Rigid Body Contact: Collision Detection to Force Computation

Brian Mirtich

TR98-01 December 1998

Abstract

The most difficult aspect of rigid-body simulation is contact modeling. Two major subproblems of contact modeling are detecting contacts and computing contact forces. Although typically studied in isolation, these problems are tightly intertwined in simulation. This paper describes a method for fitting the pieces together. It gives an overview of the V-Clip collision detection algorithm, which can rapidly and robustly locate pairs of closest features between polyhedral models. Using the results from multiple invocations of V-Clip, persistent, one- and two-dimensional contact regions can be modeled. The paper briefly describes several methods for computing contact forces that act between bodies. It also presents a new, easy to implement method, based on the singular value decomposition of the contact matrix.

Workshop on Contact Analysis and Simulation, IEEE International Conference on Robotics and Automation, May 1998

MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Rigid Body Contact: Collision Detection to Force Computation

Brian Mirtich

TR-98-01 March 1998

Abstract

The most difficult aspect of rigid-body simulation is contact modeling. Two major subproblems of contact modeling are detecting contacts and computing contact forces. Although typically studied in isolation, these problems are tightly intertwined in simulation. This paper describes a method for fitting the pieces together. It gives an overview of the V-Clip *collision detection algorithm, which can rapidly and robustly locate pairs of closest features between polyhedral models. Using the results from multiple invocations of* V-Clip, *persistent, one- and two-dimensional contact regions can be modeled. The paper briefly describes several methods for computing contact forces that act between bodies. It also presents a new, easy to implement method, based on the singular value decomposition of the contact matrix.*

Presented at Workshop on Contact Analysis and Simulation, IEEE International Conference on Robotics and Automation, May 1998

1. First printing, TR98-01, March 1998

1 Introduction

Since classical methods exist for computing the motion of rigid and articulated bodies in isolation, the most difficult aspect of general rigid body simulation is modeling the transient contacts that form between moving bodies. Two major subproblems of contact modeling are detecting contacts, often termed collision detection, and computing contact forces. This paper summarizes a new collision detection algorithm, designed for robustness and speed, that handles rigid polyhedral models. Often the issue of collision detection is considered independently from the issue of contact force computation, which is unfortunate since the processes are tightly intertwined in simulation. After describing the collision detection algorithm, this paper goes on to show how its output is connected to the computation of contact forces. Three families of methods for computing contact forces are briefly described, and a fourth, newer method is explained in more detail. This new method is easy to implement, and has been applied successfully to some simple simulation problems, but more general testing of it is needed to evaluate its practicality.

The contact detection and tracking methods described here operate on polyhedral object models. The advantage of this representation is its generality: any shape can be represented within an arbitrary tolerance by a polyhedral model. The algorithms presented are insensitive to the complexity of the polyhedral model, and so it is not detrimental from an efficiency standpoint to use highly tesselated models. In some situations, surface curvature plays a significant role in contact dynamics, but polyhedral models do not admit accurate curvature measurements. Nonetheless, previous experience suggests that highly tesselated polyhedral models are sufficient to model phenomena such as rolling spheres and cylinders [21].

A more significant limitation of the algorithms presented here is that they are designed for *piecewise convex* models. Nonconvex models are allowed if they are specified as a hierarchy of convex pieces. Some objects are difficult to represent in this manner. For these, other types of collision detection algorithms are more suitable, such as those based on octrees [1], binary space partitioning trees [24], sphere hierarchies [14], or oriented bounding boxes [12, 15]. Many of these algorithms, however, do not return an accurate estimate of the distance between objects nor the closest points between them, which is a drawback for simulation applications. They are also logarithmic in the polyhedral complexity when models are in close proximity.

2 *V-Clip* collision detection

Several algorithms have been developed for collision detection between polyhedral models specified by a boundary representation. The most recent algorithms cache witnesses that are used to verify disjointness or penetration in constant time. By exploiting the coherence properties of bodies moving continuously through space, these algorithms achieve near constant time performance.

The popular *Lin-Canny closest features* algorithm [17] computes the distance between disjoint polyhedra; it is among the fastest solutions for this problem. Convex polyhedra are decomposed into vertex, edge and face features. The algorithm tracks and caches the closest features between a pair of convex polyhedra. The closest points between the polyhedra are computable from the closest features.

Lin-Canny has two significant drawbacks. The first is that it does not readily handle the case of penetrating polyhedra. This makes it difficult to use *Lin-Canny* in the usual collision detection paradigm: detecting collisions after penetration occurs, and then backtracking to find the exact point of collision. The second drawback is a lack of robustness. *Lin-Canny* is prone to cycling behavior when presented with models in degenerate configurations. The algorithm is difficult to code, and it is also difficult to tune the various floating point tolerances. Despite these problems, *Lin-Canny's* speed, and availability through the *I-Collide* package [9] have made it a popular choice for collision detection applications.

Gilbert, Johnson and Keerthi developed a simplex-based algorithm for finding the distance between two disjoint convex polyhedra, or the distance by which they penetrate [11]. Given two polyhedra, *GJK* searches for a simplex, defined by four vertices of the Minkowski difference polyhedron, that either encloses or is nearest to the origin. Others have made several improvements on the basic *GJK* algorithm, mostly to improve efficiency [27, 8, 7]. *GJK* algorithms are more robust than *Lin-Canny*, and handle the penetration case, but the implementations generally require more floating point operations.

The Voronoi clip, or *V-Clip* algorithm is a closest features algorithm that bears a family resemblance to its ancestor, *Lin-Canny*, but which overcomes many of the limitations of the latter: *V-Clip* efficiently handles the penetration case; *V-Clip* is very robust, has no numerical tolerances, and does not exhibit cycling problems; and *V-Clip* is simpler to implement. *V-Clip's* operation count is comparable with *Lin-Canny's* and generally lower than that of Cameron's *Enhanced GJK* algorithm [7]. The remainder of this section briefly motivates the *V-Clip* algorithm; more details are given in [22].

2.1 Features and Voronoi regions

The boundary of a convex polyhedron comprises vertices, edges, and faces; these *features* are convex sets. The neighbors of a vertex are the edges incident to that vertex. The neighbors of a face are the edges bounding the face. An edge has exactly four neighbors: the two vertices at its endpoints and the the two faces it bounds. Note that the neighbor relation is symmetric. Polyhedral features are treated as closed sets. Hence, a face includes the edges that bound it, and an edge includes its vertex endpoints. Voronoi regions and planes (see Figure 1) are central to the *V-Clip* algorithm:

Definition 1 *For feature X on a convex polyhedron, the* **Voronoi region** *$\mathcal{VR}(X)$ is the set of points outside the polyhedron that are as close to X as* *to any other feature on the polyhedron. The* **Voronoi plane** $\mathcal{VP}(X, Y)$ *between neighboring features* X *and* Y *is the plane containing* $\mathcal{VR}(X) \cap \mathcal{VR}(Y)$.

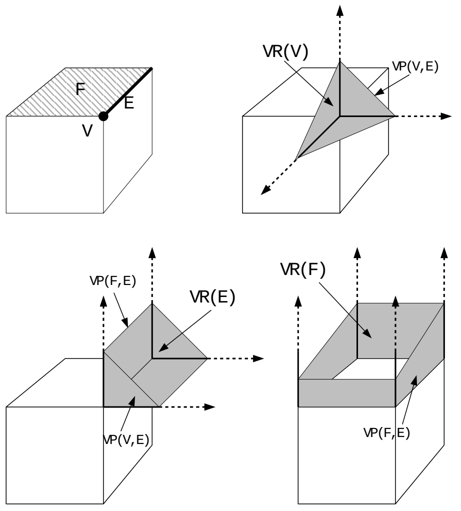

Figure 1: Top left: *A cubical polyhedron. Among its features are face* F*, edge* E*, and vertex* V. Top right: *The Voronoi region* $\mathcal{VR}(V)$*. One of the Voronoi planes bounding this region is* $\mathcal{VP}(V, E)$*, corresponding to* V*'s neighboring edge* E. Bottom left: *The Voronoi region* $\mathcal{VR}(E)$*. Two of the Voronoi planes bounding this region are* $\mathcal{VP}(V, E)$ *and* $\mathcal{VP}(F, E)$*, corresponding respectively to* E*'s neighboring features* V *and* F. Bottom right: *The Voronoi region* $\mathcal{VR}(F)$*. One of the Voronoi planes bounding this region is* $\mathcal{VP}(F, E)$*, corresponding to* F*'s neighboring edge* E*. The support plane of* F *itself also bounds* $\mathcal{VR}(F)$.

All Voronoi regions are bounded by Voronoi planes, and the regions collectively cover the entire space outside the polyhedron. Voronoi planes between neighboring features come in two varieties: vertex-edge and face-edge planes. Vertex-edge planes contain the vertex and are normal to the edge, while face-edge planes contain the edge and are parallel to the face normal (Figure 1).

2.2 Algorithm overview

The *V-Clip* algorithm is based on a fundamental theorem that is proved in [22]:

Theorem 1 *Let* X *and* Y *be a pair of features from disjoint convex polyhedra, and let* $\mathbf{x} \in X$ *and* $\mathbf{y} \in Y$ *be the closest points between* X *and* Y*. If* $\mathbf{x} \in \mathcal{VR}(Y)$ *and* $\mathbf{y} \in \mathcal{VR}(X)$*, then* $\mathbf{x}$ *and* $\mathbf{y}$ *are a globally closest pair of points between the polyhedra.*

Theorem 1 does not require the closest points on X and Y to be unique, and in degenerate situations they are not. If the conditions of the theorem are met, however, no pair of points from the two polyhedra are any closer than $\mathbf{x}$ and $\mathbf{y}$. Like *Lin-Canny*, the *V-Clip* algorithm is essentially a search for two features that satisfy the conditions of Theorem 1. At each iteration, *V-Clip* tests whether the current pair of features satisfy the conditions, and if not, updates one of the features, usually to a neighboring one.

The state diagram of Figure 2 illustrates the algorithm. Each state corresponds to a possible combination of feature types, for example, the *V-F* state means one feature is a vertex, and the other is a face. The arrows denote possible update steps from one state to another. Solid arrows mark updates that decrease the inter-feature distance; dashed arrows mark updates for which the inter-feature distance stays the same. The four primary states of the algorithm are *V-V*, *V-E*, *E-E*, and *V-F*; it may terminate in any one of these states. The fifth state, *E-F*, is special in that the algorithm cannot terminate in this state unless the polyhedra are penetrating. Figure 2 implies that the algorithm must terminate, for since there are no cycles in the graph comprising only dashed arrows, any infinite path through the graph would contain an infinite number of strict reductions in inter-feature distance, which is impossible.

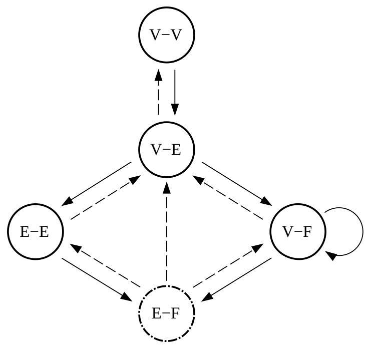

Figure 2: *States and transitions of the* V-Clip *algorithm.*

One reason for *V-Clip's* robustness in the face of degeneracy is that the face-face case, which is the most problematic case in *Lin-Canny*, is not allowed. The edge-face case, also somewhat problematic in *Lin-Canny*, is only used as a terminating state when the objects penetrate. There is a tradeoff involved. An algorithm like *Lin-Canny* can return more information to the caller. If the closest features are two parallel faces, F_1 and F_2, the caller will be informed of such. *V-Clip* is less informative. It will return some pair of features that are no farther apart than F_1 and F_2, but that are of lower total dimension, such as F_1 and one of the vertices of F_2. This has the undesirable effect of leaving more work to the application, as will be seen in Section 3, however, it provides a robustness to the collision detection algorithm. Another reason for *V-Clip's* robustness is that, unlike *Lin-Canny*, it never actually computes the points $\mathbf{x}$

and $\mathbf{y}$ of Theorem 1 during the search for closest features,[1] except in the trivial case of vertex features. The next section gives an example that shows how this is possible.

2.3 Voronoi clipping example

Suppose that the current pair of closest features is an edge E from one polyhedron, and a face F from the other. To check the condition of Theorem 1, we need to determine if the closest point on E to F lies within the $\mathcal{VR}(F)$. If E's endpoints are $\mathbf{t}$ (for tail) and $\mathbf{h}$ (for head), then $E \cap \mathcal{VR}(F)$ is either empty, or a line segment along E:

$$(1-\lambda)\mathbf{t} + \lambda\mathbf{h}, \quad \underline{\lambda} \leq \lambda \leq \overline{\lambda}.$$

The values of $\underline{\lambda}$ and $\overline{\lambda}$ are simple to calculate. One simply enumerates over each Voronoi plane P bounding $\mathcal{VR}(F)$. If ever $\mathbf{t}$ and $\mathbf{h}$ lie on opposite sides of P, an interpolation operation determines the exact value of λ at which E crosses the plane P. After discarding all of the sections of E that are clipped by the Voronoi planes, what remains is either a null interval of E, indicating that E is disjoint from $\mathcal{VR}(F)$, or the bounds $\underline{\lambda}$ and $\overline{\lambda}$ that define the portion of E within $\mathcal{VR}(F)$. Algorithm 1 gives a formal description of the process. It computes the values $\underline{\lambda}$ and $\overline{\lambda}$, and also the neighboring features $\underline{N}$ and $\overline{N}$ of F that correspond to the planes that clip E: E enters $\mathcal{VR}(F)$ as it crosses $\mathcal{VP}(X, \underline{N})$, and exits as it crosses $\mathcal{VP}(X, \overline{N})$ (Figure 3). The algorithm returns TRUE if and only if $E \cap \mathcal{VR}(F) \neq \emptyset$. The divisions that occur in steps 11 and 18 are the only divisions that occur in the entire *V-Clip* algorithm. In these cases, the divisor's magnitude must be nonzero and never less than the dividend's magnitude, thus no overflow can occur.

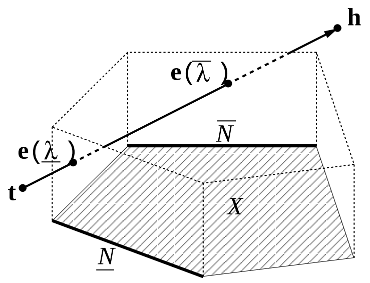
h
$\mathbf{e}(\overline{\lambda})$
$\overline{N}$
$\mathbf{e}(\underline{\lambda})$
t
X
$\underline{N}$

Figure 3: *Clipping an edge against planes bounding the Voronoi region of a pentagonal face X. The edge enters the region $\mathcal{VR}(F)$ as it crosses $\mathcal{VP}(X, \underline{N})$ at the parameterized point $\mathbf{e}(\underline{\lambda})$; it exits $\mathcal{VR}(F)$ as it crosses $\mathcal{VP}(X, \overline{N})$ at point $\mathbf{e}(\overline{\lambda})$.*

[1]Most of *Lin-Canny's* robustness problems stem from checking the conditions of Theorem 1 with explicitly computed closest points when these points are not unique.

Algorithm 1 *clipEdge. Clip the edge from* $\mathbf{t}$ *to* $\mathbf{h}$ *against the Voronoi region in* $\mathcal{VR}(F)$*. Return FALSE if the edge is completely clipped, otherwise TRUE.*

```
$\underline{\lambda} \leftarrow 0$; $\overline{\lambda} \leftarrow 1$
$\underline{N} \leftarrow \overline{N} \leftarrow \emptyset$
for all Voronoi planes $P$ bounding $\mathcal{VR}(F)$ do
  $N \leftarrow$ neighboring edge of $F$ corresponding to $P$.
  $d_t \leftarrow D_P(\mathbf{t})$
  $d_h \leftarrow D_P(\mathbf{h})$
  if $d_t < 0$ and $d_h < 0$ then
    $\underline{N} \leftarrow \overline{N} \leftarrow N$
    return FALSE
  else if $d_t < 0$ then
    $\lambda \leftarrow d_t/(d_t - d_h)$
    if $\lambda > \underline{\lambda}$ then
      $\underline{\lambda} \leftarrow \lambda$
      $\underline{N} \leftarrow N$
      if $\underline{\lambda} > \overline{\lambda}$ then
        return FALSE
  else if $d_h < 0$ then
    $\lambda \leftarrow d_t/(d_t - d_h)$
    if $\lambda < \overline{\lambda}$ then
      $\overline{\lambda} \leftarrow \lambda$
      $\overline{N} \leftarrow N$
      if $\underline{\lambda} > \overline{\lambda}$ then
        return FALSE
return TRUE
```

After clipping E against $\mathcal{VR}(F)$, it is necessary to determine whether or not the closest point on E to F lies within $\mathcal{VR}(F)$. In the case where E lies completely within or completely outside $\mathcal{VR}(F)$, this question is trivially answered. In the more general case, the answer can be inferred *without actually computing the closest point* by examining the derivatives of a certain distance function at the values $\underline{\lambda}$ and $\overline{\lambda}$.

Definition 2 *Let E be an edge, and $\mathbf{e}(\lambda)$ a parameterized point along it. For a polyhedral feature X, the* **edge distance function** $D_{E,X}(\lambda) : \Re \rightarrow \Re$ *is defined as*

$$D_{E,X}(\lambda) = \min_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{e}(\lambda)\|.$$

$D_{E,X}(\lambda)$ is the distance between $\mathbf{e}(\lambda)$ and X. It can be shown that this function is continuous, convex, and differentiable at any point λ_0 such that $\mathbf{e}(\lambda_0) \notin X$. These facts imply that it is sufficient to check the signs of the derivative $D'_{E,X}(\lambda)$ at $\underline{\lambda}$ and $\overline{\lambda}$ to localize the closest point. The minimum occurs in the interval $[0, \underline{\lambda})$ if and only if $D'_{E,X}(\underline{\lambda}) > 0$; it occurs in the interval $(\overline{\lambda}, 1]$ if and only if $D'_{E,X}(\overline{\lambda}) < 0$. If neither of the conditions hold, then the minimum must occur in the interval $[\underline{\lambda}, \overline{\lambda}]$ and hence within $\mathcal{VR}(X)$. In the current example, X is a face, and the derivative signs are easy to compute. If $\hat{\mathbf{u}}$ is a unit vector in the direction of $\mathbf{h} - \mathbf{t}$, and $\hat{\mathbf{n}}$ is the unit exterior normal to the face F,

$$\text{sign}[D'_{E,X}(\lambda)] = \begin{cases} +\text{sign}(\mathbf{u} \cdot \mathbf{n}), & D_F[\mathbf{e}(\lambda)] > 0 \\ -\text{sign}(\mathbf{u} \cdot \mathbf{n}), & D_F[\mathbf{e}(\lambda)] < 0, \end{cases}$$

where $D_F[\mathbf{e}(\lambda)]$ denotes the signed distance of $\mathbf{e}(\lambda)$ from the support plane of F. The case of determining whether the closest point on an edge lies within a face's Voronoi region has been discussed in detail, but the other cases are similar. Edges may be clipped against the Voronoi regions of vertices and other edges in a similar manner, and the clipping operations are always followed by the checking of signs of distance derivatives to localize closest points.

2.4 Experimental results

Experiments were performed to compare three collision detection algorithms: *V-Clip*, *Lin-Canny*, and *Enhanced GJK*.[2] The first round of experiments counted the number of floating point operations required to track the closest features between a pair of polyhedra at various levels of coherence. The graph shown in Figure 4 is representative of these experiments: typically, *V-Clip* and *Lin-Canny* required comparable numbers of floating point operations, while *Enhanced GJK* required 50-100% more operations. The horizontal axis in the plot is a coherence parameter: higher values correspond to larger motions of the moving model between algorithm invocations.

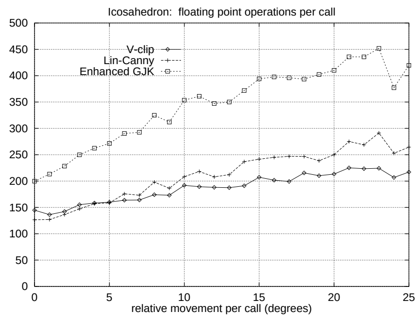

Figure 4: *Total floating-point operations for icosahedra.*

A second set of experiments tested the robustness of the algorithms in degenerate configurations. These experiments focused on critical configurations where the witnesses changed. For *Lin-Canny* and *V-Clip*, the witnesses are polyhedral

[2]Source code for all three algorithms is publicly available. *V-Clip*: see *www.merl.com/people/mirtich/vclip.html.* *Lin-Canny*: see *www.cs.berkeley.edu/~mirtich/collDet.html* or *www.cs.unc.edu/~geom/I_COLLIDE.html.* *Enhanced GJK*: see *www.comlab.ox.ac.uk/oucl/users/stephen.cameron/distances.html.*

features; for *Enhanced GJK*, they are polyhedral simplices. *Lin-Canny* exhibited problems in 4.5% of the 100,000 trials. *Enhanced GJK* exhibited problems in 0.1% of the trials when a floating point tolerance was set too tightly; using the default tolerance eliminated the problems. *V-Clip*, with no tolerances to set, exhibited no detectable problems. In short, both in terms of speed and robustness, *V-Clip* compared favorably with existing algorithms for polyhedral collision detection.

3 Handling extended contacts

In conjunction with a backtracking method, the *V-Clip* algorithm is sufficient for localizing the time at which two bodies begin to penetrate, and the closest points between the bodies just before this point. If the relative normal velocity between these points is above some small threshold, the interaction is treated as a collision, and applied collision impulses instantaneously change the body velocities, causing the bodies to separate at the collision points. See [6] for a good survey of methods for computing collision impulses algebraically, and [5, 16, 21, 29, 31] for more sophisticated methods. When the relative normal velocity between two bodies about to collide is below the threshold, the interaction is more naturally modeled as a contact. This is affected by applying a small collision impulse that brings the relative normal velocity to zero, and subsequently applying contact forces that keep the contact points from accelerating toward each other. For the remainder of this paper, *contact* means non-colliding contact, as when bodies rest on, slide along, or roll along each other.

3.1 Contact sets

A closest feature algorithm like *V-Clip* only returns information about the pair of closest features (and points) between two polyhedral models. Contact regions between three-dimensional bodies are frequently one- or two-dimensional, and so the raw output from *V-Clip* is insufficient for determining the contact region. One solution to this problem is to provide hysteresis in tracking pairs of closest features. The contact tracker maintains a *set* of contacts, each defined by a pair of features, one from each body. For each contact there is also a pair of contact points, which are the closest points between the contact features. The points are where the contact forces are applied. Consider the situation in Figure 5, where body A settles on body B, The initial contact occurs between some pair of features, such as vertex V and face F. *V-Clip* returns both this feature pair, and the closest points between the features. The contact set is initialized to contain the pair (V, F). Henceforth, equal and opposite contact forces are continuously applied at the contact points to prevent the features from penetrating. In response to this force, the bottom face of body A will continue to tilt toward the top surface of body B.

If *V-Clip* is called periodically[3] on bodies A and B while they are integrated,

[3]In our system, *V-Clip* is invoked within the inner integration loop, from the routine that

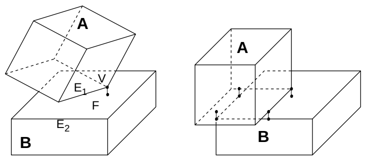

Figure 5: Left: *Initial V-F contact between bodies A and B.* Right: *Sometime later, the contact set comprises two V-F contacts, and two E-E contacts.*

penetration will be detected at some point. This triggers a binary search to locate a time t when A and B are disjoint, and the pair of closest features between them are not in the current contact set. Such a time and feature pair must exist since penetration was detected at the end of the interval, and the feature pairs in the contact set are kept apart by contact forces. For the example of Figure 5, a time t is found when the edges E_1 and E_2 are closer than V and F. At this point, the pair (E_1, E_2) is added to the contact set, and the integration proceeds onward from time t. Contact forces between E_1 and E_2 will prevent the previously detected penetration. Eventually, enough contacts are created to keep the entire objects from penetrating. The right side of Figure 5 shows a possible final resting position of bodies A and B.

3.2 Tracking contacts

A contact's feature pair remains unchanged during the lifetime of the contact, but the contact points must be tracked as they move continuously along the features. Closest points on vertex features can't move and are known trivially. This leaves only three cases to solve for:

1. *Closest point on edge to vertex.* The closest point on the edge's support line is given by
$$\mathbf{e} + [(\mathbf{v} - \mathbf{e}) \cdot \hat{\mathbf{u}}]\hat{\mathbf{u}},$$
where $\mathbf{v}$ is the location of the vertex, $\mathbf{e}$ is the location of one endpoint of the edge, and $\hat{\mathbf{u}}$ is a unit vector directed along the edge, away from $\mathbf{e}$. If this point lies between the two edge Voronoi planes perpendicular to $\hat{\mathbf{u}}$ (see Figure 1), it lies on the edge itself. Otherwise the contact is inactivated.

2. *Closest point on face to vertex.* The closest point on the face's support plane is given by
$$\mathbf{v} - [(\mathbf{v} - \mathbf{f}) \cdot \hat{\mathbf{n}}]\hat{\mathbf{n}},$$
where $\mathbf{v}$ is the location of the vertex, $\mathbf{f}$ is the location of one corner of the face, and $\hat{\mathbf{n}}$ the unit normal to the face. This point should be tested

calculates the accelerations of the contacting bodies. Profiling remains to be done, however, we expect the computational cost of rechecking the closest features to be less than the cost of computing the contact forces, which must also be done within the inner loop.

against the Voronoi planes around the face's boundary (see Figure 1). If it lies inside all of them, it is on the face itself. Otherwise the contact is inactivated.

3. *Closest points between two edges.* For each edge i, let $\mathbf{e}_i$ be the location of one endpoint of the edge and $\hat{\mathbf{u}}_i$ a unit vector directed along the edge, away from $\mathbf{e}_i$. Compute:

$$\begin{aligned} k &= \hat{\mathbf{u}}_1 \cdot \hat{\mathbf{u}}_2 \\ \mathbf{x} &= \mathbf{e}_1 + \frac{(\mathbf{e}_2 - \mathbf{e}_1) \cdot (\hat{\mathbf{u}}_1 - k\hat{\mathbf{u}}_2)}{1 - k^2} \hat{\mathbf{u}}_1. \end{aligned}$$

   Then $\mathbf{x}$ is the closest point on edge 1's support line to edge 2's support line. The closest point on edge 2's support line to edge 1's support line is found using the same approach as in Case 1, replacing $\mathbf{v}$ with $\mathbf{x}$. Both closest points are subjected to the same Voronoi checks as in Case 1. If they pass, then they lie on the edges themselves, otherwise the contact is inactivated.

The inactivation of a contact means that the features are no longer touching, as when a vertex slides off a face, for example. A contact is also inactivated if the distance between the closest points exceeds the contact threshold ε. This means that the contact features have separated and contact forces should no longer be applied. Figure 6 shows an example of contacts being tracked.

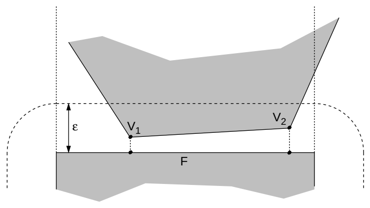

Figure 6: *Side view of two bodies in contact (the face F is perpendicular to the page). The contact set contains two contacts, defined by feature pairs (V_1, F) and (V_2, F). The former is the current pair of closest features between the bodies; it alone satisfies the conditions of Theorem 1. The pair (V_2, F) must have been the closest feature pair at some previous time when it became an active contact. If either V_1 or V_2 passes out of F's Voronoi region (bounded by the vertical dotted lines) or moves further than a distance ε from F, the corresponding contact is inactivated.*

4 Computing contact reactions

Forces or impulses must be applied at the contact points to keep contacting bodies from penetrating, and to model frictional effects. A variety of approaches have been taken for computing contact reactions, each with their own strengths and weaknesses. We briefly summarize three widely used approaches, and discuss a newer one in more detail.

4.1 Previous approaches

Penalty methods are the oldest and simplest approach to computing contact forces. These methods do not strictly enforce non-penetration; instead, they keep penetrations negligible relative to the scale of the system. A stiff spring is attached between the contact points, so that as two bodies move into one another, the spring attempts to push them apart. The more penetration, the stronger the restoring force. If the bodies move apart, the spring is destroyed. Penalty methods are used for rigid body simulation [20, 23, 28] but are most useful in deformable body simulation [10]. Penalty methods can produce stiff equations of motion due to the large spring constants needed to keep penetrations small. Choosing the spring constants is done in an ad-hoc way, and must be tailored to specific situations. Because of these drawbacks, their use use in general settings is limited.

LCP methods cast the contact force problem as a linear complementarity problem (LCP) [3, 4, 18, 19, 25, 30]. There is a linear relationship between the relative accelerations of the contact points $\mathbf{a}$ and the contact forces $\mathbf{f}$:

$$\mathbf{a} = \mathbf{A}\mathbf{f} + \mathbf{b}. \tag{1}$$

Here $\mathbf{a}$ and $\mathbf{f}$ are (respectively) stacked vectors of the relative accelerations and the contact forces between each pair of contact points. $\mathbf{A}$ and $\mathbf{b}$ are matrix and vector constants that are determined from the known configuration of the system. Inequality constraints enforce the requirements that the contact points do not accelerate toward each other and that the contact forces only push, never pull. Finally, complementarity constraints require that at each contact, either the relative normal acceleration or the normal force is zero. When there is no friction, the LCP is convex and solutions can be computed using algorithms that run in worst case exponential time but expected polynomial time in the number of contacts.

Friction can be incorporated into the same framework by modifying or adding constraints to the LCP. To do this, the usual Coulomb friction cone is linearly approximated by a pyramid. Frictional constraints destroy the convexity of the LCP and admit the possibility of having no valid solution. Baraff showed that the problem of determining the existence of a valid set of contact forces that obey Coulomb friction laws at the contacts is NP-hard when there is sliding friction [2, 3]. Pang and Trinkle [25] and Trinkle, *et. al.* [30] give comprehensive results on the existence and uniqueness of solutions for rigid body contact forces under the Coulomb friction model. Roughly speaking, existence and uniqueness can be guaranteed if the coefficients of friction are small enough. Certain problems, such as the static stability problem in which the bodies in the system are at rest, always have solutions. In general, the reliability of the algorithms decrease as the number of contact points increases, or as friction increases.

Impulse-based methods were pioneered by Hahn [13] and extended by Mirtich [21]. The fundamental strategy of this approach is to model all contact between bodies by collisions at contact points. Between collisions, the bodies move along ballistic trajectories. If friction and restitution are incorporated into the collision model, then trains of collision impulses can generate persistent contact phenomena like rolling, sliding, and settling. The major advantage of an impulse-based method is that only a single pair of contact points is handled at a given time; many of the computational problems associated with simultaneous contacts under the LCP method are avoided. There is always a valid solution when computing the collision impulses between two bodies. Unlike the LCP methods, impulse-based methods do not require contact coherence for efficiency. They work well on systems of bodies where the contacts are changing rapidly. The main disadvantage of impulse-based methods is their inability to efficiently handle more than a few simultaneous, persistent contacts. These methods will grind to a halt on systems such as a tall stack of blocks at rest, due to the large number of collisions that must be propagated up and down the stack. Ad-hoc methods must also be used to model static friction in certain cases.

4.2 SVD approach

We are currently exploring another method of computing the contact forces, based on the singular value decomposition (SVD) of a matrix. Equation (1) is used, but in a different way than with the LCP methods. At each contact, the contact normal is defined as a unit vector along the segment between the two contact points. If the component of relative velocity between the contact points in the plane perpendicular to the contact normal is below a small threshold, the contact is treated as a static friction contact, otherwise it is a dynamic friction contact. Call these static contacts and dynamic contacts for short. The direction of the contact force for dynamic contacts is completely specified by the Coulomb friction law; only the magnitude must be solved for. For static contacts, the direction of the force is not known, and so three components must be solved for. Numbering the n contacts such that the first m are static contacts, and the remainder are the dynamic contacts, Equation (1) takes the form

$$\underbrace{\left[\begin{array}{c} a_{1,x} \\ a_{1,y} \\ a_{1,z} \\ \vdots \\ a_{m,x} \\ a_{m,y} \\ a_{m,z} \\ \hline a_{m+1,z} \\ \vdots \\ a_{n,z} \end{array}\right]}_{\mathbf{a}} = \mathbf{A} \underbrace{\left[\begin{array}{c} f_{1,x} \\ f_{1,y} \\ f_{1,z} \\ \vdots \\ f_{m,x} \\ f_{m,y} \\ f_{m,z} \\ \hline \|f_{m+1}\| \\ \vdots \\ \|f_n\| \end{array}\right]}_{\mathbf{f}} + \mathbf{b}. \tag{2}$$

The vector **f** contains the three unknown components of each static contact force, and the unknown magnitudes of each dynamic contact force; the goal is to solve for **f**. The vector **a** comprises the three components of relative acceleration for each static contact, and the z-component—chosen to be the normal direction—of relative acceleration for each dynamic contact. The square matrix **A** and the nonhomogeneous term **b** are known quantities computed from the current dynamic state.

The procedure is to solve for **f** by setting **a** to **0**. This finds forces that keep the relative acceleration (normal and tangential) equal to **0** at all static contacts, and the relative normal acceleration equal to zero at all dynamic contacts. Often **A** is rank deficient, and so **f** is solved for by

$$\mathbf{f} = -\mathbf{A}^I\mathbf{b}, \tag{3}$$

where $\mathbf{A}^I$ is the "inverse" of **A** computed by SVD [26]. Assuming **b** is in the column space of **A**, which is the case in practice, the vector **f** obtained from (3) is the vector of minimum norm that satisfies (2) with **a** set to **0**. This gives a naturally symmetric force distribution when there are contact degeneracies, as shown in Figure 7.

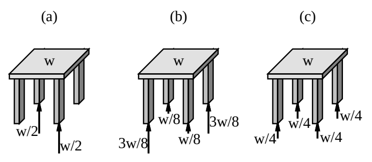

Figure 7: *For a table of weight w resting on the floor, there are many contact force configurations that satisfy frictional and contact constraints, such as the three shown above. The SVD method computes the composite force vector of minimum norm, which would be case (c).*

LCP methods enforce a complementarity constraint: the normal acceleration or the normal force must vanishes at every contact point. This constraint is not explicitly modeled in the SVD method. The justification for this is that the acceleration is almost always the quantity that vanishes. A non-zero acceleration means the bodies are separating near the contact point and the contact points will be deactivated anyway, once the distance between them exceeds the contact threshold ε.

Computing the SVD of a matrix is a robust procedure, even for very singular matrices. But two things can go wrong with the SVD method. At either a static or dynamic contact, the computed normal force might be negative, which is physically impossible. Or the force at a static contact might lie outside the friction cone. Currently, these problems are handled as follows. If a computed force at a static contact lies outside the friction cone, the contact is converted to dynamic one with the initial tangential force component in the direction of the invalid static tangential force. If the normal component of a computed contact force is negative, the contact is converted to an inactive one. In either case, the matrices $\mathbf{A}$ and $\mathbf{b}$ in (2) must be adjusted, and the SVD of $\mathbf{A}$ must be recomputed. Adjusting $\mathbf{A}$ and $\mathbf{b}$ is efficient: only a small number of rows or columns must be changed or deleted. Computing the SVD of $\mathbf{A}$ is currently done from scratch; this method would benefit from an incremental SVD algorithm that efficiently handles slightly modified matrices. This procedure is repeated until the computed contact forces are valid.

This looping procedure might not terminate with a correct set of contact forces. It assumes, for example, that a negative normal contact force indicates the contact is breaking, and so the contact force should be eliminated. However, there is no guarantee that upon eliminating this contact force and re-solving for the others, the relative acceleration at the offending contact will be positive. The algorithm has been tested and works correctly on some simple cases, but more study is needed. A more thorough characterization of when the approach fails, as Pang and Trinkle provided for the LCP approach, would be helpful. Another open question is whether the SVD approach is fast enough. All algorithms for computing the SVD of a matrix are iterative in nature and have no fixed complexity based on input size. Future tests on large contact groups will give more information on the practicality of this approach. The main advantage of the SVD approach is its simplicity.

5 Conclusion

We have presented *V-Clip*, a fast and robust algorithm for collision detection between *piecewise convex* polyhedral models. The algorithm returns a pair of closest features between objects, and can also return the closest points between these features. For disjoint bodies, *V-Clip* only returns features that result in zero-dimensional contact regions. However, one- and two-dimensional contact regions are modeled by maintaining a set of closest feature pairs returned by recent invocations of *V-Clip*. The most experimental work presented here is a new method of computing contact forces between contacting bodies. The method uses the singular value decomposition of the contact matrix to solve for forces that prevent interpenetration and model frictional effects. The method is easy to implement, but more study is needed to determine the limits of its practicality.

References

[1] D. Ayala, P. Brunet, R. Juan, and I. Navazo. Object representation by means of nonminimal division quadtrees and octrees. *ACM Transactions on Graphics*, 4(1):41–59, January 1985.

[2] David Baraff. Coping with friction for non-penetrating rigid body simulation. *Computer Graphics*, 25(4):31–40, August 1991.

[3] David Baraff. *Dynamic Simulation of Non-Penetrating Rigid Bodies.* PhD thesis, Department of Computer Science, Cornell University, March 1992.

[4] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 94. ACM SIGGRAPH, 1994.

[5] Vivek Bhatt and Jeff Koechling. Classifying dynamic behavior during three dimensional frictional rigid body impact. In *Proceedings of International Conference on Robotics and Automation.* IEEE, May 1994.

[6] Raymond M. Brach. *Mechanical Impact Dynamics; Rigid Body Collisions.* John Wiley & Sons, Inc., 1991.

[7] Stephen Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation.* IEEE, April 1997.

[8] Kelvin Chung. An efficient collision detection algorithm for polytopes in virtual environments. Master's thesis, University of Hong Kong, September 1996.

[9] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scaled environments. In *Symposium on Interactive 3D Graphics*, pages 189–196. ACM Siggraph, ACM Siggraph, April 1995.

[10] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, July 1997.

[11] Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, April 1988.

[12] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Computer Graphics Proceedings, Annual Conference Series*, Proceedings of SIGGRAPH 96. ACM SIGGRAPH, 1996.

[13] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[14] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3), July 1996.

[15] T. Hudson, M. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-collide: Accelerated collision detection for VRML. In *Proceedings of VRML*, 1997.

[16] J. B. Keller. Impact with friction. *Journal of Applied Mechanics*, 53, March 1986.

[17] Ming C. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of California, Berkeley, December 1993.

[18] Per Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 61(12):605–15, 1981.

[19] Per Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–93, June 1984.

[20] Michael McKenna and David Zeltzer. Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, pages 29–38, 1990.

[21] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* PhD thesis, University of California, Berkeley, December 1996.

[22] Brian Mirtich. *V-Clip:* fast and robust polyhedral collision detection. Technical Report TR97–05, Mitsubishi Electric Research Lab, Cambridge, MA, July 1997.

[23] Matthew Moore and Jane Wilhems. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, August 1988.

[24] Bruce F. Naylor. Interactive solid modeling via partitioning trees. In *Graphics Interface*, pages 11–18, May 1992.

[25] J.S. Pang and J.C. Trinkle. Complementarity formulations and existence of solutions of dynamic multi-rigid-body contact problems with coulomb friction. *Mathematical Programming*, 1996. (To appear).

[26] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian R. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, second edition, 1992.

[27] Rich Rabbitz. Fast collision detection of moving convex polyhedra. In Paul S. Heckbert, editor, *Graphics Gems IV*, pages 83–109, Cambridge, 1994. Academic Press, Inc.

[28] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, pages 349–358, 1991.

[29] W. J. Stronge. Unraveling paradoxical theories for rigid body collisions. *Journal of Applied Mechanics*, 58:1049–1055, December 1991.

[30] J.C. Trinkle, J.S. Pang, S. Sudarsky, and G. Lo. On dynamic multi-rigid-body contact problems with coulomb friction. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 1996. (To appear).

[31] Yu Wang and Matthew T. Mason. Modeling impact dynamics for robotic operations. In *Proceedings of International Conference on Robotics and Automation*, pages 678–685. IEEE, May 1987.