Stable Haptic Interaction with Virtual Environments Using an Adapted Voxmap-PointShell Algorithm

Matthias Renz[1,2], Carsten Preusche[1], Marco Pötke[2], Hans-Peter Kriegel[2], Gerd Hirzinger[1]

[1] *German Aerospace Center (DLR)*
Institute of Robotics and Mechatronics
Oberpfaffenhofen, D-82234 Weßling
renz@robotic.dlr.de,
Carsten.Preusche@dlr.de, Gerd.Hirzinger@dlr.de

[2] *University of Munich*
Institute for Computer Science
D-80538 Munich
poetke@informatik.uni-muenchen.de,
kriegel@informatik.uni-muenchen.de

Abstract

Despite a lot of research in recent years stable and realistic haptic interaction with virtual environments keeps an unsolved problem. Among different approaches the Voxmap-PointShell™ (VPS) method seems very promising due to constant sample rates, independent of the static environment. But there is still the stability problem to be solved globally. In this paper some adaptations based on the VPS are presented, including the dynamic object modeling and the force calculation method, to reduce the distractions of the calculated collision forces to increase stability. The PointShell points lie exactly on the surface of the dynamic object, to get a smoother surface representation. A variation of the collision force calculation leads to a reduction of the "voxel noise", that appears due to the discretization of the volume space. A design framework for the virtual coupling is presented, that enables the automatic configuration for different haptic devices. The validity will be shown with different kineasthetic hand controllers.

1. Introduction

There are numerous haptic rendering algorithms for virtual simulations, differing in object and surface modeling. Object models are mostly approximate descriptions of the simulated objects, used in the virtual environment. In many rendering algorithms, the geometrical complexity of the particular modeled objects is restricted due to high rendering times. In the VPS approach of McNeely, Puterbaugh and Troy [6], the virtual environment consists of objects divided into dynamic and static objects. Dynamic objects can freely move through the virtual space, whereas the static objects are fixed in the world coordinate system. With a haptic device, a human can touch the static objects (static environment) with the dynamic object, and the rendered collision forces will be fed back to the operator.

The haptic rendering update rate is independent from the complexity of the static environment, thus qualifying for real time applications. The environment of static objects is collectively represented by a single spatial occupancy map called a voxmap (volume map). This is created by discretizing the static environment space, which is partitioned into regions of free space, object surface, and object interior. The collection of the discrete volume elements (voxels) build the voxmap. The dynamic object is described by a collection of points (PointShell), which models its surface. Each point is assigned a surface normal vector, pointing inwards. The haptic rendering algorithm includes a fast collision detection technique based on probing the voxmap with the surface point samples of the PointShell. By using the normal vectors of the PointShell, an approximate collision force can be easily computed in constant time for each point-voxel interpenetration. Figure 1 shows a PointShell colliding with the voxmap. For each PointShell point, contained by a surface voxel, the depth of interpenetration is calculated as the distance d from the point to the tangent plane. This plane passes through the voxel center and has the same normal vector as the PointShell point normal vector.

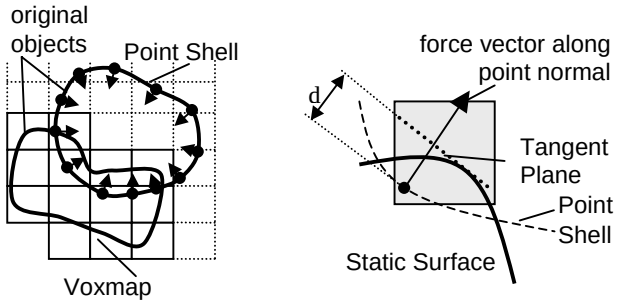

Figure 1: Voxmap-PointShell-Model™

So the time consumed to render a single frame depends only on the number of PointShell points. The real time capability qualifies this algorithm for online haptic interactions between a human operator and a virtual environment. For enhancing haptic stability, McNeely et al. [6] employ a 'virtual coupling' scheme, which connects the user's haptic motion with the motions of the dynamic object through a virtual spring and damper. This scheme was embedded in the haptic rendering algorithm. But mechanical properties of different haptic devices are not treated by this virtual coupling, and it is difficult to tune the system to obtain a stable haptic feedback.

In contrast, the approach described in this paper decouples the stability problem from the haptic rendering, as also proposed by Adams and Hannaford [1], where the virtual coupling was treated as part of the haptic interface (device and virtual environment). This allows the usage of an arbitrary haptic rendering algorithm independent from the haptic device. Almost no changes of the virtual environment parameters are required, when the haptic device changes. In chapter 2, we present adaptations to the VPS based object modeling and force calculation, for reducing the distractions of collision forces to get more stability. The 'virtual coupling' is described in chapter 3. Chapter 4 shows some results of experiments, where the validity of the methods can be seen. The paper closes with the conclusion in chapter 5.

2. Extending the VPS Algorithm

2.1 Exact PointShell Creation

The calculation of the contact force takes place in the force voxel layer which is one voxel deep. Due to the design of the VPS method the effective force layer depth around static objects amounts to half a voxel. So it can be seen easily, that surface deviations of the PointShell object, with an amplitude of more then a half voxel size, have considerable influence on force distractions. This surface variance appears due to the PointShell creation described in the original VPS, where the object first was voxelized, and the collection of all surface voxel centers build the PointShell (Fig. 3, 1-3).

Next, the extended PointShell creation is presented, where the points lie on the triangulated object surface (anti-aliasing), not only to improve the accuracy, but also to stabilize the collision force. The example in Figure 2 shows the same collision situation of a dynamic and static object, with the two different PointShell models. In the upper scene (Fig. 2 a), the PointShell was built with the original algorithm, where the distance between the PointShell points and the object surface is up to $\frac{\sqrt{3}}{2}e$, where e denotes the voxel extension. This surface deviation is very high compared to the force layer depth of e/2. In the case of Figure 2 a) the PointShell was created using the voxel raster shown with dashed lines. With this rough surface approximation only one of the four PointShell points effects a collision force, although the 'real' dynamic object surface cross three force layer voxel. Figure 2 b) shows the effected collision forces using the exact PointShell model, creating a smoother force field.

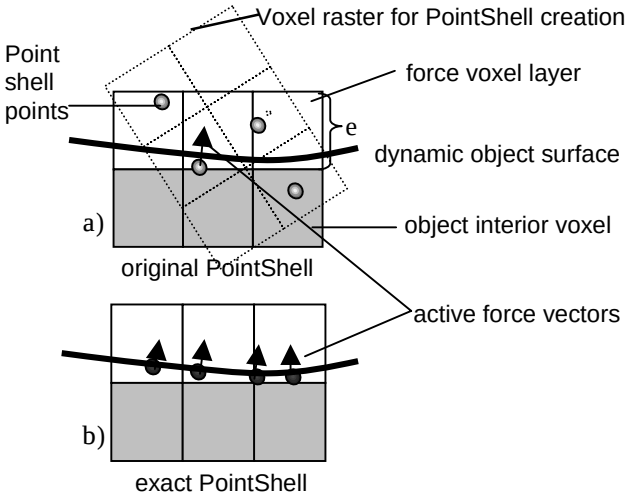

Figure 2: Different collision force effects in comparison between the original PointShell and our exact Point Shell

The exact PointShell creation process steps are shown in Figure 3. The first two steps are equal to the original PointShell creation. In order to guarantee a proper detection of collisions between dynamic and static objects, the distance between two neighbored PointShell points is limited according to the voxel resolution of the static environment. Hence the 'naive' PointShell is built by the sum of all center points of the resulting surface voxels, after creating the VoxMap of the dynamic object with the same resolution as of the static environment (Fig. 3, 1-3).

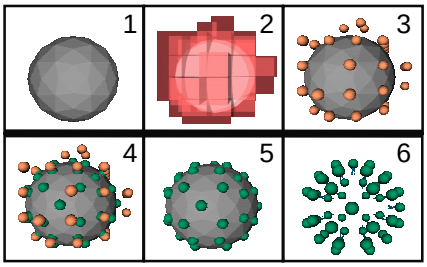

Figure 3: Steps of the exact PointShell creation algorithm

A smoother surface representation can be achieved by projecting each center point on to the triangle surface

(Fig. 3, 4-5). Two additional requirements for the new position have to be fulfilled:

1) inside the primary voxel
2) minimal distance to the voxel center

This two requirements can be inconsistent with one another, whereas the first requirement has the higher priority and the second should be optimized as far as possible. The first condition keeps the maximal neighbor point distance such that an unrecognized interpenetration of the static and dynamic object is prevented. The extension of the spatial gap between two PointShell point neighbors is restricted to the minimal spatial extension of a static object surface part (3 voxel extensions, if a force field is used as described by McNeely [6]). The property of the shortest distance tries to keep the original distance of the point neighborhood as far as the other conditions allows, for a convenient point distribution.

Algorithm: The following algorithm shows the correct point-surface-projection, keeping the above requirements with the defined priority. The object surface is described by a number of triangles. Due to the condition, that the projected point must be contained by the respective voxel, only the triangles with a common intersection point with that voxel have to be considered. For each triangle the following steps have to be done. First calculate from the voxel center the perpendicular nadir p_0 on the triangle surface. Now four cases have to be considered:

Let:
- V be the set of all points contained by the respective Voxel;
- VS be the set of all points lying on the surface of the respective Voxel; VS $\subseteq$ V;
- T be the set of all points lying on the observed triangle;

then:

case 1: $p_0 \in V$ and $p_0 \in T$:
the result is p_0;

case 2: $p_0 \in V$ and $p_0 \notin T$:
return point $p_r \in \{T \cap V\}$, which has the shortest distance to p_0;

case 3: $p_0 \notin V$ and $p_0 \in T$:
return point $p_r \in \{T \cap VS\}$, which has the shortest distance to p_0;

case 4: $p_0 \notin V$ and $p_0 \notin T$:
if ($p_r \in T$ with the shortest distance to p_0) $\in V$ then the result is p_r,
else the result is equal to the result of case 3;

Figure 4 shows two examples of case 4. The treatment in case 4 is generally valid, therefore all cases can be handled by the following algorithm:

Algorithm :

projectPoint(Voxel, Triangle)

```
p0 =    perpendicularNadir(Voxel.centerpt, Triangle.plane);
pr =    trianglePointWithShortestDistanceToPoint(p0);
if      (Voxel.contains(pr))
then    return(pr)
else    intersection_lines = intersect(Triangle, Voxel.surface);
        for each Line ∈ intersection_lines
            pl = linePointWithShortestDistanceToPoint(p0,Line);
            presult = min(distance(pl,p0), distance(presult,p0));
return presult;
end;
```

For each original PointShell point (voxel center), map it on the corresponding triangles, and choose the result point with the shortest distance to the voxel center. The set of all mapped points build the exact PointShell. Now the PointShell vectors can be calculated from the normal directions of the triangles intersecting the relevant voxel.

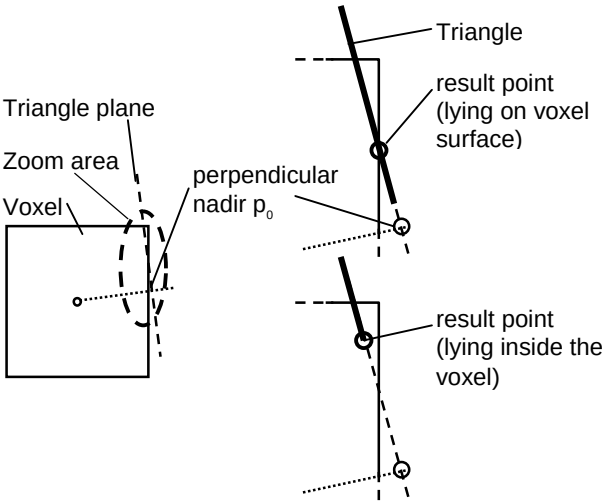

Figure 4: Two examples showing the projection treatment of case 4

2.2 Adapted Collision Force Calculation

Further force discontinuities appear, when PointShell points cross static voxel borders. Especially under sliding motion with a relatively constant interpenetration between the dynamic and static object, this effect, shown in Figure 5 (a, b), causes instability and notably disturbs the operator's correct sense of touch (vibrations). The presented variation of the VPS collision force calculation [6], leads to smoother transitions at voxel borders. In contrary to hierarchical methods, which increase the calculation time, the presented method does not need a finer resolution (granularity) of the static environment.

Instead of providing a general solution to the force stability problem, our adapted force calculation specifically targets the force noise for dynamic objects

sliding over a plane of adjacent surface voxels. When PointShell points cross the border between these adjacent voxels, instability occurs if the direction of the relevant point normal vector is not perpendicular to the sliding plane. Figure 5 a) shows a dynamic object colliding with a static object under sliding motion. Figure 5 b) and c) demonstrate the interesting area (marked with the dashed line in 5 a)). Figure 5 b) shows the problem that appears with the original force calculation method. When the PointShell point crosses the voxel border, the calculated force value is reduced to zero, which influences the overall collision force. Instead of using the individual PointShell normals to calculate the collision force, the average of all collision point normals is used to calculate the interpenetration distance d of each PointShell point (Figure 5 c)). This average direction is a more stable approximation of the sliding plane normal, and the mentioned problem can be mitigated.

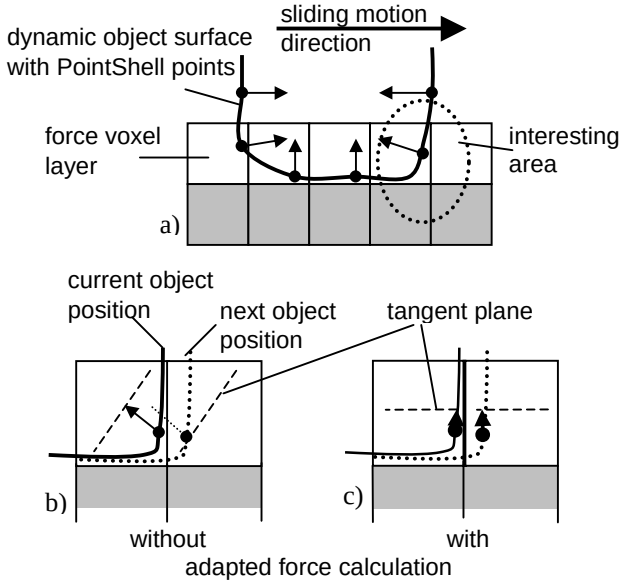

Figure 5: Force stabilization under sliding motion

3. Virtual Coupling

The schemes described above only reduce force noise, but there are still remaining distracting forces causing an unstable system. The direct way to reach the required stability is to use a 'virtual coupling' scheme between the operator and the virtual environment. A design framework for the virtual coupling is presented, that allows a consistent interaction between the haptic rendering (VPS) and different haptic devices, because for the overall stability, the dynamic properties of the haptic device play a key role.

In many other approaches (e.g. McNeely et al. [6], Adams and Hannaford [1]) the virtual coupling is treated and parameterized as a spring-damper system, which is a good mechanical interpretation, but leads in most cases to a heuristic optimization of the parameters. The spring-damper system is used to reduce the force steps, produced by discontinuities and discretization, for the human user.

The virtual coupling can also be seen as a dynamic shaping filter, as it is done in this approach (Fig. 6). The dynamics of the calculated contact force are reduced such that neither the haptic device nor the force reflection loop with the human become unstable.

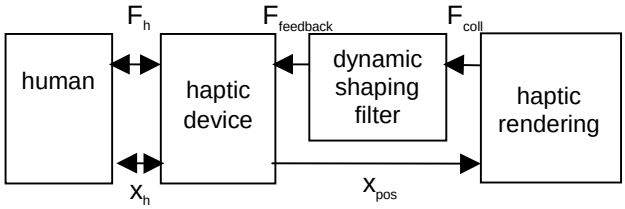

Figure 6: Signal flow diagram of the haptic simulation

Filter Construction

The filter should keep the role of the virtual coupling, thus the physically model of the virtual coupling (the spring-damper system) is used as a model for the construction part.

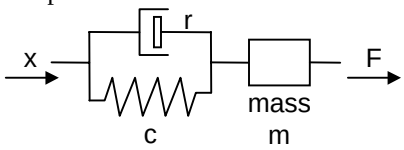

Figure 7: Mechanical spring-damper system

With this mechanical representation (Fig. 7) the following equation can be built:

$$F = c \cdot x + r \cdot \dot{x} + m \cdot \ddot{x}$$

For the filter, the translation value x must be replaced by the collision force as input, which is generated by the haptic rendering algorithm. This replacement can be done under the condition that the collision force depends directly on the penetration distance, which is true with this haptic rendering algorithm ($F_{coll} \sim x_{pos\ (collision)}$).

With $T = \sqrt{\frac{m}{c}}$, $w_0 = \frac{1}{T}$, $d = \frac{r}{2\sqrt{m \cdot c}}$, $K = \frac{1}{c}$

this function leads to the PT_2-element with the following transfer function (continuous time):

$$H_f(s) = K \cdot \frac{w_0^2}{w_0^2 + 2 \cdot d \cdot w_0 \cdot s + s^2}$$

The continuous time filter can be transformed to the discrete time filter $H_f(z)$ with the Tustin-Approximation:

$$s = \frac{2}{T_A} \cdot \frac{z-1}{z+1},$$

$$H_f(z) = K \frac{w_0^2}{w_0^2 + 2 \cdot d \cdot w_0 \cdot \frac{2}{T_A} \cdot \frac{1-z^{-1}}{1+z^{-1}} + \frac{4}{T_A^2} \cdot \frac{(1-z^{-1})^2}{(1+z^{-1})^2}}$$

The parameters used with this filter function can easily adjusted to the dynamic properties of the haptic device, without changes in the haptic rendering algorithm. Whereas the sample time T_A is chosen by the cycle time of the rendering algorithm and haptic device, the sufficient stability can be reached by adjusting the cut-off-frequency w_0 and the damping factor d.

Typically a human can sense kineasthetic impression with a bandwidth of approximately 1kHz [3] and can make controlled movements with a maximal bandwidth of 10 Hz. Studies at the DLR resulted in a bandwidth of 2 Hz in which a human performs a desired task (no reflexes). To perceive realistic kineasthetic impression the dynamic of the force controller (implemented at the device) plays the key role and limits the achievable bandwidth. That means that the force dynamic of the virtual environment can be limited to 5Hz without loosing a realistic impression of the scene.

4. Experiments

The validity will be shown in experimental results using the Phantom T-Model (Fig.14) and the DLR- light weight robot (Fig. 13) as kineasthetic hand controller [7]. In the virtual test bed the static scene consists of a cup, whereas a sphere is used as dynamic object manipulated by the operator (Fig 15).

4.1 Experiments to the Adapted Collision Force Calculation

For a benchmark of the force distractions, the results of the collision force calculations are evaluated in two experiments. The dynamic sphere was moved over the static object with a predefined path and a constant penetration. The experiments differ with respect to the surface conditions of the static object, which typically occur in discrete space models like the voxel map. The surface condition depends on the orientation of the surface plane related to the main axis planes of the voxel grid coordinate system. Force distractions, appearing in sliding motion, primarily influence the collision force direction. For demonstrating this effect, only the force parts vertical to the collision plane normal (the intuitive collision force direction) were recorded. In each experiment this record is done for both the original and the adapted collision force calculations. The direct comparison between them is depicted in the following diagrams, made for each experiment.

Experiment 1: sliding motion over a plane surface
The sliding path of the dynamic object is in this experiment a circle movement over a plane surface of the static object. In Figure 8 all the distraction arrowheads are recorded, by keeping the axis origin at the sphere center position.

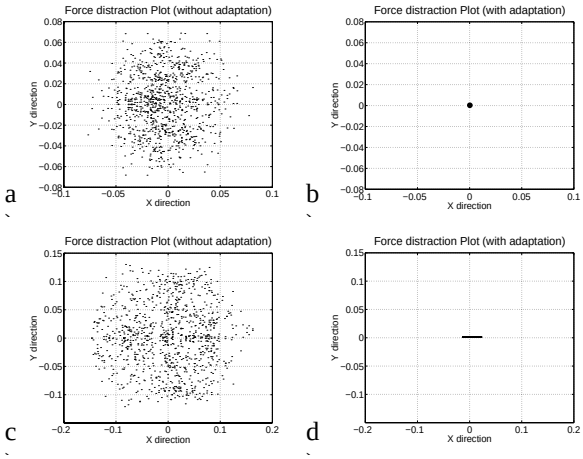

Figure 8: Collision force distractions of experiment 1

In the experiment related to Figure 8 a) and b) the sliding surface of the static object was oriented along the voxel grid main axes, whereas 8 c) and 8 d) presents the results of the motion over a not voxel grid raster oriented plane. With the adapted force calculation the distraction reduction is appreciable in both cases.

Experiment 2: sliding motion over a curved surface
In this experiment the sphere slides on a circle path around the superficies surface of a cylinder, with also a constant penetration depth into the cylinder (Fig. 10). This shows the collision force distraction behavior with many different surface condition cases. Figure 9 a) shows the distractions of the collision force over the time calculated with the original algorithm and Figure 9 b) shows the distractions with the adapted algorithm. Of cause there are still remaining distractions with the adapted form coming from the different surface conditions, but a high reduction of these distractions can be seen with the adapted collision force calculation.

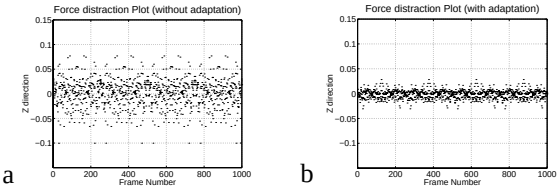

Figure 9: Collision force distractions in one direction of experiment 2

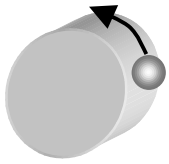

Figure 10: Benchmark for the virtual coupling

4.2 Experiments to the Virtual Coupling

To evaluate the virtual coupling, a sphere is moved over a cylindrical static object with a constant penetration (Fig. 10).

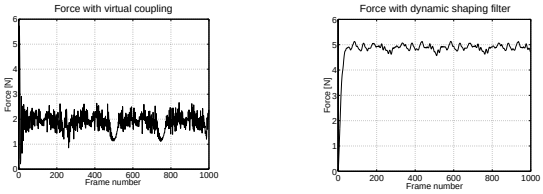

Figure 11: Force plots for different virtual couplings

In Figure 11 shows the results of a heuristic tuned virtual coupling and the dynamic shaping filter explained in chapter 3. In hardware-in-the-loop experiments, the virtual cup (Fig. 15) was touched by the virtual sphere using the DLR Light-Weight robot (LWR) (Fig. 13) and the Phantom T-Model (Fig. 14) as haptic hand controllers. The following diagrams (Fig. 12) shows the collision forces (thin line) calculated by the haptic rendering algorithm and the corresponding output force (thick line), which was returned as feedback force to the hand controllers by the virtual coupling element. Two kinds of collisions were tested, an abrupt collision, hitting the cup with the sphere (Fig. 12 a,c), and a steady collision, sliding with the sphere over the cup surface (Fig. 12 b,d).

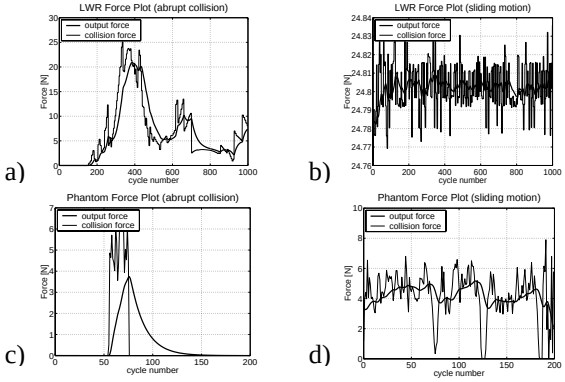

Figure 12: Force plots of the experiments with the Phantom and the LWR

5. Conclusions

In this paper some efforts were done to improve the stability and immersion of haptic feedback. Our approach is based on the VPS method, due to its constant sample rates. First the object modeling was improved towards an exact surface representation. This not only increases the accuracy, but also avoids some force distractions during contact situation with static objects. The force calculation itself was adapted such that the voxel noise, occurring during sliding motions over static objects, is substantially reduced. This leads to a smoother force evolution within the task. Finally, a design framework for virtual coupling was presented. It takes into account the dynamic properties of the human operator and of the hand controller. This approach leads to a more stable haptic interaction with virtual environments as it is shown in the presented experiments.

Further work in the fields of improving the realistic haptic feedback extending the VPS, and of automatic design of the virtual coupling will be done.

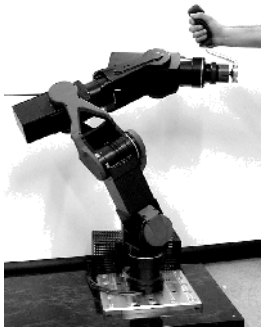

Figure 13: DLR Light-Weight robot as kineasthetic hand controller

Figure 14: Phantom T-Model from Sensable Technologies Inc.

Figure 15: Virtual test bed

6. References

[1] R. J. Adams, B. Hannaford, "Stable Haptic Interaction with Virtual Environments", *Proc. IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, June 1999, pp. 465-474.

[2] K. J. Aström, B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, 3rd ed., Prentice-Hall, Inc., 1997, NJ, ISBN 0-13-314899-8.

[3] G. Burdea, *Force and Touch Feedback for Virtual Reality*, John Wiley Sons, Inc., 1996.

[4] O. Föllinger, *Regelungstechnik: Einführung in die Methoden und ihre Anwendung.* 7th ed., Hüthig, 1992, Heidelberg, ISBN 3-7785-2136-5.

[5] T. Massie, K. Salisbury, "The PHANToM Haptic Interface: A Device for Probing Virtual Objects", *Proc. ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems*, Chicago, IL, November 1994

[6] W. A. McNeely, K. D. Puterbaugh, J. J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proc. ACM SIGGRAPH*, 401-408, 1999.

[7] C. Preusche, R. Koeppe, A. Albu-Schäffer, M. Hähnle, N. Sporer, G. Hirzinger: "Design and Haptic Control of a 6 DoF Force-Feedback Device", In: *Proc. of the 2001 Workshop on Advances in Interactive Multimodal Telepresence*

[8] C.B. Zilles, J.H. Salisbury, "A Constraint Based God-Object Method for Haptic Display", *Proc. IEEE Conf. on Intelligent Robots and Systems*, vol 3, 1995, pages 146-151