Haptic Rendering of Volume Data with Collision Detection Guarantee Using Path Finding

Roman Vlasov, Karl-Ingo Friese, and Franz-Erich Wolter

Institute of Man-Machine-Communication, Leibniz Universität Hannover, Germany
{rvlasov,kif,few}@gdv.uni-hannover.de

Abstract. In this paper we present a novel haptic rendering method for exploration of volumetric data. It 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. An improved version of the method presented here does not have the issues of the original method – oscillations of the interaction point and wrong friction force in some cases. It uses the ray casting technique for collision detection and a path finding approach for rigid collision response. 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. Additionally, the presented approach has a nearly constant time complexity independent of data resolution.

Keywords: haptics, haptic rendering, collision detection, collision response, collision resolution, ray casting, implicit surface, path finding.

1 Introduction

Nowadays 3D data processing and visualization, especially medical imaging, are widely used for analysis, diagnosis, illustration and other purposes, such as neurosurgery planning and reconstruction of volumetric data from Magnetic Resonance Imaging (MRI) and industrial CT (Computed Tomography).

When one works with 3D data, it is not very natural to navigate and manipulate it using a standard computer mouse and keyboard. A more intuitive way would be to use a device with more Degrees-of-Freedom (DoF). Several haptic devices fulfill this purpose, additionally providing an additional channel of interaction: feeling the objects – a user can both manipulate a virtual object and feel force feedback reactions. Since 3D data is widely used not only in medicine but in many different areas, such as CAD-applications, entertainment, museum display, sculpting, geology, military applications, various scientific applications and others, haptic devices could be also useful in these fields. Additionally, user studies were performed showing that a training with haptic devices gives better results than a training without them [1, 2].

There exist many different surface- and voxel-based haptic rendering methods, and almost all of them give no collision detection guarantees and/or require a special topology and/or precalculations, which is not 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. The approach is an improved version of the method presented in [3], addressing the following issues: sometimes there are oscillations of the interaction point (IP) around the locally closest surface point to the current position of the device manipulator, and the direction of the friction force could be incorrect in the case of multiple obstacles or a complex surface. Our improved method uses the ray casting technique for collision detection and a local path finding approach for rigid collision response. As in the original method, the improved approach has been implemented within the bounds of the YaDiV platform [4] – a virtual system for working with high-quality visualization of 3D volumetric data. 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.

2 Definitions

2.1 Volumetric Data

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.

Volume data, also called **volumetric data**, is a data set consisting of pairs $< coordinates, intensity\_value >$, where the intensity value is a scalar measured by a scanning device (e.g. the value of unabsorbed X-rays) [5]. One can take a look for a detailed description of volumetric data and related terms in [6].

Since 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 whether the voxel belongs to a certain structure (e.g. to kidneys or bones) or not. We use a bit cube representation of segments for this (see [7]). The development of segmentation processes is a large field of research, and different approaches for different purposes have already been proposed (see e.g. [5] and [7] for an overview and suggested methods). Further we assume that the 3D volumetric data is already segmented, i.e. that a set of segments (a set of scene objects) is provided.

2.2 Haptics

The term **haptic** (originating from the Greek *haptesthai*, meaning "to touch") is an adjective used to describe something relating to or based on the sense of touch. The word "haptic" is in relation to the word "touching" as "visual" is to "seeing" and as "auditory" is to "hearing".

Below are the definitions being used in the rest of this work.

Definition 1. *A **haptic device (or a haptic display)** is capable of tracking its own position and/or orientation and stimulates the kinesthetic sense of the user via a programmable force feedback system.*

Definition 2. *A **probe (or end-effector) (of a haptic display)** is the part of the device for which the position/orientation is tracked (passive DoF) and to which a force feedback is applied (active DoF).*

Definition 3. *A **tool (in a virtual world)** is an object in a virtual world the user manipulates via the probe. A particular case is the **(haptic) interaction point** (if the object is a 3D point).*

Definition 4. *A **handle (in a virtual world)** is a grasped part of the tool.*

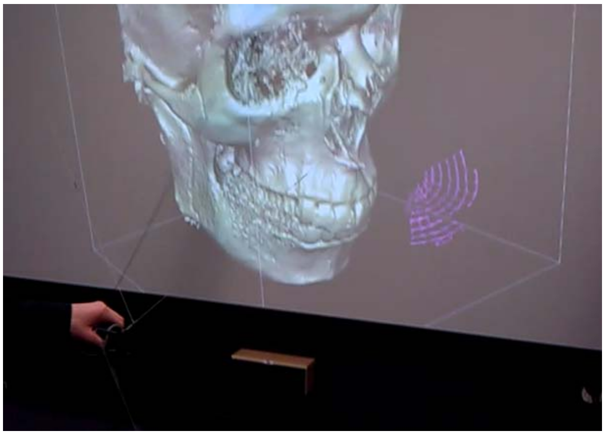

Fig. 1. Haptic rendering of the data set Head$_{big}$ using the INCA 6D device

3 Related Work

We would like to start with a description of a haptic rendering pipeline. Generally it has three stages as shown in Fig. 2. All stages are often tightly integrated in order to effectively use a solution of one task for solving others.

To communicate with the haptic device, there exist two ways:

– admittance control scheme: a user applies a force to the manipulator of the device, and the application program sends a command to the hardware to move the manipulator according to the simulation
– impedance control scheme: a user moves the manipulator, and the application program calculates forces and applies them to the manipulator. This scheme is shown in Fig. 2 and is usually used nowadays (see [8, 9]). It is also assumed in our work.

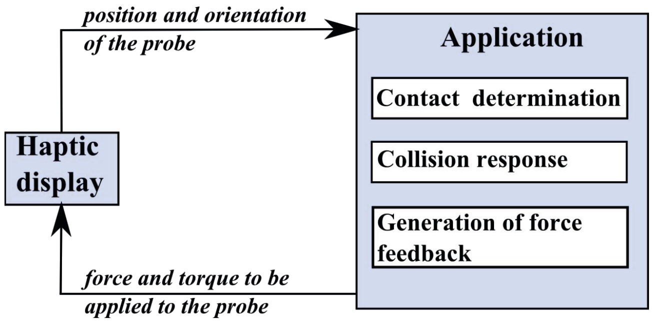
position and orientation of the probe
Application
Contact determination
Collision response
Generation of force feedback
Haptic display
force and torque to be applied to the probe

Fig. 2. Haptic rendering pipeline for the impedance control scheme

Colgate et al. showed in [10] that in order to have high quality haptic rendering it is important to compute feedback forces at a high update rate. According to Brooks et al. [11] it should be at least 0.5-1 kHz.

Remark 1. A force update rate of 1 kHz is generally not sufficient for stable haptic rendering. This means that other issues, such as too fast object movement or too strong forces, should be addressed in the system.

Further on, there exist two main techniques for dealing with the manipulation of the handle. The aim of both of them is to provide the user with stable haptic rendering. The techniques are:

- direct rendering: apply manipulations with the haptic probe directly to the handle
- virtual coupling (first proposed by Colgate et al. [10]): connect the haptic probe to the handle through a virtual spring-damper connection.

Each technique is suitable for a certain case. The direct rendering is good if all the stages of the haptic rendering pipeline can perform at 1 kHz (an update rate sufficient for a stable user interaction). The virtual coupling is good in the remaining cases, e.g. for multirate approaches, when force feedback is generated at 1 kHz, but physics simulation runs at, say, at 30 Hz.

There are different haptic rendering approaches, for which in the following we will give an overview by groups.

We start with **rigid-rigid methods**, i.e. for which the tool and all objects in the virtual world are rigid. Adachi et al. [12] and Mark et al. [13] were the first to propose an intermediate representation of the virtual environment. Zilles and Salisbury [14] proposed a god-object non-penetration approach for 3-DoF. Later the aforementioned approach was extended to 6-DoF by Ortega et al. [15]. An extension of the god-object idea in 3-DoF is a concept called "virtual proxy" [16]. 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. An extended 6-DoF technique of virtual proxy was used in [17]. McNeely et al. [18] developed a 6-DoF haptic rendering method for the Boeing Company. They proposed to use a voxmap (spatial occupancy map) and object pointshells. Volumetric data and penalty pre-contact forces were used. Later this approach was significantly improved in the next works of McNeely and others [19] and [20]. A completely different haptic rendering approach was suggested by Otaduy, Lin et al. [21, 22]. Their method allows haptic rendering of interaction between "haptically textured" triangulated models (with fine surface details stored as a height field). Collision detection between low-resolution meshes is based on sensation-preserving contact levels of detail from [23, 24]. Another interesting approach was suggested by Johnson and Willemsen [25, 26]. They used spatialized normal cone hierarchies for fast collision detection between the tool and an environmental object. Weller and Zachmann [27] presented inner sphere trees – a structure which bounds an object from inside with a set of non-overlapping bounding volumes – and employed it for haptic rendering. Vidal et al. [28] made a simulation of ultrasound guided needle puncture and proposed a proxy-based surface/volume haptic rendering for that. Palmerius et al. [29] have shown in their work how subdivision of proxy movements can improve precision of volume haptic rendering. Kim and others [30] presented a method that uses implicit surface representations and requires some preprocessing and a certain topology. An approach devoted to haptic rendering of volume-embedded isosurfaces was suggested by Chan et al. [31]. Another haptic rendering method, which uses isosurfaces defined by interpolating on tetrahedral meshes, was recently proposed by Corenthy et al. [32].

Another group consists of **rigid-defo methods**, i.e. for which the tool is rigid and the environment is deformable. The following methods could be marked out. Debunne et al. [33] presented a method for animating dynamic deformations of a visco-elastic object with a guaranteed frame-rate, built into a 6-DoF haptic rendering framework. An object is represented via a tetrahedral mesh, and the proposed physical simulation approach belongs to the domain of physics-based continuous models. Basing on [19], Barbic et al. [34, 35] proposed their own approach, which supports contact between rigid and reduced deformable models, both with complex geometry. A distributed contact between objects is allowed, i.e. an interaction with potentially several simultaneous contact sites. A pointshell-based hierarchical representation was used for the deformable object and a signed-distance field for the rest of the scene. Kuroda et al. [36] presented a simulation framework, where the manipulating point pushes a deformable object, which is in contact with another one. A work of Basdogan et al. [37] is devoted to 6-DoF haptics in minimally invasive surgical simulation and training. One more method for a "physically realistic" virtual surgery was suggested by De et al. [38]. They used the Point-Associated Finite Field (PAFF) approach. The idea is to discretize the computational domain (e.g. an organ) using a scattered set of points ("nodes") with spherical influence zone and defined a nodal shape function for it. In [39] Otaduy and Gross represented environmental deformable objects by tetrahedral meshes, In [40], a layered representation of objects is employed: a low-resolution mesh for the collision detection and haptic interaction, a deformable tetrahedral mesh for deformation computations and a detailed surface mesh for the deformable skin simulation. Luciano et al. [41] devoted their work to a local elastic point-based deformation around a contact point in 3-DoF haptic rendering. Ikits et al. [42] presented a 3-DoF constraint-based technique for haptic volume exploration. Chang et al. [43] proposed a 6-DoF haptic rendering method using the mass-spring simulation model.

For the **defo-defo methods**, i.e. for methods for which tool and environment are both deformable, the following methods should be noted. In the later work of Barbic et al. [44] the distance field was made parametrically deformable. In [45] Garre and Otaduy proposed haptic rendering of complex deformations through handle space force linearization. Duriez et al. [46] proposed a method using Signorini's contact model for deformable objects in haptic simulations with a focus on contact response. It belongs to approaches with non-penetration constraints and is independent from a collision/proximity detection. In the later work [47] the authors incorporated friction into the simulation model. Maciel et al. [48] also presented a haptic rendering method for physics-based virtual surgery by using NVIDIA's PhysX physics library, which is GPU accelerated. The latter method supports 6-DoF. Peterlik et al. [49] suggested an asynchronous approach for haptic rendering of deformable objects. In [50, 51, 52] Boettcher et al. suggested a kinesthetic haptic rendering of virtual fabrics grasped by two fingers (the HAPTEX project [53, 54, 55]). The fingers are represented by spherical tools manipulated via two 3-DoF probes. The simulation of tactile perception of the fabrics was proposed by Allerkamp et al. [56, 57]. The VR system developed in the HAPTEX project was the first one and until today still appears to be the only one offering (simultaneously) an integration of combined haptic and tactile perception, cf. [50, 51, 52, 53, 54, 55, 56, 57]. The exact physical properties of fabrics were simulated in the system, see [58]. Later on, in [59] Boettcher et al. described a generalized multi-rate coupling scheme of physical simulations for haptic interaction with deformable objects.

As was stated in the introduction, the motivation for our approach was that almost all methods referenced above can not give collision detection and non-penetration guarantees, as well as require a pre-specified topological structure of objects. We would like to provide the user with a method, which does not have these drawbacks.

4 Our Method

4.1 Collision Detection

The collision detection in our haptic rendering pipeline employs the ray casting technique (see e.g. [60, 61, 62, 63, 64, 65]), which has its roots in computer graphics. A short description was given in our work [66], and in this subsection we present it in more detail.

The idea of ray casting in visualization is to numerically evaluate the volume rendering integral in a straightforward manner. According to [67], the rendering integral $I_\lambda(x, r)$, i.e. the amount of the light of wavelength λ coming from a ray direction r that is received at location x on the image plane, is:

$$I_\lambda(x, r) = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt} ds, \tag{1}$$

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.

From the algorithmic point of view, ray casting in visualization works as follows: for each pixel of the image a ray is cast into the scene. Along the cast ray the intensity values of the volumetric data are resampled at equidistant intervals, usually using trilinear interpolation. 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 the $< coordinates, scalar\_value >$ pairs for the resampled points to colors and opacities according to a previously chosen transfer function is used.

In haptic rendering, for the collision detection of the interaction point (IP) following the position of the manipulator, we perform ray casting from its last position to the current one – Fig. 3(a). In more detail, we are going along the ray with 1-voxel steps – Fig. 3(b). If the value of any bit cube representing an obstacle at the sampled point is *true* – Fig. 3(c), – then a collision information and *true* is returned by the collision detection procedure – Fig. 3(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. To our best knowledge, there exists only one method (see [15]), which provides the same collision detection guarantees as ours, but it only works with triangulated objects and not with volumetric / voxel based data.

In case a higher precision for the collision detection is needed, ray casting at sub-voxel resolution or sampling once between each pair of consecutive intersections of the ray and a grid plane could be used. In our experiments, we found that 1-voxel step is sufficient for our data though.

In order to speed up computations further, a dynamic list of objects being determined as *collision candidates* is updated at each haptic frame. For that, we check if the ray from the last position of the IP 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 the collision candidates only. Furthermore, we put a reasonable upper limit on the maximal movement of the IP between two haptic frames. This allows to perform localized and therefore faster ray casting using the cached information from the previous frame and avoid possible haptic rendering instabilities (the last technique is also used in [35]).

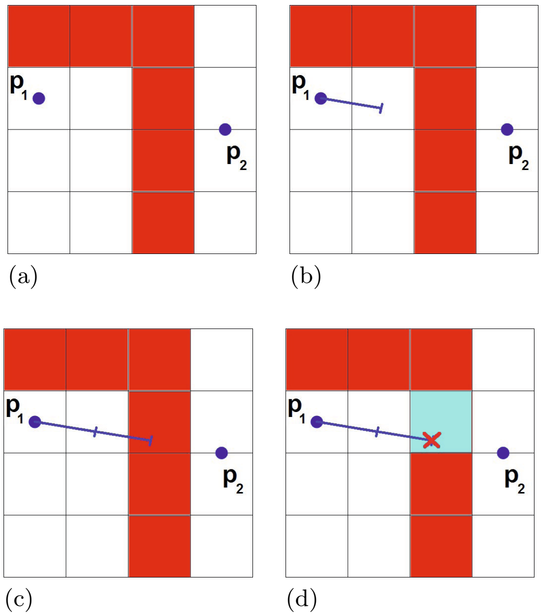

Fig. 3. The ray from the previous position p_1 to the current one p_2 is cast with 1-voxel steps until an obstacle is found or p_2 is reached

The time complexity of the method is

$$O\left(N_{obj}\frac{w_{max}}{step}\right), \qquad (2)$$

where N_{obj} – number of objects in the scene; w_{max} – maximum path length per frame, in voxels; *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.

4.2 Collision Response

The original version of our joint collision detection and response stage of the haptic rendering pipeline was proposed in our work [3]. Here we present its improved version, which uses the path-finding approach combined with the god object/proxy paradigm. It works directly with volumetric data and has no limitations. The idea of the original method from [3] was as follows.

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 approach was made to test the capacities and speed of our collision detection method and as a base for further experiments.

The position of the IP from the last frame is denoted as p_1, and the position to be calculated – as p_2. For the device's manipulator, we denote its last position as d_1 and the current one as d_2. The IP always moves in the direction of 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 [7]).

In more detail, the method is the following:

1. $p_2 := p_1$
2. Do the collision test from p_2 to d_2. If there is no collision then $p_2 := d_2$ and exit. Else move p_2 towards the collision point p_{col}, so that the distance between p_2 and p_{col} is less than the predefined $\epsilon < 1$
3. While $p_2 \neq d_2$ *and* the total path length of the IP at this haptic frame has not exceeded w_{max} (see section 4.1) *and* it is not shorter just to move directly from p_2 to d_2 do:
   (a) Locate empty-space border voxels neighbouring to p_2
   (b) Select a voxel with the maximal dot product (voxel-p_2, d_2-p_2) > 0. If there is no such voxel then go to step 4
   (c) Move p_2 to this voxel. If 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 $\leq w_{max}$ *and* $p_2 \neq d_2$ *and* $p_2 \neq$ the value of p_2 at the beginning of step 2, then go to step 2. Else exit.

Remark 2. There are some additional checks and minor details, which we omitted in the above description for clarity. A complete listing of the algorithm can be found in [3].

An example of how the method works is shown in Fig. 4. After the initialization at step 1, Fig. 4(a), the collision test is performed at step 2, Fig. 4(b). There is a collision, so the "sliding along the surface" part of the algorithm – step 3 – is executed, Fig. 4(c). Then the conditions for the outer loop (steps 2-4) are checked at step 4. As long as they are fulfilled, step 2, Fig. 4(d), and step 3, Fig. 4(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 cannot come closer to d_2 this time, so nothing is changed, and the algorithm stops at step 4.

We have found out that the use of the dot product of the vectors at step 3b 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 d_2 (lets denote it as 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 d_2-p_1 according to the conditions at step 3b, the IP may pass 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 p'_2. If d_2 remains unchanged at the next haptic frame then the IP will go the way back and will also pass p'_2 backwards direction and go further because of the same reason. At the next haptic frame the IP will go in the same

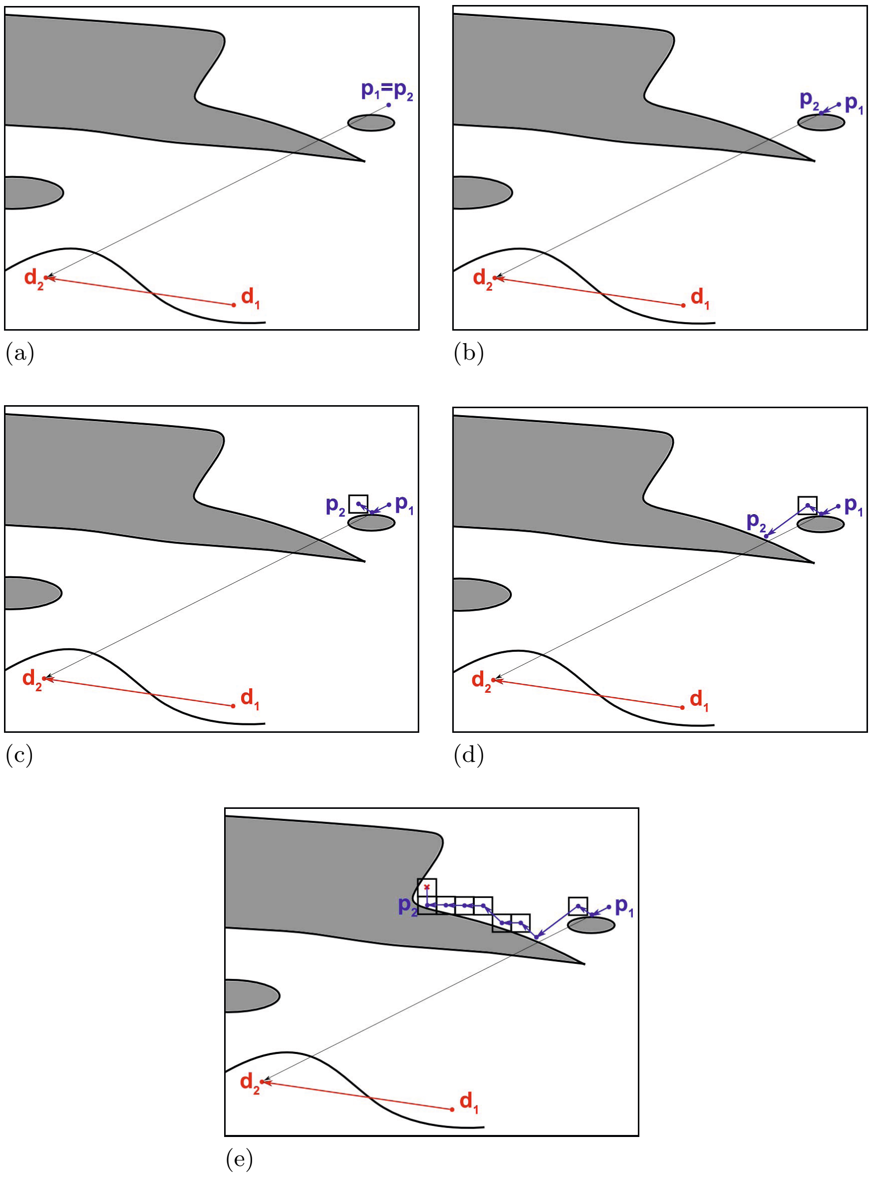

Fig. 4. Example of execution of the original "sliding along the surface" approach

direction as at the first haptic frame and will pass 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 d_2. In other words, we suggested to use a path finding algorithm looking for a locally optimal path to 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 $p_1$, $d_1$, $d_2$
2: $p_2 := p_1$ // Initialize $p_2$
3: Set $p_{2last}$ to be unequal to $p_2$
4: $w := 0$ // Path length travelled by the IP at this frame
5: while ($p_2 \neq d_2$ and $w < w_{max}$ and $p_{2last} \neq p_2$) do
6:    $p_{2last} := p_2$
7:    Make the collision test from $p_2$ to $d_2$
8:    if (no collision) then
9:       Move $p_2$ towards $d_2$ for the distance $\min(||d_2 - p_2||_2, w_{max} - w)$
10:      $w := w +$ (the above movement of $p_2$)
11:      break
12:   else
13:      Move $p_2$ towards the collision point $p_{col}$ so that it is at the given $\epsilon < 1$
         before $p_{col}$, or for the distance $(w_{max} - w)$ from $p_2$ in case the last is
         shorter
14:      $w := w +$ (the above movement of $p_2$)
15:      // Find a path to $d_2$ along the obstacle's surface, so that
16:      // the path is locally optimal at each step:
17:      while $w < w_{max}$ and $p_2 \neq d_2$ do
18:         // Is it shorter just to move from $p_2$ towards $d_2$
19:         // without following the surface?
20:         if ($p_2$ will not be inside any obstacle if moved by 1 voxel towards $d_2$)
            then
21:            // We will move directly to $d_2$ 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 $p_2$
26:         $dist\_sq := \infty$
27:         Select a voxel with the smallest square distance to $d_2$, and remember
            this distance as $dist\_sq$
28:         if ($dist\_sq = \infty$) then
29:            break
30:         end if
31:         Move $p_2$ towards the selected voxel for the distance $\min(||voxel - p_2||_2,$
            $w_{max} - w)$
32:         if ($p_2$ is inside another obstacle) then
33:            Cancel the above movement of $p_2$
34:            break
35:         end if
36:         $w := w +$ (the above movement of $p_2$)
37:      end while
38:   end if
39: end while
```

Note 1. Here and further in this work we assume that the empty-space border voxels are not precalculated. If they are precalculated for each segment at the preprocessing step then it will give 25% speed-up.

Additionally, we would like to note, that we still use the ideas from the original approach combined with the one presented here in order to add mass to the IP. This work is in progress.

4.3 Force Feedback

We improved our force feedback generation comparing to the one in [3]. This was necessary because the direction of the friction force F_{fr} could be wrong in the case of multiple obstacles or a complex surface, since we used the direction of p_2-p_1. Additionally, in the formula for F_{fr} we used w_{bv}, *the path length* which the IP travelled through empty-space border voxels, instead of *the number* of those empty-space border voxels. This was done, since the IP could move less than one voxel in the inner loop of the algorithm above.

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 $F = F_c + F_{fr}$, where F_c is a coupling force. If F exceeds a maximum for a given haptic device then we scale it as to fit to the device limitations. We define F_c as

$$F_c = -\frac{\mathrm{d}_2 - \mathrm{p}_2}{\|\mathrm{d}_2 - \mathrm{p}_2\|_2}(k\|\mathrm{d}_2 - \mathrm{p}_2\|_2) = k(\mathrm{p}_2 - \mathrm{d}_2), \tag{3}$$

where k is the coefficient of the spring.

For F_{fr} the updated expression could be written as

$$F_{fr} = -\mu\frac{\mathrm{v}_{bv}}{\|\mathrm{v}_{bv}\|_2}|F_c \cdot n|\frac{w_{bv}}{w}, \tag{4}$$

where μ is the friction coefficient; $\mathrm{v}_{bv} = \sum_i \mathrm{v}_i$, and v_i are linear path segments being travelled by the IP through the empty-space border voxels at this haptic frame; n – a normal vector being perpendicular to v_{bv} and located on the plane spanned by v_{bv} and d_2-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 $|F_c \cdot n|$ could be rewritten as $\|F_c\|_2 - \left|F_c \cdot \frac{\mathrm{v}_{bv}}{\|\mathrm{v}_{bv}\|_2}\right|$.

We suggest the new formula for F_{fr} as opposed to [3] because at the end of a haptic frame the IP is moved from p_1 to p_2, so it is logical to turn 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 F_{fr} to be proportional to the part of F_c which is perpendicular to 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. We would like to note that making the forces related to physical properties of certain materials was not our goal on this stage of research.

5 Implementation

5.1 Prototype System

We developed our interactive VR system as a plug-in for the YaDiV Open-Source platform [4]. YaDiV is used for teaching and in various research projects, and was developed in Java. The last 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. Our system is independent from a haptic display, so that a wide range of devices can be used, including Phantom Omni, High-end Phantom Premium 1.5 6-DOF and INCA 6D with a very large workspace of approx. 2x1x1.6m (Fig. 1). The size of the virtual workspace can be scaled and varies from case to case.

Since Java is executed on a Virtual Machine (VM), 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. We conducted experiments and found out that the delays appear even with the simplest Java application. The authors of [68], [69] wrote that a real-time VM can provide a deterministic execution time, i.e. to eliminate the aforementioned issue. We conducted experiments with two common real-time VMs: Sun JavaRTS and IBM Web Sphere Real Time. We followed all recommendations of the developers, like installation of Linux with a real-time core and fine tuning of the VM. As a result, we found out that there are still delays of 1-3 ms. We would like to point out that the observed results differ from the information stated in [68] and [69], which was officially presented by IBM and Sun respectively.

5.2 Synchronization Issues

The graphical representation of objects in YaDiV is re-rendered upon request. That is, when properties (color, position, ...) of a scene object are changed, the scene is redrawn. Together with haptic interaction, this rendering scheme leads to synchronization problems. If we would 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.

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. 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.

6 Results

Using the improved method, we repeated the tests as stated in [3]. Since we improved our method, it was necessary to perform the tests again. We used the same real tomography data sets as in our last work, including Torso (520x512x512, Fig. 5), Head_{big} (464x532x532, Fig. 1) and Head_{small} (113x256x256, Fig. 6).

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 (8 x Intel Xeon CPU W5580 @ 3.20GHz, 24 GB RAM, NVIDIA Quadro FX 5800). 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. We have measured the timings of each step and found out that the update rate is lower because of the required Java-C++ communication (transferring of the device transformations and forces), since the haptic device dependent part was developed using C++ (see section 5). All values for the data sets for the joint collision detection and response approach are shown in table 1. *Triangles* denotes the number of triangles in the scene for the graphics rendering as a reference. The 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.

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. The system was tested under Microsoft Windows and 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 comparing to the normal graphics mode.

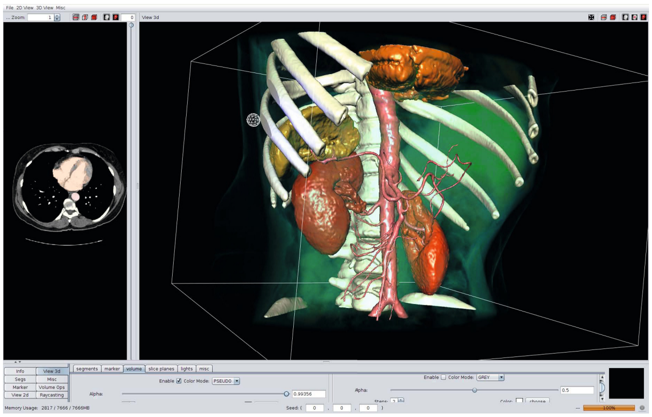

Fig. 5. Working with the Torso data set

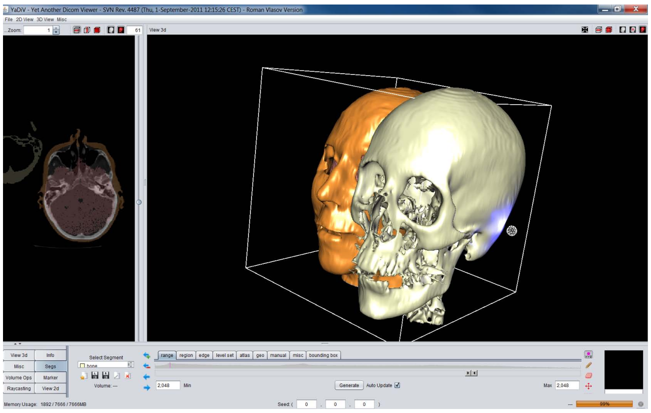

Fig. 6. The data set Head$_{small}$

Table 1. Resulting update rates

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

7 Summary and Future Work

In this work we presented an improved version of our haptic rendering approach originally proposed in [3]. 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. 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. Additionally, we presented an improved force feedback generation scheme, which does not suffer issues of the original scheme given in [3]. 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. The prototype was implemented as a plug-in of the YaDiV system and supports different haptic devices and operation systems.

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, we plan to introduce object-object interactions, where the controlled object is represented as a set of points, and implement the collision detection stage on GPUs, since ray casting could be efficiently parallelized (see e.g. [61], [65]). This will allow us to make computations faster and therefore represent the controlled object with more points and/or perform a more sophisticated collision response. For the latter case, we plan to use FEM-based approaches for simulation of elastic deformations, as e.g. in [47], [39], [44], but we will work directly with volumetric data. 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. For that, it is planned to make an assessment of our VR system by physicians from Hanover Medical School (MHH).

Acknowledgements. This research is sponsored by a grant provided by Siemens/DAAD Postgraduate Programme (DAAD - German Academic Exchange Service). The authors would like to thank B.Berger, P.Blanke, A.Vais and R.Buchmann for helpful comments.

References

[1] Nakao, M., Kuroda, T., Komori, M., Oyama, H.: Evaluation and user study of haptic simulator for learning palpation in cardiovascular surgery. In: Proc. of Int. Conference of Artificial Reality and Tele-Existence (ICAT 2003), pp. 203–208 (2003)
[2] Sewell, C., Blevins, N.H., Peddamatham, S., Tan, H.Z., Morris, D., Salisbury, K.: The effect of virtual haptic training on real surgical drilling proficiency. In: 2nd Joint EuroHaptics Conf. and Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems (WHC 2007), pp. 601–603 (2007)
[3] Vlasov, R., Friese, K.-I., Wolter, F.-E.: Haptic rendering of volume data with collision determination guarantee using ray casting and implicit surface representation. In: Proc. of Cyberworlds 2012 Int. Conference, pp. 91–99 (September 2012)
[4] Friese, K.-I., Blanke, P., Wolter, F.-E.: Yadiv – an open platform for 3d visualization and 3d segmentation of medical data. The Visual Computer 27, 129–139 (2011)
[5] Chen, M., Correa, C., Islam, S., Jones, M., Shen, P.Y., Silver, D., Walton, S.J., Willis, P.J.: Manipulating, deforming and animating sampled object representations. Computer Graphics Forum 26(4), 824–852 (2007)
[6] Kaufman, A., Cohen, D., Yagel, R.: Volume graphics. IEEE Computer 26(7), 51–64 (2007)
[7] Friese, K.-I.: Entwicklung einer Plattform zur 3D-Visualisierung und -Segmentierung medizinischer Daten. PhD thesis, Leibniz Universitat Hannover, Germany (2010)
[8] Glencross, M., Chalmers, A.G., Lin, M.C., Otaduy, M.A., Gutierrez, D.: Exploiting perception in high-fidelity virtual environments. ACM SIGGRAPH 2006 Courses (July 2006)
[9] Otaduy Tristan, M.A.: 6-dof haptic rendering using contact levels of detail and haptic textures. PhD thesis, University of North Carolina at Chapel Hill (2004)
[10] Colgate, J.E., Stanley, M.C., Brown, J.M.: Issues in the haptic display of tool use. In: Proc. of IEEE/RSJ International Conf. on Intelligent Robots and Systems, pp. 140–145 (1995)

[11] Brooks Jr., F.P., Ouh-Young, M., Batter, J.J., Kilpatrick, P.J.: Project grope - haptic displays for scientific visualization. ACM SIGGRAPH Computer Graphics 24(4), 177–185 (1990)
[12] Adachi, Y., Kumano, T., Ogino, K.: Intermediate representation for stiff virtual objects. In: Virtual Reality Annual International Symposium, pp. 203–210 (1995)
[13] Mark, W.R., Randolph, S.C., Finch, M., Van, J.M., Russell, V., Taylor II, M.: Adding force feedback to graphics systems: issues and solutions. In: Proc. of the 23rd Annual Conf. on Comp. Graphics and Interactive Techniques, pp. 447–452 (1996)
[14] Zilles, C.B., Salisbury, J.K.: A constraint-based god-object method for haptic display. In: Proc. of the Int. Conf. on Intelligent Robots and Systems, vol. 3, pp. 31–46 (1995)
[15] Ortega, M., Redon, S., Coquillart, S.: 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 (2007)
[16] Ruspini, D.C., Kolarov, K., Khatib, O.: The haptic display of complex graphical environments. In: Proc. of the 24th Ann. Conf. on Comp. Gr. and Interact. Techn., pp. 345–352 (1997)
[17] Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M., Manocha, D.: Six degree-of-freedom haptic display of polygonal models. In: Proc. of the Conf. on Vis. 2000, pp. 139–146 (2000)
[18] McNeely, W.A., Puterbaugh, K.D., Troy, J.J.: Six degree-of-freedom haptic rendering using voxel sampling. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 401–408 (July 1999)
[19] Wan, M., McNeely, W.A.: Quasi-static approximation for 6 degrees-of-freedom haptic rendering. In: Proc. of the 14th IEEE Vis. Conf. (VIS 2003), pp. 257–262 (2003)
[20] McNeely, W.A., Puterbaugh, K.D., Troy, J.J.: Voxel-based 6-dof haptic rendering improvements. Journal of Haptics-e 3(7) (2006)
[21] Otaduy, M.A., Jain, N., Sud, A., Lin, M.C.: Haptic display of interaction between textured models. In: Proc. of the Conf. on Visualization 2004, pp. 297–304 (October 2004)
[22] Otaduy, M.A., Lin, M.C.: A perceptually-inspired force model for haptic texture rendering. In: Proc. of the 1st Symp. on App. Perception in Graphics and Vis., pp. 123–126 (2004)
[23] Otaduy, M.A., Lin, M.C.: Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. In: Proc. of the 1st Joint Eurohaptics Conf. and Symp. on Hapt. Interf. for Virt. Env. and Tel. Syst., pp. 247–256 (2005)
[24] Otaduy, M.A., Lin, M.C.: A modular haptic rendering algorithm for stable and transparent 6-dof manipulation. IEEE Trans. on Robotics 22(4), 751–762 (2006)
[25] Johnson, D.E., Willemsen, P.: Six degree-of-freedom haptic rendering of complex polygonal models. In: Proc. of the 11th Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS 2003), pp. 229–235 (2003)
[26] Johnson, D.E., Willemsen, P., Cohen, E.: Six degree-of-freedom haptic rendering using spatialized normal cone search. IEEE Transactions on Visualization and Computer Graphics 11(6), 661–670 (2005)
[27] Weller, R., Zachmann, G.: A unified approach for physically-based simulations and haptic rendering. In: Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games, pp. 151–160 (August 2009)

[28] Vidal, F., John, N., Healey, A., Gould, D.: 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)

[29] Lundin Palmerius, K., Baravdish, G.: Higher precision in volume haptics through subdivision of proxy movements. In: Ferre, M. (ed.) EuroHaptics 2008. LNCS, vol. 5024, pp. 694–699. Springer, Heidelberg (2008)

[30] Kim, L., Kyrikou, A., Desbrun, M., Sukhatme, G.: An implicit-based haptic rendering technique. In: Proc. of the IEEE/RSJ International Conf. on Intelligent Robots (2002)

[31] Chan, S., Conti, F., Blevins, N., Salisbury, K.: Constraint-based six degree-of-freedom haptic rendering of volume-embedded isosurfaces. In: W. Hapt. Conf. 2011, pp. 89–94 (2011)

[32] Corenthy, L., Martin, J.S., Otaduy, M., Garcia, M.: Volume haptic rendering with dynamically extracted isosurface. In: Proc. of Haptics Symp. 2012, pp. 133–139 (2012)

[33] Debunne, G., Desbrun, M., Cani, M.P., Barr, A.H.: Dynamic real-time deformations using space & time adaptive sampling. In: Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp. 31–36 (2001)

[34] Barbic, J., James, D.: Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. In: Proc. of the 2007 ACM SIGGRAPH/Eurogr. Symp. on Comp. Animation, pp. 171–180 (2007)

[35] Barbic, J.: Real-time reduced large-deformation models and distributed contact for computer graphics and haptics. PhD thesis, Carnegie Mellon University, Pittsburgh (2007)

[36] Kuroda, Y., Nakao, M., Hacker, S., Kuroda, T., Oyama, H., Komori, M., Matsuda, T., Takahashi, T.: Haptic force feedback with an interaction model between multiple deformable objects for surgical simulations. In: Proceedings of Eurohaptics 2002, pp. 116–121 (July 2002)

[37] Basdogan, C., De, S., Kim, J., Muniyandi, M., Kim, H., Srinivasan, M.A.: Haptics in minimally invasive surgical simulation and training. IEEE Computer Graphics and Applications 24(2), 56–64 (2004)

[38] De, S., Lim, Y.J., Manivannan, M., Srinivasan, M.A.: Physically realistic virtual surgery using the point-associated finite field (paff) approach. Presence: Teleoperators and Virtual Environments 15(3), 294–308 (2006)

[39] Otaduy, M.A., Gross, M.: Transparent rendering of tool contact with compliant environments. In: Proc. of the 2nd Joint EuroHaptics Conf. and Symp. on Haptic Interfaces for Virt. Env. and Teleoperator Systems, pp. 225–230 (2007)

[40] Galoppo, N., Tekin, S., Otaduy, M.A., Gross, M., Lin, M.C.: Interactive haptic rendering of high-resolution deformable objects. In: Shumaker, R. (ed.) HCII 2007 and ICVR 2007. LNCS, vol. 4563, pp. 215–223. Springer, Heidelberg (2007)

[41] Luciano, C.J., Banerjee, P., Rizzi, S.H.R.: Gpu-based elastic-object deformation for enhancement of existing haptic applications. In: Proc. of the 3rd Annual IEEE Conf. on Automation Science and Engineering, pp. 146–151 (2007)

[42] Ikits, M., Brederson, J.D., Hansen, C.D., Johnson, C.R.: A constraint-based technique for haptic volume exploration. In: Proceedings of the 14th IEEE Visualization 2003 (VIS 2003), pp. 263–269 (October 2003)

[43] Chang, Y.H., Chen, Y.T., Chang, C.W., Lin, C.L.: Development scheme of haptic-based system for interactive deformable simulation. Computer-Aided Design 42(5), 414–424 (2010)

[44] Barbic, J., James, D.L.: Six-dof haptic rendering of contact between geometrically complex reduced deformable models. IEEE Trans. on Haptics 1(1), 39–52 (2008)

[45] Garre, C., Otaduy, M.A.: Haptic rendering of complex deformations through handle-space force linearization. In: Proc. of the World Haptics Conf., pp. 422–427 (2009)
[46] Duriez, C., Andriot, C., Kheddar, A.: Signorini's contact model for deformable objects in haptic simulations. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 32–37 (2004)
[47] Duriez, C., Dubois, F., Kheddar, A., Andriot, C.: Realistic haptic rendering of interacting deformable objects in virtual environments. IEEE Transactions on Visualization and Computer Graphics 12(1), 36–47 (2006)
[48] Maciel, A., Halic, T., Lu, Z., Nedel, L.P., De, S.: Using the physx engine for physics-based virtual surgery with force feedback. Int. Journal of Medical Robotics and Computer Assisted Surgery 5(3), 341–353 (2009)
[49] Peterlik, I., Duriez, C., Cotin, S.: Asynchronous haptic simulation of contacting deformable objects with variable stiffness. In: 2011 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2608–2613 (2011)
[50] Boettcher, G.: Haptic Interaction with Deformable Objects. Springer (2011)
[51] Boettcher, G., Allerkamp, D., Wolter, F.-E.: Virtual reality systems modelling haptic two-finger contact with deformable physical surfaces. In: Proc. of HAPTEX 2007, pp. 292–299 (October 2007)
[52] Boettcher, G., Allerkamp, D., Gloeckner, D., Wolter, F.-E.: Haptic two-finger contact with textiles. Visual Computer 24(10), 911–922 (2008)
[53] Salsedo, F., Fontana, M., Tarri, F., Ruffaldi, E., Bergamasco, M., Magnenat-Thalmann, N., Volino, P., Bonanni, U., Brady, A., Summers, I., Qu, J., Allerkamp, D., Boettcher, G., Wolter, F.-E., Makinen, M., Meinander, H.: Architectural design of the haptex system. In: Proc. of the HAPTEX 2005 Workshop on Haptic and Tactile Perception of Deformable Objects (peer-reviewed), pp. 1–7 (December 2005)
[54] Magnenat-Thalmann, N., Volino, P., Bonanni, U., Summers, I.R., Bergamasco, M., Salsedo, F., Wolter, F.-E.: From physics-based simulation to the touching of textiles: The haptex project. Int. Journal of Virtual Reality 6(3), 35–44 (2007)
[55] Fontana, M., Marcheschi, S., Tarri, F., Salsedo, F., Bergamasco, M., Allerkamp, D., Boettcher, G., Wolter, F.-E., Brady, A.C., Qu, J., Summers, I.R.: Integrating force and tactile rendering into a single vr system. In: Proc. of HAPTEX 2007, pp. 277–284 (October 2007)
[56] Allerkamp, D., Boettcher, G., Wolter, F.-E., Brady, A.C., Qu, J., Summers, I.R.: A vibrotactile approach to tactile rendering. Vis. Computer 23(2), 97–108 (2007)
[57] Allerkamp, D.: Tactile Perception of Textiles in a Virtual-Reality System, vol. 10. Springer, Heidelberg (2011)
[58] Volino, P., Davy, P., Bonanni, U., Magnenat-Thalmann, N., Boettcher, G., Allerkamp, D., Wolter, F.-E.: From measured physical parameters to the haptic feeling of fabric. In: Proc. of the HAPTEX 2005 Workshop on Haptic and Tactile Perception of Deformable Objects (peer-reviewed), pp. 17–29 (December 2005)
[59] Boettcher, G., Allerkamp, D., Wolter, F.-E.: Multi-rate coupling of physical simulations for haptic interaction with deformable objects. Visual Computer 26(6-8), 903–914 (2010)
[60] Hadwiger, M., Ljung, P., Salama, C.R., Ropinski, T.: Advanced illumination techniques for gpu-based volume raycasting. In: ACM SIGGRAPH 2009 Courses (2009)

[61] Kruger, J., Westermann, R.: Acceleration techniques for gpu-based volume rendering. In: Proc. of the 14th IEEE Visualization 2003 (VIS 2003), pp. 287–292 (October 2003)
[62] Levoy, M.: Efficient ray tracing of volume data. ACM Transactions on Graphics 9(3), 245–261 (1990)
[63] Mensmann, J., Ropinski, T., Hinrichs, K.: Accelerating volume raycasting using occlusion frustums. In: IEEE/EG Int. Symp. on Vol. and Point-Based Graphics, pp. 147–154 (2008)
[64] Engel, K., Hadwiger, M., Kniss, J.M., Lefohn, A.E., Salama, C.R., Weiskopf, D.: Real-time volume graphics. ACM SIGGRAPH 2004 Course Notes (2004)
[65] Ropinski, T., Kasten, J., Hinrichs, K.H.: Efficient shadows for gpu-based volume raycasting. In: Proc. of the 16th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2008), pp. 17–24 (2008)
[66] Vlasov, R., Friese, K.-I., Wolter, F.-E.: Ray casting for collision detection in haptic rendering of volume data. In: I3D 2012 Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, p. 215 (March 2012)
[67] Bruckner, S.: Efficient volume visualization of large medical datasets. Master's thesis, Vienna University of Technology, Austria (May 2004)
[68] Stoodley, M., Fulton, M., Dawson, M., Sciampacone, R., Kacur, J.: Real-time Java, Part 1: Using Java code to program real-time systems (April 2007)
[69] Oracle: Sun java real-time system 2.2 update 1 technical documentation (April 2010),
`http://download.oracle.com/javase/realtime/rts_productdoc_2.2u1.html`