Natural and Robust Interaction in Virtual Assembly Simulation

Gabriel Zachmann

Informatik II
University of Bonn
Römerstraße 164
53117 Bonn, Germany
zach@cs.uni-bonn.de

Alexander Rettig

Visualization and Virtual Reality
Fraunhofer Institute for Computer Graphics
Rundeturmstraße 6
64283 Darmstadt, Germany
arettig@igd.fhg.de

Abstract

Virtual assembly simulation is one of the most challenging applications of virtual reality. Robust and natural interaction techniques to perform the assembly tasks under investigation are essential as well as efficient methods for choosing from a large number of functionalities from inside the virtual environment. In this paper we present such techniques and methods, in particular multimodal input techniques including speech input and gesture recognition for controlling the system. We address precise positioning by novel approaches for constraining interactive motion of parts and tools, while a new natural grasping algorithm provides intuitive interaction. Finally, sliding contact simulation allows the user to create collision-free assembly paths efficiently. Preliminary results show that the array of functionality and techniques described in this paper is sufficiently mature so that virtual assembly simulation can be applied in the field.

Keywords: Virtual prototyping, virtual reality, constraint-based interaction, physically-based interaction.

1 Introduction

Virtual reality[1] is gradually being accepted as a tool for digital prototyping in manufacturing industries, because it offers many advantages: rapid design/test cycles, low prototyping costs, efficient learning (especially when compared to blueprints), a convenient platform for simultaneous and even distributed engineering teams.

Assembly simulation, however, is one of the most challenging applications of virtual reality (VR). This is mostly due to the very high interactivity: it is not only the high amount of functionality needed, but also because some of the interaction must be as natural as possible. After all, it is the interaction itself which is to be simulated; and that interaction mostly involves the human hand. This is in contrast to other VR applications like styling review, design review, or lighting simulation: there, interaction is just a means for studying an object (the car body, car interior, tools, etc.), so it does not need to be natural. And in some applications, like styling review, the amount of immersive functionality is much less than in virtual assembly simulation.[2]

By natural interaction, we understand interaction which imitates that same interaction in the real world as close as possible. Actually, in the early days of virtual reality, this was one of its driving goals. However, it has become obvious, that there are many applications (like design review) where natural interaction is not possible or not efficient. Today, virtual reality focuses on creating *intuitive* interaction, instead. A special case, of course, is natural interaction.

In the real world, a worker utilizes natural constraints to obtain precise and efficient manipulation of parts and tools. The same effect can be achieved within a virtual environment (VE) by making the system constrain the user's motions. Force-feedback could be used to achieve that, but it is not available in some work environments and it is not appropriate for certain types of constraints.

[1] By *virtual reality* we understand the use of *immersive* displays and novel, *intuitive* input devices, together with system providing real-time rendering and simulation. This is in contrast to a broader application of the term in the manufacturing realm, where somtimes even a desktop VRML browser is called "virtual reality".

[2] Of course, virtual styling review presents other challenges.

During a VR session the user not only interacts with the virtual environment but also with the system itself. This interaction must be as robust and efficient as possible, otherwise the system will not be accepted as a tool in the design process. On the 2D desktop, most users are familiar with WIMP[3]. Therefore, these concepts should be adopted for VR, although there is a different pointing device (dataglove or flying joystick, for instance) and no keyboard. Because precise handling of the pointing device in VR is more difficult than on the desktop, voice input as an additional input channel can be very helpful and efficient.[4]

In the following, we will describe some related work (see Section 2), then we will discuss various issues related to interaction with the system (Section 3). In Section 5, we will describe an algorithm for preventing collisions between parts, and in Section 6 we will describe an algorithm for natural grasping. Finally, we will draw some conclusions (Section 7) and discuss directions for further work.

2 Related Work

A number of systems for virtual assembly simulation have been reported, among them are [JCL97, JJWT99, LD97, Zac99]. The latter is a commercial system actually being deployed in industries, while the former are research systems.

Planning assembly sequences is generally most efficient in a virtual environment, compared to blueprints or a desktop environment [BBYD99]. Similar findings are reported by [CDG97] for VR-based CAD.

In general, by applying virtual prototyping techniques to the product development process, considerable time and cost savings can be achieved — as [LMO96] report: “80% of development costs and 70% of life cycle costs of a product are determined during its conceptual phase.” Similar numbers are reported by [Pra95, Ull92].

An overview of 3D interaction techniques is presented by [Han97]. A software architecture which facilities run-time constraint detection and the maintenance of constraint consistencies in order to support real-time constraint-based modeling has been presented by [FMTW99]. We use the term “constraint” in a broader sense, however, and propose several paradigms for constraining the user’s interaction rather than algorithms for solving constraint-based modeling.

3 Multi-modal input

Virtual reality systems try to use many input channels in order to increase intuitivity of the interaction. These are usually head and hand tracking, a dataglove (i.e., finger tracking), and we also suggest the use of voice input.

Finger tracking (through a dataglove) is necessary, because the hand is the user’s most important tool for assembly. Since flex data are available, one usually also does some gesture recognition. While there are many algorithms for gesture recognition [ZLB+87, SZ94], we have found that a relatively simple algorithm based on adaptive scaling and classification within the $\{-1, 0, +1\}^d$ cube is very effective, robust and user-independent [Zac00]. Besides, in our experience, only a minimum of about 3 gestures should be used at all.

We have experimented with many different ways how to implement menus in virtual environments. Virtual menu can be realized as an array of virtual 3D buttons. From an implementation point of view, this is very appealing, but two main problems arise with that approach: (1) where to place them, and (2) how to select an entry. There are applications where the number of entries in virtual menus and the frequency of their utilization is small (games, for instance). Then, virtual 3D menus can and should be used.

When the number of entries in menus becomes large, positioning and selecting virtual 3D menus can become very difficult. This is due to the problem that unconstrained 6D pointing is much less accurate and the HMD’s resolution and sharpness is much less than on a desktop screen. We have tried stationary, head-centered, and body-centered positioning, with various ways of selecting entries, namely actual touching, ordinary ray casting, and eye-to-finger ray casting. If both hands of the user are tracked, then hand-centered positioning can be utilized [PNW98, PBBW95, WMB98].

Because of all these problems, we believe that for virtual assembly simulation, menus should be realized as some kind of 2D overlay on the 3D scene, which we have done for all our automotive VR applications. Such 2D menus can be implemented in the familiar way of hierarchical lists, or as “mark menus” [Kur93].

Voice input is one of the most natural ways of human communication. Since speech recognition has become quite robust and almost real-time, it is a matter

[3] Windows, Icons, Menus, Pointers
[4] In fact, even on the desktop voice input is becoming more and more widespread as a third input channel.

of course to use this input channel for interacting with virtual environments.

In order to be easily configurable and to facilitate a consistent command language, we have developed a simple grammar for specifying speech commands (sentences). This grammar is used in specifications of VEs in order to trigger actions. The general syntax is

$$w_{11}|w_{12}|\ldots\ n_1\ w_{21}|w_{22}|\ldots\ n_2\ \ldots$$

This sentence will trigger when the word w_{11}, or w_{12}, etc., is received, followed by at most n_1 "noise words", followed by the word w_{21}, or w_{22}, etc. The concept of noise words allows to discard unimportant utterances of users (e.g., "path [go to] [the] next position"), and it is another way to allow variants.

Some commands include a parameter which must be conveyed to the action. For instance, the command "rotate by n degrees about axis x" contains two parameters. Therefore, instead of specifying a word (or several alternative words), the grammar also allows to specify a (formal) parameter. When the sentence has been matched, the corresponding actual parameters are delivered to the action. The sentence matching engine does not perform any "type checking" (it would require that words have a type); this is done by the action.

In our experience, it is important that the structure of voice commands is similar to the structure of the menu. That way, the user can use the menu and learn the corresponding voice command as she goes through the tree of sub-menus.

4 Constraining interactive object motion

Constraining the motion of virtual objects which are interactively manipulated by the user facilitates precise positioning in immersive VEs, which otherwise would almost be impossible. This is because the user's real hand always moves in free space, there are no mechanical points of reference other than his body, and there are no rests or supports for the user's hand. Constraints can also be used to enhance abstract interaction, or to simulate mechanical constraints of the real world (see below). Especially for the latter, force-feedback would enhance the realism of the simulation significantly, but often force-feedback is not applicable: in CAVE-like environments, a force-feedback device disturbs the immersion because it occludes parts of the screen. Besides, the concept of constraints is valid and useful even when force-feedback is available.

4.1 Abstract constraints for precise positioning

In virtual assembly simulation, like in other CAD systems, objects must often be positioned or moved precisely. This is a situation, where preciseness and robustness takes precedence over natural interaction.

One method is to position objects by abstract commands given via voice input, immersive menu, desktop GUI, or keyboard. That way, an object can be translated, rotated, or scaled about one coordinate axis. The user can choose between car coordinate system and object coordinate system. In our experience, voice commands containing parameters are the most flexible and most efficient way to control abstract positioning in a fully immersive VE, but also menus have been proven to work well.

The second method provides a more intuitive but still quite precise way to position parts. The user first selects a point, axis, or plane in the car coordinate system or object coordinate system. This will constrain the object's degrees of freedom. Then, the object is linked to the user's hand so that it tries to follow the hand's motion but *only* within the constraint. Similarly, rotational constraints limit an object's rotation to an arbitrary axis and angular range in space. All of these constraints can be implemented such that they work with parts and compounds, and such that they work regardless of the source of the motion of parts.

These constraints are not only useful for moving parts, but also for auxiliary geometry like interactive clipping planes or grids. Usually, both need to be aligned to the car reference frame in order to match those in the CAD system. Furthermore these constraints provide an easy way to define simple interactive kinematics, hinges, or slide joints.

4.2 Aligning tool axes

One of the goals of assembly simulation is to verify that there is enough space to perform a certain operation with a certain tool. Very often, this is some kind of screwing operation (with a screw driver or wrench, for instance). Two problems arise when investigating this in VR: it is difficult for the user to hit a screw exactly with the wrench, for instance, and it is almost impossible to maintain the correct contact during the screwing operation.

A solution for both problems is to constrain the tool's motion in the following way: the tool center point (TCP), usually at the tip of the tool, is kept coincident with the contact point of the screw (SCP), usually at its head, and the tool axis is kept aligned with the screw axis.[5]

When applying such a constraint to the tool, the virtual hand may behave in two different ways:

1. follow the user's hand motion
2. stay attached to the tool

Whereas the first alternative often seems to be less confusing for the user, the second is important for a correct verification of the space requirements.

Usually, the user's hand motion directly changes the transformation of the virtual hand relative to the virtual body's reference frame.[6] For a tool being grabbed the relative transformation T_{rel} between hand and tool must stay invariant (see Figure 1). T_{rel} is therefore given by

$$T_{rel} = W_{tool}\, W_{hand}^{-1}$$

where W_{obj} denotes the transformation in world coordinates of *obj* at the moment *obj* is being grabbed. When the user moves the hand, T_{tool} is updated such that the invariant T_{rel} is maintained.

The proposed constraint requires some manipulations in the transformation chain to map the user's motion to the tool in an intuitive way. For the first variant, only the tool transformation has to be modified. The second variant requires to modify the hand transformation coming from the tracking device. A good way to do this is to insert new transformation nodes S_{tool} and S_{hand} into the scene graph before the tool or the hand transformation, respectively. Thus, the snapping transformation can be specified conveniently with these nodes.

The algorithm conceptually works as follows: in each frame, it starts with the unconstrained position of the tool (which is being grabbed by the user). Then an offset transformation is computed which rotates the tool around the TCP such that the tool axis becomes parallel to the screw axis and the TCP coincides with the SCP. Setting this transformation for S_{tool} will snap the tool to the screw without changing the position of the hand. In order to keep the virtual hand attached to the tool, the same transformation has to be set for S_{hand} (in the body's reference frame).

In our implementation, a tool does not immediately snap to a screw when it is moved close to it, but only after it has collided with it. This has proved to be an intuitive solution to simulate the problem to hit the screw in tight compartments and dense environments. Furthermore, we implemented a thresholding mechanism to switch off the snapping constraint when the user moves the tool too far away from the screw location. Like in reality, each tool matches only certain screws. This is modelled in our system by defining sets of assoications between screws and their proper tools.

5 Sliding contact for interactive part movement

In recent years, packaging of parts has become so tight, that there are many parts for which there is no assembly path such that the part never touches other parts. In fact, the part usually touches other parts over a relatively long interval along the path. And even if there would be a collision-free path — a human worker will always "create" assembly paths with touching collisions in the real world.

So, in the virtual world, we need a way to prevent parts from penetrating each other, while still allowing touching collisions such that the user can move parts in a sliding fashion along the surface of other parts.

If the VR system has no control over the user's real hand (via force feedback), then the commonly used rigid grasping metaphor has to be changed slightly towards a less rigid one. This is a variant of grasping where the transformation from hand coordinate system to object coordinate system is no longer invariant. For the sake of clarity, let us assume that the object is collision-free at the time when it is being attached to the hand. Let us assume further that at that moment we make a copy of the object which is allowed to penetrate all other objects and which is being grasped rigidly by the hand. We will call this copy the "*ghost*" of the object. It marks the position where the object would really be if there was no collision. There are, at least, three non-rigid grasping metaphors:

- The rubber band metaphor: the object is connected to the ghost by a rubber band. This tries to pull the object as close to the ghost as possible without penetrating.

- Rubber band and spiral spring: like the plain rubber band metaphor, but the object is also connected by a spiral spring to the ghost (this is a little bit difficult to picture). In the plain rubber band metaphor, the user has no control over the orientation of the object — it is completely determined by the simulation.

[5] Such axes and contact points must be part of the tools' description.
[6] Following the "flying carpet" paradigm as introduced in [Zac00].

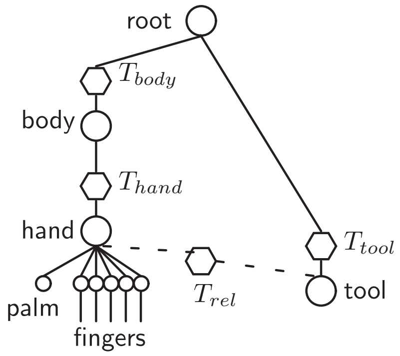

Figure 1: Grabbing parts should be implemented by maintaining the relative transformation T_{rel} invariant.

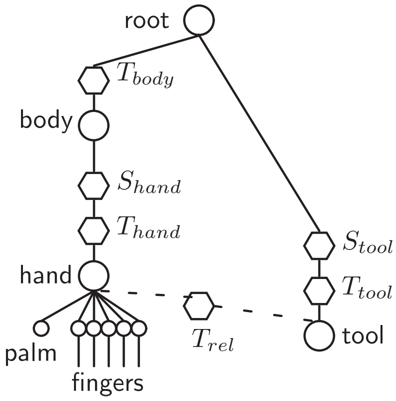

Figure 2: Snapping of tools can be implemented easily by inserting nodes to hold the snapping transformations S_{tool} and S_{hand}.

- Incremental motion: when the ghost has moved by a certain delta the object will try to move about the same delta (starting from its current position). If there is a collision during that delta, then the simulation will determine a new direction. So, alternatingly the object is under the control of the user and under simulation control.

The algorithm we describe in the following allows for implementation of any of these metaphors.

If the VR system does have control over the user's real hand via force feedback, then the rigid (and more natural) grasping metaphor can be retained. Still, a simulation algorithm is needed to create forces appropriate to keep the user's hand from generating interpenetrations. The algorithm we will present below can be used to render such forces.

In the remainder of this section, we will explain that algorithm in more detail. The reader should keep in mind, that the goal was *not* to make the sliding behavior of objects as physically correct as possible. Readers interested in physically correct simulations should refer to the wealth of literature, for instance [Bar94, GVP91, Hah88, SS98, BS98].

Instead, the goal was to develop an efficient algorithm which helps the user to move the object exactly where he wants it, in minimal time, and even in closely packed environments (such as the interior of a car door).

However, we believe that our approach is more general than the one presented by [KYK98]. There, the approach is to constrain the motion of objects by certain faces identified through collision detection; then, the number of faces determines the number of remaining degrees of freedom (for instance, with two faces only one translational degree remains). In addition, constraining faces exhibit a certain "stickiness". The overall approach is not physically-based, but just meant as an aid for assembling simple "block-shaped" objects.

5.1 The main loop

For several reasons, the collision detection module runs concurrently in our VR system. Therefore, the sliding simulation is implemented as a finite state machine, so that it can handle that. This has the additional advantage, that the simulation module itself runs concurrently in our VR system. The three modules communicate with each other as depicted in Figure 3.

In the simulation, there is a so-called *collision object* (short *collobj*) which is invisible.[7] It is used to check intermediate position for collisions. The *visible object* is the one users are really seeing. It is never placed at invalid (i.e., colliding) positions. So the user only sees a valid, i.e., collision-free path of the object.

The algorithm works, simply put, as follows:

```
loop:
  while no collision
    move visible and coll.  object
    according to hand motion
  {now the coll.-object is penetrating}
  approximate exact contact point
  classify contact
  calculate new direction
```

[7] The geometry of the collision object is, usually, exactly the same as that of the visible object. If the scene graph API allows for it, they can share their geometry.

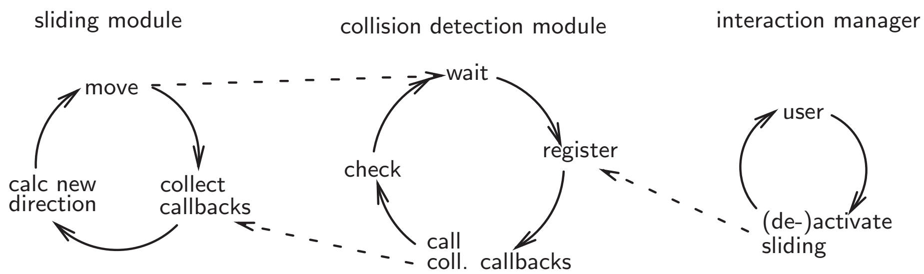

Figure 3: The physically-based simulation module for sliding runs concurrently to the other two main loops of the collision detection module and the interaction manager. Dashed arrows mark rendezvous points.

As mentioned before, this is implemented as a finite state machine. A more detailed picture and description of that can be found in [Zac00]. There, you can also find several algorithms for fast collision detection.

In our algorithm, the contact approximation is done by interval bisection and a number of static collision checks. [ES99] propose a dynamic collision detection algorithm. However, it is not clear that this would really speed up the simulation in this case, since the dynamic algorithm takes about 3–5 times longer and an exact contact point is usually not needed here.

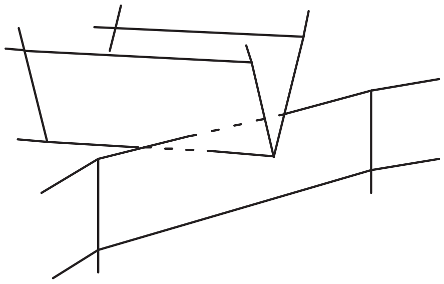
Figure 4: Contact classification can be done by looking at adjacent polygons and counting edge/face intersections.

5.2 New directions

Let us assume that we have the exact contact position. Let us further assume that we need to handle only practically relevant contact situations. Then we will need to deal only with the following cases: 1 contact point, 2 contact points, and ≥ 3 contact points, which we will discuss in the following.

In each case, we must be able to deal with "wrong" surface normals. In general, polygonal geometry imported from CAD programs has "random" surface normals in the sense that the vertex order is *not* consistent across adjacent polygons. But even if it were, we would have to be able to deal with such a situation, because unclosed objects (like sheet metal) does not have "inside" and "outside". With such "sheet objects" we might be colliding from either side.

Our implementation of the sliding algorithm presented here allows for arbitrarily pointing normals. They can even be "inconsistent" in the sense that adjacent polygons' normals can point on different side.

In order to be able to compute new forces, each contact point must be classified. In theory, we need to handle only two contact situations: vertex/face and edge/edge. Given one pair of touching polygons (p, q), we can determine the contact situation by the following simple procedure: let n_p be the number of polygons adjacent to p and touching q; define n_q analogously. If $n_p = n_q = 1$, then we have the edge/edge case. Otherwise, either n_p or n_q must be > 1 (but not both), and we have the vertex/face (or face/vertex) case.

In practice, a few more cases can happen. Partly, this is due to the mere approximation of the contact position, partly, it is due to non-closed geometry.[8]

If $n_p = 0$, for instance, then this polygon might be at the rim of the object. In order to decide that, we need to check if any edge of p intersects q (see Figure 4). If so, we have got the edge/edge situation. If not, then it is the vertex/face situation. Note that both $n_p = n_q = 0$ is possible. It is not sufficient to just check edge/face intersections of the two intersecting polygons (counter-example: two intersecting wedges).

When all contact points have been classified, we can compute new forces and velocities. See [Zac00] for the mathematical details. Depending on the kind of metaphor we have chosen (see above), we must then evaluate a stopping criterion before iterating the next sliding cycle. Actually, the kind of grasping metaphor is determined by the stopping criterion. Our criterion is a combination of several sub-criteria involving the number of iterations so far and different distance measures. See [Zac00] for more details.

[8] Non-closed geometry is fairly frequent in virtual prototyping: all sheet metal is non-closed. Now imagine the possible contact situations when a pipe is to be fitted into a hole of sheet metal.

6 Natural grasping

Grasping objects is one of the most fundamental interaction techniques in VEs, in particular in virtual assembly simulation, which comes as no surprise since it is also one of the most frequent activities in the real world.

In order to be able to make true verifications of assemblability and serviceability, it is important, that the virtual hand grasps virtual parts just like the real hand would grasp their real counterparts. So, the VR system must make sure that the virtual fingers never penetrate parts, but still allows them to close tight around them (see Figure 5). In addition, in order to make virtual grasping natural, the VR system has to determine when an object is grabbed firmly. So, the user would not need to remember a command, and objects cannot be grabbed by the back of the hand.

Like with force-feedback devices, we need to distinguish between (at least) four types of grasping:

1. *Precision grasping* with three sub-types [Jon97]: tip pinch, three-jaw chuck, and key grasp,
2. *cigarette grasping*,
3. *3-point pinch grasping*,
4. *Power grasping* (or just *grasping*),
5. *Gravity grasping* (or *cradling*).

Precision grasping involves 2 fingers, usually the thumb and one of the other fingers; it is used for instance to grasp a screw. Cigarette grasping involves two neighboring fingers; it is usually used to "park" long thin objects, such as a cigarette or pencil. 3-point grasping involves three fingers (one of them being the thumb), giving the user a fairly firm grip, and allowing him to rotate the object without rotating the hand. Power grasping involves the whole hand, in particular the palm. With this type of grasp the object is stationary relative to the hand. Gravity grasping is actually a way of carrying an object.

For power grasping, the algorithm consists of two simple parts: clasping the fingers around the object, and analyzing the contact. The former will be done by an iteration, while the latter is implemented by a simple heuristic.

The position of the hand is completely specified by (M, F), where M is a matrix specifying the position of the hand root, and F is the *flex vector* (usually 22-dimensional). Given a new target hand position (M^n, F^n), the goal is to minimize $(|MM^{n-1}|, |F - F^n|)$ such that (M, F) is collision-free. Note that the position of a finger-joint depends on its flex value and all flex values higher up in the chain and the position of the hand root. Therefore, we suspect that there are several local minima, even if we only consider flex values during optimization (and keep the position fixed). However, this should not be a problem if the minimization process is fast enough, so that consecutive collision-free hand positions are not too "distant" from each other.

Minimization must not be done using the visible model of the hand; otherwise, the user would "witness" the process (because the renderer runs concurrently). So, a copy of the hand tree is used for collision detection, and only after minimization has finished, the new position/flex values are copied to the visible hand. This minimization should be as fast as possible, so it runs as a concurrent process in our VR system; otherwise, the user might notice considerable latency.

In order to find an optimal flex vector, our algorithm uses an iteration process interpolating non-colliding (i.e., valid), and colliding (i.e., invalid), flex values. Here, a flex value is *colliding* if its associated finger-joint is colliding *or* any finger-joint depending on it. A finger-joint J' is *depending* on a finger-joint J, if it is further down the kinematic chain, i.e., if J moves, then J' moves, too (see [Zac00] for a detailed depiction of the finite state machine of this process). Note that a finger-joint can have many depending finger-joints. (In this context, the palm is a "finger-joint" like all the others.)

During the iteration process, the position M of the hand is treated like any other flex value, i.e., it is interpolated. The only differences are that interpolation is done on matrices instead of single real numbers. The "joint" associated with it is usually the palm or the forearm.

After a few iteration steps, some flex values will be approximated "close enough" (when the range between valid and invalid flex value is small enough). Then, they will be *fixed*. Depending flex values must be considered for fixing, too: they may or may not be close enough. So, several flex values in a row may become fixed at the same time. As long as a flex value is not fixed, it will be interpolated *and* all its depending flex values.[9]

After all flex values have been fixed, the second phase of the algorithm tries to analyze the type of grasp. While the grasping algorithm is in general applicable to any hierarchical kinematic chain, the analysis algorithm needs to know more about its "seman-

[9] An alternative would be not to interpolate depending flex values, but since depending finger-joints need to be checked for collision anyway, we can as well interpolate them, too. Thus, we probably achieve an optimum faster.

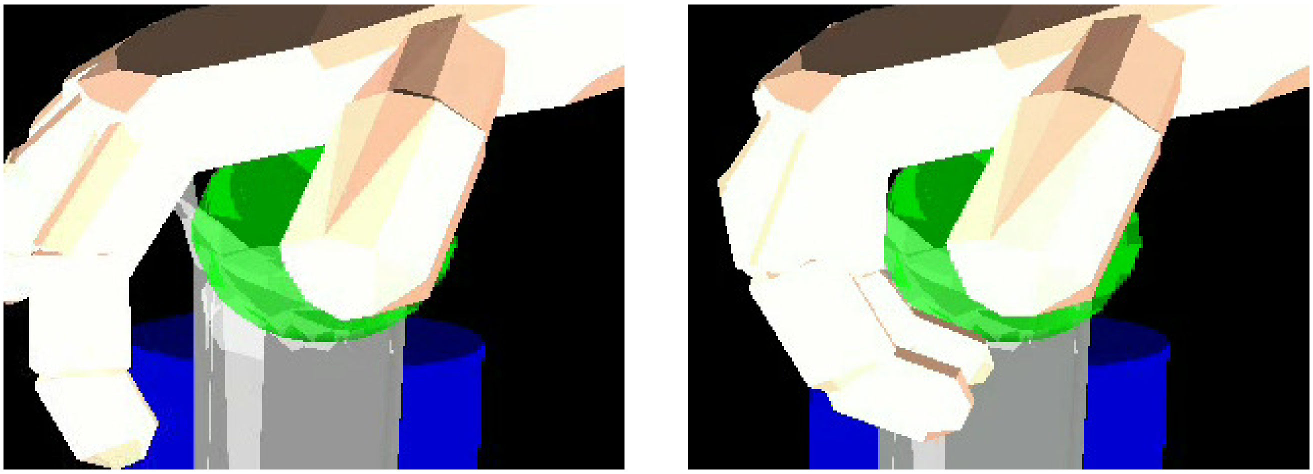

Figure 5: Natural grasping is basically a minimization problem for the flex vector under the constraint that finger-joints (and palm) must not penetrate the object.

tics", i.e., it has to know about a palm, it needs to know which finger-joints belong to the same finger, etc. The heuristic we have implemented is very simple:

1. only one finger-joint or palm is touching → push;
2. several finger-joints are touching, and none of them is part of the thumb, and the palm is not touching → push;

   This part of the heuristic would need to be more sophisticated if cigarette grasping should be recognized. However, this type of grasp is not needed for virtual assembly simulation.
3. one or more finger-joints is touching, one or more thumb-joints is touching, and the palm is not touching → precision grasp;
4. one or more finger-joints (possibly a thumb joint) and the palm are touching, and at least one of the finger-joints is a middle or outer joint → power grasp;
5. one or more finger-joints and the palm are touching, but all finger-joints are inner joints → push.

In our algorithm, motion of the cart is handled specially. A cart motion indicates that the user's (virtual) body is changing place. Since the hand is attached below the cart, a cart motion always brings on a motion of the hand. In that case, the algorithm does not try to clasp the hand tightly around an object, because that might cause the hand to be left behind, which is probably not what the user wanted. Unfortunately, navigation might cause the hand to end up in an invalid (i.e., colliding) place, so after navigation has stopped, the clasping algorithm cannot begin until the whole hand has been moved to a collision-free place by the user.

7 Conclusion

In this paper, we have shown that interaction metaphors for virtual assembly simulation must be balanced between naturalness, robustness, precision and efficiency. This is often a compromise which has to be decided on a case-by-case basis.

Precise positioning of parts is made possible by constraining interactive object motions and by abstract positioning via command interfaces. Robust and efficient interaction with the system is achieved by utilizing all input channels available, namely gesture recognition, tracking, voice input, and menus. We have reported on a robust algorithm for gesture recognition, a way to robustly recognize voice commands, and on our experiences with robust menu interaction. Robust tracking can be achieved by predictive filtering [Zac00].

Natural grasping can be solved with an algorithm presented in this paper. This is a robust interaction, even without relying on abstract gestures. Similarly, a robust physically-based simulation has been proposed in order to create collision-free assembly paths. In the absence of control over the user's hand, naturalness must be sacrificed over correctness; force-feedback would be needed to retain both correctness and naturalness.

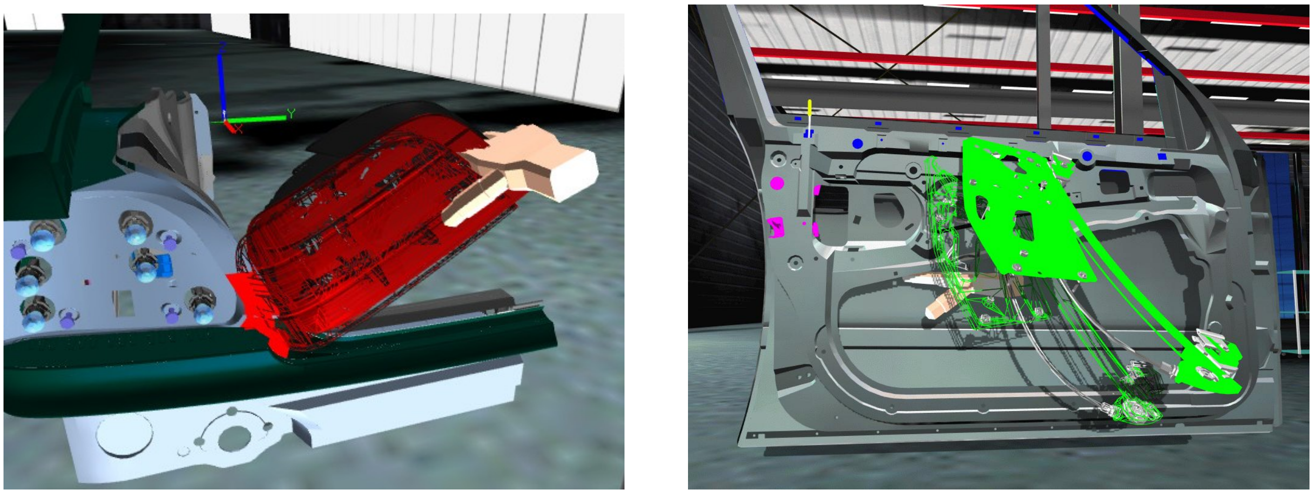

Figure 6: All the algorithms and paradigms presented in this paper have been integrated in our VR system *Virtual Design II*, which is used by car industries to carry out virtual assembly simulation in scenarios like the two shown above.

All of these algorithms and metaphors have been developed in cooperation with automotive industry partners and are integrated into our VR system *Virtual Design II*, which is being utilized for the virtual prototyping process in applications as design review, styling review, immersive scientific visualization, and others. In particular, the virtual assembly module containing the functionality and human machine interface described in this paper has proven by usability tests to be efficient and mature for assembly simulation [**?**].

8 Future Work

In order to make the sliding simulation and natural grasping even more efficient, collision detection algorithms should provide some measure of the amount of interpenetration. This information could be used to estimate the contact points more quickly.

A particularly difficult problem for the sliding algorithm is presented by the way slide-in units are designed (for instance, car radios): these parts and their compartments are usually designed such that their respective hulls overlap precisely, so that the virtual parts will actually have a collision when in final position.

Interacting with the system itself will become an area for further research. As the number of functionalities offered by the VR system increases, and as the amount of information about the current state of the system and about the virtual environment increases, interaction with the system can become more and more difficult. Given the type of VR input devices needed for the respective application, it is important not to confuse the user yet provide an efficient interface.

Force feedback is currently an active area of research. For simple cases, algorithms and devices are available, but as of yet, there is no device suitable for virtual assembly simulation. However, force feedback would greatly increase intuitivity and user efficiency as well as the transferability of the results obtained in interactive simulations.

Building on our algorithms presented in Section 5, we are currently developing haptic simulation algorithms for a novel force feedback device. These will be integrated with our VR system *Virtual Design II* and applied to complex scenarios such as Figure 6. A main focus is laid on stability of the simulation and user safety. In addition, we are investigating the perceptual quality of haptic rendering. In order to achieve stable behavior of the system, very high update rates must be guaranteed. This requires further increase of the speed of collision detection [Zac01].

References

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Andrew Glassner, Ed., Computer Graphics Proceedings, Annual Conference Series, pages 23–34. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[BBYD99] A. Banerjee, P. Banerjee, N. Ye, and F. Dech. Assembly planning effectiveness using virtual reality. *Presence*, 8(7):204–217, 1999.

[BS98] Matthias Buck, and Elmar Schömer. Interactive rigid body menipulation with obstacle contacts. *The Journal of Visuazlization and Computer Animation*, 9:243–257, 1998.

[CDG97] Chi-Cheng P. Chu, Tushar H. Dani, and Rajit Gadh. Multi-sensory user interface for a virtual-reality-based computer-aided design system. *Computer-Aided Design*, 29(10):709–725, 1997.

[ES99] Jens Eckstein, and Elmar Schömer. Dynamic collision detection in virtual reality applications. In *Proc. The 7-th International Conference in Central Europe on Computer Graphics, Visualization, and Interactive Digital Media '99 (WSCG'99)*, pages 71–78. University of West Bohemia, Plzen, Czech Republic, February 1999.

[FMTW99] Terrence Fernando, Norman Murray, Kevin Tan, and Prasad Wimalaratne. Software architecture for a constraint-based virtual environment. In *Proc. VRST '99*. ACM, 1999.

[GVP91] Marie-Paule Gascuel, Anne Verroust, and Claude Puech. Animation and collisions between complex deformable bodies. In *Proceedings of Graphics Interface '91*, pages 263–270, June 1991.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, John Dill, Ed., vol. 22, pages 299–308, August 1988.

[Han97] Chris Hand. A survey of 3D interaction techniques. *Computer Graphics Forum*, 16(5): 269–281, 1997. ISSN 1067-7055.

[JCL97] Sankar Jayaram, Hugh I. Connacher, and Kevin W. Lyons. Virtual assembly using virtual reality techniques. *Computer-aided Design*, 29(8):575–584, 1997.

[JJWT99] Sankar Jayaram, Uma Jayaram, Yong Wang, and Hrishikesh Tirumal. VADE: A virtual assembly design environment. *IEEE Computer Graphics & Applications*, 19(6):44–50, November, December 1999.

[Jon97] Lynette Jones. Dextrous hands: Human, prosthetic, and robotic. *Presence*, 6(1):29–56, February 1997.

[Kur93] Gordon Paul Kurtenbach. *The Design and Evaluation of Marking Menus.* PhD dissertation, University of Toronto, Graduate Department of Computer Science, 1993. URL `http://reality.sgi.com/gordo_tor/papers/PhdThesis/PhDthesis.html`.

[KYK98] Yoshifumi Kitamura, Amy Yee, and Fumio Kishino. A sophisticated manipulation aid in a virtual environment using dynamic constraints among object faces. *Presence*, 7(5): 460–477, October 1998.

[LD97] Valerie D. Lehner, and Thomas A. DeFanti. Projects in VR: Distributed virtual reality: Supporting remote collaboration in vehicle design. *IEEE Computer Graphics and Applications*, 17(2):13–17, March/April 1997. CODEN ICGADZ. ISSN 0272-1716.

[LMO96] Joseph S. Lombardo, Edward Mihalek, and Scott R. Osborne. Collaborative virtual prototyping. *Johns Hopkins APL Technical Digest*, 17(3), 1996.

[PBBW95] Randy Pausch, Tommy Burnette, Dan Brockway, and Michael E. Weiblen. Navigation and locomotion in virtual worlds via flight into Hand-Held miniatures. In *SIGGRAPH 95 Conference Proceedings*, Robert Cook, Ed., Annual Conference Series, pages 399–400. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[PNW98] I. Poupyrev, T. Numada, and S. Weghorst. Virtual notepad: handwriting in immersive vr. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98*, pages 126–132. Atlanta, Georgia, March 1998.

[Pra95] M. J. Pratt. Virtual prototypes and product models in mechanical engineering. In *Virtual Prototyping – Virtual environments and the product design process*, Joachim Rix, Stefan Haas, and José Teixeira, Eds., chapter 10, pages 113–128. Chapman & Hall, 1995.

[SS98] Jörg Sauer, and Elmar Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. In *Proc. VRST '98*, pages 153–161. ACM, Taipei, Taiwan, November 1998.

[SZ94] David J. Sturman, and David Zeltzer. A survey of glove-based input. *IEEE Computer Graphics and Applications*, 14(1):30–39, January 1994. CODEN ICGADZ. ISSN 0272-1716.

[Ull92] D. G. Ullman. *The Mechanical Design Process.* McGraw-Hill, 1992.

[WMB98] G. Williams, I. E. McDowell, and M. T. Bolas. Human scale interaction for virtual model displays: A clear case for real tools. In *Proc. of The Engineering Reality of Virtual Reality.* SPIE, January 1998.

[Zac99] Antonino Gomes de Sá, and Gabriel Zachmann. Virtual reality as a tool for verification of assembly and maintenance processes. *Computers & Graphics*, 23(3):389 –403, 1999.

[Zac00] Gabriel Zachmann. *Virtual Reality in Assembly Simulation — Collision Detection, Simulation Algorithms, and Interaction Techniques*. PhD dissertation, Darmstadt University of Technology, Germany, Department of Computer Science, May 2000. URL `http://web.informatik.uni-bonn.de/~zach/papers/diss.html`. Fraunhofer IRB Verlag, ISBN 3-8167-5628-X; also: `http://www.geocities.com/gabriel_zachmann/`.

[Zac01] Gabriel Zachmann. Optimizing the collision detection pipeline. In *Proc. of the First International Game Technology Conference (GTEC)*, January 2001.

[ZLB+87] Thomas G. Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. In *Proceedings of Human Factors in Computing Systems and Graphics Interface '87*, J. M. Carroll and P. P. Tanner, Eds., pages 189–192, April 1987.