A Platform for Bimanual Virtual Assembly Training with Haptic Feedback in Large Multi-Object Environments

Mikel Sagardia*, Thomas Hulin, Katharina Hertkorn, Philipp Kremer, and Simon Schätzle
German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Wessling, Germany

Abstract

We present a virtual reality platform which addresses and integrates some of the currently challenging research topics in the field of virtual assembly: realistic and practical scenarios with several complex geometries, bimanual six-DoF haptic interaction for hands and arms, and intuitive navigation in large workspaces. We put an especial focus on our collision computation framework, which is able to display stiff and stable forces in 1 kHz using a combination of penalty- and constraint-based haptic rendering methods. Interaction with multiple arbitrary geometries is supported in realtime simulations, as well as several interfaces, allowing for collaborative training experiences. Performance results for an exemplary car assembly sequence which show the readiness of the system are provided.

Keywords: haptic rendering, virtual assembly, haptic devices, interaction techniques

Concepts: •**Software and its engineering** → **Virtual worlds training simulations;** •**Human-centered computing** → *Haptic devices;* •**Computing methodologies** → *Virtual reality;*

1 Introduction

In Virtual Assembly (VA) simulations with haptic feedback, users typically select, move and manipulate objects in a synthetic environment by means of a haptic device with the goal of placing and fixing them in their target positions. They not only see the interactions of the manipulated parts, but also feel the collisions through the haptic device. Realtime VA simulations have been for long very appealing to product development and manufacturing industries, since they enable revision and optimization of parts in a faster, more efficient and more economic way [Zorriassatine et al. 2003]. While traditionally difficult to build and expensive real prototypes have been tested before production, in Virtual Environments (VE), engineers can instantaneously check and update their provisional virtual mockups, or even the experience and expertise of assembly line technicians can be incorporated to earlier stages of product design.

The field of VA comprises many research and development areas, starting from basic technical issues like data format conversion, standardization and handling, and continuing with assembly verification [Gomes de Sá and Zachmann 1999], optimal and automatic assembly sequence planning [Sung et al. 2001], natural realtime simulation and interaction [Zachmann and Rettig 2001], evaluation of ergonomic and performance factors [Lim et al. 2007], or training and skill transfer [Gutiérrez et al. 2010].

We focus on the particular topic of interactive multimodal virtual simulations which cover visual and haptic feedback. Haptic feedback has been shown to improve manipulation performances [Garbaya and Zaldivar-Colado 2007], and, in particular, force feedback was reported to be more efficient than other modalities when it comes to build an accurate mental picture of tasks during virtual manipulations [Sagardia et al. 2012].

In this line, we present a system that addresses some of the challenges that current VA systems with haptic interactions need to target, as highlighted in recent surveys [Seth et al. 2011], [Liu et al. 2015]:

- (i) realtime (1 kHz) collision handling with several realistic and non-simplified geometries directly exported from CAD environments,
- (ii) bimanual six-DoF force and tactile interaction for hands and arms via interfaces that try to be as transparent as possible to the user,
- (iii) and intuitive upper body navigation in large workspaces that overcomes the spatial limitation present in many desktop environments.

Our system supports collaboration through several interfaces that can interact simultaneously, allowing the instructor to train the trainee during the same session. Additionally, assembly sequences can be comfortably defined having the CAD parts and their target poses. Exemplary performance results are reported and the attached additional video illustrates them.

2 Related Work

For a realistic and immersive interaction in virtual assembly environments with haptic feedback, (i) **collisions** between objects and assembly features must be similar to the ones in the physical reality, (ii) the employed user **interfaces** must operate synchronized and according to the capabilities of the human senses, and (iii) natural or at least intuitive **interaction** techniques must be provided. In the following, we describe these requirements, focusing on collision computation and display.

Collision computation is considered to be the most expensive task in this process and it consists in, first, detecting or avoiding interferences between geometries, and, second, rendering contact forces upon overlap. Next, those forces can be displayed to the user via haptic interfaces or the motion equations that depend on them can be integrated to obtain the position values of the next time step. Both processes of detecting collisions and computing forces can be resolved simultaneously or sequentially; in any case, collision detection is usually the bottleneck, and it becomes more expensive as the complexity of objects increases (i.e., number of polygons or non-convex shapes). Therefore, still nowadays, many solutions simplify the geometries in the scene in order to achieve realtime rates. Typically, force computation algorithms are classified to be

*e-mail: mikel.sagardia@dlr.de

VRST '16, November 02 - 04, 2016, Garching bei München, Germany
ISBN: ACM 978-1-4503-4491-3/16/11
DOI: http://dx.doi.org/10.1145/2993369.2993386

one of three types according to the principle they are based on. First, the probably computationally less expensive *penalty-based* approaches compute forces using overlap metrics (i.e., penetration) between objects. The Voxelmap-Pointshell (VPS) algorithm [McNeely et al. 1999] is a well known example. Second, the *impulse-based* methods apply velocity changes (or impulses) necessary to avoid the contacts that occur in collision cases, such as in [Mirtich and Canny 1994]. And, finally, usually the most complex *constraint-based* algorithms compute optimum object configurations so that overlapping objects do not intersect. A notable example is the *god object* method in [Ortega et al. 2007].

Virtual coupling [Colgate et al. 1995] is a widely used method to achieve stiff but stable display of forces. With this method, the virtual representation of the device position (haptic handle) is connected to the grasped object (the one visualized) with a spring-damper system. The stiffness of the displayed forces and torques is the one of the coupling.

Another approach to provide collision feedback to the user consists in displaying forces related to selected geometry features that define assembly constraints (i.e., hole axes, support planes, etc.) [Tching et al. 2010]. In general, these methods are computationally less expensive and more accurate when objects are close to the mating configuration, but they require usually more manual definition and lead to less realistic interaction.

Multimodal VEs comprise **interfaces** that target visual, haptic and auditory senses. The simulation and also the interfaces must materialize feedback with the smallest latencies possible and in synchronization with all other modalities in order to cause a sensation of reality and immersion, and avoid cybersickness [LaViola Jr 2000]. In this sense, haptics and its interfaces face up challenging update rates of 1 kH, not only due to the human haptic perception system, but also due to stability issues that arise with slower frequencies [Srinivasan and Basdogan 1997]. Moreover, haptic devices operate in demanding high dynamic ranges: they must be able to move transparently and instantaneously constrain the movement of their motors with the highest torque possible. Additional requirements and concrete values related to the needed human-machine haptic range, resolution and bandwidth are reported in [Tan et al. 1994]. A plethora of haptic interfaces with different targeted properties (desktop devices, bimanual, wearable, tactile, etc.) have appeared in recent years, both experimental and commercial. Reviewing them is out of the scope of this work, but we refer the reader to [Talvas et al. 2014], which surveys bimanual haptic systems.

In strong relation with the user interfaces there are the **interaction techniques** that enable the human-machine communication. According to [Bowman and Hodges 1999], there are four main classes of 3D interaction: (i) navigation, (ii) selection, (iii) manipulation, and (iv) system control. The early work in [Gomes de Sá and Zachmann 1999] presented, for instance, a rich variety of exemplary multimodal interaction techniques. In that work, selection and manipulation was achieved with natural grasping, which was simulated with a virtual hand controlled via a data glove with tactile feedback. As for system control, hand gestures were recognized, natural speech recognition was implemented and the users could interact with overlaid menus.

Several non-commercial **VA systems with haptic feedback** have been presented in the past years. [Seth et al. 2011] and more recently [Liu et al. 2015] have surveyed the field, the latter focussing on physically-based interactions. In the following, we describe some VA platforms and applications that appeared in the last decade. Most of them are desktop-based and many use different versions of the Phantom[1] device for force feedback. Additionally, in many of them, conventional physics engines are used for collision computation, which often force to simplify geometries or keep a moderate complexity in the scene. In our system, on the other hand, upper body movements are supported and the collision detection engine allows for large-scale scenarios with several complex objects (non-convex) composed of millions of triangles.

The MIVAS platform [Wan et al. 2004] is a multimodal VA system with which the users can manipulate complex objects in a CAVE (cave automatic virtual environment) with a virtual hand avatar moved through a CyberGrasp[2] hand exo-skeleton. Collision forces are displayed at the fingers. Manipulations with a virtual hand can lead to very natural interactions, but can also become uncomfortable due to the complexity of the interfaces. Additionally, the system receives voice inputs. SHARP [Seth et al. 2006], on the other hand, is a bimanual virtual assembly simulator that uses two Phantoms and physically-based modeling with the Boeing VPS collision computation algorithm. Their system supports head-mounted displays (HMD) and also a CAVE, and they computed swept volumes for later assembly analysis. Similarly, the work in [Howard and Vance 2007] presents a system where complex parts (which are simplified for collision detection) can be bimanually manipulated with Phantom Omni devices. The authors test the performance of the system while handling some motor parts and in a drop-peg-in-hole scenario when using low- and high-end desktop computers. Also bimanual, IMA-VR [Gutiérrez et al. 2010] is a multimodal platform that focusses on training assembly skills. The dominant hand controls the haptic device (their own developed LIFhAM or a Phantom) and the motion of the non-dominant is captured for gesture commands. Diverse direct and indirect aids are implemented in the simulation to accelerate the learning of the trainee, including tele-mentoring, in which trainers can guide the movement of the apprentice. In [Xia et al. 2011], HVAS is described, a virtual assembly system that implements a hierarchical scene graph divided into several layers, from the assembled product itself to polygons. A Phantom device is used and the authors prove in a user study that their geometry constraints and guidance forces improve performance. HAMS [Gonzalez-Badillo et al. 2014] is another example of bimanual haptic assembly and manufacturing system able to handle manipulation with two Phantoms. The authors perform collision feedback with a mixed approach in which part collision detection and assembly constraints are displayed. Trajectories are visualized with colored spheres which encode movement properties for later analysis. The system is additionally validated with questionnaires after a user study where assemblies of realistic objects such as a bearing puller or a gear oil pump are performed. Bimanual and covering upper body movements, VR-OOS [Sagardia et al. 2015] is a virtual reality system for satellite on-orbit servicing simulations based on the previous version of the setup presented in this work. Interactions are possible with two DLR/KUKA Light Weight Robots transformed as haptic devices. The system focusses on collision detection and physical motion simulation of parts in a space environment. The goal of the authors is to research on systems able to test maintenance scenarios in space and eventually generate (virtual) experience data for astronauts or robotic systems. More recently, VMASS [Al-Ahmari et al. 2016] was presented, a virtual manufacturing assembly simulation system which focuses on the integration of several interfaces and software modules with the goal of providing the most adequate feedback to trainees. The system consists of a powerwall (but supports also HMDs) and objects are manipulated with a Phantom Desktop and a 5DT[3] data glove with vibrotactile feedback.

As mentioned, and now illustrated, most of the VA systems are

[1] http://www.geomagic.com/en/

[2] http://www.cyberglovesystems.com/cybergrasp/

[3] http://www.5dt.com

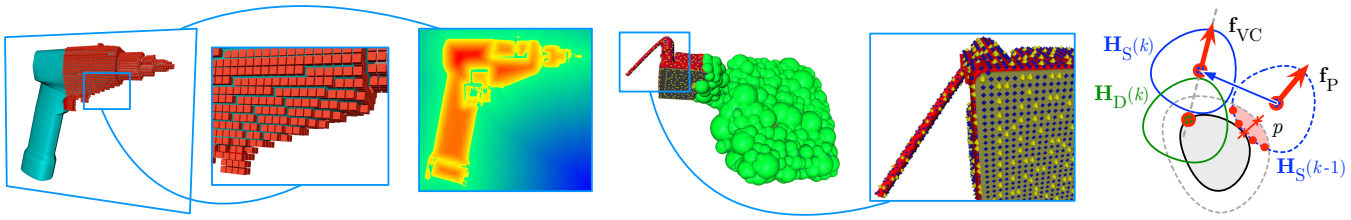

Figure 1: *Point sampled and voxelized representations of a virtual electronic box and a screw driver. (a) Partially voxelized screw driver. (b) Close up of the voxelized screw driver. (c) Sagital section of the voxelized screw driver: distance (turquoise-blue) and penetration (yellow-red) values embedded in the voxelized structure. (d) Point-sphere tree of the electronic control box: one sphere level in green and two successive point levels. (e) Close up of the two last point levels: the blue set contains 4× more points than the yellow, which is also contained in the blue. (f) Constraint-based computation: although the device frame ($\mathbf{H}_D$) is overlapping, only the non-penetrating god object is displayed on the contact surface ($\mathbf{H}_S$); the virtual coupling forces ($\mathbf{f}_{VC}$) are computed out of the difference of the device and surface frames.*

desktop-based and use physics engines that simplify geometries for collision rendering. Yet, real assembly scenarios can be large and consist of multiple complex objects. In that sense, our platform brings up a novel system to the playground that is able to tackle those aforementioned shortcomings. Analyzing the effect of using either a desktop-sized or a larger interface for the presented scenario is certainly a necessary step, as commented in Section 6, but out of the focus of this paper. Nonetheless, we believe that increasing the workspace results in increasing realism.

3 Simulation Framework

This section deals with the different software engines used in order to create the realtime multi-body assembly simulation. We especially focus on the collision computation modules in Section 3.1 and Section 3.2.

3.1 Penalty- and Constraint-Based Collision Computation of an Object Pair

Our core collision computation engine is founded on the *penalty-based* Voxelmap Pointshell (VPS) Algorithm [McNeely et al. 1999], and the force computation is performed with a *constraint-based* approach similar to the *god object* method introduced in [Salisbury and Tarr 1997].

The VPS algorithm is able to render six-DoF (net forces and torques) contact forces between arbitrarily complex geometries in 1 kHz using voxelized representations and point-clouds, as shown in Figure 1. Our VPS implementation was built from the scratch and enhanced with signed distance fields and point-sphere trees, as proposed by several authors [Barbič and James 2008], [Sagardia et al. 2014]. Since each level of the tree samples the whole object, time critical level-of-detail traverses are possible. These data structures are created offline in few seconds from CAD polygonal models, without manual pre-processing. Independently of the tessellation quality, only the vertices of the original model are used for the computation ignoring the normal vectors, which overcomes many issues still present in CAD-VR geometry handling and conversion tools [Liu et al. 2015]. These input vertices are used to re-sample all object surfaces with the desired resolution and to compute automatically correct normals that point inwards.

Unfortunately, the VPS method requires the geometries slightly to overlap, and, therefore, they can pop through thin surfaces. For that reason, a constraint-based force rendering method is used on top of the penalty-based VPS algorithm, as described in [Sagardia and Hulin 2016]. As shown in Figure 1(f), this method computes a *god object* proxy that remains on the surface ($\mathbf{H}_S$); this proxy is the one visualized to the user, not the overlapping device frame ($\mathbf{H}_D$). In order to compute the *god object* pose, the distance field object is artificially dilated (usually two voxel layers are enough). Then, the penetration (p) and penalty forces ($\mathbf{f}_P$, $\mathbf{t}_P$) of the last *god object* configuration ($\mathbf{H}_S(k-1)$) are computed with our VPS implementation. The unconstrained movement of the proxy is the one which tries to go from the last *god object* pose ($\mathbf{H}_S(k-1)$) to the current device pose ($\mathbf{H}_D(k)$). If we constrain that transformation (from $S(k-1)$ to $D(k)$) with the penalty values (p, $\mathbf{f}_P$, $\mathbf{t}_P$), we obtain the current *god object* pose: $\mathbf{H}_S = \mathbf{H}_S(k)$. That is essentially performed by blocking any translation or rotation component that opposes to the penalty forces and torques. Finally, as it is done in virtual coupling approaches, the difference between the invisible device ($\mathbf{H}_D$) and the surface proxy ($\mathbf{H}_S$) frames is used to compute the displayed forces ($\mathbf{f}_{VC}$, $\mathbf{t}_{VC}$). This *god object* heuristic allows for stiff contacts even between thin, non-watertight objects and displays all manipulated objects on the collision surface.

3.2 Multibody Collision Computation Module

While the previous section discussed the collision detection and force computation between two objects, this section describes the module to compute multibody collisions. Given the critical rendering frequency of 1 kHz required in haptics, building a multibody environment that dynamically handles several complex objects is not straightforward. In the following, we introduce the general structure and the workflow of our library, for later explaining the methods used to alleviate the computational effort and deliver contact information for each object in 1 msec.

An overview of our multibody library architecture is visualized in Figure 2. The scenario is described to the system in a configuration file in which, basically, a list of all objects and their properties is specified, such as name, stiffness, haptic data structure filenames, etc. The parser processes the configuration file and loads all necessary entities: a database with all haptic structures, object nodes within the framework and the relation links between the objects. Each object node has its own I/O ports for forces, poses and visualization data. The module receives the poses of the objects and writes their corresponding forces in them. Additionally, there are a collision matrix and configuration state matrix port. The first summarizes in an object vs object table if an object pair is colliding. The second is a user interface for enabling/disabling collision detection between a specific object pair, also implemented in an object vs object table. Objects can be attached to user interfaces, e.g., to

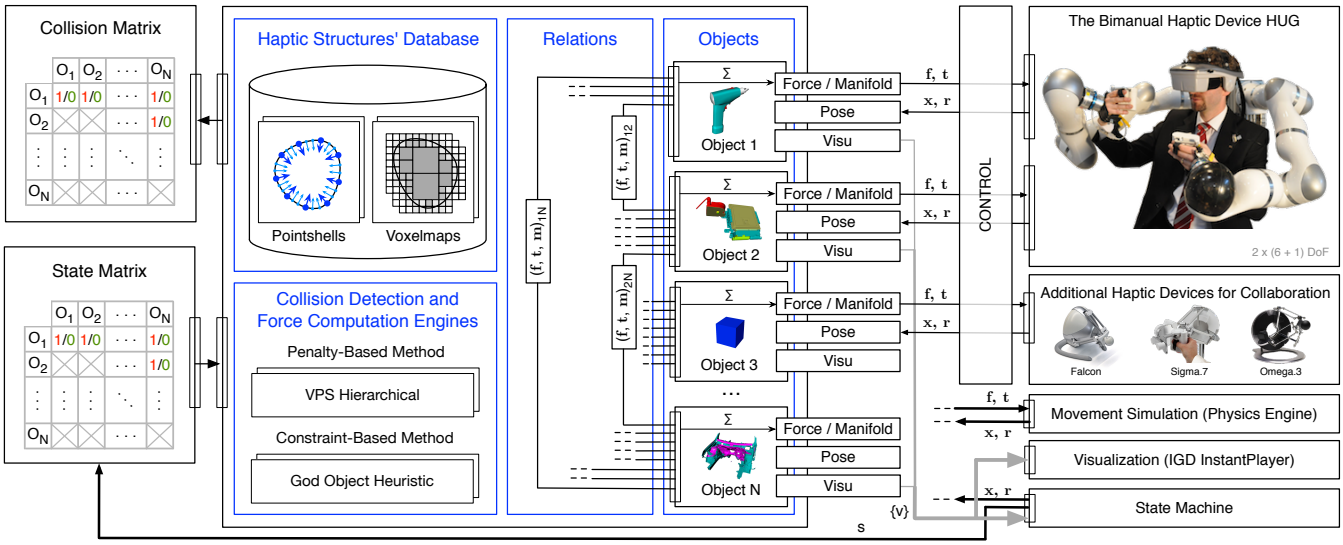

Figure 2: *Overview of the multi-body simulation framework focussing on collision computation. The collision detection module contains a data base of all object in the scene. Several haptic devices or other modules can be connected to the objects or other appropriate input ports of it; for instance, a physics engine or a game state machine can define the movement of an object. Our framework uses InstantPlayer as visualization tool, but other scene graphs could be connected as well. The system is easily scalable. The images of the Omega.3 and the Sigma.7 are courtesy of Force Dimension, Switzerland.*

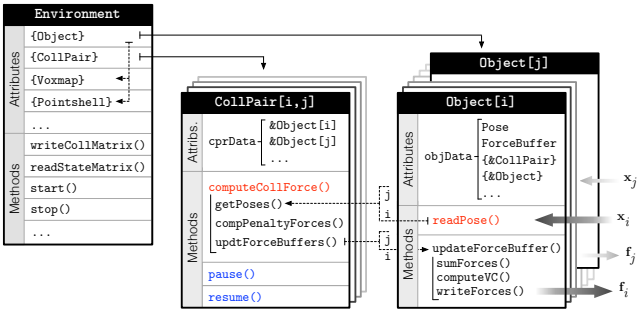

Figure 3: *Multibody library architecture. The class* `Environment` *contains and supervises the states of all* `Objects` *and* `CollPairs`*. Each* `Object` *contains both haptic structures that represent it.* `CollPairs` *call the collision computation method that checks the contacts between the* `Objects` *they relate.*

the bimanual haptic device HUG described in Section 4.1, or to a physics engine that integrates the collision forces to obtain object poses.The communication between the collision computation module and the devices is realized using thread-safe shared memories and the UDP communication protocol.

In the following, we describe the architecture of the C++ implementation, which has three main classes, as shown in Figure 3:

1. Instantiations `Object[i]` of class `Object` represent a body i in the scenario. The class contains the ports for pose ($\mathbf{x}_i$) and force ($\mathbf{f}_i$) data available to the user and methods (e.g., `readPose()`) associated to them. `Objects[i]` contain both voxelized and point-sampled representations and use the appropriate one for each situation.
2. The instance `CollPair[i,j]` of the class `CollPair` uses the pose information ($\mathbf{x}_i$ and $\mathbf{x}_j$) from its associated `Objects` in order to compute forces according to the algorithms introduced in Section 3.1. Each of the instantiations of `CollPair` is related to a cell in the collision matrix.
3. The class `Environment` contains arrays of `Objects` and `CollPairs`, all interconnected. In `Environment`, the collision matrix is written and the configuration state matrix read. All data structures can be accessed by the user via this class, and therefore, the class also has methods to `start()` and `stop()` the functions of `Objects` and `CollPairs`.

Our architecture resembles a graph, where `Objects` are nodes that store poses and forces and the `CollPairs` are links that generate force data in dedicated threads after reading poses. The system is asynchronous and the usual workflow is the following (see Figure 3):

(i) The callback thread function `computeCollForce()` from `CollPair[i,j]` is continuously executed. First, it calls all corresponding `readPose()` in order to update pose data: $\mathbf{x}_i$ from `Object[i]` and $\mathbf{x}_j$ from `Object[j]`.

(ii) Next, penalty force computation is carried out as explained in Section 3.1, which yields $\mathbf{f}_{ij}$ and its corresponding contact manifold.

(iii) After that, the thread function passes the penalty force to all related objects via `updateForceBuffer()`. This last method performs several operations on the `ForceBuffer` before sending the total force to the user. The `ForceBuffer` is an array that contains a force cell for each algorithm that is using the `Object`. The total penalty force upon the object is the sum of all forces in the `ForceBuffer`.

(iv) If the module decides to send the force to the user, first, all penalty forces are summed. Different sending policies are supported: for example, we could prefer to send total force values every time a new force arrives, or every time the cell with the oldest update time stamp is refreshed. We have not experienced any issues due to lack of synchronization yet. If

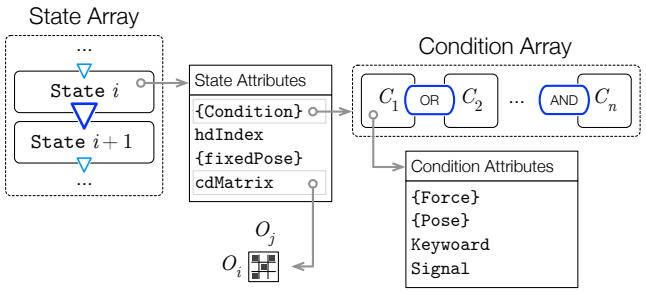

Figure 4: *Game control workflow. Each state dictates which object is moved by the user (`hdIndex`) and specifies the position of the other ones (either free or fixed, {`fixedPose`}). In order to jump to the next state, a series of conditions or tasks related to force or pose values must be fulfilled.*

that were the case, the `ForceBuffer` would have to be extended, for instance, to extrapolate force values with history data every time a force summation needs to be delivered. We leave this analysis for future work.

(v) Next, if the current object is held by the user, the *constraint-based* force rendering explained in Section 3.1 [Sagardia and Hulin 2016] is executed with the penalty force summation and the biggest penetration value.

(vi) Finally, force and visualization data are sent to their corresponding object ports.

In those large-scale environments with multiple objects, one of the main challenges consists in dealing with the quadratic nature of collision detection. Several strategies have been proposed to reduce the computational complexity, such as sweep and prune of pairs using bound boxes [Cohen et al. 1995]. To cope with multi-body environments while keeping the 1 kHz frequency required by haptics, we exploit these techniques: (i) *object grouping* and parallelization, (ii) *spatio-temporal coherence* [McNeely et al. 2006], and (iii) *graceful degradation* [Barbič and James 2008].

The first one, *object grouping* and parallelization, consists in taking advantage of the multiple cores of modern CPUs. As previously introduced, `computeCollForce()` (see Figure 3) is an interface that can run in a separate thread for each object pair. Higher parallelization degrees are desirable for scenarios with few objects (less than five) and multi-core CPUs, whereas the user should serialize collision computation calls in case the number of geometries increases, due to the thread overload that might be incurred (that degree is currently a parameter in the configuration file). Additionally, objects can be packed in groups in the configuration file so that all objects from the same group are not checked for collision between them. A practical use of that is the kinematic chain formed by the hand and the arm: collisions between the different limbs can be neglected by packing them into the same group.

Secondly, we use the *spatio-temporal coherence* property of multi-body assembly environments before calling `computePenaltyForces()`; the notion behind it is that the distance between two objects is very similar in two consecutive time steps. Since our *penalty-based* approach yields the signed distance p between objects, assuming these are allowed to move no faster than $v_{\max} = 1\,\mathrm{m/s}$, we know two objects will not collide during the period $p/v_{\max}$. This saves in practice hundreds of collision detection calls in a second even in scenarios where objects are relatively close to each other. It is worth to mention that the value of $v_{\max}$ may have to be adjusted for elongated objects. In that case, even small rotations might produce high velocities in the extremities. Nevertheless, the selected value has sufficed in our scenarios.

And finally, the third method coping with expensive computations consists in *graceful degradation*, which is possible due to the multi-resolution nature of the point-sphere trees. The volume and surface of all objects is evaluated at the beginning and resource percentages for worst-case scenarios (e.g., when all objects are in contact) are assigned. Additionally, load and efficiency factors are computed online in `computePenaltyForces()`, i.e., the number of point and sphere collision checks and which of them yield positive. Under worst-case conditions or high loads, pointshells can automatically limit the number of levels allowed in the hierarchy traverse. Since each level samples the whole object, an approximate but valid result is computed by the algorithm. Force values are scaled with the number of colliding points and point densities, hence, we can guarantee that the relative order of magnitude is maintained. A practical example of that is given in Section 5.1.

3.3 Game Control

In order to enable assembly sequences, we developed a finite machine that externally attaches to the multibody collision computation module explained in Section 3.2. The user can easily extend the configuration file of the multibody collision detection module specifying which objects have to be mounted in which position and in which order. As shown in Figure 4, this is implemented in an array of *states* and, for each *state*, an array of *conditions*.

A state establishes which objects are moved by the user(s) (`hdIndex`). Additionally, objects can have a `fixedPose` (e.g., mounted) or can have no assigned pose, and therefore be waiting for a external pose data. Each state can also update the collision detection state matrix explained in Section 3.2. This allows for dynamically managing resources; e.g., when an object is mounted, it should not be checked for collision against other mounted pieces, thus, the collision calls between them should be paused.

The condition array is a set of tasks that must be carried out. Usually, each condition is related to target `Pose[j]` and force `Force[j]` values expected on the object `j` that has to be assembled (including tolerances). However, the machine can also receive `Keyboard` commands, such as `"fulfilled"`, or external `Signals` (e.g., from other interfaces, gestures, etc.). Conditions are concatenated with `AND|OR` operators and the user can decide if they need to occur simultaneously or not. When all conditions are fulfilled, an exit event occurs and the machine jumps to the next state.

Simple yet powerful, our game control machine handles the simulation and takes all logic decisions necessary to accomplish assembly sequences.

3.4 Completing the Jigsaw Puzzle

As shown in Figure 2, we use as visualization tool the InstantPlayer[4].We preferably display the scene to the user with a nVisor SX60[5] head-mounted display (HMD), which is optically tracked by a Vicon Bonita[6] system. Additionally, the Bullet[7] physics engine has been successfully tested. Our framework operates in a distributed manner, having for each module a dedicated desktop computer. The whole system is controlled with an experimental process

[4]http://www.instantreality.org
[5]http://www.nvisinc.com
[6]http://www.vicon.com/products/camera-systems/bonita
[7]http://bulletphysics.org/wordpress/

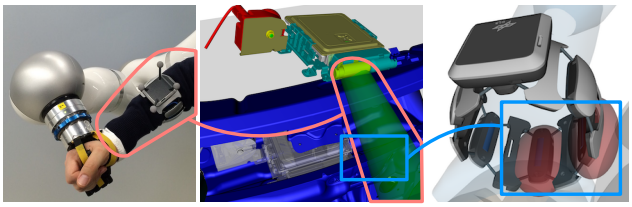

Figure 5: *The VibroTac provides tactile feedback on the forearm using six vibration motors equally distributed. The markers on the device are optically tracked (left). The movement of the elbow is mapped to its virtual representation (middle, in green). If the virtual forearm collides against the scenario (middle), the corresponding segments of the VibroTac provide vibration feedback to the user (right, in red).*

manager middleware which is additionally able to communicate in realtime with robotic interfaces.

4 Interaction Devices and Techniques

This section describes the interaction devices and techniques as key elements of the training platform. Our goal is to provide realistic haptic feedback to the hand and the arm while maintaining the safety and increasing the usability (i.e., intuitive experience, minimal restriction of movements wrt. dynamics and workspace).

4.1 The Bimanual Haptic Device HUG

In order to promote the training effect in assembly simulations and to create an immersive experience for the human operator, a haptic device is required that provides both a large workspace to enable unrestricted movements of the human hand and appropriate force capabilities to generate realistic haptic feedback. For this reason, the bimanual haptic device HUG was chosen in the present application [Hulin et al. 2011]. HUG is equipped with two DLR/KUKA light-weight robot arms[8] and an additional force-torque sensor at each robot wrist (see Figure 2 top right). Each robot has seven revolute joints with position and torque sensors, and the electronics integrated in them, running at 3 kHz. Although six joints would be enough for displaying six-DoF force feedback, the seventh is useful for optimizing the robot configuration. The torque-controlled robots (impedance control is used) are mounted behind the operator's upper body in order to maximize the usable workspace of the human arm [Zacharias et al. 2010]. Each arm can reach up to 150 N peak forces. Although the haptic rendering algorithms exposed in Section 3.1 could generate these force values, 50 N are rarely exceeded in a normal session. Additionally, the user is attached to the device with a magnetic clutch that is decoupled stopping the robots if forces greater than 80 N are applied in any direction. HUG supports a variety of hand interfaces that range from passive data gloves to active gripping-force devices, that can be changed depending on the application. In our version presented here, the users operate with the bare hand, but new hand interfaces are contemplated as future work.

4.2 The Vibrotactile Arm Band VibroTac

With the haptic feedback applied by HUG on the human hand, the user is able to perceive the collision of a manipulated object in a virtual scene. However, in certain situations, it is decisive to know about collisions of the human operator with the virtual environment,

[8] http://www.kuka-lbr-iiwa.com

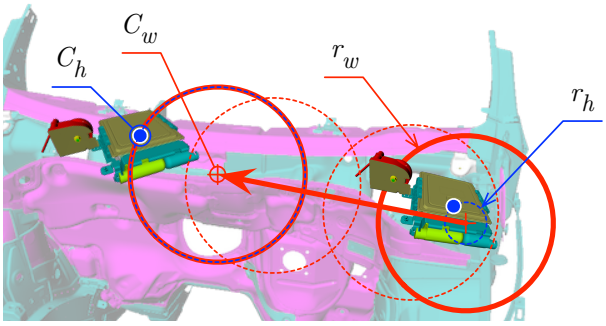

Figure 6: *Scenarios which are bigger than the device workspace require indexing. This is achieved by moving the virtual workspace (red sphere with center C_w and radius r_w) to the grasping point C_h which is outside the boundary.*

e.g., to validate if there is enough space for the human arm inside a motor compartment. This problem can be treated by the usage of additional haptic devices. Arm-exoskeletons may pose the most realistic kind of feedback to the human arm by mechanically coupling the arm with the device.

Due to the lack of available mechanical solutions, however, we use an alternative solution to generate this additional feedback to the operator's forearm. Instead of force feedback, we employ tactile feedback provided by a vibrotactile arm band. The used tactile feedback device VibroTac [Schätzle et al. 2010] is a battery driven and wireless controllable wrist band with six vibration motors (so-called tactors). Due to the different tactor locations, the location and direction of an impact can be indicated to the user's forearm. The strength of collision can be displayed by varying the intensity of the stimuli or by generating different vibration patterns, which can be controlled separately for each tactor.

As shown in Figure 5, we added reflective markers to the VibroTac in order to track it optically. The user sees a green transparent arm in the virtual scene; the wrist is coupled to the end effector of the HUG, whereas the elbow corresponds to the movement of the VibroTac. The HUG can provide six-DoF collision feedback at the palm and the VibroTac extends the touch sensation to the forearm: when the virtual arm collides with a certain force against the objects in the scenario, the corresponding segments are activated with the intensity suited to this virtual force. As a result, it is possible to evaluate and train assemblies where the whole forearm configuration with respect to the mounted part has to be considered.

4.3 Workspace Navigation

Interacting with virtual environments larger than the workspace of the interface requires movement mapping and control strategies. Conti and Khatib give an overview of those methods [Conti and Khatib 2005] and propose a *workspace drifting control* scheme with which the virtual workspace is shifted with a velocity controlled by the distance from the device workspace origin to the position of the endeffector. As these authors report, the most common navigation approach consists in performing *position control* with a selected *scaling* factor, or *indexing* the position of the end effector with respect to the moved avatar after the interface has been decoupled from the simulation (also known as *de-/clutching*). It is also possible to carry out *ballistic control* [Salisbury et al. 2004], with which the scaling factor depends on the velocity of the end effector, or *rate control*, where the velocity of the avatar or moved object increases lineally with the distance from avatar to workspace origin.

[Dominjon et al. 2006] tested some of these methods in a user study and concluded that their *bubble* technique yielded best performance results in a three-DoF painting task that required precise activities in large environments. The *bubble* technique is a hybrid *position/rate control* approach where the optimal workspace around the endeffector (a bubble or sphere) is displayed both visually and haptically to the user. Whenever the boundaries of the bubble are reached, it is moved with a velocity cubically proportional to the trespassed distance and the user feels restoring forces in the direction opposite to the movement.

As reported in [Pavlik and Vance 2011], workspace visualization can become distracting in complex six-DoF assemblies, and restoring forces can lead to the perception of artifacts or they can make the discrimination of collision forces difficult. Therefore, in our implementation we proceed similarly as in the *bubble* technique, but without displaying the workspace visually or haptically. The user experiences two modi with the haptic interface: (i) fast controllable *navigation* in order to relocate the workspace (*rate control*) and (ii) *manipulation* with 1:1 mapping (unscaled *position control*).

As shown in Figure 6, we abstract the workspace with a sphere with center C_w and a constant radius $r_w = 0.2\,\text{m}$. Additionally, we call here r_h the distance between the grasping point C_h related to the object moved by the haptic interface and the current workspace center. This workspace center C_w is considered the anchor frame with respect to the device movements, i.e., the center of the real workspace of the user is mapped to this point and the moved object is transformed with respect to it.

Our workspace shifting works as follows: if the grasping point of the moved object is inside the workspace ($r_h \leq r_w$), the workspace will not move ($C_w(t+\Delta t) = C_w(t)$). Otherwise ($r_h > r_w$), and in absence of collision forces, the workspace is translated towards the grasping point, as summarized in (1):

$$C_w(t+\Delta t) \quad = \quad C_w(t) + \lambda_w \Delta t \big(1 - \tfrac{r_w}{r_h}\big) \big(\underbrace{C_h(t) - C_w(t)}_{\text{vector length } r_h(t)} \big), \qquad (1)$$

where $\lambda_w = 1.5\,\text{s}^{-1}$ is the gain and Δt the time step between two consecutive cycles. Our method moves the workspace in the scenario if the grasped object is outside this workspace and the moved objects are not colliding; the velocity of the translation is linear to the distance from the grasping point to the surface of the workspace.

4.4 Collaboration with Additional Haptic Interfaces

Shared and collaborative virtual environments with haptic feedback are challenging due to the user-user interaction, the required synchronization, and their delay and frequency demands. As mentioned in Section 2, a collaborative tele-mentoring functionality is presented in [Gutiérrez et al. 2010]. In it, a trainer can teach a trainee how to perform specific assembly tasks: position and force data are sent unidirectionally between them (usually from the tutor to the apprentice) so that skills can be transferred faster and interactively. On the other hand, a peer-to-peer collaborative framework for assembly simulations was presented in [Iglesias et al. 2006]. The work focuses on the technical strategies required to maintain synchronicity for each peer simulation. The consistency is achieved basically by keeping track of the last valid (non-colliding) pose of object.

Since our framework is modularly built and the objects of the collision computation database transfer pose and contact information independently, the system allows for collaboration between users

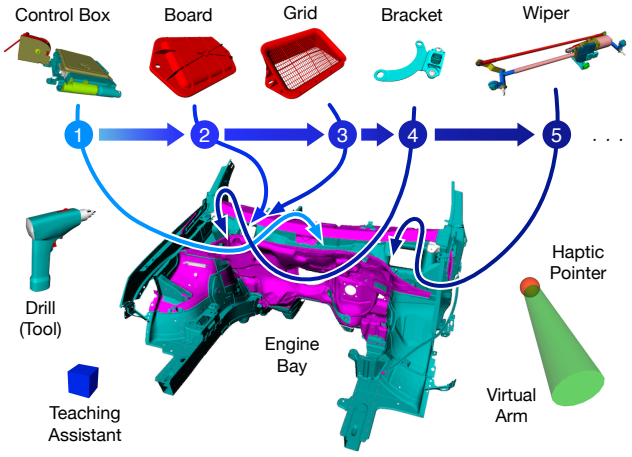

Figure 7: *Schematic scenario description and assembly sequence. The user has to mount five parts into a car engine bay in a row; instructions are given in the upper left corner of the display. When no object has been grabbed, the user sees a red sphere at the right hand (haptic pointer); the left hand can carry tools, such as the drill illustrated in the picture. A teaching assistant box can be controlled by another interface.*

by simply connecting another haptic (or other) interface to an object node, as it is shown in Figure 2. In our current implementation we tested a Falcon[9] and a Sigma.7[10] successfully. The interaction occurs through an extra pointer in the scene which is moved by the second user, called *teaching assistant* (blue cube in Figure 7). The teacher can show the trainee how to move to perform the assemblies and correct wrong movements with small pushes. Both users see and feel collisions. Synchronicity or consistency is achieved with the `ForceBuffer` described in Section 3.2, as if the *teaching assistant* pointer were another object in the scene.

5 Exemplary Scenario: Car Assembly Sequence

This section introduces a practical use case in which all methods explained in previous sections are used.[11] As shown in Figure 7, the virtual scene consists of ten objects:

- (i) a car engine bay where several objects can be mounted,
- (ii) a haptic pointer (red ball) symbolizing the right hand coupled to the right forearm of the HUG (Section 4.1) with which the user can grab an object to mount it,
- (iii) a virtual forearm (transparent and green) moved with the tracked VibroTac (Section 4.2),
- (iv) five selected car parts that have to be assembled in a specified sequence: a control box, a covering board, a grid, a bracket, and a wiper mechanism,
- (v) an electric drill held by the left arm,
- (vi) and a teaching assistant cube moved by another user with a different interface than the HUG. Thanks to this object, the experienced user can haptically correct the trials of novice operators online.

[9] http://www.novint.com/index.php/novintfalcon

[10] http://www.forcedimension.com/products/sigma-7/overview

[11] The complementary video shows the scenario, the interaction techniques and the introduced tasks.

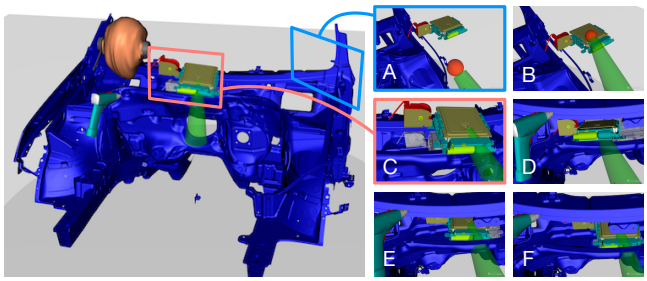

Figure 8: *Left: Snapshot of the whole scenario; user head is displayed by an avatar. Right: Assembly steps of the control box as seen by the user through the head mounted display moved by the head and optically tracked: (A) Approach to the control box to be assembled with the haptic pointer; (B) Touch and grab the control box; (C) Move (indexing) close to the place where the object has to be mounted (displayed with a grey transparent replica); (D) and (E) Insert the control box to the engine cavity; (F) Reach target pose. The haptic pointer appears and the user has to keep on with the next object.*

All objects are real CAD parts and the interaction occurs without scaling (e.g., 1:1). It is worth to mention that the user receives only vibrotactile feedback (not force feedback) when the green transparent virtual arm collides. Therefore, the virtual arm might overlap with the other objects if the user does not actively re-configure his arm as the vibrations indicate.

The scenario is easily extensible to different or more objects by modifying the configuration scripts of the multibody framework (see Section 3.2) and the state machine (see Section 3.3). The assembly workflow of the control box is shown on the right side of Figure 8, and it is similar to all other objects to be assembled. First, the part to be mounted appears on the right and the user has to navigate to it. The navigation takes place thanks to the indexing explained in Section 4.3. If the user remains in contact 1 s with the object to be mounted, the red haptic pointer is replaced by it. Next, the user can navigate towards the region where a grey transparent replica of the part is shown. The following steps consist in assembling the part into its correct location. If this is achieved within the translation and rotation deviation tolerance specified in the configuration file, the part is fixed in its assembly pose and the red haptic pointer appears again. Then, after testing the space for the tool held with the non-dominant hand, the user has to keep on with the next part. Instructions are displayed in a text box on the left upper corner of the simulation frame.

A previous version of our system has also been tested with a virtual satellite maintenance scenario [Sagardia et al. 2015]. Compared to that setting, the use case discussed in the present work focusses on multibody interactions with more complex and realistic objects; additionally, we describe and validate the implementation of techniques that make possible interacting in large virtual assembly training environments (e.g., navigation, collaboration, and game control).

5.1 Performance Results

The evaluation of the assembly process of the control box presented in Figure 8 is shown in this section. Force, penetration, load and computation time diagrams of the results as well as the details of the used data structures are reported in Figure 9.

During the first 12 s, the user is in navigation mode (see Section 4.3) and shifts the workspace center to the desired position. Around frame D, the first collision occurs and the position is corrected again (13.2 s – 17.1 s), for finally proceeding with the insertion into the assembly cavity (17.1 s – end, frames E and F). As expected, it is in those collision situations when the load and the computation time increase considerably.

The load value is computed every cycle and consists of the sum of two load ratios related to the amount of spheres and points:

$$\eta = 100(\omega_S \frac{N_S^V}{N_S^T} + \omega_P \frac{N_P^C}{N_P^V}), \tag{2}$$

where

N_S^V is the number of spheres visited or checked for collision,
N_S^T the total number of spheres,
N_P^C the number of colliding points,
N_P^V the number of points visited or checked for collision (which are inside the colliding spheres),

and $\omega_S = \omega_P = 0.5$ are the weighting factors chosen for the sphere and point loads, respectively.

As mentioned in Section 2, the number of levels swept in the pointshell can be decreased still checking all collision regions if the *critical load* is reached. This pointshell specific *critical load* can also be updated during runtime depending on the number of collision threads that are active and their respective loads. In the shown assembly experiment part, only three collision threads were active (the ones between the engine bay, the drill and the control box), and the maximum achieved load $\eta_{\max} = 18.89\,\%$ (at time stamp 25.86 s) was smaller than the critical $\eta_{\text{crit}} = 70\,\%$. Positively and strongly correlated to the load is the required computation time, which achieves a peak of 0.75 msec around simulation time stamp 27.07 s, below the 1 msec convention for stable and realistic haptic interaction.

Force magnitude oscillates below 4 N (maximum 3.88 N at time stamp 22.97 s) depending on the penetration of the colliding points and the reaction force applied by the user. The values shown in Figure 9 are the raw values computed by the collision computation module before sending them to the haptic device. Depending on the interface, filtering or similar control algorithms can be applied. Note, additionally, that the penetration values are smaller than 5 mm or even 4 mm most of the time. This penalty value is the necessary error between the displayed god object and the virtual object attached to the haptic device.

Overall, we consider that the results show a system able to render realtime assembly simulations with haptic feedback even in large scenarios with numerous complex geometries exported directly from CAD frameworks. The video attached to this paper further shows how the system behaves.

6 Conclusions and Perspectives

We presented a virtual assembly training platform that supports bimanual haptic interactions with several arbitrarily complex objects simultaneously. We put an especial focus on our collision detection engine based on a combination of penalty- and constraint-based force rendering methods. The framework is able to handle scenarios with multiple geometries imported from CAD environments. Additionally, in contrast to usual desktop systems, unscaled large upper body movements can be performed and force and tactile feedback is provided to hand and forearm, respectively. We adapted some intuitive navigation methods from the literature based in hybrid position/rate control for a more efficient interaction in large realistic

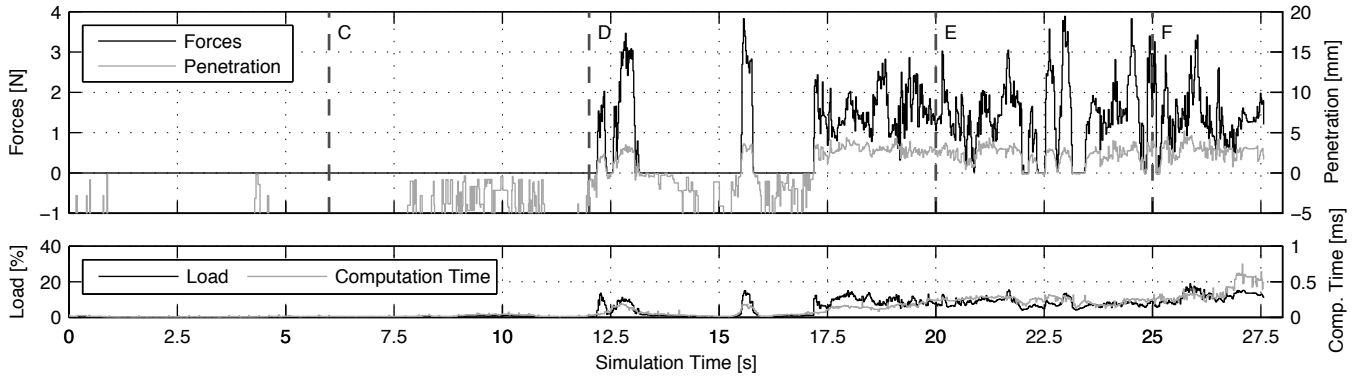

Figure 9: *Force, penetration, load and computation time values obtained during the assembly of the control box displayed in Figure 9. Assembly steps C – F are marked in the upper diagram. Negative penetration values correspond to the approximate distance values used during spatio-temporal coherence computation, which are unsteadier since they correspond to sphere distances. However, these data are scaled to show penetration and force values, because they are considered to be the most relevant in assembly simulations. The car's engine bay has a voxel edge of 3 mm, resulting in a grid of* $1022 \times 788 \times 541$ *elements compacted in 286 MB. It was computed from the original model consisting of 1,83M of triangles in few minutes. The assembled control box consists of 10507 points divided in 3506 clusters and 7 levels compacted in 875 KB (see Figure 1). It was computed from the original model consisting of 101472 of triangles in few seconds. The offline computation time required for data generation depends on the resolution and selected number of layers. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Suse SLED 11 (not realtime).*

car assembly scenarios. Performance results that validate the system are also provided.

Our future work contemplates following main topics, based in part on the current limitations of our system:

- User evaluation of the presented system in comparison with desktop-based systems. We plan to test the car assembly scenario presented in Section 5 and more abstract ones with the HUG and a Sigma.7 in order to measure user and device performance.
- More realistic and natural bimanual interaction. Currently, the dominant hand grabs the parts to assemble and the non-dominant one holds the tool in order to check whether there is enough space for it. In our future system, both hands will grab and assemble parts; after that, users will have to fix them with the tool held with the dominant hand and, for example, using the non-dominant one as support.
- Integration of data gloves compatible with the HUG. Current hand and forearm are simplified to two objects. Our plan is to include both complete mechanical chains of the arms and hands, and to provide them with haptic feedback.

Acknowledgements

We are thankful to Volkswagen AG for providing us the geometries used in this work. The images of the Omega.3 and the Sigma.7 in Figure 2 are courtesy of Force Dimension, Switzerland.

References

AL-AHMARI, A. M., ABIDI, M. H., AHMAD, A., AND DARMOUL, S. 2016. Development of a virtual manufacturing assembly simulation system. *Advances in Mechanical Engineering 8*, 3.

BARBIČ, J., AND JAMES, D. L. 2008. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Trans. on Haptics 1*, 1, 39 –52.

BOWMAN, D. A., AND HODGES, L. F. 1999. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *J. of Visual Languages & Computing 10*, 1, 37–53.

COHEN, J. D., LIN, M. C., MANOCHA, D., AND PONAMGI, M. 1995. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of ACM Interactive 3D Graphics Conference*, ACM, 189–ff.

COLGATE, J. E., STANLEY, M. C., AND BROWN, J. M. 1995. Issues in the haptic display of tool use. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, vol. 3, 140–145.

CONTI, F., AND KHATIB, O. 2005. Spanning large workspaces using small haptic devices. In *Proc. IEEE World Haptics Conference*, IEEE, 183–188.

DOMINJON, L., ANATOLE, L., BURKHARDT, J.-M., AND SIMON, R. 2006. A Comparison of Three Techniques to Interact in Large Virtual Environments Using Haptic Devices with Limited Workspace. *J. of Material Forming 4035*, 288–299.

GARBAYA, S., AND ZALDIVAR-COLADO, U. 2007. The affect of contact force sensations on user performance in virtual assembly tasks. *Virtual Reality 11*, 4, 287–299.

GOMES DE SÁ, A., AND ZACHMANN, G. 1999. Virtual reality as a tool for verification of assembly and maintenance processes. *Computers and Graphics 23*, 3, 389–403.

GONZALEZ-BADILLO, G., MEDELLIN-CASTILLO, H., LIM, T., RITCHIE, J., AND GARBAYA, S. 2014. The development of a physics and constraint-based haptic virtual assembly system. *Assembly Automation 34*, 1, 41–55.

GUTIÉRREZ, T., RODRÍGUEZ, J., VELAZ, Y., CASADO, S., SUESCUN, A., AND SÁNCHEZ, E. J. 2010. Ima-vr: a multimodal virtual training system for skills transfer in industrial maintenance and assembly tasks. In *Proc. IEEE Int. Symp. on Robots and Human Interactive Communications (ROMAN)*, IEEE, 428–433.

HOWARD, B. M., AND VANCE, J. M. 2007. Desktop haptic virtual assembly using physically based modelling. *Virtual Reality 11*, 4, 207–215.

HULIN, T., HERTKORN, K., KREMER, P., SCHÄTZLE, S., ARTIGAS, J., SAGARDIA, M., ZACHARIAS, F., AND PREUSCHE, C. 2011. The dlr bimanual haptic device with optimized workspace (video). In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, 3441–3442.

IGLESIAS, R., CASADO, S., GUTIERREZ, T., GARCIA-ALONSO, A., YAP, K. M., YU, W., AND MARSHALL, A. 2006. A peer-to-peer architecture for collaborative haptic assembly. In *Proc. IEEE Int. Symp. on Distributed Simulation and Real-Time Applications*, IEEE, 25–34.

LAVIOLA JR, J. J. 2000. A discussion of cybersickness in virtual environments. *ACM SIGCHI Bulletin 32*, 1, 47–56.

LIM, T., RITCHIE, J. M., DEWAR, R. G., CORNEY, J. R., WILKINSON, P., CALIS, M., DESMULLIEZ, M., AND FANG, J.-J. 2007. Factors affecting user performance in haptic assembly. *Virtual Reality 11*, 4, 241–252.

LIU, K., YIN, X., FAN, X., AND HE, Q. 2015. Virtual assembly with physical information: a review. *Assembly Automation 35*, 3, 206–220.

MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. ACM SIGGRAPH*, ACM, 401–408.

MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. 2006. Voxel-based 6-dof haptic rendering improvements. *Haptics-e: The Electronic Journal of Haptics Research 3*, 7, 1–12.

MIRTICH, B., AND CANNY, J. 1994. Impulse-based dynamic simulation. Tech. rep., University of California at Berkeley.

ORTEGA, M., REDON, S., AND COQUILLART, S. 2007. A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Trans. on Visualization and Computer Graphics 13*, 3, 458–469.

PAVLIK, R. A., AND VANCE, J. M. 2011. Expanding haptic workspace for coupled-object manipulation. In *Proc. World Conf. on Innovative Virtual Reality (ASME)*, American Society of Mechanical Engineers (ASME), 293–299.

SAGARDIA, M., AND HULIN, T. 2016. A fast and robust six-dof god object heuristic for haptic rendering of complex models with friction. In *Proc. ACM Symp. on Virtual Reality and Software Techonology (VRST)*, ACM. (Accepted and pending for publication).

SAGARDIA, M., WEBER, B., HULIN, T., PREUSCHE, C., AND HIRZINGER, G. 2012. Evaluation of visual and force feedback in virtual assembly verifications. In *Proc. IEEE Virtual Reality (VR)*, IEEE, 23–26.

SAGARDIA, M., STOURAITIS, T., AND E SILVA, J. L. 2014. A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation. In *EuroVR: Conf. and Exhibition of the European Association of Virtual and Augmented Reality*, Eurographics Association, 65–76.

SAGARDIA, M., HERTKORN, K., HULIN, T., SCHATZLE, S., WOLFF, R., HUMMEL, J., DODIYA, J., AND GERNDT, A. 2015. VR-OOS: The DLR's virtual reality simulator for telerobotic on-orbit servicing with haptic feedback. In *Proc. IEEE Aerospace Conf.*, 1–17.

SALISBURY, K., AND TARR, C. 1997. Haptic rendering of surfaces defined by implicit functions. In *Proc. Annual ASME Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, vol. 61, 61–67.

SALISBURY, K., CONTI, F., AND BARBAGLI, F. 2004. Haptic rendering: Introductory concepts. *IEEE Computer Graphics and Applications 24*, 2, 24–32.

SCHÄTZLE, S., ENDE, T., WUESTHOFF, T., AND PREUSCHE, C. 2010. VibroTac: an ergonomic and versatile usable vibrotactile feedback device. In *Proc. IEEE Int. Symp. on Robots and Human Interactive Communications (ROMAN)*, 705–710.

SETH, A., SU, H.-J., AND VANCE, J. M. 2006. Sharp: a system for haptic assembly and realistic prototyping. In *Proc. ASME Int. Design Engineering Technical Conf. and Computers and Information in Engineering Conf.*, American Society of Mechanical Engineers (ASME), 905–912.

SETH, A., VANCE, J. M., AND OLIVER, J. H. 2011. Virtual reality for assembly methods prototyping: a review. *Virtual Reality 15*, 1, 5–20.

SRINIVASAN, M. A., AND BASDOGAN, C. 1997. Haptics in virtual environments: Taxonomy, research status, and challenges. *Computers & Graphics 21*, 4, 393–404.

SUNG, R. C., CORNEY, J. R., AND CLARK, D. E. 2001. Automatic assembly feature recognition and disassembly sequence generation. *Journal of Computing and Information Science in Engineering 1*, 4, 291–299.

TALVAS, A., MARCHAL, M., AND LECUYER, A. 2014. A survey on bimanual haptic interaction. *IEEE Trans. on Haptics 7*, 3, 285–300.

TAN, H. Z., SRINIVASAN, M. A., EBERMAN, B., AND CHENG, B. 1994. Human factors for the design of force-reflecting haptic interfaces. *Dynamic Systems and Control 55*, 1, 353–359.

TCHING, L., DUMONT, G., AND PERRET, J. 2010. Interactive simulation of cad models assemblies using virtual constraint guidance. *Int. J. on Interactive Design and Manufacturing (IJIDeM) 4*, 2, 95–102.

WAN, H., GAO, S., PENG, Q., DAI, G., AND ZHANG, F. 2004. Mivas: a multi-modal immersive virtual assembly system. In *Proc. Int. Design Engineering Technical Conf. (ASME)*, American Society of Mechanical Engineers (ASME), 113–122.

XIA, P., LOPES, A., AND RESTIVO, M. 2011. Design and implementation of a haptic-based virtual assembly system. *Assembly Automation 31*, 4, 369–384.

ZACHARIAS, F., HOWARD, I. S., HULIN, T., AND HIRZINGER, G. 2010. Workspace comparisons of setup configurations for human-robot interaction. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 3117–3122.

ZACHMANN, G., AND RETTIG, A. 2001. Natural and robust interaction in virtual assembly simulation. In *Proc. Int. Conf. on Concurrent Engineering: Research and Applications (ISPE)*, vol. 1, 425–434.

ZORRIASSATINE, F., WYKES, C., PARKIN, R., AND GINDY, N. 2003. A survey of virtual prototyping techniques for mechanical product development. *Proc. of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 217*, 4, 513–530.