A FRAMEWORK FOR COLLISION DETECTION AND RESPONSE

C. Lennerz, E. Schömer, T. Warken
Computer Science Department
Saarland University
P.O.Box 151150
D-66041 Saarbrücken, Germany
http://www-hotz.cs.uni-sb.de/silvia/silvia.html

ABSTRACT

Detecting collisions and calculating physically correct collision responses play an important role when simulating the dynamics of colliding rigid bodies. Virtual reality applications such as virtual assembly planning and ergonomy studies can especially profit from advances in these directions, because they enable an interactive and intuitive manipulation of objects in virtual environments. This paper presents new algorithms for the real-time simulation of multi-body systems with unilateral contacts. The algorithms for dynamic collision detection and for the calculation of contact forces are part of the software library SiLVIA, a simulation library for virtual reality applications.

1 INTRODUCTION

Virtual reality techniques are becoming more and more attractive to engineers, who design complex mechanical systems. This trend is supported by an intensified application of virtual prototyping. The construction of virtual prototypes simplifies the whole design process and is a prerequisite for VR-based assembly planning. In order to interactively simulate an assembly of mechanical parts the simulation software must be able to detect collisions and calculate reaction forces efficiently. This enables the assembly engineer to intuitively manipulate all objects in spite of missing force feedback mechanisms. Figure 1 demonstrates the principle:

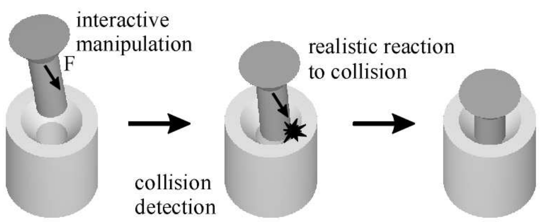

Figure 1: The insertion of a bolt into a countersunk hole.

The manual insertion of a bolt into a hole is a difficult task in a virtual environment, if the motion simply stops as soon as a collision occurs. In reality the bolt automatically slides into the right direction when it comes into contact with the conical boundary of the hole. It is desirable to simulate this effect in order to perform virtual fitting operations in a more intuitive manner. A more elaborate example can be found in figure 3.

Our paper is organized as follows: After a short review of some previous work, we start with explaining the structure of our software library SiLVIA. It includes a collection of algorithms providing a broad basis for the real-time simulation of colliding rigid bodies in virtual environments. Sections 2.1 and 2.2 sketch the data structures necessary for an efficient implementation of our algorithms for collision detection and distance calculation. Both are described in section 2.3 and 2.4, respectively. In section 3 we describe the mathematical approach we use to determine the reaction forces for colliding bodies. We evaluate our concepts by simulating the fitting of a screwdriver into a crosshead screw. We conclude with some remarks on further research.

1.1 Related Work

There is an extensive literature dealing with interference and distance calculations for polygonal models. The simplest method to detect collisions consists in examining all objects for overlap at discrete points in time during the motion sequence. This method is called static collision detection. It has a serious drawback: the existence of collisions has to be checked at very short intervals, so that objects cannot unnoticedly "tunnel" through each other. In contrast to this method, the dynamic collision detection actually considers the continuous motion of the objects. For a moving and a stationary object this means, that not the moving object itself but its swept volume is intersected with the stationary object. In spite of the increased cost of the dynamic approach in comparison to the static one, the dynamic version admits a larger step-size without the risk to miss collisions.

A further method to avoid collisions between two moving objects is the following: calculate the shortest distance between both objects and test whether both objects can travel this distance during the given time interval. Of course this decision depends on the velocities of the objects [CK86, Hub95]. The period of time during which the objects can move unrestrictedly can be easily determined, if an upper bound of the velocities is known in advance. When this time has elapsed, the distance has to be calculated anew. This method works very well unless the objects approach too much. The smaller the distance becomes, the more often it has to be updated.

The basics for all collision and distance calculations are the elementary tests whether two surface patches intersect (during their motion) and the determination of their minimal distance respectively. Since objects are often described by thousands of surface patches, it is very inefficient to compare all faces of one object with all faces of a second object. By wrapping objects in preferably small and simply shaped envelopes an overlap of the envelopes (and thus of the objects themselves) can be excluded in many cases and a lower bound of their distance can be obtained. These envelopes, which are also called bounding volumes, can be spheres [Qui94, PG95, Hub96], spherical shells [KPLM98], axis aligned boxes [ZF95], oriented boxes [GLM96] or fixed-directions hulls [HKM$^+$96, Zac98].

For reasons of simplicity virtual objects are often modelled as rigid bodies, which are not subject to any deformations when they collide. A popular method to calculate the forces acting during a collision consists in using springs. If two objects are going to interpenetrate, a spring force depending on the penetration depth pushes them apart. Physically more correct models for the determination of contact forces use a complementarity formulation, which reflects the unilateral nature of the contacts [Löt82, Bar94, BS98, SS98]. These methods are well suited to cope with situations in which the bodies mutually touch at many contact points. Even the consideration of friction is possible. An alternative method, called impulse-based simulation [Mir96], also takes plastic and elastic impacts between the bodies into account. It provides a very accurate and consistent model for the physical process of collision, but is confined to single contact problems.

1.2 What is New?

Our intention is to provide an open and extendable software library for the real-time simulation of multi-body systems with unilateral contacts. We use self-developed algorithms as well as known results from the fields of computational geometry and computational mechanics in order to obtain efficient and robust implementations for collision detection and collision response. From the algorithmic point of view our main contributions are:

- a *dynamic* version of collision detection. It can handle virtual scenarios of moderate size at interactive rates and does not fail when dealing with degenerate geometric configurations, which are especially error-prone, but often arise in fitting simulations.

- a distance calculation algorithm permitting collision detection for non-interactive simulations. In the course of the algorithm the proximity information is continuously improved so that a lower bound of distance can be provided when its execution is suspended.

- a new method for the determination of contact forces, which is based on a complementarity formulation. In contrast to [Bar94] the contact forces and the contact distances are the complementary variables and not any derivatives thereof. This avoids an accumulation of numerical inaccuracies and ensures that the geometric contact conditions are satisfied quite exactly.

2 THE ARCHITECTURE OF SIlVIA

Our fundamental data structure to represent the boundary of an object follows the ACIS® specification, which clearly distinguishes between the topological and the geometric entities of an object. The basic topological entities are vertices, edges and faces which are geometrically embedded on points, lines and planes. The explicit representation of all adjacencies between the topological entities is important for the robust treatment of degenerate geometric cases.

This kernel is encapsulated by a module for hierarchical bounding volumes and one for object culling. The use of both techniques is indispensable in order to efficiently handle complex objects and scenes with many objects.

The next level of functionality comprises algorithms for efficient collision detection, distance calculation and some typical CAD operations. Detecting collisions and calculating distances between pairs of moving objects are essential for all subsequent mechanical simulations because they provide the necessary geometric information about the admissible spatial positions and orientations of the bodies involved. The two major criteria these algorithms have to fulfil are efficiency and robustness.

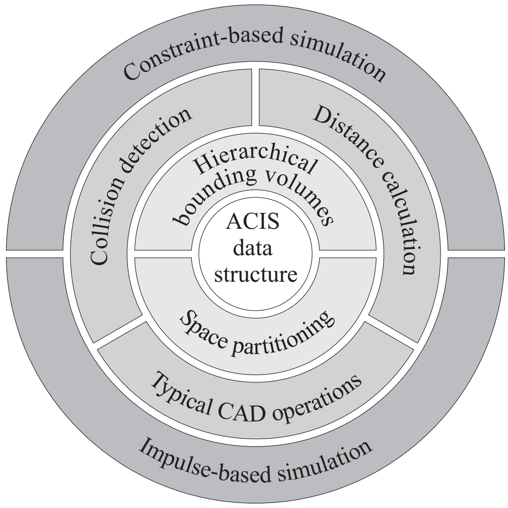

Figure 2: The structure of SiLVIA.

The main purpose of the SiLVIA library is to supply procedures to simulate the dynamics of rigid bodies and to interactively manipulate these bodies in virtual environments. In order to achieve a realistic object behaviour, we use constraint- and impulse-based simulation techniques [Mir96]. Thereby we focus on unilateral contacts between colliding rigid bodies. The whole concept is illustrated in figure 2.

2.1 Object Culling

In order to reduce the computational effort spent on collision detection and distance calculation, object culling tries to quickly remove most object pairs from consideration.

Depending on the application (static or dynamic collision detection/distance calculation) we compute an axis-aligned box for every object wrapping it at a single instant in time or over a time interval. Obviously we have to consider only the pairs with intersecting bounding volumes for pairwise collision detection.

SiLVIA provides two alternative methods. The first one is based on space partitioning and tries to find possible intersections among boxes in space using a hierarchical hashing table. The basic technique was first presented in [Ove92] and then modified in [Mir96] to exploit coherence between moving objects. The second method uses coordinate sorting to find all intersections between the boxes.

2.2 Hierarchical Bounding Volumes

On top of the ACIS® kernel, we built a data structure that allows collision detection and distance calculation queries following *divide-and-conquer* strategies. The collision detection and distance computation problem can be reduced to the problem of finding an intersection and the smallest distance among all faces of both objects, respectively. In order to prevent the consideration of all face pairs, we use a binary tree of bounding volumes covering a hierarchical decomposition of the object.

The root of the tree is associated with the full face set and every inner node corresponds to a proper subset of its parent's faces. Thereby both children form a partition of their parent's face set. Consequently we speak of a leaf if no further partitioning of the face set is possible.

So, every node represents a part of the object that we want to approximate by a simpler shape, the so-called bounding volume. In SiLVIA several types of bounding volumes are available: spheres [Hub95], (arbitrarily) oriented bounding boxes (OBBs) [GLM96] and fixed-directions hulls (FDHs or k-DOPs) [HKM+96], [Zac98]. Axis-aligned bounding boxes (AABBs) are a special case of the latter.

We build the tree in a top-down approach. In doing so, we recursively divide the original face set by creating "balanced" partitions and compute tight bounding volumes on every tree node. The recursion terminates if no further subdivision is possible (leaf). Partitioning proceeds as follows: We pick a splitting plane and assign every face to the side of the plane where its reference point lies. Thereby the splitting plane is defined by containing the mean of all reference points and being orthogonal to the principal axis of inertia that minimizes the sum of the volumes of both children.

Considering the computation of the bounding volumes, we refer to the publications mentioned above.

2.3 Collision Detection

Now that we have described the data structure as the foundation of efficient collision detection queries, we skip to the algorithmic part of the computation.

The collision detection algorithm is very simple and outlined in many publications before [GLM96]. The basic idea is the following: Let v_1 and v_2 be nodes of the respective object trees. If the face sets F_1 and F_2 associated with v_1 and v_2 intersect, then the bounding volumes of F_1 and F_2 do intersect as well. If we negate this statement, we obtain the following rejection test: whenever the bounding volumes of F_1 and F_2 do not intersect, the covered face sets must be separated as well. In the case of intersection we have to check if v_1 and v_2 are both leaves. If neither v_1 nor v_2 has children, we apply an elementary collision test to all face pairs associated with v_1 and v_2. Otherwise we divide the problem and ask if any pair of children of v_1 and v_2 do intersect.

The termination condition depends on the desired information. So as to learn if the objects intersect, we only have to find a single leaf pair for which the elementary test indicates an intersection. Are we interested in obtaining all intersecting features though, we have to check all leaf pairs that cannot be excluded by bounding volume rejection tests.

For the elementary collision test between face pairs we use a dynamic method. First we observe that there are only two types of collisions between polyhedral objects: vertex-face and edge-edge collision. Vertex-edge and vertex-vertex collisions can be seen as special cases of these two types. Given a face pair our algorithm for the elementary test proceeds as follows. Each vertex of the first face is tested for collision with the second face and vice versa. Then each edge of the first face is tested against each edge of the second face. We want to give a short description of the two methods used for these two types of tests.

Vertex-face collision Given a vertex $\boldsymbol{v}$ and an oriented face F that lies in the plane with the equation $\boldsymbol{n}^T\boldsymbol{x} = n_0$ we define the signed distance between $\boldsymbol{v}$ and F as $d = \boldsymbol{n}^T\boldsymbol{v} - n_0$. We denote the current coordinates of the vertex as $\boldsymbol{v}^t$ and the desired coordinates as $\boldsymbol{v}^{t+\Delta t}$ (also for $\boldsymbol{n}$ and n_0) and compute the corresponding distances d^t and $d^{t+\Delta t}$. In contrast to a static collision detection algorithm which would only take the desired coordinates under consideration we interpolate the distance linearly: $d(t) = d^t + t/\Delta t \cdot (d^{t+\Delta t} - d^t)$. In the case $d^t > d^{t+\Delta t}$ we compute $t_c = \Delta t d^t/(d^t - d^{t+\Delta t})$. If $d^t \leq d^{t+\Delta t}$ the vertex moves parallel to or away from the face or comes from the "wrong" side ($d^t < 0$). t_c is the time of collision between $\boldsymbol{v}$ and the plane in which F lies. If $t_c < 0$ or $t_c > \Delta t$ there is no collision during the desired movement. Otherwise we have to check whether $\boldsymbol{v}$ will lie in F at time t_c or not. In order to obtain the coordinates of $\boldsymbol{v}$ and F at this time we interpolate the position and orientation parameters of both objects linearly. For reasons of stability we do not check whether $\boldsymbol{v}$ will lie exactly in F but in the prism defined by F and its normal $\boldsymbol{n}$.

Edge-edge collision Now we are given two edges e_1 and e_2 with endpoints $\boldsymbol{a}_1, \boldsymbol{b}_1$ and $\boldsymbol{a}_2, \boldsymbol{b}_2$, respectively. Let g_1 and g_2 be the lines in which e_1 and e_2 lie. Let furthermore F be a face adjacent to e_2 and assume that F lies in the plane with the equation $\boldsymbol{n}^T\boldsymbol{x} = n_0$. We cannot define a signed distance between the edges as we did in the vertex-face case. But we observe that e_1 and e_2 cannot collide if both endpoints of e_1 are always on the same side of the face F (unless e_1 is always parallel to F. In this case we interchange the roles of e_1 and e_2). So we determine all maximal intervals $I = [t_1, t_2]$ for which $\boldsymbol{a}_1$ and $\boldsymbol{b}_1$ lie on different sides of F, that means such that for all $t \in I$ we have

$$\begin{array}{rcll} \boldsymbol{n}(t)^T\boldsymbol{a}_1(t) - n_0(t) & < & 0 & \text{and} \\ \boldsymbol{n}(t)^T\boldsymbol{b}_1(t) - n_0(t) & > & 0 & \text{or vice versa.} \end{array} \tag{1}$$

Here the time argument t means that the coordinates are interpolated linearly. For all these intervals we do the following. First we determine the (signed) distances d_1 and d_2 between the lines g_1 and g_2 at time t_1 and t_2, respectively:

$$d_i = \frac{\det(\boldsymbol{b}_1 - \boldsymbol{a}_1, \boldsymbol{b}_2 - \boldsymbol{a}_2, \boldsymbol{a}_1 - \boldsymbol{a}_2)}{|(\boldsymbol{b}_1 - \boldsymbol{a}_1) \times (\boldsymbol{b}_2 - \boldsymbol{a}_2)|} \tag{2}$$

where $\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{a}_2, \boldsymbol{b}_2$ are evaluated at time t_i for $i = 1, 2$. As in the vertex-face test we interpolate the distances linearly and compute $t_c = d_1/(d_1 - d_2)$. As the inequations (1) ensure that g_1 and g_2 are non-parallel, t_c is the time of collision between these lines. If $t_c < t_1$ or $t_c > t_2$ there is no collision between g_1 and g_2 and hence e_1 and e_2 do not collide. Otherwise we have to check whether the point of intersection lies on the edges or not. As before we use linear interpolation to determine the positions of the edges at time t_c. For reasons of stability we do not compute the point of intersection (because of the interpolations the lines might not intersect exactly) but the pair of closest points on the lines.

2.4 Distance Calculation

Measuring proximity between moving bodies plays an important role in robot motion planning and collision detection for non-interactive physical simulations [Mir96]. Dynamical and proximity information allow the computation of lower bounds of the time of impact between two objects in motion. Prioritizing object pairs according to this estimation yields time intervals during which the dynamical system can be evaluated and a collision-free movement of the objects is guaranteed.

The fastest algorithms so far, presented in [LC91], [Mir97], [Cam97] focus on convex objects. Non-convex objects have to be partitioned in convex pieces, which itself is a non-trivial problem. The algorithm implemented in SiLVIA does not rely on the property of convex objects.

Comparable with the collision detection algorithm described above, we try to cull away the parts of the object which do not contribute to find the optimal solution. We use the hierarchy of bounding volumes to solve the distance minimization problem according to a *branch-and-bound* strategy. A lower bound of the distance between two node face sets is given by the distance between the corresponding bounding volumes. As we try to cut off subtrees, we also need an upper bound to which we can compare the bounding volume distance. The upper bound corresponds to the minimal distance computed between two faces of both objects so far. This provides us with a simple rejection test. If the distance between the bounding volumes is not less than the current upper bound, we can prune the respective subtrees. The elementary distance computation invoked at a leaf pair computes the smallest distance between the respective face pairs using efficient edge-to-edge-distance and vertex-to-face-distance routines.

The availability of lower bounds of distance between node pairs is exploited to create heuristic strategies determining the order of the child node pairs to be visited. We developed two dominating strategies. The first can be characterized as a local acting greedy heuristic choosing the child pair with minimum bounding volume distance.

The second one stores all visited node pairs, keyed by their bounding volume distance, in a sorted sequence structure. At every iteration we expand the node pair with lowest key and after updating the upper bound we cut off all pairs at the end of the sequence with key greater or equal to the current upper bound. In comparison to the former strategy the more global view of the latter reduces the number of visited nodes but the sorted sequence insertion costs lead to slightly higher running times.

A considerable advantage of the second heuristic is the possibility of maintaining a lower bound of distance between the objects that is continuously improved during the tree traversal. This is particularly important when it comes to fulfil real-time demands. In collision detection applications conservative estimations on distances are useful when there is no time left to compute exact proximity information. This property allows adaptiveness by assigning the current available computation time to the distance calculation routine and getting a conservative bound on the optimal distance when the time assignment is consumed.

Adaptiveness is not the only aspect our algorithm differs from related work. We do not restrict ourselves to the usage of a particular type of bounding volumes. The data structures and algorithms are designed with regard to high flexibility concerning the usage of bounding volumes. The distance calculation algorithm abstracts from the underlying bounding volume type making the system extensible. In SiLVIA we implemented lower bound computations for the bounding volume types mentioned above and evaluated the performance of the distance calculation using different types. We observed that in most cases the extremely fast lower bound computations between spheres do pay for their rather worse approximation properties.

3 CALCULATION OF CONTACT FORCES

In this section we want to consider an important aspect of the constraint-based dynamics simulation: the calculation of the contact forces. The contact forces prevent the objects from interpenetrating at the points of collision and influence their velocities. Let us consider a multi-body system consisting of n rigid bodies in mutual contact at K contact points. Suppose that body B_{i_k} touches body B_{j_k} at the k-th contact point $\boldsymbol{p}_k$. The interpenetration of these bodies at $\boldsymbol{p}_k$ is prevented by a pair of opposite directed contact forces $\pm\boldsymbol{F}_k = \pm f_k \boldsymbol{n}_k$. In the absence of friction $\boldsymbol{F}_k$ acts in direction of the normal vector $\boldsymbol{n}_k$ of the contact plane, which is tangential to the surface of both objects at the contact point. Let the vector $\boldsymbol{r}_{kl} = \boldsymbol{p}_k - \boldsymbol{c}_l$ point from the center of mass of object B_l to the k-th contact point. The position and orientation of object B_l are described by the vector $\boldsymbol{c}_l$ and a quaternion $\boldsymbol{q}_l$. $\boldsymbol{v}_l$ and $\boldsymbol{\omega}_l$ specify the linear and angular velocity of body B_l and m_l and $\mathbf{I}_l$ denote its mass and its inertia matrix. If B_l is fixed in space, then we set $\boldsymbol{v}_l = \boldsymbol{\omega}_l = \mathbf{0}$ and $m_l = \infty$ and $\mathbf{I}_l = \infty\mathbf{E}$, where $\mathbf{E} \in I\!R^{3\times 3}$ is the identity matrix. If B_l is interactively moved we interpret the user's input as velocities and set $\boldsymbol{v}_l$ and $\boldsymbol{\omega}_l$ appropriately.

The generalized Newton-Euler equations of motion for the objects B_l are:

$$
\begin{aligned}
\dot{\boldsymbol{c}}_l &= \boldsymbol{v}_l, \qquad \dot{\boldsymbol{q}}_l = \frac{1}{2}\boldsymbol{\omega}_l \boldsymbol{q}_l, \\
\dot{\boldsymbol{v}}_l &= m_l^{-1} \sum_{\{k | j_k = l\}} \boldsymbol{F}_k - m_l^{-1} \sum_{\{k | i_k = l\}} \boldsymbol{F}_k + \boldsymbol{g}, \\
\dot{\boldsymbol{\omega}}_l &= \mathbf{I}_l^{-1} \sum_{\{k | j_k = l\}} \boldsymbol{r}_{kl} \times \boldsymbol{F}_k \\
&- \mathbf{I}_l^{-1} \sum_{\{k | i_k = l\}} \boldsymbol{r}_{kl} \times \boldsymbol{F}_k - \mathbf{I}_l^{-1} \boldsymbol{\omega}_l \times \mathbf{I}_l \boldsymbol{\omega}_l
\end{aligned}
$$

Our objective is to determine the constraint-forces $\boldsymbol{F}_k$ for $k = 1, \ldots, K$. To give up the component-wise description the following vectors and matrices are quite useful: the generalized velocity vector $\boldsymbol{u} = [\boldsymbol{v}_1, \boldsymbol{\omega}_1, \ldots, \boldsymbol{v}_n, \boldsymbol{\omega}_n]^T \in I\!R^{6n}$, the generalized position vector $\boldsymbol{s} = [\boldsymbol{c}_1, \boldsymbol{q}_1, \ldots, \boldsymbol{c}_n, \boldsymbol{q}_n]^T \in I\!R^{7n}$, the vector of the magnitudes of the contact forces $\boldsymbol{f} = [f_1, f_2, \ldots, f_K]^T \in I\!R^K$, the vector of external forces $\boldsymbol{f}_{ext} = [m_1\boldsymbol{g}, -\boldsymbol{\omega}_1 \times \mathbf{I}_1 \boldsymbol{\omega}_1, \ldots, m_n\boldsymbol{g}, -\boldsymbol{\omega}_n \times \mathbf{I}_n \boldsymbol{\omega}_n]^T \in I\!R^{6n}$, the matrix

$$\mathbf{S} = \mathrm{diag}(\mathbf{E}, \mathbf{Q}_1, \ldots, \mathbf{E}, \mathbf{Q}_n) \in I\!R^{7n \times 6n}$$

where $\mathbf{Q}_l \in I\!R^{4\times 3}$ imitates the quaternion product $\frac{1}{2}\boldsymbol{\omega}_l \boldsymbol{q}_l = \mathbf{Q}_l \boldsymbol{\omega}_l$, the generalized mass matrix

$$\mathbf{M} = \mathrm{diag}(m_1\mathbf{E}, \mathbf{I}_1, \ldots, m_n\mathbf{E}, \mathbf{I}_n) \in I\!R^{6n \times 6n}$$

and the matrix of contact conditions $\mathbf{C} \in I\!R^{6n \times K}$. The transposed matrix $\mathbf{C}^T$ has the following structure indicated by its k-th row:

$$
\begin{array}{c}
\begin{array}{ccccccc}
 & \overset{2i_k - 1}{\downarrow} & \overset{2i_k}{\downarrow} & & \overset{2j_k - 1}{\downarrow} & \overset{2j_k}{\downarrow} & \\
\end{array} \\
\left[\begin{array}{ccccccc}
\mathbf{0}\ldots\mathbf{0} & -\mathbf{n}_k & -\mathbf{n}_k \times \mathbf{r}_{ki_k} & \mathbf{0}\ldots\mathbf{0} & \mathbf{n}_k & \mathbf{n}_k \times \mathbf{r}_{kj_k} & \mathbf{0}\ldots\mathbf{0}
\end{array}\right]
\end{array}
$$

Using this notation, the equations of motion can be formulated in their continuous and discretized (Euler-scheme) version:

$$
\begin{aligned}
\dot{\boldsymbol{s}} &= \mathbf{S}\boldsymbol{u} \\
\dot{\boldsymbol{u}} &= \mathbf{M}^{-1}(\mathbf{C}\boldsymbol{f} + \boldsymbol{f}_{ext}) \\
\Rightarrow \boldsymbol{s}^{t+\Delta t} &= \boldsymbol{s}^t + \Delta t \mathbf{S} \boldsymbol{u}^{t+\Delta t} & (3) \\
\boldsymbol{u}^{t+\Delta t} &= \boldsymbol{u}^t + \Delta t \mathbf{M}^{-1}(\mathbf{C}\boldsymbol{f} + \boldsymbol{f}_{ext}) & (4)
\end{aligned}
$$

Now we define the function $\boldsymbol{\delta} : I\!R^{7n} \to I\!R^K$ that computes the contact distances $\boldsymbol{\delta}(\boldsymbol{s}) = [\delta_1, \ldots, \delta_K]^T$ for the generalized position vector $\boldsymbol{s}$. Then δ_k is the distance between the parts of the objects involved in the k-th (potential) contact. Note, that $\boldsymbol{\delta}$ is non-linear.

Finally, we need the function $\boldsymbol{\sigma} : \mathbb{R}^K \to \mathbb{R}^{7n}$ taking the magnitudes of the contact forces as arguments and computing the configuration $\boldsymbol{s}^{t+\Delta t}$. We obtain $\boldsymbol{\sigma}$ by inserting equation (4) into equation (3):

$$\boldsymbol{\sigma}(\boldsymbol{f}) = \boldsymbol{s}^t + \Delta t \mathbf{S}(\boldsymbol{u}^t + \Delta t \mathbf{M}^{-1}(\mathbf{C}\boldsymbol{f} + \boldsymbol{f}_{ext}))$$

3.1 A Naive Method

If we assume that all contacts are bilateral, the condition

$$\boldsymbol{\delta}(\boldsymbol{\sigma}(\boldsymbol{f})) = \mathbf{0} \tag{5}$$

with $\mathbf{0} = [0, \ldots, 0]^T \in \mathbb{R}^K$ must hold. This is a non-linear equation system with the contact forces as unknown quantities. E.g. it can be solved by the Newton-Raphson method, which requires the successive solution of a K dimensional linear system of equations. The number of iteration steps is small because we know good start values for $\boldsymbol{f}$ from the previous time step. Since the contacts are not really bilateral, we release the k-th contact if f_k becomes negative.

3.2 Reduction to a System of Equations

Now suppose that all contacts are unilateral. Then a disadvantage of the first method is that we allow negative contact forces for one simulation step, which cause the objects to stick together for a short time. We can avoid this by replacing condition (5) by the complementarity condition

$$\boldsymbol{\delta}(\boldsymbol{\sigma}(\boldsymbol{f})) \geq \mathbf{0} \quad \text{compl. to} \quad \boldsymbol{f} \geq \mathbf{0}. \tag{6}$$

Note that '$\boldsymbol{a}$ compl. to $\boldsymbol{b}$' is equivalent to $\boldsymbol{a}^T\boldsymbol{b} = 0$ for $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$. Condition (6) means that we do not allow negative distances (i.e. interpenetration) or negative (i.e. attractive) contact forces and that at each contact point the distance or the force must be equal to zero. In order to solve this non-linear complementarity problem (NCP) we use the same technique as in [BS98] and consider the so-called Fischer function $\varphi : \mathbb{R}^2 \to \mathbb{R}$ which is defined by $\varphi(a, b) = \sqrt{a^2 + b^2} - a - b$. It is obvious that the following holds for each $a, b \in \mathbb{R}$:

$$\varphi(a, b) = 0 \Longleftrightarrow a \geq 0 \quad \text{compl. to} \quad b \geq 0.$$

We define the function $\boldsymbol{\Phi} : \mathbb{R}^{2K} \to \mathbb{R}^K$ by

$$\boldsymbol{\Phi}(\boldsymbol{f}) = [\varphi(f_1, \delta_1(\sigma(\boldsymbol{f}))), \ldots, \varphi(f_K, \delta_K(\sigma(\boldsymbol{f})))]^T$$

Now we can transform the NCP into the non-linear equation system which can again be solved by the Newton-Raphson method.

$$\boldsymbol{\Phi}(\boldsymbol{f}) = \mathbf{0}, \tag{7}$$

In contrast to the classical method [Bar94] this approach uses the contact distances instead of the contact accelerations as variables complementary to the contact forces. In this way the integration of the motion equations can be performed in a stable way with respect to the geometric constraints. The classical method however has to face the problem that the deviations from the exact constraints accumulate during the integration process.

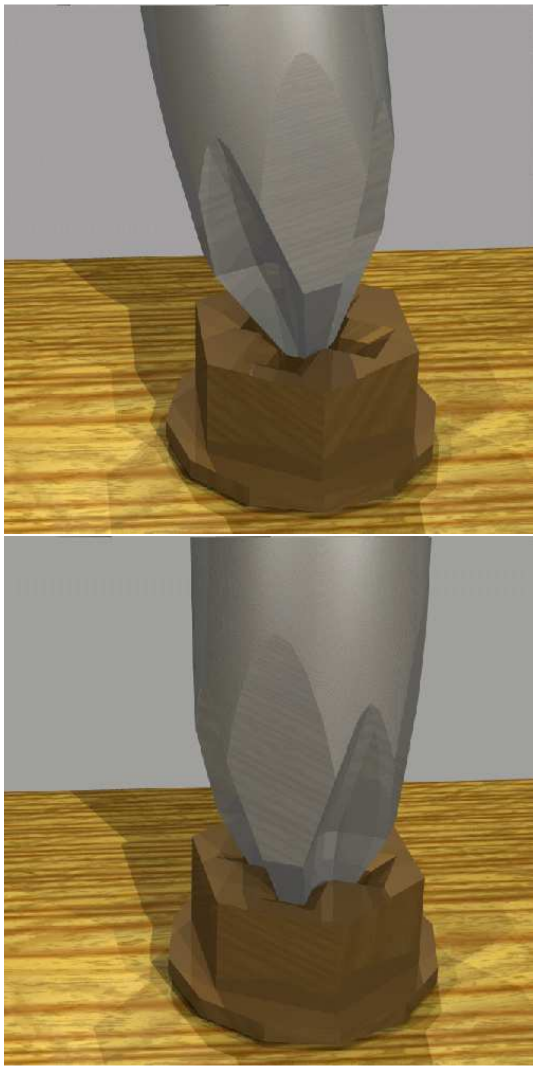

Figure 3: An example for a simulation by SiLVIA.

3.3 Evaluation

By analogy with the example in the introduction we have chosen a typical fitting operation as an evaluation example. We designed a crosshead screw and screwdriver according to the international norm ISO 4757 "Cross recesses for screws". This norm specifies the measurements and geometric tolerances for both objects and ensures that the screwdriver correctly fits into the screw. Then we simulated the fitting and twisting process, as you can see in figure 3. The user exerted a light pressure while twisting the screwdriver, which automatically found its way into the proper position as soon as the first contact was established. Although the scene only consisted of circa 200 polygons, the resulting geometric contact configurations were quite complex. Up to 20 mutual contacts between the screwdriver, the screw and the underlying block were nothing unusual.

Although a strict proof of the convergence of the approaches presented in section 3.2 is still missing the empirical results indicate that the methods behave well in spite of large time steps.

4 CONCLUSIONS AND FURTHER RESEARCH

A simulation system for interactive virtual assembly planning should provide different levels of abstraction from the real physical laws: At the base level all objects are treated as rigid bodies, which are forbidden to penetrate each other. The next level comprises the modelling of kinematic constraints, represented by bilateral and unilateral contacts. Based upon that the dynamics of the system of rigid bodies can be simulated, i. e. inertial and gravitational forces, contact forces, and interactively exerted forces and torques come into play. A further step is the modelling of plastic and elastic impacts still assuming the rigidity of the bodies. The next major step to improve the physical model is the integration of friction. But an even more subtle point is the modelling of deformable objects.

In this paper we presented some first steps on the long way to attain these ends.

BIOGRAPHY

Elmar Schömer is a research assistant at the computer science department of Saarland University where he received his doctorate in computer science in 1994. His fields of research include computational geometry and computational mechanics. He is currently working on algorithms for interactive virtual assembly planning in cooperation with the Virtual Reality Competence Center of DaimlerChrysler.

REFERENCES

[Bar94] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *SIGGRAPH*, pages 174–203, July 1994.

[BS98] M. Buck and E. Schömer. Interactive rigid body manipulation with obstacle contacts. In 6^{th} *Int. Conference in Central Europe on Computer Graphics and Visualization, WSCG'98*, pages 49–56, 1998.

[Cam97] S. Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *IEEE Int. Conf. Robotics and Automation*, 1997.

[CK86] R.K. Culley and K.G. Kempf. A collision detection algorithm based on velocity and distance bounds. In *IEEE International Conference of Robotics and Automation*, pages 1064–1068, 1986.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, pages 171–180, August 1996. Proc. SIGGRAPH'96.

[HKM+96] M. Held, J. T. Klosowski, J. S. B. Mitchell, H. Sowizraland, and K. Zirkan. Efficicient collision detection using bounding volume hierarchies of k-DOPs. In *ACM SIGGRAPH'96 Visual Proceedings New Orleans*, August 1996.

[Hub95] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. on Visual. and Comput. Graph.*, 1(3):218–230, September 1995.

[Hub96] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996.

[KPLM98] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shells: A higher order bounding volume for fast proximity queries. In *Proc. 1998 Workshop Algorithmic Found. Robot.*, 1998.

[LC91] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 1008–1014, 1991.

[Löt82] P. Lötstedt. Mechanical systems of rigid bodies subject to unilateral constraints. *SIAM Journal of Applied Mathematics*, 42(2):281–296, 1982.

[Mir96] B. Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD thesis, University of California, Berkeley, 1996.

[Mir97] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, MERL, 201 Broadway, Cambridge, MA 02139, USA, 1997.

[Ove92] M. H. Overmars. Point location in fat subdivisions. *Inform. Process. Lett.*, 44:261–265, 1992.

[PG95] I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere-trees. *Proc. Eurographics*, 14(2):105–116, 1995.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

[SS98] J. Sauer and E. Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. In *Proc. ACM Symposium on Virtual Reality Software and Technology*, pages 153–161, 1998.

[Zac98] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE, VRAIS'98 Atlanta*, March 1998.

[ZF95] G. Zachmann and W. Felger. The BoxTree: Enabling real time and exact collision detection of arbitrary polyhedra. In 1^{st} *Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.