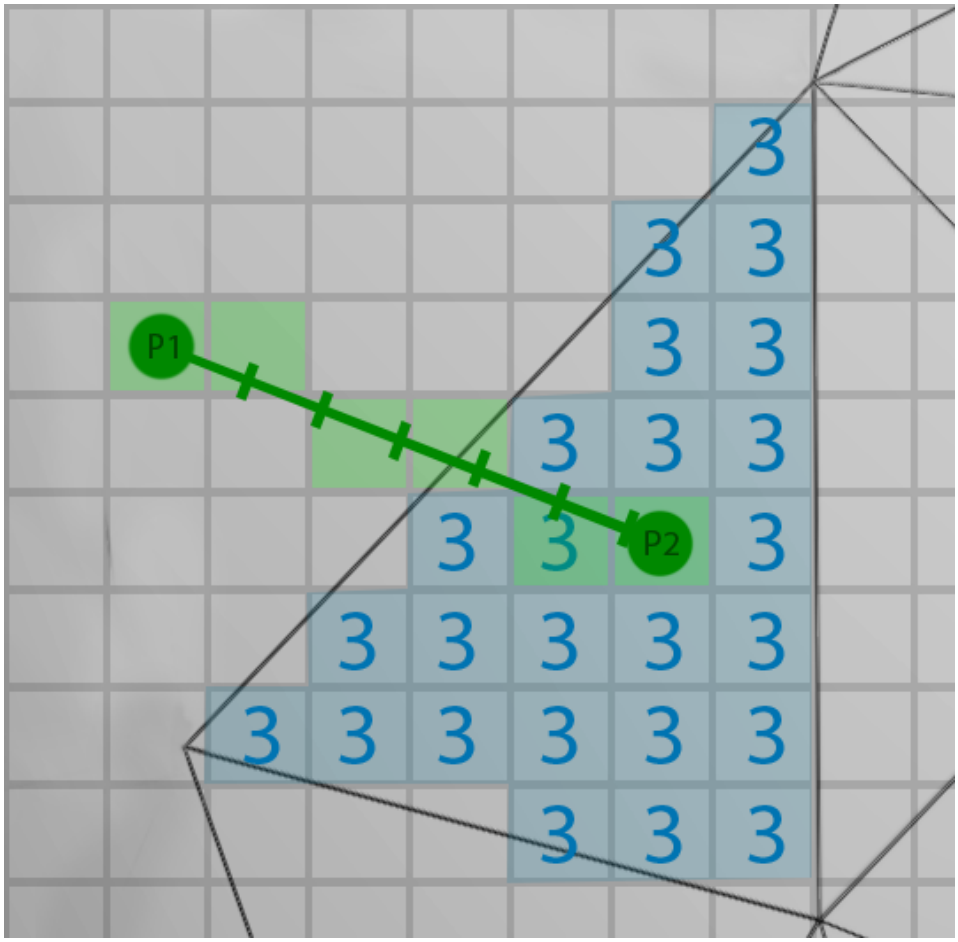


Figure 5.20: Ray casting with 1–voxel from step P1 to P2 to find the hit voxel for the voxel cube with triangle index coding (source: [194])

Table 5.5: Haptic update rates for the approach presented in [194]

Data set	**Triangles**	**Update Rate (no collision)**	**Update Rate (collision)**
001_pelvis_final_l Improved_Goal	478480	187 kHz	92 kHz
pelvis_r_5	594624	195 kHz	115 kHz
ydm_testsphere1	700	289 kHz	115 kHz

for voxelization of the given object.

The resulting update rates for the described approach are shown in table 5.5. The same high-end user PC as in section 4.14 was used for the tests (8 x Intel Xeon CPU W5580 @ 3.20 GHz, 24 GB RAM, NVIDIA Quadro FX 5800). The data sets used for the tests are shown in figures 5.21, 5.22 and 5.23. As concluded in [194], the haptic update rate is much higher than the required minimum of 1 kHz. As expected, the haptic update rate without a collision is higher than the one during a collision. Further on, the number of triangles does not affect the update rate, because the volumetric representation is used for the haptic interaction. The difference in update rate can appear due to different resolution of the voxel grid, which is currently determined by the side of the smallest triangle edge.

Further research is planned to find more optimal ways to determine the resolution of the generated voxel grid depending on the input, as well as ways to make an adaptive grid or to employ local voxelization only around the IP. Another research direction is to incorporate a deformable model within our deformation framework to provide the user with an advanced force feedback and to allow to deform the marked areas.

5.28.2 MultiScaleHuman Project

Our prototype system was presented on the CeBIT international computer expo 2013 and 2015 within the scope of the MultiScaleHuman project [186] (figure 5.24). The Marie Curie ITN MultiScaleHuman project, funded by the European Union, visualizes the functionality and articulation [146] of the human body under a dynamic 3D multi-scale approach [221] – see figure 5.25. The goal of the project is to obtain a better understanding of joint diseases and to enable a more efficient diagnosis and treatment of patients, such as a musculoskeletal disease of the human knee.

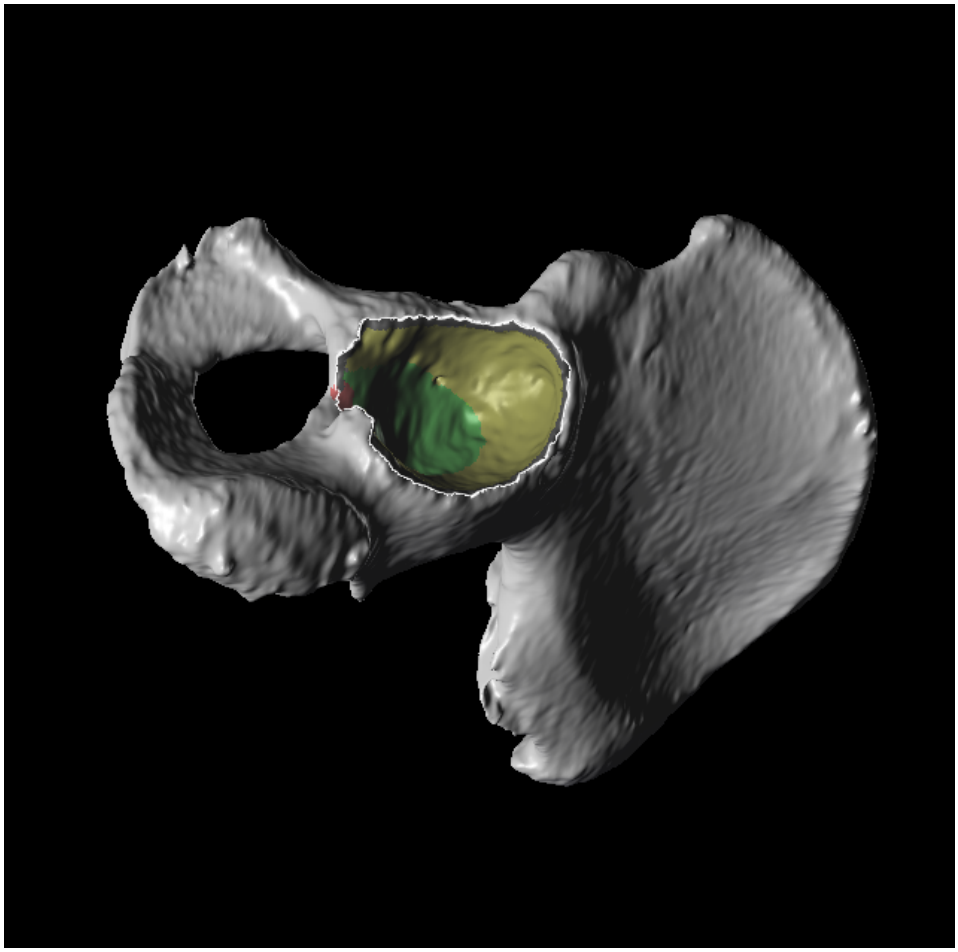


Figure 5.21: The *001_pelvis_final_l Improved_Goal* data set used in [194]) (source: [194])

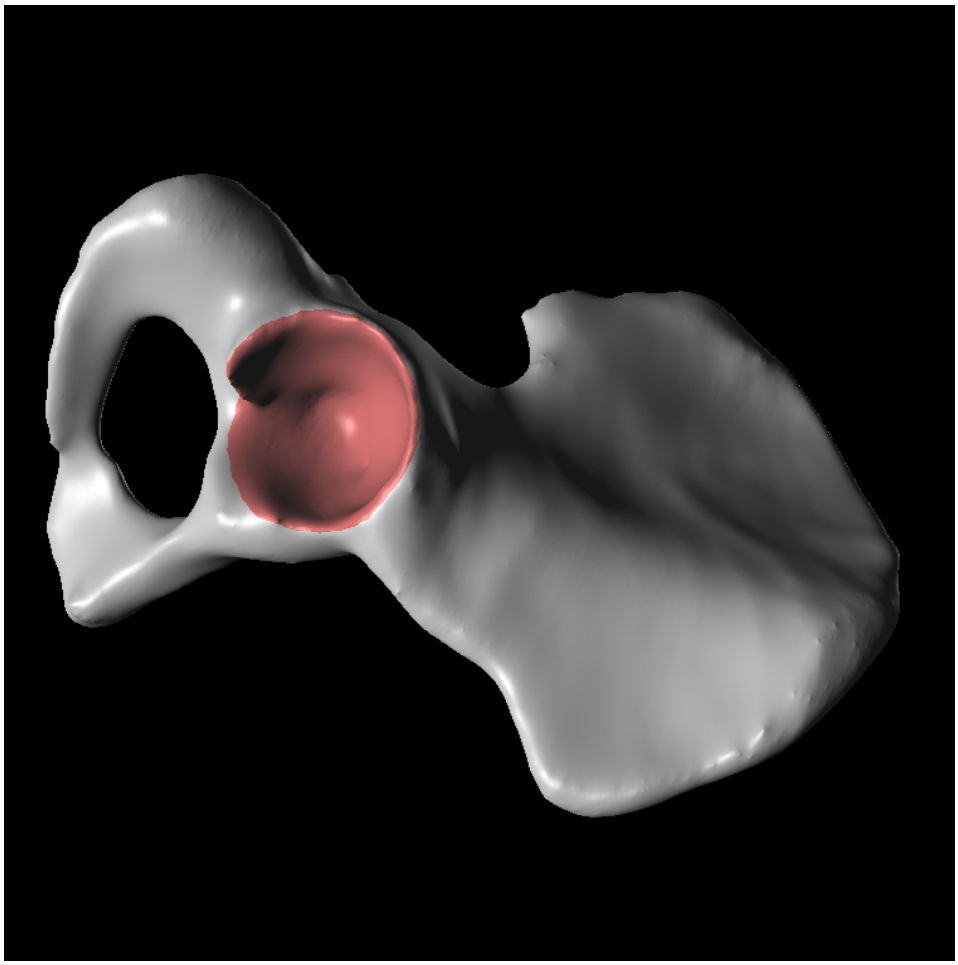


Figure 5.22: The *pelvis_r_5* data set used in [194]) (source: [194])

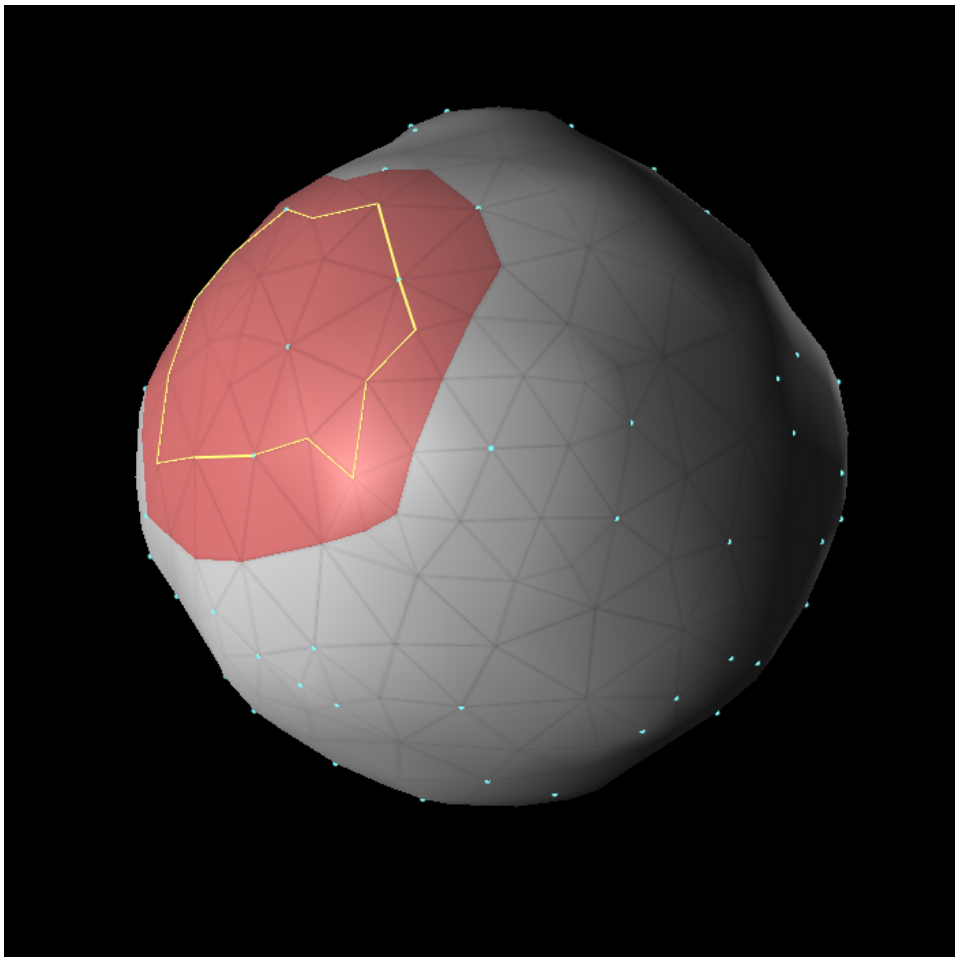


Figure 5.23: The *ydm_testsphere1* data set used in [194]) (source: [194])

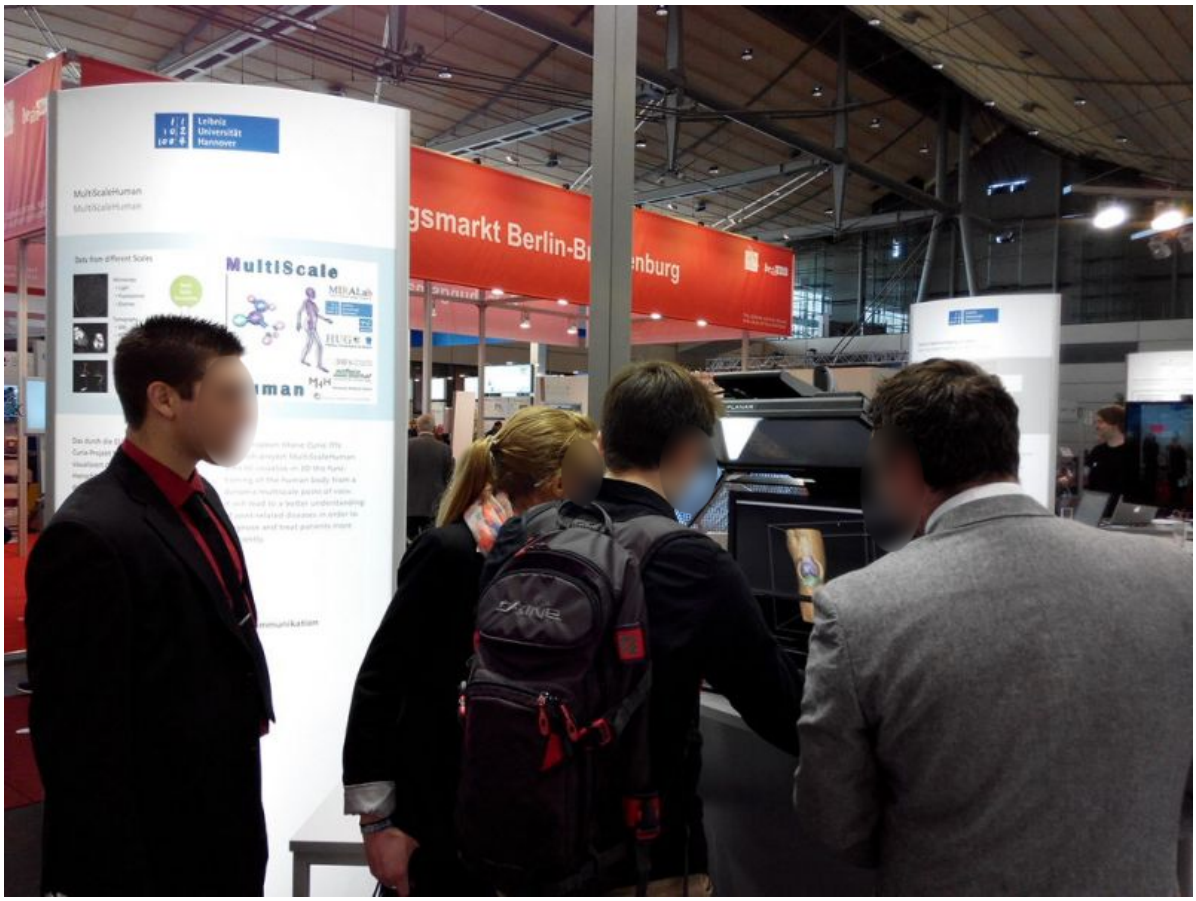


Figure 5.24: Our prototype system presented on the CeBIT 2015 within the scope of the MultiScaleHuman project (source: [186])

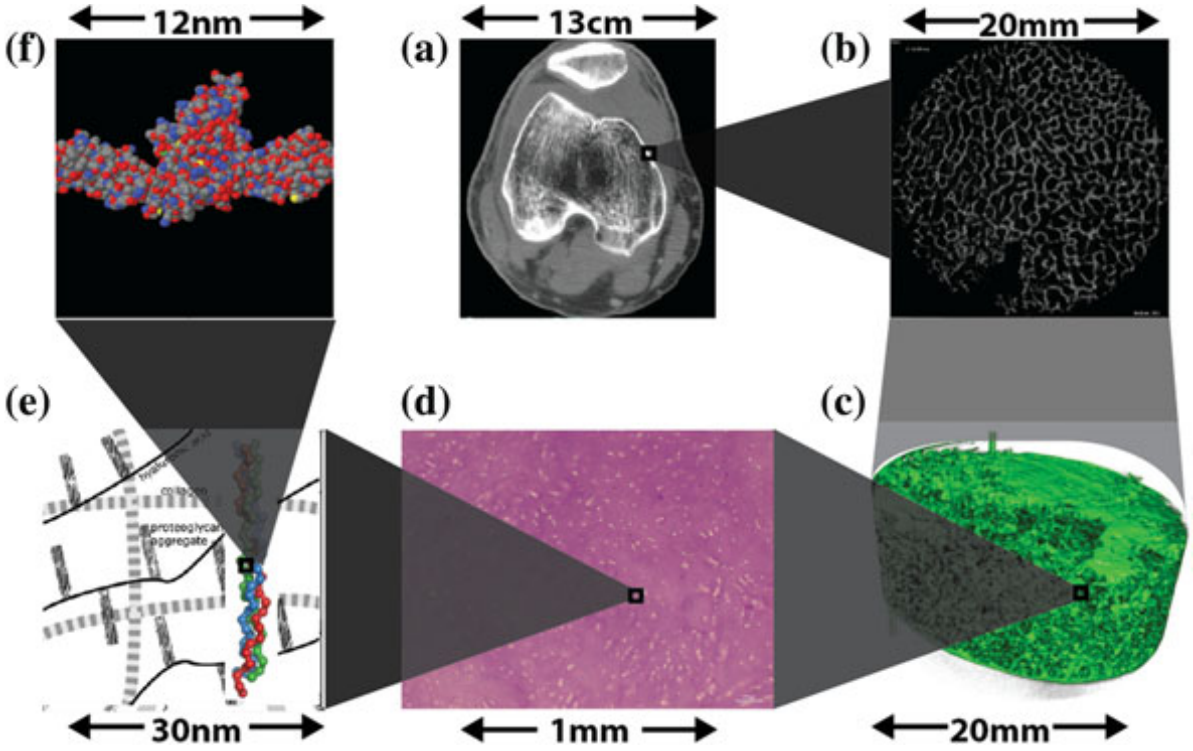


Figure 5.25: Example of the knee joint multi-scale data set (source: [221])

The MultiScaleHuman project proposes a multi-modal interaction with a focus on natural 3D interaction [199], as well as a semantically adaptable integrated visualization from different spatio-temporal scales [220] and a visualization from the multi-scale biomedical knowledge represented by an underlying ontology [4]. The haptic interaction was based on our work (see [220]).

5.28.3 Simulation

The developed local deformation simulation approaches together with our deformation framework can be used for simulation of drilling and for simulation of needle insertion.

Haptic techniques can be used for simulation of drilling or content removal. Among the existing methods, one can mark out the following. Agus et al. [136] used haptic rendering for drilling within the mastoidectomy simulator. "Subjective input" was used to tune the parameters that control force feedback. In [5] the authors developed an analytical model of bone erosion as a function of applied drilling force and rotational velocity. Petersik et al. [183] presented a penetration-based approach with the main application being a simulator for petrous bone surgery. They used the volume modification approach from

[184], but without a simulation model of the material itself. Authors of [29] presented a system where the tool's voxels intersecting the object's (bone) voxels remove an amount of the object depending on the tool's voxels' drilling power. Acosta et al. [1] presented a craniotomy surgical simulator which uses a modified voxmap–point-shell model. The bone erosion model in this work is based on density, on the point's erosion factor and on the tool's bit rotational speed, but there is no physical model of the material itself. Sewell et al. [205] made a study about the effect of haptic training on surgical drilling proficiency showing the benefit of haptic training. The authors used a horizontal plane which "resists" until a certain force is applied for a certain time. Kim et al. [2] presented an approach based on point-shell (surface) and signed-distance fields (tool). The authors used a penalty based collision response. They also used boolean operations on the tool and on the object as the material removal model, with material stiffness as an additional parameter (no physical model was employed). Wu et al. [245, 244] presented a voxel-based approach to simulate dental drilling. The authors defined two layers of voxels on the drill, where the boundary voxels are only employed to compute force feedback and the interior voxels are adopted to remove materials from teeth. The authors used a physical model for resistance force but a geometrically based one for material removal. Rhienmora et al. [193] presented a dental training simulator that uses an open source library called PolyVox [241]. The authors of [182] proposed algorithms to generate realistic cut simulations using a mass-spring model. Additionally, they presented a drill effect being implemented by removing the voxels which are located inside the virtual tool. No physical model of the material behavior during material removal (drilling) is employed. Bogone et al. [34] presented a method supporting multiple materials and material removal. Collision detection and force-feedback are based on our collision detection and force-feedback from [227]. The material removal approach depends on material density, tool's drill and wear coefficient, and there is no simulation of material itself. Stredney et al. [217] made a simulation of procedural drilling techniques for neurosurgical training. Force feedback is calculated on the basis of intensities of volume data. No further details regarding the haptic rendering method and the physical model are provided.

As was noted in the above overview, most of drilling approaches do not have a physical model for the material itself. It is replaced by a model describing when to "disable" a certain voxel. Contrary, our "regular" and cuboid potential fields based local simulation approaches can simulate material removal not by "disabling" voxels but by their rearrangement within the object. The scenario where the user drills the bone is shown in figure 5.26. The force feedback over 1000 ms of the deformation simulation for this scenario is shown in figure 5.27. As shown in the figure, the force-feedback is stable and

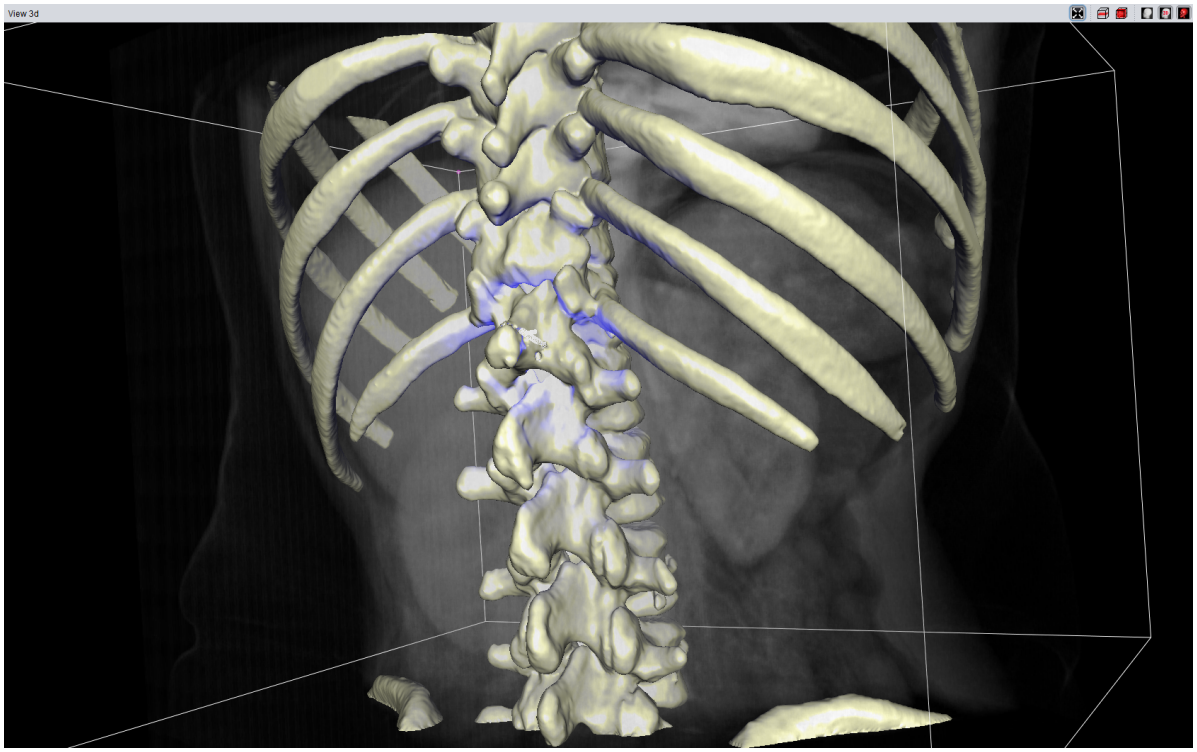


Figure 5.26: The Bone segment for the bone drilling scenario

has small fluctuations reflecting the rearrangement of voxels within the object. Similar charts were obtained for other segments which we used for the drilling experiments.

For the simulation of needle insertion using haptic techniques, one can mark out the following works. Coles et al. [49] proposed an interventional radiology procedures simulator with augmented reality techniques. Webster et al. [234] presented a suturing prototype. Heng et al. [91] proposed a Chinese acupuncture learning and training system employing an approximation defining different tissue states and employing break limit based, viscosity based and penalty based techniques for soft and hard tissues.

Our potential fields based local deformation simulation model allows simulation and feeling of different tissues. For the test we use a scenario where the user penetrates the hepar (liver) but cannot penetrate the bone – see figure 5.30. The force feedback over time for this scenario is shown in figure 5.31. In another test scenario, the user penetrates the skin but cannot penetrate the skull bone – see figure 5.28. The force feedback over time for this scenario is shown in figure 5.29. As shown in the charts, the force feedback increases when the bone is hit, preventing the user to easily penetrate into it. Further on, the force feedback keeps increasing as long as the user presses stronger and stronger trying to penetrate into the bone.

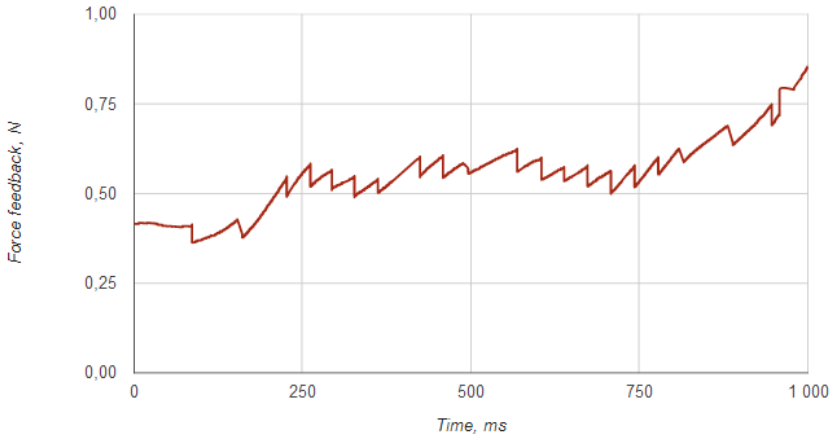


Figure 5.27: The force feedback for the bone drilling scenario

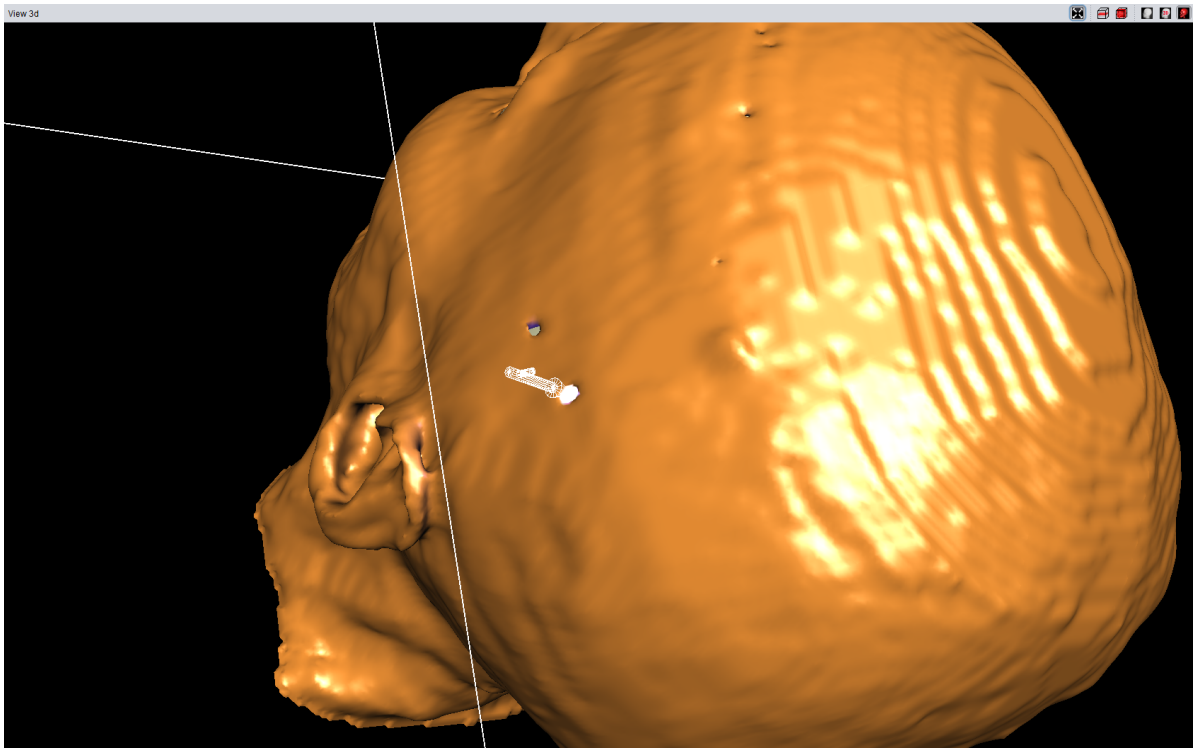


Figure 5.28: The Skin and the Skull segments for the needle insertion scenario

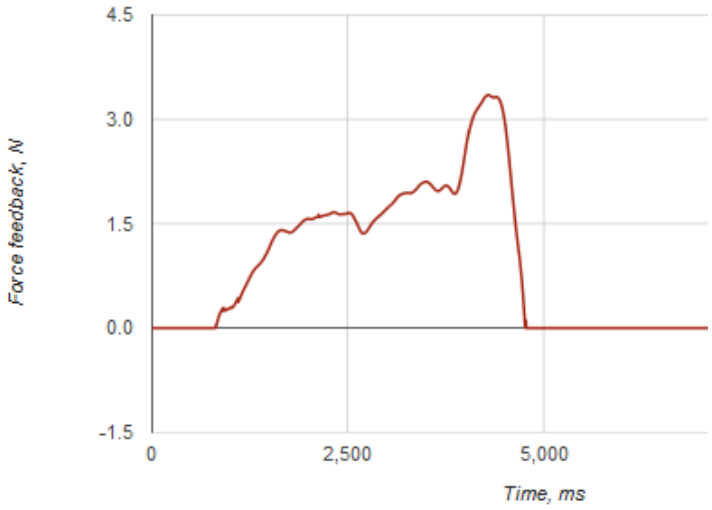


Figure 5.29: The force feedback for the skin and skull bone penetration scenario. The force feedback increases the first time starting from 800 ms – when the skin is penetrated. The force feedback increases the second time starting from 4000 ms – when the bone is hit

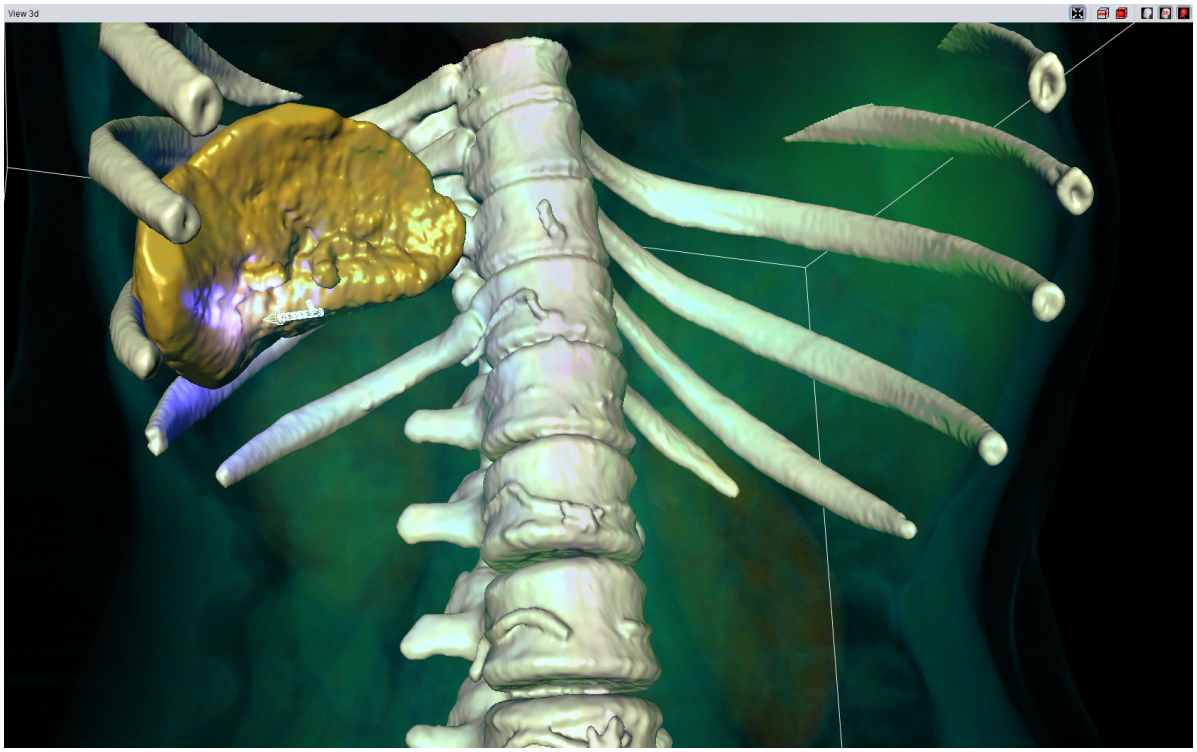


Figure 5.30: The Liver and the Bone segments for the needle insertion scenario

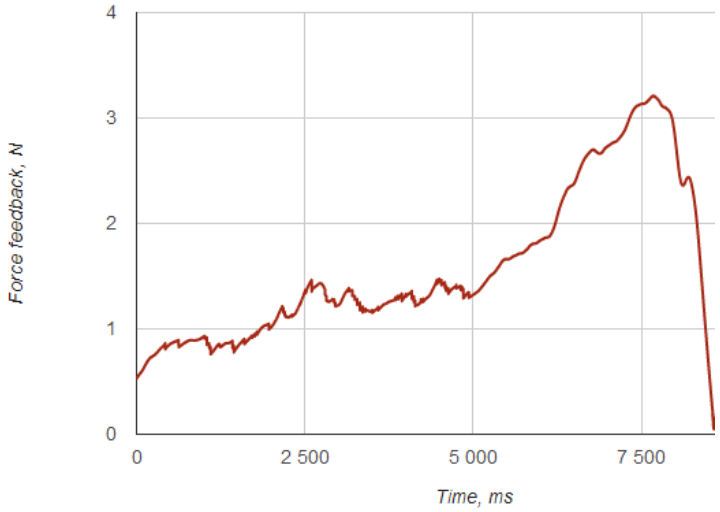


Figure 5.31: The force feedback for the liver and bone penetration scenario. The force feedback increases starting from 5000 ms – when the bone is hit

The following parameters for the "regular" and cuboid potential fields based local deformation simulation models were used for the test cases for drilling and needle insertion (see sections 5.14, 5.15, 5.17 and 5.19 for details of how the parameters are used):

- For the bone: $[E_{min}, E_{max}] = [5 \cdot 10^9, 21 \cdot 10^9]$ Pa, $[\nu_{min}, \nu_{max}] = [0.30, 0.32]$, $[HU_{min}, HU_{max}] = [700, 3000]$ HU, $[\rho_{min}, \rho_{max}] = [920, 1900]$ kg/m^3
- For the soft tissue: $[E_{min}, E_{max}] = [3.4 \cdot 10^4, 3.5 \cdot 10^4]$ Pa, $[\nu_{min}, \nu_{max}] = [0.47, 0.48]$, $[HU_{min}, HU_{max}] = [10, 60]$ HU, $[\rho_{min}, \rho_{max}] = [1100, 1200]$ kg/m^3.

The parameters were taken from different sources, including [62], [166], [68], [246]. As proposed in section 5.15, in order to take into account the heterogeneity of the simulated material, we make interaction forces for each pair of potential fields depend on intensities of the corresponding voxels.

5.29 Discussion and Future Outlook

We presented a flexible deformation framework allowing us to use our improved approach of haptic rendering of volume data with collision detection guarantee which has been presented in chapter 4 together with different deformation approaches. The framework,

as well as our deformation simulation approaches, was fully developed by us from scratch, without the use of any third party libraries.

In section 5.1 we proposed our framework and described it in detail. Further on, we proposed an approach to interactively visualize the results of the deformation simulation for the chosen deformation model within our prototype system. In more detail, we showed how to transfer changes in the object's structure to its volume representation, as well as how to effectively update the graphics surface representation from the changed volume representation. In more detail, in section 5.2 we showed how to perform fast update of the part of the object's surface triangulation affected by the deformation. Further on, we showed how to effectively smooth the retriangulated area so that its borders match with the borders of the rest of the surface.

In order to validate our deformation framework, we chose the method of potential fields in order to introduce our novel local deformation simulation approaches. Additionally, we introduced the prototype of the global potential fields based deformation approach. The potential fields approaches are a good "illustration", because they initially do not have the properties of our haptic rendering approach. Firstly, we presented our potential fields based local deformation simulation approach with the moving local simulation area and how the IP interacts with the objects. Further on, in section 5.18 we introduced the novel cuboid potential fields and showed that they fit well for the representation of volumetric data since the volumetric data consists of voxels being cuboids. Based on the cuboid potential fields, we proposed our cuboid potential fields based local deformation simulation approach. Next, we showed how to establish the correspondence of our proposed potential fields deformation simulation models to parameters of real materials and showed how we took heterogeneity of the simulated material into account. Additionally, we extended the classical potential fields approach in other aspects, such as adding additional forces and parameters to the model. Further on, we showed how we set initial positions and velocitites of potential fields, how we reuse potential fields objects, which kind of speed-up structures we used to find collisions, how we ensured stability of the simulation system, how we computed force-feedback, and other aspects of the described deformation models. Additionally, for cuboid potential fields we showed that the equilibrium distance was chosen correctly. Further on, a prototype of a global potential fields based deformation approach was presented.

As discussed in section 5.27, it was easy to built in different deformation simulation approaches into our deformation framework, as expected, because we designed our deformation framework in this way. Furthermore, as expected our previously developed haptic rendering approach added its properties including collision detection guaran-

tee and non-penetration guarantee to the employed deformation simulation approaches. This is especially important for such delicate procedures as pre-operation planning. Additionally, the resulting combined approach does not require any pre-calculated structure and works "on the fly". Further on, the results of tests with real volumetric data showed that the haptic update rate of our deformation framework remained stable when a deformation simulation was added. It did not decrease for both local and global simulation approaches. Furthermore, the haptic update rate was still an order of magnitude higher than the required 1 kHz. Further on, the results showed that our novel cuboid potential fields approach provides a simpler and a more natural simulation for volumetric data with one to one correspondence between potential fields and voxels within the local simulation area. The approach has the same time complexity as the "classical" potential fields approach, while still ensuring stability and smoothness of the force feedback. Furthermore, if normalized by the number of voxels in the simulation area, the simulation speed for the cuboid potential fields approach is faster than for the "classical' potential fields approach. In order to cover a larger simulation area and to have much shorter integration step resulting in more precise deformation simulation, the potential fields simulation can be speeded-up by parallelization on GPUs and/or on multi-processor systems.

A number of possible practical use cases were presented in section 5.28. Such, in section 5.28.1 we presented an approach to add meta–information to virtual surfaces simply and naturally using a haptic device. This is needed e.g. for marking anatomic regions and landmarks. The author of the current PhD thesis was a co-supervisor of this Bachelor thesis. In the Bachelor thesis, the virtual surface was voxelized. But instead of using a bit cube for the voxel representation of an object (a segment), the triangle index coding was used. This information was used later to effectively find the hit triangle during the interaction using a haptic device. Further on, our prototype system was presented on the CeBIT international computer expo 2013 and 2015 within the scope of the Marie Curie ITN MultiScaleHuman project, which was funded by the European Union (see section 5.28.2). Additionally, in section 5.28.3 we showed that the developed deformation framework can be used for the simulation of drilling and for the simulation of needle insertion. Our local potential fields model allows simulation and feeling of different tissues. Such, we presented an interaction scenario where the user can penetrate the liver but cannot penetrate the bone.

As a future work, we plan to make areas within the same object being deformable or non-deformable by setting which areas should and which areas should not be updated by the potential fields approach, or which areas should be considered empty space. Moreover,

as mentioned above, the potential fields simulation can be speeded-up by parallelization using GPUs and/or multi-processor systems in order to cover a larger simulation area and to have much shorter integration step resulting in a more precise deformation simulation. Another research direction would be to improve the global deformation simulation approach and combine it with the local simulation approach. Further on, as mentioned in section 5.28.1 the approach to add meta–information to a virtual surface can be enriched by incorporating a deformable model within our deformation framework to provide the user with an advanced force feedback and to allow deformations of the marked areas.

Chapter 6

Summary and Outlook

In this work we presented theoretical background and novel methods for effective haptic rendering of volumetric data.

We started with basics and definitions of haptic rendering and visualization of volumetric data, followed by an extensive literature overview and classification of existing haptic rendering and visualization methods. The general challenges for haptic rendering are a huge amount of volumetric data per object, stability of haptic rendering and that haptic rendering requires an update rate of at least 1 kHz. As follows from the literature overview, there exist many different surface-based and voxel-based haptic rendering methods, and almost all of them have drawbacks that the manipulated object, when moved too quickly, can go through or inside an obstacle. Additionally, either a specific topological structure for the collision objects is needed, or extra speed-up data structures should be prepared. These issues could make it difficult to use a method in practice.

In this work we proposed a new haptic rendering approach, which is free of such drawbacks. This is especially important, because in practice the real medical data we work with can have any structure if segmentation has been done automatically. Our haptic rendering approach employs a novel collision detection technique based on ray casting concepts known from computer graphics. The approach was published in [227, 225]. The method gives collision detection guarantees that a manipulated object does not pass through "thin" obstacles and is never inside any of them while not requiring any special topological object structure. The collision detection was extensively tested with a new "slide along a surface" approach using an implicit surface representation "on the fly". The results confirm our approach to be a viable alternative to existing techniques avoiding most common drawbacks. The prototype was implemented as a plug-in of the

YaDiV VR system and supports different haptic devices and operation systems. Furthermore, we presented an improved version of our haptic rendering approach. The improved approach has all properties of the original method (including an implicit surface representation "on the fly") and does not have the drawbacks described in section 4.9. It was published in [226]. The method employs local path finding and ray casting concepts. Further on, we presented an improved force feedback generation scheme. The scheme of our prototype system was presented in section 4.12. The system is independent from a haptic display, so that a wide range of devices are supported. The results show that our haptic rendering approach is a good alternative to existing techniques, while avoiding most common drawbacks. Furthermore, it contrasts most triangle-based approaches, where millions of triangles would be generated and complex speeding-up traversing structures are required for the collision detection with the same guarantees.

Further on, in section 4.13 we described our experience of dealing with indeterministic delays from a few milliseconds to tens of milliseconds from time to time during the run of our prototype system on the Java VM. These delays were a serious drawback of the Java VM, since the haptic update rate should be constantly at least 1 kHz. In order to find the solution, we conducted experiments with two common real-time VMs: Sun JavaRTS and IBM Web Sphere Real Time. We followed all recommendations of the developers, but found out that there are still delays of 1-3 ms. The observed results differed from the information stated in [216] and [167], which was officially presented by IBM and Sun respectively. As a result, we used the standard VM and added virtual coupling into our C++ part having nearly constant update rate of at least 1 kHz. Using this approach, a sufficient and stable haptic update rate was always provided to the user.

For the advanced contact resolution, we introduced our flexible deformation framework which allows us to use our above mentioned improved approach of haptic rendering of volumetric data together with deformation models. The scheme of the framework was presented in section 5.1. Furthermore, we showed that our haptic rendering approach adds its properties including collision detection guarantee and non-penetration guarantee to the selected deformation model within the proposed deformation framework. Furthermore, we proposed an approach to interactively visualize the results of the deformation simulation for the chosen deformation model. We showed how to transfer changes in the object's structure to its volume representation, as well as how to effectively update the graphics surface representation from the changed volume representation.

Further on, in order to validate our framework, we proposed two local deformation simulation approaches based on the method of potential fields, where potential fields can be

considered as specific finite elements, i.e. discrete carriers of properties of the medium. The first approach uses "regular" potential fields. The second approach uses our novel cuboid potential fields. Further on, we demonstrated that cuboid potential fields are better suited to haptic rendering of volumetric data. Furthermore, we showed how to establish the correspondence of parameters of our proposed deformation simulation models to parameters of real materials, and proposed a way to take the heterogeneity of the simulated material into account. Additionally, we introduced the prototype of the global potential fields based deformation approach. The potential field based deformation simulation approaches were a good "illustration", because they initially did not have the "nice" properties of our haptic rendering approach. Furthermore, the resulting combined haptic rendering approach with our proposed deformation simulation approaches within our deformation framework does not require any pre-calculated structure and works "on the fly". The haptic update rate of our deformation framework remains stable when a deformation simulation was added. It does not decrease for both local and global simulation approaches. Furthermore, the results of tests with real volumetric data showed that haptic update rate is still orders of magnitude higher than the required 1 kHz.

Our deformation framework and all our haptic rendering and deformation simulation approaches were fully developed by us from scratch, without the use of any third party libraries.

Further on, we presented a number of possible practical use cases. Such, we presented an approach to add meta–information to virtual surfaces simply and naturally using a haptic device. This is needed e.g. for marking anatomic regions and landmarks. Further on, our prototype system was presented on the CeBIT international computer expo 2013 and 2015 within the scope of the Marie Curie ITN MultiScaleHuman project, which was funded by the European Union. Additionally, we showed that the developed deformation framework can be used for the simulation of drilling and for the simulation of needle insertion. Further on, our local potential fields model allows simulation and feeling of different tissues. Such, we presented an interaction scenario where the user can penetrate the liver but cannot penetrate the bone.

There are numerous possible future directions of research. As an ongoing research, object-object interactions could be introduced, where the controlled object is represented as a set of points, and the collision detection stage could be implemented on GPUs. As was shown e.g. in [117], [196], ray casting could be efficiently parallelized using GPUs and/or multi-processor systems. This will allow making computations faster and therefore representing the controlled object with more points and/or performing a more

sophisticated collision response. We plan to conduct the tests on the hardware which we already have at our Institute. It includes the high-end Tesla cluster granted by NVIDIA in the context of a Professor Partnership Program, modern graphics hardware, multi-core processor systems and an IBM Cell Cluster. Further on, for our deformation approaches we plan to make areas within the same object being deformable or non-deformable by setting which areas should and which areas should not be updated by the potential fields approach, or which areas should be considered as empty space. Moreover the potential fields simulation can be speeded-up by parallelization using GPUs and/or multi-processor systems in order to cover a larger simulation area and to have much shorter integration step resulting in a more precise deformation simulation. Another research direction would be to improve the global deformation simulation approach and combine it with the local simulation approach. Further on, the approach to add meta–information to a virtual surface presented in section 5.28.1 can be enriched by incorporating a deformable model within our deformation framework to provide the user with an advanced force feedback and to allow deformations of the marked areas. Further on, since the voxel data is discrete, a user can feel one-voxel "stairs", especially in the case of low resolution of the volumetric data. Therefore smoothing techniques can be introduced in order to provide a smoother force feedback. On another hand, in practice it can be important to feel the real segment, and not its smoothed version. The other practical use cases of our VR system with haptic interaction could be assembling a fractured bone being an important step for pre-operation planning in facial surgery, and correction of the results of automatic approaches. A broader outlook includes using our approaches with the future generation of VR haptic devices, using our haptic rendering system to feel nano-structures and to interactively model new nano-structures, and using our approach as a base for controlling of a bionic prosthesis of hand.

Bibliography

[1] Eric Acosta and Alan Liu. Real-time volumetric haptic and visual burrhole simulation. In *Virtual Reality Conference, 2007. VR'07. IEEE*, pages 247–250. IEEE, 2007.

[2] Eric Acosta and Alan Liu. Real-time volumetric haptic and visual burrhole simulation. In *Virtual Reality Conference, 2007. VR'07. IEEE*, pages 247–250. IEEE, 2007.

[3] Y. Adachi, T. Kumano, and K. Ogino. Intermediate representation for stiff virtual objects. *Virtual Reality Annual International Symposium*, pages 203–210, 1995.

[4] Asan Agibetov, Ricardo Manuel Millán Vaquero, Karl-Ingo Friese, Giuseppe Patanè, Michela Spagnuolo, and Franz-Erich Wolter. Integrated visualization and analysis of a multi-scale biomedical knowledge space. In *Proceedings of the EuroVis Workshop on Visual Analytics*, volume 36, pages 1–5. Springer, 2014.

[5] Marco Agus, Andrea Giachetti, Enrico Gobbetti, Gianluigi Zanetti, and Antonio Zorcolo. Real-time haptic and visual simulation of bone dissection. *Presence: Teleoperators and Virtual Environments*, 12(1):110–122, 2003.

[6] T. Aila and S. Laine. Understanding the efficiency of ray traversal on gpus. *In HPG 09: Proceedings of the Conference on High Performance Graphics 2009*, pages 145–149, 2009.

[7] T. Akenine-Moller, E. Haines, and N. Hoffman. *Real-time rendering.* A K Peters, Ltd., third edition, 2008.

[8] D. Allerkamp. *Tactile Perception of Textiles in a Virtual-Reality System.* Springer, 2011.

[9] Dennis Allerkamp. *Generation of Stimuli Supporting Tactile Perception of Textiles in a VR System*. PhD thesis, Leibniz Universitat Hannover, Faculty of Electrical Engineering and Computer Science, Welfenlab, Germany, 2009.

[10] Dennis Allerkamp, Guido Boettcher, Franz-Erich Wolter, Alan C. Brady, Jianguo Qu, and Ian R. Summers. A vibrotactile approach to tactile rendering. *Visual Computer*, 23(2):97–108, January 2007.

[11] J. Arvo. Backward ray tracing. *In ACM SIGGRAPH 86 Course Notes - Developments in Ray Tracing*, 12(3):259–263, 1986.

[12] R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. *Proceedings of the 7th conference on Visualization '96*, pages 197–204, October 2006.

[13] C. Bajaj, I. Ihm, S. Park, and D. Song. Compression-based ray casting of very large volume data in distributed environments. *Proceedings of the The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, 2:720–725, May 2000.

[14] Y. Bar-Cohen, C. Mavroidis, C. Pfeiffer, C. Culbert, and D. Magruder. Haptic interfaces. In Y. Bar-Cohen, editor, *Automation, Miniature Robotics and Sensors for Non-Destructive Testing and Evaluation*. Editor, April 1999.

[15] J. Barbic. *Real-time reduced large-deformation models and distributed contact for computer graphics and haptics*. PhD thesis, Carnegie Mellon University, Pittsburgh, 2007.

[16] J. Barbic and D. James. Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 171–180, August 2007.

[17] J. Barbic and D. L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. *ACM Transactions on Graphics*, 24(3):982–990, 2005.

[18] J. Barbic and D. L. James. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, 1(1):39–52, January 2008.

[19] C. Basdogan, S. De, J. Kim, M. Muniyandi, H. Kim, and M. A. Srinivasan. Haptics in minimally invasive surgical simulation and training. *IEEE Computer Graphics and Applications*, 24(2):56–64, March 2004.

[20] C. Basdogan, M. Sedef, M. Harders, and S. Wesarg. Vr-based simulators for training in minimally invasive surgery. *IEEE Computer Graphics and Applications*, 27(2):54–66, March 2007.

[21] Vincent Baudet, Fabrice Jaillet, and Behzad Shariat. Fitting a 3d particle system model to a non-dense data set in medical applications. *Journal for Geometry and Graphics*, 7(1):65–74, 2003.

[22] Matthias Becker. Modellbasierte orbita segmentierung und die automatisierte bestimmung anatomisch relevanter parameter. Master's thesis, Leibniz Universitat Hannover, Faculty of Electrical Engineering and Computer Science, Welfenlab AND Kiefer-, Mund und Geschitschirurgie der MHH (Hannover Medical School), Germany, April 2011.

[23] Matthias Becker, Karl-Ingo Friese, Franz-Erich Wolter, Nils-Claudius Gellrich, and Harald Essig. Development of a reliable method for orbit segmentation & measuring. *IEEE international symposium on medical measurements and applications (MeMeA 2015)*, pages 285–290, May 2015.

[24] C. Benthin, I.Wald, M. Scherbaum, and H. Friedrich. Ray tracing on the cell processor. *In Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 15–23, 2006.

[25] H. Bentoumi, P. Gautron, and K. Bouatouch. Gpu-based volume rendering for medical imagery. *International Journal of Computer Systems Science and Engineering 2007*, 1(1):36–42, 2007.

[26] Michaël Beuve, Mourad Amrani, Fabrice Jaillet, and Behzad Shariat. Physically based modelling with particle systems. In *International Conference on Concurrent Engineering: Research and Applications*, 2003.

[27] B. Bickel, M. Baecher, M. Otaduy, W. Matusik, H. Pfister, and M. Gross. Capture and modeling of non-linear heterogeneous soft tissue. *Proceedings of ACM SIGGRAPH 2009 papers, article 89*, 2009.

[28] J. Bigler, A. Stephens, and S. G. Parker. Design for parallel interactive ray tracing systems. *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pages 187–196, 2006.

[29] Nikolas H Blevins and Sabine Girod. Visuohaptic simulation of bone surgery for training and evaluation. *IEEE Comput Graph Appl*, 26:48–57, 2006.

[30] G. Boettcher. *Haptic Interaction with Deformable Objects*. Springer, 2011.

[31] Guido Boettcher. *Modelling VR Systems for Haptic Interaction with Deformable Objects, especially Textiles*. PhD thesis, Leibniz Universitat Hannover, Faculty of Electrical Engineering and Computer Science, Welfenlab, Germany, 2010.

[32] Guido Boettcher, Dennis Allerkamp, Daniel Gloeckner, and Franz-Erich Wolter. Haptic two-finger contact with textiles. *Visual Computer*, 24(10):911–922, September 2008.

[33] Guido Boettcher, Dennis Allerkamp, and Franz-Erich Wolter. Multi-rate coupling of physical simulations for haptic interaction with deformable objects. *Visual Computer*, 26(6-8):903–914, January 2010.

[34] Tales Nereu Bogoni and Márcio Sarroglia Pinho. Haptic technique for simulating multiple density materials and material removal. *21st International Conference on Computer Graphics, Visualization and Computer Vision*, pages 151–160, 2013.

[35] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald. Interactive distribution ray tracing. *Technical report UUSCI-2006-022, SCI Institute, University of Utah*, June 2006.

[36] Robert Bridson and Matthias Müller-Fischer. *Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page*. ACM, 2007.

[37] F. P. Brooks Jr., M. Ouh-Young, J. J. Batter, and P. J. Kilpatrick. Project grope - haptic displays for scientific visualization. *ACM SIGGRAPH Computer Graphics*, 24(4):177–185, August 1990.

[38] S. Bruckner. Efficient volume visualization of large medical datasets. Master's thesis, Vienna University of Technology, Austria, May 2004.

[39] S. Bruckner, S. Grimm, A. Kanitsar, and M.E. Groeller. Illustrative context-preserving volume rendering. *Proc. EuroVis '05*, pages 69–76, 2005.

[40] M.-P. Cani and A. Angelidis. Towards virtual clay. *ACM SIGGRAPH 2006 Courses*, pages 67–83, 2006.

[41] S. Chan, F. Conti, N.H. Blevins, and K. Salisbury. Constraint-based six degree-of-freedom haptic rendering of volume-embedded isosurfaces. *W. Hapt. Conf.'11*, pages 89–94, 2011.

[42] Y.-H. Chang, Y.-T. Chen, C.-W. Chang, and C.-L. Lin. Development scheme of haptic-based system for interactive deformable simulation. *Computer-Aided Design*, 42(5):414–424, May 2010.

[43] M. Chen, C. Correa, S. Islam, M. W. Jones, P.-Y. Shen, D. Silver, S. J. Walton, and P. J. Willis. Manipulating, deforming and animating sampled object representations. *Computer Graphics Forum*, 26(4):824–852, 2007.

[44] W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive ewa volume splatting. *Proceedings of the conference on Visualization '04*, pages 67–74, October 2004.

[45] A. Chihoub, Y. Chen, and M. Nadar. System and method for fast 3-dimensional data fusion. *US patent application 7439974 B2*, 2008.

[46] Gabriel Cirio, Maud Marchal, Aurélien Le Gentil, and Anatole Lécuyer. tap, squeeze and stir the virtual world: Touching the different states of matter through 6dof haptic interaction. In *Virtual Reality Conference (VR), 2011 IEEE*, pages 123–126. IEEE, 2011.

[47] Gabriel Cirio, Maud Marchal, Miguel A Otaduy, and Anatole Lécuyer. Six-oof haptic interaction with fluids, solids, and their transitions. In *World Haptics Conference (WHC), 2013*, pages 157–162. IEEE, 2013.

[48] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228. ACM, 2005.

[49] Timothy R Coles, Nigel W John, Derek Gould, Darwin G Caldwell, et al. Integrating haptics with augmented reality in a femoral palpation and needle insertion training simulation. *Haptics, IEEE Transactions on*, 4(3):199–209, 2011.

[50] J. E. Colgate and G. G. Schenkel. Passivity of a class of sampled-data systems: Application to haptic interfaces. *Journal of Robotic Systems*, 14(1):37–47, 1997.

[51] J. E. Colgate, M. C. Stanley, and J. M. Brown. Issues in the haptic display of tool use. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 140–145, 1995.

[52] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, January 1984.

[53] L. Corenthy, J. S. Martin, M.A. Otaduy, and M. Garcia. Volume haptic rendering with dynamically extracted isosurface. *Proceedings of Haptics Symposium 2012*, pages 133–139, 2012.

[54] F. Cosco, C. Garre, F. Bruno, M. Muzzupappa, and M. A. Otaduy. Augmented touch without visual obtrusion. *In the Proc. of the International Symposium on Mixed and Augmented Reality*, pages 99–102, October 2009.

[55] C. Crassin, F. Neyret, S. Lefebvre, and E. Eisemann. Gigavoxels: ray-guided streaming for efficient and detailed voxel rendering. *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 15–22, 2009.

[56] S. De, Y.-J. Lim, M. Manivannan, and M. A. Srinivasan. Physically realistic virtual surgery using the point-associated finite field (paff) approach. *Presence: Teleoperators and Virtual Environments*, 15(3):294–308, June 2006.

[57] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr. Dynamic real-time deformations using space & time adaptive sampling. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, August 2001.

[58] Ph. Decaudin and F. Neyret. Volumetric billboards. *Computer Graphics Forum*, 28(8):2079–2089, 2009.

[59] D. E. DeMarle. Distributed interactive ray tracing for large volume visualization. Master's thesis, University of Utah, USA, 1999.

[60] D. E. DeMarle, S. Parker, M. Hartner, C. Gribble, and C. Hansen. Distributed interactive ray tracing for large volume visualization. *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94, October 2003.

[61] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. 1996.

[62] K Donina and A Yarusskaya. Density of bone. *The Physics. Factbook, Ed. Glenn Elert*, 2002.

[63] C. Duriez, C. Andriot, and A. Kheddar. Signorini's contact model for deformable objects in haptic simulations. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2004*, pages 32–37, 2004.

[64] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot. Realistic haptic rendering of interacting deformable objects in virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 12(1):36–47, January 2006.

[65] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. Rezk Salama, and D. Weiskopf. Real-time volume graphics. *ACM SIGGRAPH 2004 Course Notes*, 2004.

[66] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. *In Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2001*, pages 9–16, 2001.

[67] K. Engel and G. Paladini. Sliding texture volume rendering. *US patent application 7460117 B2*, 2008.

[68] Timothy G Feeman. *The mathematics of medical imaging: a beginner's guide*. Springer Science & Business Media, 2010.

[69] M. Foskey, M. A. Otaduy, and M. C. Lin. Artnova: Touch-enabled 3d model design. *In Proceedings of IEEE Virtual Reality*, pages 119–126, 2002.

[70] S. Frank and A. Kaufman. Dependency graph scheduling in a volumetric ray tracing architecture. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 127–135, September 2002.

[71] H. Friedrich, J. Guenther, A. Dietrich, M. Scherbaum, H.-P. Seidel, and P. Slusallek. Exploring the use of ray tracing for future games. *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 41–50, 2006.

[72] Karl-Ingo Friese. *Entwicklung einer Plattform zur 3D-Visualisierung und -Segmentierung medizinischer Daten*. PhD thesis, Leibniz Universitat Hannover, Germany, 2010.

[73] Karl-Ingo Friese, Philipp Blanke, and Franz-Erich Wolter. Yadiv – an open platform for 3d visualization and 3d segmentation of medical data. *The Visual Computer*, 27:129–139, 2011.

[74] EM Galimov and AM Krivtsov. Origin of the earth–moon system. *Journal of earth system science*, 114(6):593–600, 2005.

[75] N. Galoppo, M. A. Otaduy, S. Tekin, M. Gross, and M. C. Lin. Interactive haptic rendering of high-resolution deformable objects. *Proceedings of the 2nd international conference on Virtual reality*, pages 215–223, 2007.

[76] E. Gamma, R. Helm, R. Johnson, and J. Vlissideset. Design patterns: Elements of reusable object-oriented software. *Addison-Wesley*, 1995.

[77] C. Garre and M. A. Otaduy. Haptic rendering of complex deformations through handle-space force linearization. *In the Proceedings of the World Haptics Conference*, pages 422–427, 2009.

[78] C. Garre and M. A. Otaduy. Toward haptic rendering of full-hand touch. *In the Proc. of CEIG (Spanish Computer Graphics Conference)*, 2009.

[79] A. Van Gelder and K. Kim. Direct volume rendering with shading via three-dimensional textures. *In 1996 Symposium on Volume Visualization*, pages 23–30, 1996.

[80] FA Gilabert, AM Krivtsov, and A Castellanos. Computer simulation of mechanical properties for powder particles using molecular dynamics. *Proc. of XXX Summer School" Advanced Problems in Mechanics", St. Petersburg, Russia*, pages 230–239, 2002.

[81] FA Gilabert, AM Krivtsov, and A Castellanos. Molecular dynamics modelling of the adhesive interaction between fine particles. *Powders and Grains 2005. Proceedings of the 5th International 5th International Conference on Micromechanics of Granular Media. Stuttgart, Germany*, 1:513–516, 2005.

[82] M. Glencross, A. G. Chalmers, M. C. Lin, M. A. Otaduy, and D. Gutierrez. Exploiting perception in high-fidelity virtual environments. *ACM SIGGRAPH 2006 Courses*, July 2006.

[83] Leslie Greengard. *The rapid evaluation of potential fields in particle systems*. MIT press, 1988.

[84] A. Gregory, M. C. Lin, S. Gottschalk, and R. Taylor. A framework for fast and accurate collision detection for haptic interaction. *ACM SIGGRAPH 2005 Courses*, pages 34–41, 2005.

[85] A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin, and D. Manocha. Six degree-of-freedom haptic display of polygonal models. *Proceedings of the conference on Visualization '00*, pages 139–146, October 2000.

[86] S. Grimm, S. Bruckner, A. Kanitsar, and E. Groeller. Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. *Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*, pages 1–8, October 2004.

[87] M. Hadwiger, A. Kratz, C. Sigg, and K. Buehler. Gpu-accelerated deep shadow maps for direct volume rendering. *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, September 2006.

[88] M. Hadwiger, P. Ljung, C. Rezk Salama, and T. Ropinski. Advanced illumination techniques for gpu-based volume raycasting. *ACM SIGGRAPH ASIA 2008 Courses*, 2008.

[89] M. Hadwiger, P. Ljung, C. Rezk Salama, and T. Ropinski. Advanced illumination techniques for gpu-based volume raycasting. *ACM SIGGRAPH 2009 Courses*, 2009.

[90] Takahiro Harada, Masayuki Tanaka, Seiichi Koshizuka, and Yoichiro Kawaguchi. Real-time coupling of fluids and rigid bodies. *Proc. of the APCOM*, pages 1–13, 2007.

[91] Pheng-Ann Heng, Tien-Tsin Wong, Rong Yang, Yim-Pan Chui, Yong Ming Xie, Kwong-Sak Leung, and Ping-Chung Leung. Intelligent inferencing and haptic simulation for chinese acupuncture learning and training. *Information Technology in Biomedicine, IEEE Transactions on*, 10(1):28–41, 2006.

[92] B. Hibbard. Vis files: computational field visualization. *ACM SIGGRAPH Computer Graphics*, 35(4):5–9, November 2001.

[93] Roger W Hockney and James W Eastwood. *Computer simulation using particles*. CRC Press, 1988.

[94] N. Hogan. Impedance control – an approach to manipulation. i – theory. ii – implementation. iii – applications. *ASME Transactions, Journal of Dynamic Systems, Measurement and Control*, 107:1–24, March 1985.

[95] Xiyuan Hou and Olga Sourina. Haptic rendering algorithm for biomolecular docking with torque force. In *Cyberworlds (CW), 2010 International Conference on*, pages 25–31. IEEE, 2010.

[96] S.-U. Hwang, B.-C. Lee, J. Ryu, K. H. Lee, and Y.-G. Lee. Adaptive haptic rendering for time-varying haptic and video frame rates in multi-modal interactions. *Computer Animation and Virtual Worlds*, 21(1):25–38, January 2010.

[97] R. M. Taylor II. Haptics for scientific visualization. *ACM SIGGRAPH 2005 Courses*, pages 174–179, 2005.

[98] M. Ikits, J. D. Brederson, C. D. Hansen, and C. R. Johnson. A constraint-based technique for haptic volume exploration. *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 263–269, October 2003.

[99] DA Indeitsev, AM Krivtsov, and PV Tkachev. Molecular dynamics analysis of the relation between the spall strength and strain rate for solids. In *Doklady Physics*, volume 51, pages 154–156. Springer, 2006.

[100] B. E. Insko. *Passive haptics significantly enhances virtual environments.* PhD thesis, The University of North Carolina at Chapel Hill, USA, 2001.

[101] Simon Jackson and Richard Thomas. Introduction to ct physics. *Cross-Sectional Imaging Made Easy. Churchill Livingston*, page 7, 2004.

[102] D. L. James and D. K. Pai. Bd-tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. on Graphics*, 23(3):393–398, August 2004.

[103] D. Johnson and P. Willemsen. Accelerated haptic rendering of polygonal models through local descent. *12th International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)*, pages 18–23, 2004.

[104] D. E. Johnson and P. Willemsen. Six degree-of-freedom haptic rendering of complex polygonal models. *Proceedings of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'03)*, pages 229–235, March 2003.

[105] D. E. Johnson, P. Willemsen, and E. Cohen. A haptic system for virtual prototyping of polygonal models. *Proceedings of DETC2004: 2004 ASME Design Engineering Technical Conferences*, pages 84–88, 2004.

[106] D. E. Johnson, P. Willemsen, and E. Cohen. Six degree-of-freedom haptic rendering using spatialized normal cone search. *IEEE Transactions on Visualization and Computer Graphics*, 11(6):661–670, November 2005.

[107] David Kahaner, Cleve Moler, and Stephen Nash. Numerical methods and software. *Englewood Cliffs: Prentice Hall*, 1, 1989.

[108] B. Kainz, M. Grabner, A. Bornik, S. Hauswiesner, J. Muehl, and D. Schmalstieg. Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore gpus. *ACM SIGGRAPH Asia 2009*, 2009. article 152.

[109] S. Kashyap, R. Goradia, P. Chaudhuri, and S. Chandran. Real time ray tracing of point-based models. *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010. article 4.

[110] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *IEEE Computer*, 26(7):51–64, July 2007.

[111] L. Kim, A. Kyrikou, M. Desbrun, and G. Sukhatme. An implicit-based haptic rendering technique. *In Proc. of the IEEE/RSJ International Conf. on Intelligent Robots*, 2002.

[112] R. L. Klatzky, S. J. Lederman, C. Hamilton, M. Grindley, and R. H. Swendsen. Feeling textures through a probe: Effects of probe and surface geometry and exploratory factors. *Perception and Psychophysics*, 65(4):613–631, 2003.

[113] J. Kniss, S. Premoze, C. Hansen, and D. Ebert. Interactive translucent volume rendering and procedural modeling. *In Proc. of IEEE Visualization 2002*, pages 168–176, 2002.

[114] A. Knoll, I. Wald, and C. Hansen. Coherent multiresolution isosurface ray tracing. *The Visual Computer*, 25(3):209–225, 2009.

[115] AM Krivtsov. *Deformation and fracture of solids with microstructure*. Moscow, Fizmatlit. (in Russian), 2007.

[116] AM Krivtsov and NV Krivtsova. Method of particles and its application to mechanics of solids. *Far Eastern Mathematical Journal (in Russian)*, 3(2):254–276, 2002.

[117] J. Kruger and R. Westermann. Acceleration techniques for gpu-based volume rendering. *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 287–292, October 2003.

[118] U. Kuehnapfel, H. K. Cakmak, and H. Maab. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers and Graphics*, 24:671–682, October 2000.

[119] Y. Kuroda, M. Nakao, S. Hacker, T. Kuroda, H. Oyama, M. Komori, T. Matsuda, and T. Takahashi. Haptic force feedback with an interaction model between multiple deformable objects for surgical simulations. *Proceedings of Eurohaptics2002*, pages 116–121, July 2002.

[120] Vitaly A Kuzkin, Anton M Krivtsov, and Aleksandr M Linkov. Computer simulation of effective viscosity of fluid-proppant mixture used in hydraulic fracturing. *The Journal of Mining Science*, 50(1):1–9, 2014.

[121] Vitaly A Kuzkin, Anton M Krivtsov, and Aleksandr M Linkov. Computer simulation of effective viscosity of fluid-proppant mixture used in hydraulic fracturing. *Fiziko-Tekhnicheskie Problemy Razrabotki Poleznykh Iskopaemykh (in Russian)*, (1):3–12, 2014.

[122] P. Lacroute. *6-dof haptic rendering using contact levels of detail and haptic textures*. PhD thesis, Stanford University, USA, 1995. Technical report CSL-TR-95-678.

[123] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Proc. SIGGRAPH '94*, pages 451–458, July 1994.

[124] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[125] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics (TOG)*, 9(3):245–261, July 1990.

[126] T. Lewiner, H. Lopes, A. Wilson, and G. Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *J. Graphics Tools*, 8:1–15, 2003.

[127] W. Li. Invisible space skipping with adaptive granularity for texture-based volume rendering. *US patent application 7460119 B2*, 2008.

[128] M. C. Lin, D. Manocha, Y. Kim, and M. A. Otaduy. Six degree-of-freedom haptic rendering. *Department of Computer Science, University of North Carolina at Chapel Hill*, February 2004.

[129] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, July 1987.

[130] C. J. Luciano, P. Banerjee, and S. H. R. Rizzi. Gpu-based elastic-object deformation for enhancement of existing haptic applications. *Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering*, pages 146–151, September 2007.

[131] H. Ludvigsen and A. C. Elster. Real-time ray tracing using nvidia optix. *EUROGRAPHICS 2010*, 2010.

[132] E.B. Lum, B. Wilson, and K.L. Ma. High-quality lighting and efficient preintegration for volume rendering. *In Proc. of Eurographics/IEEE Symposium on Visualization 2004*, pages 25–34, 2004.

[133] C. Lundstroem. *Efficient Medical Volume Visualization - an Approach Based on Domain Knowledge*. PhD thesis, Linkoeping University, Sweden, 2007.

[134] A. Maciel, R. Boulic, and D. Thalmann. Efficient collision detection within deforming spherical sliding contact. *IEEE Transactions on Visualization and Computer Graphics*, 13 (3):518–529, May 2007.

[135] A. Maciel, T. Halic, Z. Lu, L. P. Nedel, and S. De. Using the physx engine for physics-based virtual surgery with force feedback. *In International Journal of Medical Robotics and Computer Assisted Surgery*, 5(3):341–353, September 2009.

[136] EE Marco Agus, Andrea Giachetti, Enrico Gobbetti, Gianluigi Zanetti, and Antonio Zorcolo. Mastoidectomy simulation with combined visual and haptic feedback. *Medicine Meets Virtual Reality 02/10: Digital Upgrades, Applying Moore's Law to Health*, 85:17, 2002.

[137] William R. Mark, Scott C. Randolph, Mark Finch, James M. Van, Verth Russell, and M. Taylor II. Adding force feedback to graphics systems: issues and solutions. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 447–452, August 1996.

[138] S. R. Marschner and R. J. Lobb. An evaluation of reconstruction filters for volume rendering. *In Proceedings of Visualization '94*, pages 100–107, October 1994.

[139] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[140] M. L. McLaughlin, J. Hespanha, and G. Sukhatme. Introduction to haptics. In M. L. McLaughlin, J. Hespanha, and G. Sukhatme, editors, *Touch in virtual environments: Haptics and the design of interactive systems*. 2002.

[141] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 401–408, July 1999.

[142] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Voxel-based 6-dof haptic rendering improvements. *Journal of Haptics-e*, 3(7), 2006.

[143] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90, October 2000.

[144] J. Mensmann, T. Ropinski, and K. Hinrichs. Accelerating volume raycasting using occlusion frustums. *In IEEE/EG International Symposium on Volume and Point-Based Graphics*, pages 147–154, 2008.

[145] J. Meyer-Spradow, T. Ropinski, and K. Hinrichs. Supporting depth and motion perception in medical volume data. *Visualization in Medicine and Life Sciences, Springer*, pages 121–133, 2007.

[146] Ricardo Manuel Millán-Vaquero, Sean Dean Lynch, Benjamin Fleischer, Jan Rzepecki, Karl-Ingo Friese, Christof Hurschler, and Franz-Erich Wolter. Enhanced visualization of the knee joint functional articulation based on helical axis method. In *Bildverarbeitung für die Medizin 2015*, pages 449–454. Springer, 2015.

[147] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. *Proceedings of the conference on Visualization '98*, pages 239–245, October 1998.

[148] K. Mueller, T. Moeller, and Roger Crawfis. Splatting without the blur. *Proceedings of the conference on Visualization '99: celebrating ten years*, pages 363–370, October 1999.

[149] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. High-quality splatting on rectilinear grids with efficient culling of occluded voxels. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):116–134, April 1999.

[150] Richard T Mull. Mass estimates by computed tomography: physical density from ct numbers. *American journal of roentgenology*, 143(5):1101–1104, 1984.

[151] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[152] Matthias Müller and Nuttapong Chentanez. Solid simulation with oriented particles. In *ACM Transactions on Graphics (TOG)*, volume 30, page 92. ACM, 2011.

[153] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54. ACM, 2002.

[154] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, July 2005.

[155] Matthias Müller, Barbara Solenthaler, Richard Keiser, and Markus Gross. Particle-based fluid-fluid interaction. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 237–244. ACM, 2005.

[156] M. Nakao, T. Kuroda, M. Komori, and H. Oyama. Evaluation and user study of haptic simulator for learning palpation in cardiovascular surgery. *In Proceedings of International Conference of Artificial Reality and Tele-Existence (ICAT) 2003*, pages 203–208, 2003.

[157] A. Nealen, M. Mueller, R. Keiser, E. Boxerman, and M. Carlson. Physically-based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2005.

[158] G. M. Nielson. Dual marching cubes. *Proc. of the conference on Visualization '04*, pages 489–496, October 2004.

[159] Daniel Nixon and Richard Lobb. A fluid-based soft-object model. *IEEE Computer Graphics and Applications*, 22(4):68–75, 2002.

[160] Shūichi Nosé. A molecular dynamics method for simulations in the canonical ensemble. *Molecular physics*, 52(2):255–268, 1984.

[161] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of chemical physics*, 81(1):511–519, 1984.

[162] Shūichi Nosé. An extension of the canonical ensemble molecular dynamics method. *Molecular Physics*, 57(1):187–191, 1986.

[163] NVIDIA-Corporation. Cuda programming guide. www.nvidia.com/object/cuda_home_new.html.

[164] NVIDIA-Corporation. Nvidia optix ray tracing engine. https://developer.nvidia.com/optix.

[165] NVIDIA-Corporation. Physx library. developer.nvidia.com/physx.

[166] University of Cambridge. Mechanical properties of bone. http://www.doitpoms.ac.uk/tlplib/bones/bone_mechanical.php.

[167] Oracle. Sun java real-time system 2.2 update 1 technical documentation. download.oracle.com/javase/realtime/rts_productdoc_2.2u1.html, April 2010.

[168] M. Ortega, S. Redon, and S. Coquillart. A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):458–469, May 2007.

[169] M. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross. Implicit contact handling for deformable objects. *In Eurographics '09*, 28(2):559–568, 2009.

[170] M. A. Otaduy. Haptic rendering pipeline. *IEEE International Conference on Robotics and Automation 2007, Workshop on Haptic Perception and Rendering*, 2007.

[171] M. A. Otaduy and M. Gross. Transparent rendering of tool contact with compliant environments. *Proceedings of the Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 225–230, March 2007.

[172] M. A. Otaduy, N. Jain, A. Sud, and M. C. Lin. Haptic display of interaction between textured models. *Proceedings of the conference on Visualization '04*, pages 297–304, October 2004.

[173] M. A. Otaduy and M. C. Lin. A perceptually-inspired force model for haptic texture rendering. *Proceedings of the 1st Symposium on Applied perception in graphics and visualization*, pages 123–126, August 2004.

[174] M. A. Otaduy and M. C. Lin. Introduction to haptic rendering. *ACM SIGGRAPH 2005 Courses*, 2005.

[175] M. A. Otaduy and M. C. Lin. Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 247–256, March 2005.

[176] M. A. Otaduy and M. C. Lin. A modular haptic rendering algorithm for stable and transparent 6-dof manipulation. *IEEE Transactions on Robotics*, 22(4):751–762, 2006.

[177] Miguel Angel Otaduy Tristan. *6-dof haptic rendering using contact levels of detail and haptic textures*. PhD thesis, University of North Carolina at Chapel Hill, 2004.

[178] G. Paladini. A memory efficient shear-warp voxel projection algorithm. *US patent application 6570952 B2*, 2003.

[179] K.L. Palmerius and G. Baravdish. Higher precision in volume haptics through subdivision of proxy movements. *Proc. of EuroHaptics '08*, pages 694–699, 2008.

[180] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):238–250, July 1999.

[181] S. G. Parker, S. Boulos, J. Bigler, and A. Robison. Rtsl: a ray tracing shading language. *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 149–160, September 2007.

[182] Jie Peng, Ling Li, and Andrew Squelch. Hybrid surgery cutting using snapping algorithm, volume deformation and haptic interaction. *Journal of Man, Machine and Technology*, 2(1):35–46, 2013.

[183] Andreas Petersik, Bernhard Pflesser, Ulf Tiede, Karl-Heinz Höhne, and Rudolf Leuwer. Realistic haptic interaction in volume sculpting for surgery simulation. In *Surgery Simulation and Soft Tissue Modeling*, pages 194–202. Springer, 2003.

[184] Bernhard Pflesser, Andreas Petersik, Ulf Tiede, Karl Heinz Höhne, and Rudolf Leuwer. Volume cutting for virtual petrous bone surgery. *Computer Aided Surgery*, 7(2):74–83, 2002.

[185] Ekaterina Podolskaya, Artem Panchenko, and Anton Krivtsov. Stability and structural transitions in crystal lattices. In *Surface Effects in Solid Mechanics*, pages 123–133. Springer, 2013.

[186] "MultiScaleHuman Project". http://www.welfenlab.de/multiscalehuman.html.

[187] W. Qi. Geometry based haptic interaction with scientific data. *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 401–404, 2006.

[188] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. *Proceedings of Eurographics (Computer Graphics Forum)*, 21(3):279–288, 2002.

[189] L. Ren, H. Pfister, and M. Zwicker. Object space ewa surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum*, 21(3):461–470, 2002.

[190] C. Rezk-Salama. *Volume Rendering Techniques for General Purpose Graphics Hardware*. PhD thesis, University of Siegen, Germany, 2001.

[191] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. *In Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 109–118, 2000.

[192] C. Rezk-Salama and A. Kolb. Opacity peeling for direct volume rendering. *Computer Graphics Forum*, 25(3):596–606, 2006.

[193] Phattanapon Rhienmora, Kugamoorthy Gajananan, Peter Haddawy, Matthew N Dailey, and Siriwan Suebnukarn. Augmented reality haptics system for dental surgical skills training. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, pages 97–98. ACM, 2010.

[194] Jakob Riga. Entwurf und entwicklung einer haptischen eingabemethode zur erfassung von metainformationen auf 3d-oberflachen. Bachelor's thesis, Leibniz Universitat Hannover, Faculty of Electrical Engineering and Computer Science, Welfenlab, Germany, September 2015.

[195] F. Roessler, R. P. Botchen, and T. Ertl. Dynamic shader generation for flexible multi-volume visualization. *In Proc. of IEEE Pacific Visualization Symposium 2008 (PacificVis '08)*, pages 17–24, 2008.

[196] T. Ropinski, J. Kasten, and K. H. Hinrichs. Efficient shadows for gpu-based volume raycasting. *In Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization (WSCG08)*, pages 17–24, 2008.

[197] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, and K. H. Hinrichs. Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Computer Graphics Forum (Eurographics 2008)*, 27(2):567–576, 2008.

[198] D. C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 345–352, August 1997.

[199] Jan Rzepecki, Ricardo Manuel Millán Vaquero, Alexander Vais, Karl-Ingo Friese, and Franz-Erich Wolter. Multimodal approach for natural biomedical multi-scale exploration. In *Advances in Visual Computing*, pages 620–631. Springer, 2014.

[200] Mikel Sagardia, Katharina Hertkorn, Thomas Hulin, Simon Schätzle, Robin Wolff, Johannes Hummel, Janki Dodiya, and Andreas Gerndt. Vr-oos: The dlr's virtual reality simulator for telerobotic on-orbit servicing with haptic feedback. In *2015 IEEE Aerospace Conference*, pages 1–17. IEEE, 2015.

[201] Mikel Sagardia, Thomas Hulin, Carsten Preusche, and Gerd Hirzinger. Improvements of the voxmap-pointshell algorithm-fast generation of haptic data-structures. In *53rd IWK-Internationales Wissenschaftliches Kolloquium, Ilmenau, Germany*, 2008.

[202] Mikel Sagardia, Bernhard Weber, Thomas Hulin, Gerd Hirzinger, and Carsten Preusche. Evaluation of visual and force feedback in virtual assembly verifications. In *2012 IEEE Virtual Reality Workshops (VRW)*, pages 23–26. IEEE, 2012.

[203] K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles. Haptic rendering: programming touch interaction with virtual objects. *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 123–130, April 1995.

[204] M. Sedef, E. Samur, and C. Basdogan. Visual and haptic simulation of linear viscoelastic tissue behavior based on experimental data. *2006 International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'06)*, pages 201–208, 2006.

[205] C. Sewell, N. H. Blevins, S. Peddamatham, H. Z. Tan, D. Morris, and K. Salisbury. The effect of virtual haptic training on real surgical drilling proficiency. *Second Joint EuroHaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC'07)*, pages 601–603, 2007.

[206] S.Gottschalk. *Collision Queries Using Oriented Bounding Boxes*. PhD thesis, University of North Carolina at Chapel Hill, USA, 2000.

[207] M. Shih, Y.-F. Chiu, Y.-C. Chen, and C.-F. Chang. Real-time ray tracing with cuda. *In ICA3PP 09: Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, pages 327–337, 2009.

[208] Nosé Shuichi. Constant temperature molecular dynamics methods. *Progress of Theoretical Physics Supplement*, 103:1–46, 1991.

[209] S. Signorini. Sopra akune questioni di elastostatica. *Atti della Societa ltaliana per il Progresso delle Scienze*, 1933.

[210] J. Singh and P. Narayanan. Real-time ray tracing of implicit surfaces on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):261–272, 2010.

[211] L. M. Sobierajski and R. S. Avila. A hardware acceleration method for volumetric ray tracing. *Proceedings of the 6th conference on Visualization '95*, pages 27–34, 1995.

[212] L. M. Sobierajski and A. E. Kaufman. Volumetric ray tracing. *Proceedings of the 1994 symposium on Volume visualization*, pages 11–18, October 1994.

[213] Barbara Solenthaler, Jürg Schläfli, and Renato Pajarola. A unified particle model for fluid–solid interactions. *Computer Animation and Virtual Worlds*, 18(1):69–82, 2007.

[214] L. R. Speer. An updated cross-indexed guide to the ray-tracing literature. *ACM SIGGRAPH Computer Graphics*, 26(1):41–72, Januar 1992.

[215] Anthony James Merrill Spencer. *Continuum mechanics*. Courier Corporation, 2004.

[216] M. Stoodley, M. Fulton, M. Dawson, R. Sciampacone, and J. Kacur. Real-time Java, Part 1: Using Java code to program real-time systems, April 2007.

[217] Don Stredney, Ali R Rezai, Daniel M Prevedello, J Bradley Elder, Thomas Kerwin, Bradley Hittle, and Gregory J Wiet. Translating the simulation of procedural drilling techniques for interactive neurosurgical training. *Neurosurgery*, 73(4):74–80, 2013.

[218] David Love Tonnesen. *Dynamically coupled particle systems for geometric modeling, reconstruction, and animation*. PhD thesis, University of Toronto, 1998.

[219] A. van der Ploeg. Interactive ray tracing, the replacement of rasterization? *B.Sc. Thesis, Vrije University*, December 2006.

[220] Ricardo Manuel Millán Vaquero, Asan Agibetov, Jan Rzepecki, Marta Ondrésik, Alexander Vais, Joaquim Miguel Oliveira, Giuseppe Patane, Karl-Ingo Friese, Rui Luis Reis, Michela Spagnuolo, et al. A semantically adaptable integrated visualization and natural exploration of multi-scale biomedical data. In *Information Visualisation (iV), 2015 19th International Conference on*, pages 543–552. IEEE, 2015.

[221] Ricardo Manuel Millán Vaquero, Jan Rzepecki, Karl-Ingo Friese, and Franz-Erich Wolter. Visualization and user interaction methods for multiscale biomedical data. In *3D Multiscale Physiological Human*, pages 107–133. Springer, 2014.

[222] A. Vashisth and S. Mudur. Deforming point-based models using an electronic glove. *Proceedings of the 2008 C3S2E conference*, pages 193–197, 2008.

[223] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.

[224] F.P. Vidal, N.W. John, A.E. Healey, and D.A. Gould. Simulation of ultrasound guided needle puncture using patient specific data with 3d textures and volume haptics. *Journal of Visualization and Computer Animation*, 19:111–127, 2008.

[225] Roman Vlasov, Karl-Ingo Friese, and Franz-Erich Wolter. Ray casting for collision detection in haptic rendering of volume data. *I3D '12 Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 215, March 2012.

[226] Roman Vlasov, Karl-Ingo Friese, and Franz-Erich Wolter. Haptic rendering of volume data with collision detection guarantee using path finding. In *Transactions on Computational Science XVIII*, pages 212–231. Springer, 2013.

[227] Roman Vlasov, K.-I Friese, and F.-E Wolter. Haptic rendering of volume data with collision determination guarantee using ray casting and implicit surface representation. *In Proc. of Cyberworlds 2012 Int. Conference*, pages 91–99, September 2012.

[228] I. Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, Germany, 2004.

[229] I. Wald. The rtrt core. *ACM SIGGRAPH 2005 Courses*, July 2005.

[230] I. Wald, C. Benthin, and P. Slusallek. Openrt - a flexible and scalable rendering engine for interactive 3d graphics. *Technical Report TR-2002-01, Saarland University*, 2002.

[231] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics (TOG)*, 26(1), January 2007. article 6.

[232] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics (TOG)*, 25(3), July 2006.

[233] M. Wan and W. A. McNeely. Quasi-static approximation for 6 degrees-of-freedom haptic rendering. *Proceedings of the 14th IEEE Visualization Conference (VIS03)*, pages 257–262, 2003.

[234] Roger W Webster, Dean I Zimmerman, Betty J Mohler, Michael G Melkonian, and Randy S Haluck. A prototype haptic suturing simulator. *Medicine Meets Virtual Reality*, 81:567–569, 2001.

[235] R. Weller and G. Zachmann. A unified approach for physically-based simulations and haptic rendering. *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, pages 151–160, August 2009.

[236] R. Westermann and B. Sevenich. Accelerated volume ray-casting using texture mapping. *Proceedings of the conference on Visualization '01*, pages 271–278, October 2001.

[237] L. Westover. Interactive volume rendering. *Proceedings of the Chapel Hill Workshop on volume visualization*, pages 9–16, May 1989.

[238] L. Westover. Footprint evaluation for volume rendering. *ACM SIGGRAPH Computer Graphics*, 24(4):367–376, August 1990.

[239] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[240] Martin Wicke, Philipp Hatt, Mark Pauly, Matthias Müller, and Markus Gross. Versatile virtual materials using implicit connectivity. In *Proceedings of the 3rd Eurographics/IEEE VGTC conference on Point-Based Graphics*, pages 137–144. Eurographics Association, 2006.

[241] D WILLIAMS. Polyvox technology. open source software available at. http://www.thermite3d.org.

[242] J. Williams. A method for accelerating the generation and display of volume-rendered cut-away-view of three dimensional images. *US patent application 6573891 B1*, 2003.

[243] O. Wilsona, A. VanGelde, and J.Wilhelms. Direct volume rendering via 3d textures. *Technical report UCSC-CRL-94-19, University of Santa Cruz*, 1994.

[244] Jun Wu, Dangxiao Wang, Charlie CL Wang, and Yuru Zhang. Toward stable and realistic haptic interaction for tooth preparation simulation. *Journal of Computing and Information Science in Engineering*, 10(2):021007–1–021007–9, 2010.

[245] Jun Wu, Ge Yu, Dangxiao Wang, Yuru Zhang, and Charlie CL Wang. Voxel-based interactive haptic simulation of dental drilling. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 39–48. American Society of Mechanical Engineers, 2009.

[246] JZ Wu, RG Dong, and DE Welcome. Analysis of the point mechanical impedance of fingerpad in vibration. *Medical engineering & physics*, 28(8):816–826, 2006.

[247] L. Yang and D. Xue. Systems and methods of gradient assisted volume rendering. *US patent application 7675517 B2*, 2010.

[248] S.-E. Yoon, C. Lauterbach, and D. Manocha. R-lods: Fast lod-based ray tracing of massive models. *The Visual Computer (Pacific Graphics) 2006, Technical report TR06-009, University of North Carolina at Chapel Hill, USA*, 2006.

[249] X. Yuan, M. Nguyen, B. Chen, and D. Porter. High dynamic range volume visualization. *IEEE Visualization '05 Proceedings*, pages 327–334, 2005.

[250] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. *Proceedings of the International Conference on Intelligent Robots and Systems*, 3:31–46, August 1995.

[251] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Ewa volume splatting. *Proceedings of the conference on Visualization '01*, pages 29–36, October 2001.

Curriculum Vitae

Name: Roman VLASOV
Born on: 14th January 1986 in Leningrad
E-mail: rovlasovfp@gmail.com
LinkedIn: https://www.linkedin.com/in/rovlasov

EXPERIENCE

Senior Software Engineer
Adpack TV (Germany) From 08 / 2016
Responsible for Production & Innovation

Software Engineer
Freelance 01 / 2016 – 06 / 2016

Software Engineer / PhD Researcher
Institute of Man-Machine-Communication, Leibniz University of Hanover (Germany) 04 / 2015 – Present
Finalizing the PhD

Software Engineer / Research Associate
Nanyang Technological University (Singapore) 06 / 2013 – 03 / 2015
Virtual Reality, Visualization Frameworks

Software Engineer / PhD Researcher
Institute of Man-Machine-Communication, Leibniz University of Hanover (Germany) 09 / 2009 – 05 / 2013
Virtual Reality, Visualization, Haptic Rendering
Sponsored by Siemens/DAAD, Exhibited on CeBIT '13 and '15

Part-Time Software Engineer
SoftDev SPb (Russia) 03 / 2008 – 08 / 2009
CAD Software

Part-Time Junior Software Engineer
Driver Inter Ltd. (Russia) 09 / 2006 – 09 / 2007
Golf Simulator, Samsung MenuTool

Researcher (Internship)
Department of Theoretical Mechanics, Saint-Petersburg State Polytechnical University (Russia) 05 / 2006 – 08 / 2006

Other Experience:

Teaching Assistant
Leibniz University of Hanover (Germany) 10 / 2010 – 02 / 2011

EDUCATION

Leibniz University of Hanover (Germany) 2009 – Present
Finalizing the Ph.D. in Computer Science

Saint-Petersburg State Polytechnical University (Russia) 2003 – 2009
M.Sc. in Applied Mathematics and Informatics

Physical and Mathematical Lyceum 239 (Russia) 2001 – 2003
High School