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

V-Clip: Fast and Robust Polyhedral Collision Detection

Brian Mirtich

TR-97-05 June 1997

Abstract

This paper presents the Voronoi-clip, *or* V-Clip, *collision detection algorithm for polyhedral objects specified by a boundary representation.* V-Clip *tracks the closest pair of features between convex polyhedra, using an approach reminiscent of the* Lin-Canny closest features *algorithm.* V-Clip *is an improvement over the latter in several respects. Coding complexity is reduced, and robustness is significantly improved; the implementation has no numerical tolerances and does not exhibit cycling problems. The algorithm also handles penetrating polyhedra, making it useful for nonconvex polyhedral collision detection. This paper presents the theoretical principles of* V-Clip, *and gives a pseudocode description of the algorithm. It also documents various tests that compare* V-Clip, Lin-Canny, *and the* Enhanced GJK *algorithm, a simplex-based algorithm that is widely used for the same application. The results show* V-Clip *to be a strong contender in this field, comparing favorably with the other algorithms in most of the tests, in terms of both performance and robustness.*

Submitted to ACM Transactions on Graphics, July, 1997

1. First printing, TR97-05, June 1997

1 Introduction

Collision detection is a central problem in many computer graphics applications. It is becoming more important with the rise of new applications in VR, simulation, and physically based animation. For polyhedral models specified by a boundary representation, the algorithms fall into two main categories: feature-based and simplex-based. Both varieties use coherence to obtain sublinear performance when objects move continuously through space.

1.1 Feature-based algorithms

The features of a polyhedron are the vertices, edges, and faces forming its boundary. Feature-based algorithms perform geometric computations on these elements to determine if a pair of polyhedra are disjoint and possibly to compute the distance between them. One example is Baraff's algorithm, which maintains a separating plane that embeds a face of one of the polyhedra, or one edge from each polyhedra [2]. The polyhedra are disjoint if and only if such a separating plane can be found. The separating plane is cached from one invocation to the next.

The *Lin-Canny closest features* algorithm [11] is a more sophisticated feature-based algorithm that computes the distance between disjoint polyhedra; it has traditionally been considered among the fastest solutions for this problem. The publicly available *I-Collide* collision detection package [5] uses *Lin-Canny* to perform the low-level collision checks. The algorithm tracks and caches the closest features between a pair of convex polyhedra. Once these features are known, the closest points between them, and therefore between the polyhedra, can be determined.

Lin-Canny has two drawbacks. The first is that it does not handle the case of penetrating polyhedra. If presented with such an instance, the termination criteria are never satisfied, and the algorithm cycles forever. This behavior can be prevented by forcing termination after a maximum iteration count is reached; however this approach is slow, requires choice of an arbitrary threshold, and does not return any measure of penetration depth. This limits *Lin-Canny's* usefulness for determining exact collision times by root-finding methods,[1] and for detecting collisions among nonconvex polyhedra using standard hierarchical techniques. *Lin-Canny's* second drawback is its lack of robustness. The cycling behavior also occurs in geometrically degenerate situations, so cycle detection is not a guarantee of penetration. The algorithm may be tweaked by adjusting various numerical tolerances—the implementation in *I-Collide* has six obvious ones, plus a few buried in the code, which are interdependent in subtle ways. Choosing suitable values for all applications is probably impossible. A related problem is the coding complexity of the algorithm: much effort is devoted to handling degenerate situations correctly. Despite these problems, *Lin-Canny's*

[1] In specific cases, it is possible to determine the collision time using *Lin-Canny* with a one-sided root-finding approach [13].

speed, and availability through the *I-Collide* package have made it a popular choice for collision detection applications.

1.2 Simplex-based algorithms

Rather than focusing on polyhedral features, simplex-based algorithms treat a polyhedron as the convex hull of a point set, and perform operations on simplices defined by subsets of these points. An algorithm designed by Gilbert, Johnson and Keerthi (*GJK*) was one of the earliest examples of this type [7]. Given two polyhedra, *GJK* searches for a simplex, defined by vertices of the Minkowski difference polyhedron, that either encloses or is nearest to the origin. If the origin is not enclosed, the distance between the origin and the nearest simplex of the difference polyhedron is equal to the distance between the original polyhedra. If the origin is enclosed, the polyhedra are penetrating, and a measure of the penetration is available.

The *GJK* algorithm is the essential core of an algorithm by Rabbitz, which advances the original by making better use of coherence [15]. *Q-Collide* is a collision detection library spawned from *I-Collide*, which replaces *Lin-Canny* with Rabbitz's algorithm for the low-level collision detection [4]. (All of *Q-Collide's* code and data structures to manage large numbers of objects are taken directly from *I-Collide*.) Cameron has recently developed the fastest descendent of *GJK*: it includes mechanisms to exploit coherence, and also uses topological vertex information to more carefully choose new simplices when the current simplices fail to satisfy the termination criteria. With these improvements, the algorithm attains the same *almost-constant time* complexity as *Lin-Canny* [3]. In this paper, *Enhanced GJK* refers to Cameron's algorithm and implementation.

1.3 V-Clip

This paper presents the *Voronoi-clip*, or *V-Clip*, algorithm, a closest features algorithm that bears a family resemblance to its ancestor, *Lin-Canny*. The motivation for designing *V-Clip* was to overcome the chief limitations of *Lin-Canny*:

1. *V-Clip* handles the penetration case, typically reporting penetration witness features in the same almost-constant time required for the disjoint case.

2. *V-Clip* is robust. Degenerate configurations are not problematic, and the implementation of the algorithm does not contain a single numerical tolerance. Divisions are rare, and never involve a divisor of smaller magnitude than the dividend. *V-Clip* does not exhibit cycling problems.

3. The code for *V-Clip* is significantly simpler than that of *Lin-Canny*. The *V-Clip* specification involves none of the sort of conditions that make *Lin-Canny* difficult to implement, for example, testing if a face or edge is parallel to another face or edge.

Enhanced GJK also exhibits many of the advantages listed above. Cameron's implementation contains two numerical tolerances, and divisors are occasionally many orders of magnitude smaller than dividends, but *Enhanced GJK* is quite robust in practice. It is probably the simplest of all of the algorithms to code. It requires more floating-point operations than the other algorithms, however.

Sections 2 discusses underlying principles and gives an overview of the *V-Clip* algorithm, Section 3 describes the fundamental Voronoi clip operation, Section 4 discusses how penetration is handled, Section 5 gives a pseudocode specification of the algorithm, Section 6 details several experiments involving *V-Clip* and alternative algorithms, and Section 7 makes some final remarks. Proofs of key theorems are in Appendix A.

1.4 Nonconvex objects

The algorithms that efficiently handle penetration can be easily adapted to nonconvex polyhedra by building hierarchies of convex components. Collision checking is performed between the convex hulls of various subsets of these components, and when the hulls are pierced, they are unwrapped so that collision checking may be performed between the individual pieces. This strategy works well when a convex decomposition is available with a moderate number of pieces, and a hierarchy not more than a few levels deep; it breaks down for utterly nonconvex objects. Other types of collision detection algorithms are then more suitable, such as those based on octrees [1], binary space partitioning trees [14], sphere hierarchies [9], or oriented bounding boxes (OBBs). These algorithms also often provide robustness in the presence of modeling errors, such as improperly oriented or missing facets. For example, the *Rapid* collision detection library, based on the OBB algorithm by Gottschalk, *et. al.*, adeptly handles "polygon soup." [8] While libraries like *Rapid* are the best choice in applications like complex walk through environments, the algorithms discussed in this paper are faster and preferable when the models are well-behaved, of moderate size, and not exceedingly nonconvex. This is the case in many applications that focus on object motion and interaction, and less on complex geometry.

2 Preliminaries

The boundary of a convex polyhedron in $\Re^3$ contains vertices, edges, and faces; these *features* are convex sets, here denoted by uppercase letters. Lowercase boldface letters denote points and vectors in $\Re^3$. If P is a plane and $\mathbf{y}$ is point, both in $\Re^3$, $D_P(\mathbf{y})$ denotes the signed distance of $\mathbf{y}$ from P. If

$$P = \{\mathbf{x} \in \Re^3 : \hat{\mathbf{n}} \cdot \mathbf{x} + w = 0\},$$

where $\hat{\mathbf{n}}$ is the unit normal to P, then

$$D_P(\mathbf{y}) = \hat{\mathbf{n}} \cdot \mathbf{y} + w.$$

Polyhedral edges are treated as vectors from a tail point $\mathbf{t}$ to a head point $\mathbf{h}$ in $\Re^3$. A point $\mathbf{e}$ along an edge may be parameterized by a scalar λ:

$$\mathbf{e}(\lambda) = (1-\lambda)\mathbf{t} + \lambda\mathbf{h}, \quad 0 \leq \lambda \leq 1. \tag{1}$$

2.1 Convex polyhedra topology

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 are central to the *V-Clip* algorithm:[2]

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 $\mathcal{VR}(X) \cap \mathcal{VR}(Y)$.*

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). The Voronoi region of a vertex V is bounded by a homogeneous set of planes: each is a vertex-edge plane between V and one of its neighboring edges. The Voronoi region of an edge E is bounded by four planes: two vertex-edge planes and two face-edge planes. The Voronoi region of an s-sided face F is bounded by $s+1$ planes: a face-edge plane for each edge bounding F, plus the support plane of F itself; $\mathcal{VR}(F)$ is a semi-infinite polygonal prism.

2.2 Algorithm overview

The *V-Clip* algorithm is based on a fundamental theorem, proved in Appendix A:

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

[2] Lin gives a slightly different definition in which the Voronoi regions are open sets [11].

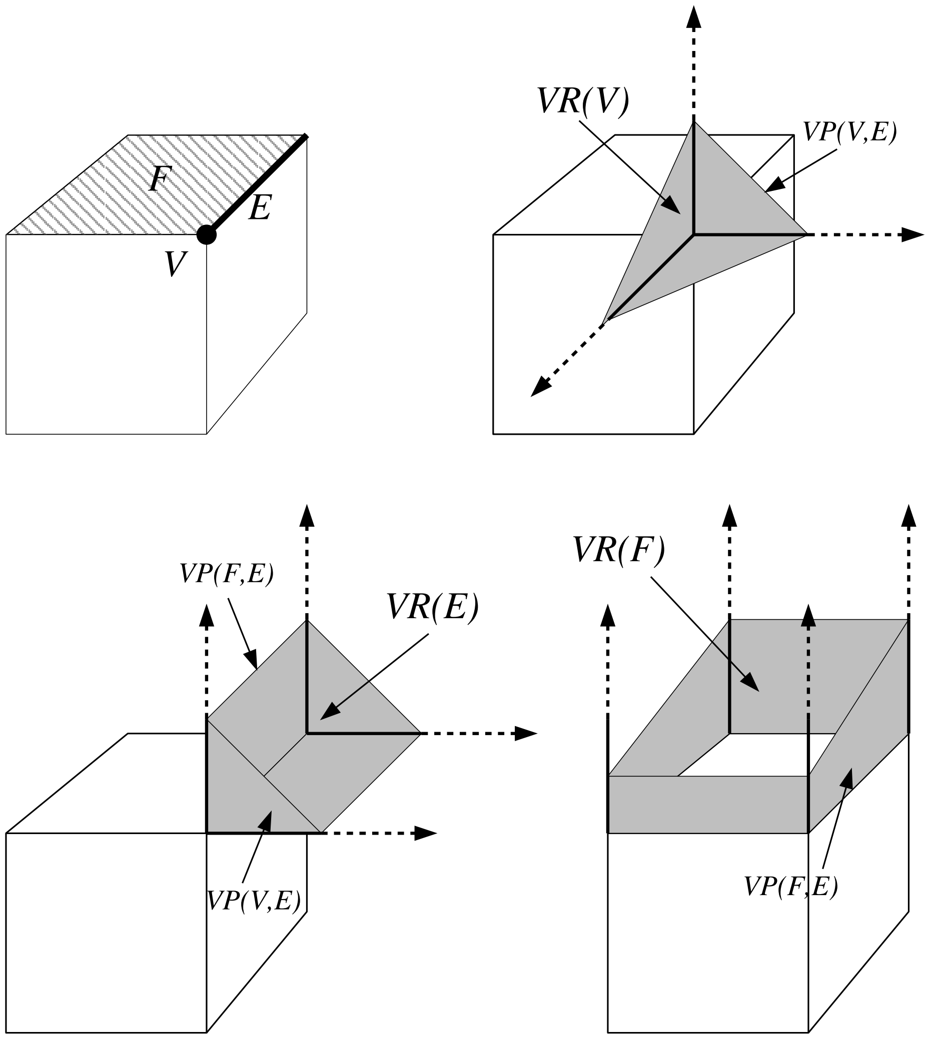


Figure 1: Top left: *A cubical polyhedron. Among its features are face F, edge E, and vertex V.* Top right: *The Voronoi region $\mathcal{VR}(V)$. One of the Voronoi planes bounding this region is $\mathcal{VP}(V, E)$, corresponding to V's neighboring edge E.* Bottom left: *The Voronoi region $\mathcal{VR}(E)$.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

features, usually to a neighboring one. If the new feature is of higher dimension than the old one, then the inter-feature distance strictly decreases. If the new feature is of lower dimension than the old one, the distance remains unchanged.

(When updating to a lower dimensional neighboring feature, there is no hope of decreasing the distance since the old feature includes the new one. However, such an update does improve the localization of the closest point, and may trigger subsequent updates that strictly reduce the inter-feature distance.)

In contrast to the *Lin-Canny* algorithm, *V-Clip* never actually computes the closest point on one feature to another, although if the former is a vertex, this is known trivially. This is what gives the algorithm its robustness in the face of degeneracy. Furthermore, the *V-Clip* iteration does not depend on the polyhedra being disjoint, and it correctly converges to a suitable pair of witness features when there is penetration.

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 solid arrows, each denoting a strict reduction in the inter-feature distance. Since there are only finitely many feature pairs, this is impossible.

Coherence is exploited by caching the pair of closest features from one invocation of the algorithm to the next. The method of initializing the feature pair is not critical, since the algorithm converges to a pair of closest features from any starting pair.

3 Voronoi clipping

This section and the next develop some building blocks that are used in the description of the complete algorithm in Section 5. During execution of the algorithm, the following problem occurs repeatedly:

Problem 1 *Given a pair of features, X and Y, one from each polyhedron, determine if the closest point on Y to X lies within $\mathcal{VR}(X)$. If not, update X in a way that decreases the inter-feature distance, or lowers the dimension of X while keeping the inter-feature distance constant.*

Basically, this problem is that of checking one of the symmetric conditions of Theorem 1, and updating the feature pair when it is not met. If Y is a vertex, the solution is trivial: Y contains but a single point which is easily checked for sidedness against the planes bounding $\mathcal{VR}(X)$. When Y does not lie in $\mathcal{VR}(X)$, the planes that are violated indicate how X should be updated.

The remainder of this section discusses the solution of Problem 1 in the more difficult case when feature Y is an edge E. This situation occurs in the V-E,

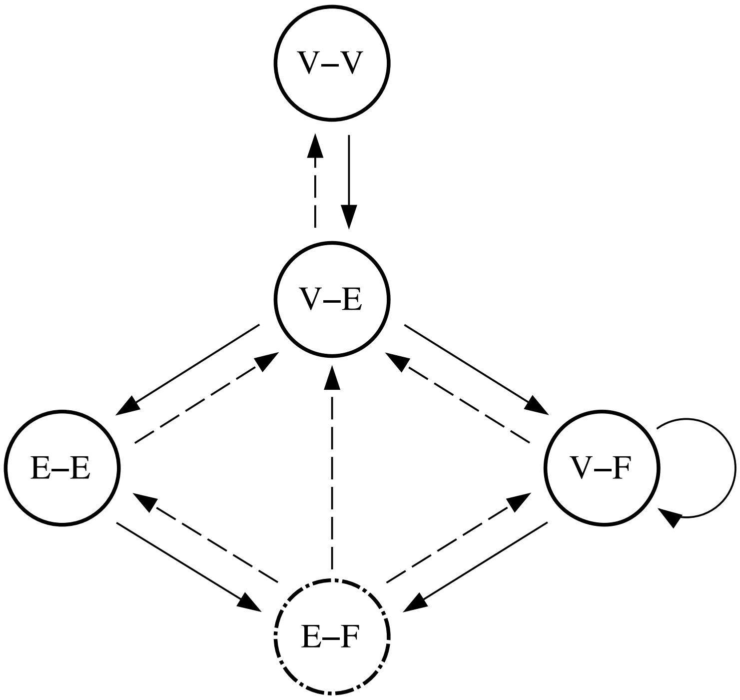


Figure 2: *States and transitions of the* V-Clip *algorithm.*

E-E, and E-F states of Figure 2, where X is a vertex, edge, or face, respectively. The general procedure is called *Voronoi clipping*.

3.1 Clipping edges

Let S be a subset of the set of oriented planes bounding $\mathcal{VR}(X)$. Each plane in S imposes a linear inequality constraint that points in $\mathcal{VR}(X)$ satisfy. Collectively, the planes in S define a convex region $K \supseteq \mathcal{VR}(X)$. If E is an edge from $\mathbf{t}$ to $\mathbf{h}$, then $E \cap K$ is either empty, or a line segment along E:

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

Algorithm 1 computes the values $\underline{\lambda}$ and $\overline{\lambda}$, and also the neighboring features $\underline{N}$ and $\overline{N}$ of X that correspond to the planes that clip E: E enters K as it crosses $\mathcal{VP}(X, \underline{N})$, and exits as it crosses $\mathcal{VP}(X, \overline{N})$. Figure 3 shows an example. If $\mathbf{t} \in K$, $\underline{N} = \emptyset$; If $\mathbf{h} \in K$, $\overline{N} = \emptyset$. If $E \cap K = \emptyset$, then the algorithm returns FALSE, otherwise it returns TRUE. 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.

If X is a vertex, a single invocation of *clipEdge* is used to clip an edge against all of the Voronoi planes ($K = \mathcal{VR}(X)$). If X is an edge, clipping against the

Algorithm 1 *clipEdge. Clip the edge from* $\mathbf{t}$ *to* $\mathbf{h}$ *against the Voronoi planes in* S*. 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 in S **do**
  $N \leftarrow$ neighbor feature of X 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

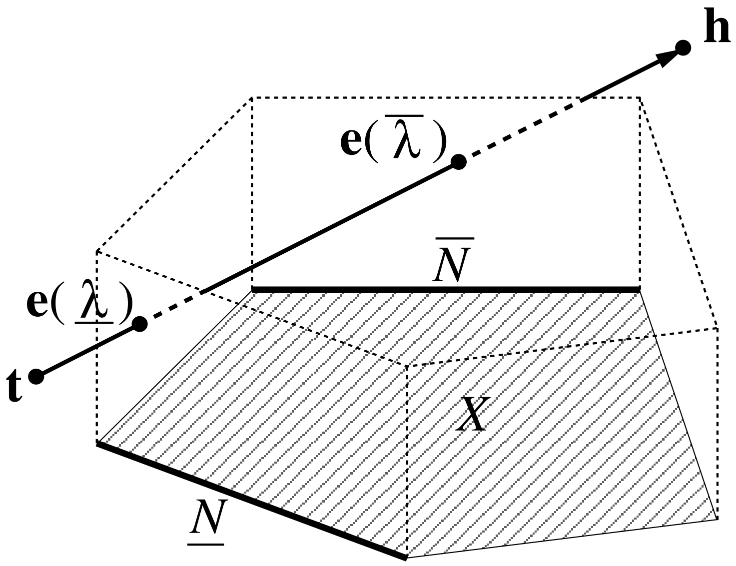


Figure 3: *Clipping an edge against planes bounding the Voronoi region of a pentagonal face X. The edge enters the region K as it crosses $\mathcal{VP}(X, \underline{N})$ at the parameterized point $\mathbf{e}(\underline{\lambda})$; it exits K as it crosses $\mathcal{VP}(X, \overline{N})$ at point $\mathbf{e}(\overline{\lambda})$.*

vertex-edge and face-edge planes bounding $\mathcal{VR}(X)$ is done separately. Finally, if X is a face, S is the set of all planes between X and its neighboring edges; clipping against the support plane of X is handled separately. Details are in Sections 4 and 5. After an edge E is clipped, the next step is to determine if the closest point on E to X lies within $K \supseteq \mathcal{VR}(X)$, and if not, how to update X.

3.2 Intersection case

First suppose edge E intersects K, so that *edgeClip* returns TRUE.

Definition 2 *Let E be an edge, and $\mathbf{e}(\lambda)$ a parameterized point along it, as in (1). For a polyhedral feature X, the* **edge distance function** $D_{E,X}(\lambda) : \Re \to \Re$ *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. Some important properties of this function are given by the following theorem, proved in Appendix A.

Theorem 2 *The edge distance function $D_{E,X}(\lambda)$ is continuous and convex. If $\mathbf{e}(\lambda_0) \notin X$, then $D_{E,X}$ is differentiable at λ_0.*

The question at hand is whether the minimum value of $D_{E,X}$, over $[0,1]$ occurs in the range $[\underline{\lambda}, \overline{\lambda}]$. Because of Theorem 2, this question is answered simply by checking the signs of its derivative at $\underline{\lambda}$ and $\overline{\lambda}$. 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 $D'_{E,X}(\overline{\lambda}) < 0$. In light of this, Algorithm 2 performs any necessary update. If X is updated to a neighboring feature, then the inter-feature distance

Algorithm 2 *Post-clipping derivative checks.*

```
1: if N̲ ≠ ∅ and D'_{E,X}(λ̲) > 0 then
2:    Update X to N̲
3: else if N̄ ≠ ∅ and D'_{E,X}(λ̄) < 0 then
4:    Update X to N̄
```

stays the same if the neighboring feature is of lower dimension than X, and strictly decreases if the neighboring feature is of higher dimension than X. If X is not updated, than the closest point on E to X lies within K.

The formulas for evaluating the signs of $D'_{E,X}$ follow from basic geometry. Let $\mathbf{u}$ be a vector directed along E, from tail to head. If X is a vertex at position $\mathbf{v}$, then

$$\mathrm{sign}[D'_{E,X}(\lambda)] = \mathrm{sign}[\mathbf{u} \cdot (\mathbf{e}(\lambda) - \mathbf{v})]. \tag{2}$$

If X is a face in the plane P with outward normal $\hat{\mathbf{n}}$, then

$$\mathrm{sign}[D'_{E,X}(\lambda)] = \begin{cases} +\mathrm{sign}(\mathbf{u} \cdot \mathbf{n}), & D_P[\mathbf{e}(\lambda)] > 0 \\ -\mathrm{sign}(\mathbf{u} \cdot \mathbf{n}), & D_P[\mathbf{e}(\lambda)] < 0 \end{cases} \tag{3}$$

A formula for the case where X is an edge is unnecessary. In this case, the derivative can be evaluated with respect to the relevant neighboring feature ($\underline{N}$ or $\overline{N}$), which must be a vertex or face. The derivative with respect to this neighboring feature is equal to the derivative with respect to X at the point where E crosses the Voronoi plane.

Equation (2) is invalid when $\mathbf{e}(\lambda) = \mathbf{v}$, and Equation (3) is invalid when $D_P[\mathbf{e}(\lambda)] = 0$; both of these are degenerate situations in which the distance function is not differentiable at $\mathbf{e}(\lambda)$. In these cases, however, the algorithm can simply report penetration since the edge E intersects the other feature.

3.3 Non-intersection case

Next consider the case where *edgeClip* returns FALSE, indicating that E lies completely outside $\mathcal{VR}(X)$. There are two ways this might be detected. The first, termed *simple exclusion* is detected when both endpoints of E are found to lie on the "outside" of a single Voronoi plane. The second, termed *compound exclusion*, is detected when $\underline{\lambda}$ exceeds $\overline{\lambda}$; this means that no point on E simultaneously satisfies the constraints imposed by two Voronoi planes. The two cases are distinguished by whether or not $\underline{N} = \overline{N}$. In either case, X must be updated.

Some care is required because of a subtle difference between vertex-edge and face-edge Voronoi planes. A vertex-edge Voronoi plane defines two half-spaces: points in one half-space are strictly closer to the edge while points in the other are equidistant from the edge and the vertex. A face-edge Voronoi plane does not partition space in this way. Points on either side of the Voronoi plane may be strictly closer to the face than to the edge (Figure 4). A consequence of

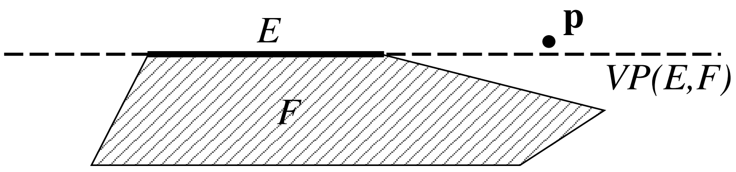


Figure 4: *An edge-face junction. This is a 2-D projection into the plane of F. Even though point* $\mathbf{p}$ *lies on the edge side of the Voronoi plane, it is strictly closer to the face.*

these facts is that the derivative checks and update steps of Algorithm 2 are valid wherever an edge crosses a vertex-edge Voronoi plane, even if the crossing is not on the boundary between the corresponding Voronoi regions. In contrast, if an edge crosses a face-edge Voronoi plane, the derivative checks and update steps are valid only if the crossing occurs on the boundary between the Voronoi regions.

3.3.1 Vertex Voronoi region exclusion

If E lies completely outside a vertex V's Voronoi region, the update is straightforward. For simple exclusion, the entire edge E lies on the edge side of a vertex-edge Voronoi plane. Every point on E is strictly closer to the neighboring edge than to V, and so V is updated to this edge. For compound exclusion, since both plane crossings are with vertex-edge Voronoi planes, Algorithm 2 is used. To see that an update must occur in this case, consider the convex distance function $D_{E,V}(\lambda)$ defined over $[0, 1]$. If compound exclusion occurs, then $0 < \overline{\lambda} < \underline{\lambda} < 1$. If $D_{E,V}$ attains its minimum in $[0, \overline{\lambda}]$ then the condition of line 1 is true. If $D_{E,V}$ attains its minimum in $[\underline{\lambda}, 1]$ then the condition of line 3 is true. Finally, if $D_{E,V}$ attains its minimum in $(\overline{\lambda}, \underline{\lambda})$, both conditions are true.[3] Hence, V will always be updated to a neighboring edge, and the distance to E will strictly decrease.

3.3.2 Edge Voronoi region exclusion

The Voronoi region of an edge X is bounded by two vertex-edge planes and two face-edge planes. The key to clipping an edge E against $\mathcal{VR}(X)$ is to handle the interactions with the vertex-edge planes first. If E crosses either vertex-edge plane bounding $\mathcal{VR}(X)$, X is possibly updated to the corresponding vertex, based on the result of derivative check (2). If E is simply excluded by either of the vertex-edge planes, X is immediately updated to the vertex. Only if none

[3]For the exclusion case, only the condition on line 1 need be tested, for if it is false, the condition on line 3 must be true.

of these tests triggers an update of X does testing with respect to the face-edge planes occur. If E is simply clipped by a face-edge plane, X is updated to the face. If compound clipping occurs with a vertex-edge and a vertex-face plane, E is updated to the vertex if prescribed by derivative check (2), and to the face otherwise. Compound clipping cannot occur with the two vertex-edge planes, since they are parallel. It can occur with the two face-edge planes. This happens exactly when edge E passes "underneath" edge X, piercing the planes of X's neighboring faces. In this case, derivative check (3) is valid. This check is applied for one of the faces, and X is updated to that face or the other, based on the result. This series of tests always produces some update of X to a neighboring feature. If X is updated to a vertex, the distance to E is unchanged, and if X is updated to a face, the distance to E strictly decreases.

3.3.3 Face Voronoi region exclusion

When an edge E is excluded from the Voronoi region of face F, much less information about how X should be updated is immediately available from the *clipEdge* algorithm. Consider Figure 5, in which E is simply excluded from $\mathcal{VR}(F)$ by the Voronoi plane corresponding to edge S_7, and compoundly excluded by the Voronoi planes corresponding to edges S_6 and S_8 (these planes are all shown as dashed lines in the figure). Either of these conditions might be the one detected by *clipEdge*, however the closest point on F to E might not lie on any of S_6, S_7, or S_8. In fact, the closest point on F to E might lie on any edge from S_3 to S_{11}, depending on the signed distances of $\mathbf{t}$ and $\mathbf{h}$ from the plane of F.

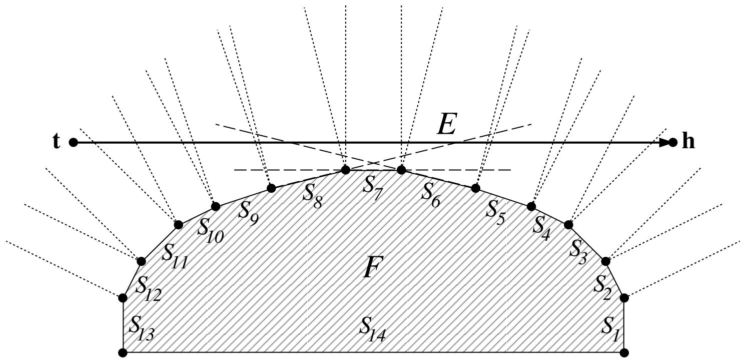


Figure 5: *A bad case for determining how to update F. This is a 2-D projection into the plane of F; the edge endpoints* $\mathbf{t}$ *and* $\mathbf{h}$ *are generally not in this plane.*

A safe way of determining how to update F is to scan along the perimeter of F, beginning at one of the features returned by *clipEdge*, clipping E against the vertex-edge planes that it cuts (shown as dotted lines in the figure), and applying the derivative check (2). The results of these clip tests and derivative checks indicate when the closest feature on F to E has been reached; it may either be an edge or a vertex along F's boundary. When F is updated to this lower-dimensional feature, the inter-feature distance remains constant.

4 Handling penetration

Because of the criteria *V-Clip* uses to update features, handling penetrating polyhedra can be done efficiently and with little extra effort. The *clipEdge* algorithm is used to clip an edge E against the face-edge planes of a face F's Voronoi region. Assuming E is not entirely clipped, the points on the edge defined by $\underline{\lambda}$ and $\overline{\lambda}$ are tested for sidedness relative to the plane of F. If they lie on opposite sides, then E must pierce F and the algorithm terminates with these two features as witnesses to penetration.

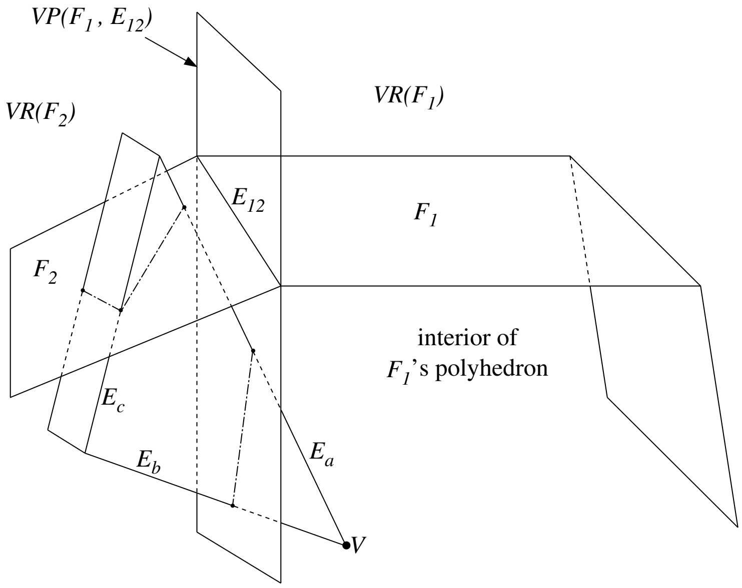


Figure 6: *A case of penetration (only relevant parts of F_1's polyhedron are shown).*

As an example, consider Figure 6, and assume that the current pair of features are V and F_1. The algorithm determines that it can decrease the inter-feature distance by updating V to its neighboring edge E_b (it may have alternatively chosen E_a). It makes no difference that V is inside F_1's polyhedron, nor does the algorithm have any concept of this—yet. It simply tries to find a neighboring feature of V that is closer to F_1. After V is updated to E_b, the algorithm clips E_b against the face-edge planes of $\mathcal{VR}(F_1)$, and discovers that E_b intersects $\mathcal{VP}(F_1, E_{12})$. A derivative check at this intersection point triggers an update of F_1 to E_{12}. After a series of update steps, the algorithm reaches the feature pair E_c-F_2. After clipping E_c against the face-edge planes of $\mathcal{VR}(F_2)$, it discovers that the endpoints of E_c lie on opposite sides of F_2 and signals penetration.

4.1 Escaping local minima

During the feature update process, it is possible to get lodged in a local minimum in the vertex-face state. This happens when the vertex satisfies the face-edge plane constraints of $\mathcal{VR}(F)$, lies "below" the plane of F, and has neighboring edges that are all directed away from F. In this situation, any update to a neighboring feature would either increase the inter-feature distance, or increase the dimension of a feature while keeping the inter-feature distance constant. Neither of these are valid updates. When the algorithm reaches such a state, the polyhedra may or may not be penetrating (Figure 7). If they are penetrating, the algorithm should terminate and report this. If not, a new pair of features must be found that escape the local minimum. Algorithm 3 handles this situation. It tests V against each face plane of F's polyhedron. If V lies on the negative side of all of them, it is inside the polyhedron and a penetration flag is returned. Otherwise, F is updated to the face that V has *maximum* signed distance from. If Algorithm 3 updates F to F_0, V is at least as close to F_0 as to any other face on F's polyhedron, and is strictly closer to F_0 than to F. Hence, there is no possibility of descent back into the same local minimum between V and F.

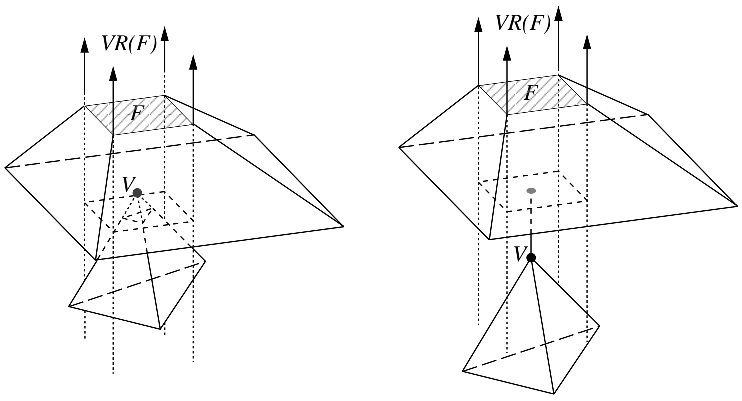


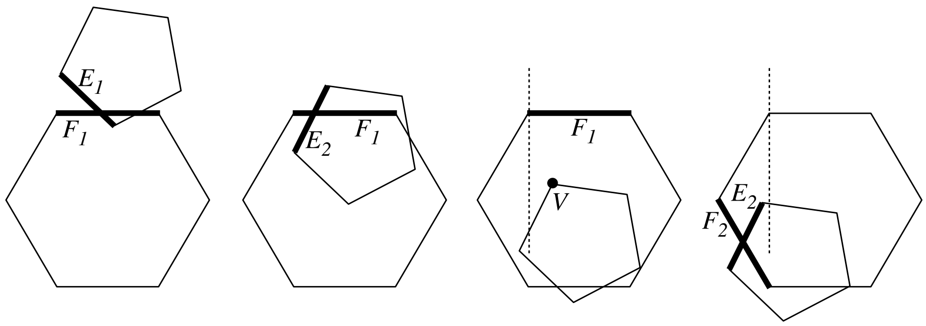
Figure 7: *Local minimum states with and without penetration.*

V-Clip detects penetration in one of two states: in the edge-face state, when the edge pierces the face; and in the vertex-face state, when a vertex lies inside of the other polyhedron. The test based on an edge-face witness pair is very fast since it only examines the local geometry near a penetration point; the test is independent of the complexity of the polyhedra. The test based on a vertex-face pair is much slower, since the vertex must be tested for sidedness against all n planes of the face's polyhedron: an $O(n)$ calculation. Fortunately, the edge-face witnesses to penetration are much more common in practice. When polyhedra

Algorithm 3 *handleLocalMin. Detect penetration or dislodge from a local minimum.*

$d_{\max} \leftarrow -\infty$
for all faces F' on F's polyhedron **do**
  $P' \leftarrow \text{plane}(F')$
  $d = D_{P'}(V)$
  if $d > d_{\max}$ **then**
    $d_{\max} \leftarrow d$
    $F_0 \leftarrow F'$
if $d_{\max} \leq 0$ **then**
  return PENETRATION
$F \leftarrow F_0$
return CONTINUE

penetrate a small amount relative to their sizes, the witness pair will be edge-face. Only if a polyhedron moves almost completely through another one *might* vertex-face penetration witnesses occur. Figure 8 shows a sequence in which a pair of polyhedra pass through each other, and vertex-face penetration witnesses momentarily occur. Usually, polyhedra pass though each other without ever generating vertex-face penetration witnesses.



(a) (b) (c) (d)

Figure 8: *Polyhedra passing through each other. This is a 2-D projection with faces appearing as line segments. In (a) and (b), the penetration witnesses are edge-face pairs. At (c), one polyhedron has moved almost completely through the other, and a vertex-face pair becomes the penetration witness. The dashed line represents one of the Voronoi planes of* $\mathcal{VR}(F_1)$. *Once* V *crosses this plane, as in (d), the algorithm escapes the* V*-*F_1 *local minimum, and again locates an edge-face penetration witness pair.*

5 Algorithm

In the implementation, each state of Figure 2 is handled by a function that can return one of three values: DONE means the conditions of Theorem 1 are satisfied, and so the current feature pair is the closest feature pair; PENETRATION means that penetration witnesses have been found; and CONTINUE means that the conditions of Theorem 1 are not met, one of the features has been updated, and the algorithm should continue. This section details the five state functions.

5.1 Vertex-vertex state

Algorithm 4 *VVstate(Vertex V_1, Vertex V_2)*

1: search for Voronoi plane $\mathcal{VP}(V_1, E)$ that V_2 violates
2: **if** violated plane exists **then**
3: $\quad V_1 \leftarrow E$
4: $\quad$ return CONTINUE
5: *repeat steps 1–4, swapping roles of V_1 & V_2*
6: return DONE

Algorithm 4 is the vertex-vertex state handler. A vertex feature contains only a single point, which is easily tested for membership in the other feature's Voronoi region. If V_2 lies outside $\mathcal{VP}(V_1, E)$, then V_2 is strictly closer to E than to V_1, and V_1 is updated to E. Similarly, if V_1 lies outside $\mathcal{VP}(V_2, E)$, then V_2 is updated to E. If no such updates are possible, V_1 and V_2 lie inside each others Voronoi regions, and the function returns DONE.

5.2 Vertex-edge state

Algorithm 5 VEstate(Vertex V, Edge E)

1: search for Voronoi plane $\mathcal{VP}(E, N)$ that V violates
2: **if** violated plane exists **then**
3: $\quad E \leftarrow N$
4: $\quad$ return CONTINUE
5: clip E against $\mathcal{VR}(V)$ [Algo. 1]
6: **if** $\underline{N} = \overline{N} \neq \emptyset$ **then**
7: $\quad V \leftarrow \underline{N}$
8: **else**
9: $\quad$ check derivatives, possibly update V [Algo. 2]
10: **if** V was updated **then**
11: $\quad$ return CONTINUE
12: **else**
13: $\quad$ return DONE

Algorithm 5 is the vertex-edge state handler. Steps 1–4 check that V lies within $\mathcal{VR}(E)$, and update E if it does not. The search in step 1 should check the vertex-edge planes before the face-edge planes. The remaining steps check that the closest point on E to V lies in $\mathcal{VR}(V)$, and update V if it does not. Steps 6–7 handle the simple exclusion case, and step 9 handles both the compound exclusion case, as well as the case when E intersects $\mathcal{VR}(V)$.

5.3 Vertex-face state

Algorithm 6 *VFstate(Vertex V, Face F)*

1: search for Voronoi plane $\mathcal{VP}(F, E)$ that V *maximally* violates
2: **if** violated plane exists **then**
3:    $F \leftarrow E$
4:    return CONTINUE
5: $P \leftarrow$ plane(F)
6: search for edge E, incident to V and V', such that $|D_P(V)| > |D_P(V')|$
7: **if** E exists **then**
8:    $V \leftarrow E$
9:    return CONTINUE
10: **if** $D_P(V) > 0$ **then**
11:    return DONE
12: return result of handleLocalMin [Algo. 3]

Algorithm 6 is the vertex-face state handler. When testing V against the Voronoi region of F, the algorithm does not stop at the first Voronoi plane which is violated, but searches for the maximally violated plane. This avoids the situation of Figure 4 and ensures that when F is updated in step 3, the inter-feature distance does not increase. The search in step 6 is for an edge incident to V that points toward F. It is equivalent to checking if the closest point on F to V lies within $\mathcal{VR}(V)$. If the algorithm reaches step 12, then it is lodged in a local minimum, and the *handleLocalMin* algorithm dislodges it or verifies penetration.

5.4 Edge-edge state

Algorithm 7 is the edge-edge state handler. As discussed in Section 3.3.2, it is important to handle the two vertex-edge Voronoi planes first. This is done in steps 1–7. Steps 8–14 handle the two face-edge Voronoi planes; this is a continuation of the clipping in steps 1–7, that is, the values of $\underline{\lambda}$ and $\overline{\lambda}$ are not reset to 0 and 1 at the step 8 clipping. If the algorithm reaches step 15, then E_2 intersects $\mathcal{VR}(E_1)$ and the closest point on E_2 to E_1 lies within this Voronoi region. Symmetric checks with the edge roles swapped complete the tests.

Algorithm 7 *EEstate(Edge E_1, Edge E_2)*

1: clip E_2 against vertex-edge planes of $\mathcal{VR}(E_1)$ [Algo. 1]
2: **if** E_2 simply excluded by $\mathcal{VP}(E_1, V)$ **then**
3:     $E_1 \leftarrow V$
4: **else**
5:     check derivatives, possibly update E_1 [Algo. 2]
6: **if** E_1 was updated **then**
7:     return CONTINUE
8: clip E_2 against face-edge planes of $\mathcal{VR}(E_1)$ [Algo. 1]
9: **if** E_2 simply excluded by $\mathcal{VP}(E_1, F)$ **then**
10:     $E_1 \leftarrow F$
11: **else**
12:     check derivatives, possibly update E_1 [Algo. 2]
13: **if** E_1 was updated **then**
14:     return CONTINUE
15: *repeat steps 1–14, swapping roles of E_1 & E_2*
16: return DONE

Algorithm 8 *EFstate(Edge E, Face F)*

1: clip E against $\mathcal{VR}(F)$ [Algo. 1]
2: **if** E excluded from $\mathcal{VR}(F)$ **then**
3:     $F \leftarrow$ closest edge or vertex on F to E
4:     return CONTINUE
5: $\underline{d} \leftarrow D_{E,F}(\underline{\lambda})$
6: $\overline{d} \leftarrow D_{E,F}(\overline{\lambda})$
7: **if** $\underline{d} * \overline{d} \leq 0$ **then**
8:     return PENETRATION
9: **if** $D'_{E,F}(\underline{\lambda}) \geq 0$ **then**
10:     **if** $\underline{N} \neq \emptyset$ **then**
11:         $F \leftarrow \underline{N}$
12:     **else**
13:         $E \leftarrow \mathrm{tail}(E)$
14: **else**
15:     **if** $\overline{N} \neq \emptyset$ **then**
16:         $F \leftarrow \overline{N}$
17:     **else**
18:         $E \leftarrow \mathrm{head}(E)$
19: return CONTINUE

5.5 Edge-face state

Algorithm 8 is the edge-face state handler. This handler either determines that the polyhedra are penetrating, or updates one of the features to a lower dimensional neighbor without changing the inter-feature distance; it never returns DONE. If E is excluded from $\mathcal{VR}(F)$, F is updated to the feature which is closest to E, among F's neighboring edges and the vertices between them (line 3). This process is discussed in Section 3.3.3.

If E intersects $\mathcal{VR}(F)$, the distances from the points on E parameterized by $\underline{\lambda}$ and $\overline{\lambda}$ to the plane of F are computed. If these distances are not both positive or both negative, then the edge E pierces F and PENETRATION is returned. Otherwise, either F is updated to one of its boundary edges or E is updated to one of its endpoints in a way which does not change the inter-feature distance.

6 Experimental results

Comparing speeds of different collision detection algorithms is difficult, and the literature is not without contradictory claims. The speeds of the algorithms depend heavily on many factors that are not constant among the results reported:

- The particular problem instances: how the objects are shaped, how they are moving, how they are clustered.
- Implementation issues: language, compiler, optimizer, floating-point number format, amount of inlined code and hand-coded assembly instructions, etc.

Despite these difficulties, we attempted the design of a test suite to obtain some gross measures of the relative performance and robustness of the *Lin-Canny*, *Enhanced GJK*, and *V-Clip* algorithms. The *V-Clip* implementation[4] is written in C++. The *Lin-Canny* implementation[5] is written in C, and is essentially the version used in *I-Collide*[6]. The *Enhanced GJK* implementation[7] is also written in C.

6.1 Coherence tests

The first set of tests were designed to measure performance on a variety of problems under varying levels of coherence. The task of designing tests was simplified since none of the tested algorithms are meant to be complete collision detection packages that handle thousands of objects, such as *I-Collide*[5] and *V-Collide* [10], but rather the lowest level collision detection routines in such systems. Thus, it suffices to test the algorithms on systems involving only two objects.

[4] Available from: TO BE ANNOUNCED...
[5] Available from: www.cs.berkeley.edu/~mirtich/collDet.html
[6] Available from: www.cs.unc.edu/~geom/I_COLLIDE.html
[7] Available from: www.comlab.ox.ac.uk/oucl/users/stephen.cameron/distances.html.

In each test, Object 1 was held fixed at the origin while Object 2, identical in shape, followed a continuous course through space around it.[8] The three coordinates of Object 2's center varied sinusoidally, as it also rotated around an axis. Algorithm 9 details the test procedure. The parameter ω corresponds roughly to the frequency of the revolution and rotation of Object 2. It controls the level of coherence between calls to the collision detectors: when $\omega = 0$ the objects remain fixed relative to each other throughout the test, and as ω increases the coherence decreases. For our tests, the value of ω was varied from 0 to 25 degrees per invocation.

Algorithm 9 Coherence testing. The angular velocity ω is an input parameter, and A is chosen to avoid penetration.

position O_1 at origin, with no rotation
for loop = 1 to 10 **do**
    $\delta_x \leftarrow \text{random}(0, 2\pi)$
    $\delta_y \leftarrow \text{random}(0, 2\pi)$
    $\delta_z \leftarrow \text{random}(0, 2\pi)$
    $\mathbf{a} \leftarrow$ random vector on unit sphere
    for i = 1 to 1000 **do**
        $\theta \leftarrow \omega * i$
        $x \leftarrow A\cos(\theta + \delta_x)$
        $y \leftarrow A\cos(\theta + \delta_y)$
        $z \leftarrow A\cos(\theta + \delta_z)$
        $R \leftarrow$ rotation by angle θ about axis $\mathbf{a}$
        position O_2 at location (x, y, z), orientation R
        collisionDetect(O_1, O_2)

These tests were performed with four different types of objects. An ordinary cube was chosen as the first test shape, since bounding boxes are common in collision detection applications. A regular icosahedron (20 triangular faces) was chosen as a medium complexity model. A polygonized disk was chosen as the third test shape. The aspect ratio of disk radius to thickness was 10:1. A distinguishing feature of this test shape is that it contains high complexity faces: two of the faces are 60-sided polygons. Finally, a tesselated sphere with 642 vertices, 1920 edges, and 1280 triangular faces was chosen as a high overall complexity model. Each algorithm was run on an identical set of problem instances.

Our interest was in comparing the *algorithms*—not the implementations—as fairly as possible. To this end, floating-point operation counts were used as the benchmark. Floating-point operations are the dominant cost in all of the algorithms, and are independent of language, compiler, and (in this case) optimizer. Fair timing tests are more difficult to devise, and were not included in the study. Additions (including subtractions and comparisons) were the most common floating-point operation, followed by multiplications. In all of the al-

[8] For all of the algorithms, there is no computational advantage or disadvantage to holding one object fixed; only relative position matters.

gorithms, the proportions of these two categories were roughly the same, and together they accounted for at least 98% of the floating-point operations. Figures 9–12 plot the average number of floating-point operations per invocation versus the coherence parameter ω for each of the algorithms and test objects. For the *Lin-Canny* and *V-Clip* algorithms, the counts reported include the cost of computing the closest points since these algorithms cannot compute the distance without them. *Enhanced GJK* can, however, and in the tests it was not asked to provide the closest points. If these are desired, *Enhanced GJK* can provide them at an additional cost of about 16 floating-point operations.

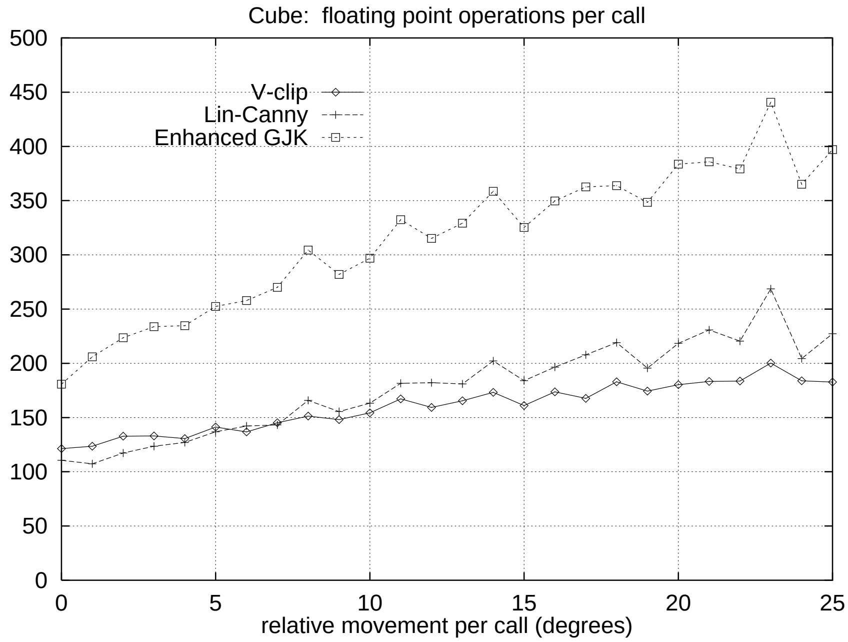


Figure 9: *Total floating-point operations for cubes.*

For the cube and icosahedron tests, *V-Clip* and *Lin-Canny* had comparable operation counts, with *Lin-Canny* holding a slight edge at high levels of coherence, and *V-Clip* holding the edge elsewhere. Since these algorithms are based on the same approach, the similarity in their operation counts is not surprising. *Enhanced GJK* generally required 50–100% more operations than the other two algorithms for these two test shapes.

For the disk test shape, *V-Clip* used the fewest operations over all coherence levels. *Lin-Canny* started out well but quickly attained the highest operation counts of any of the algorithms as coherence decreased. The operation-count curves for these two algorithms are fairly erratic. This is probably due to the fact that the disk is much less spherically symmetric than the other objects; rotation about an axis parallel to the plane of the disk is a quite different motion than rotation about an axis within the plane of the disk. *Enhanced GJK* seems less

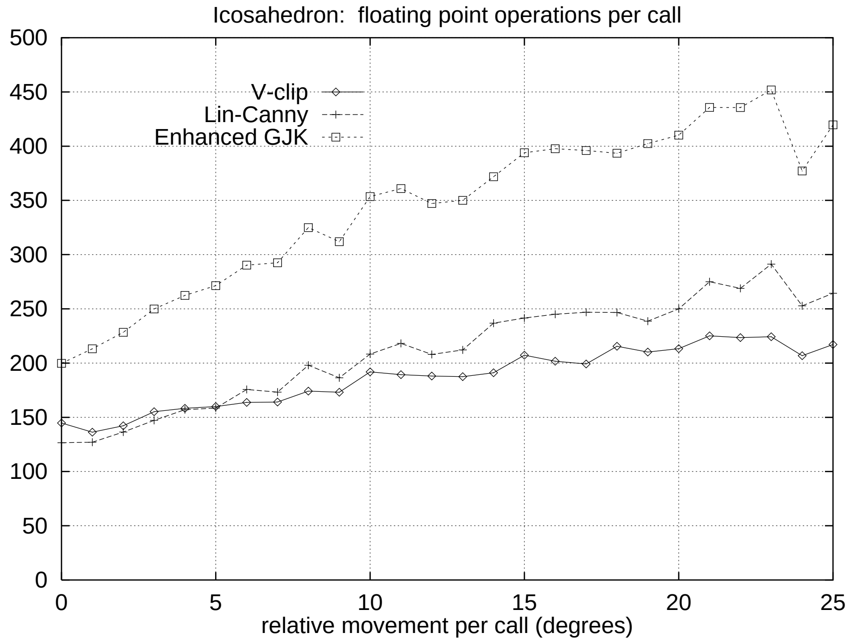


Figure 10: *Total floating-point operations for icosahedra.*

bothered by this fact, and its operation-count curve exhibits a more steady increase as coherence decreases.

The curves for the sphere tests are similar to those for the disk. *V-Clip* and *Lin-Canny* start out ahead, and *Enhanced GJK* overtakes *Lin-Canny* fairly quickly. *Enhanced GJK* also overtakes *V-Clip* at low levels of coherence. A likely explanation for this behavior lies in the iteration methods for the different algorithms. *V-Clip* and *Lin-Canny* are surface-based iterators: they move along the surfaces of the polyhedra as they search for the closest features. The *Enhanced GJK* iteration is based on simplices formed by sets of polyhedral vertices. This enables *Enhanced GJK* to *tunnel* through objects. The advantage of this method is most pronounced when the polyhedra have high complexity and the coherence level is low. In these cases, *V-Clip* and *Lin-Canny* must travel through many surface features while *Enhanced GJK* takes a more direct route, converging in fewer iterations.

6.2 Accuracy and degeneracy tests

The above tests also afforded the opportunity to gather data on algorithm accuracy. The distances reported by the three algorithms after each invocation were checked for consistency. Each algorithm's result was compared to the minimum result returned by any of the three algorithms; a warning was flagged if the difference exceeded a predefined tolerance of 10^{-6}. *V-Clip* had the best perfor-

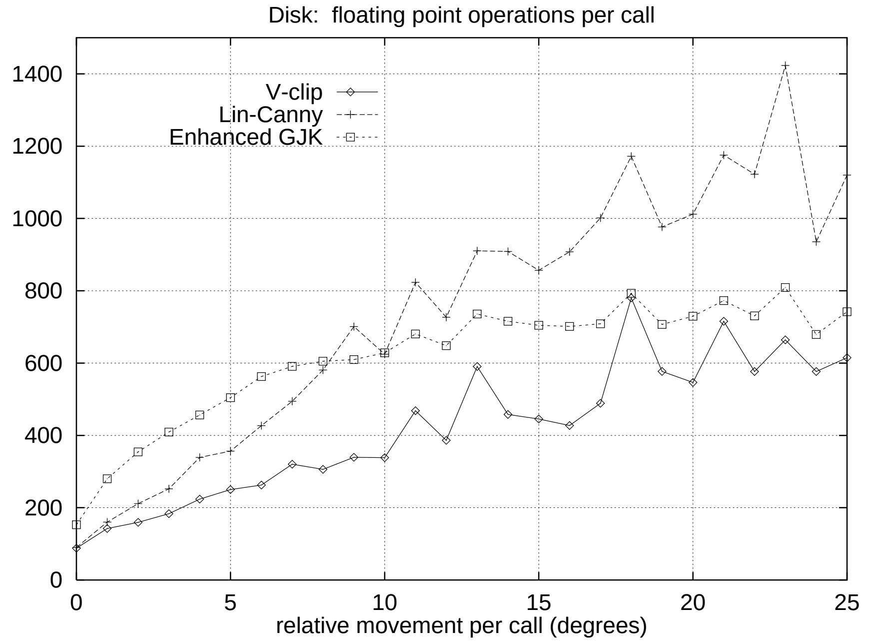


Figure 11: *Total floating-point operations for disks.*

mance, generating no warnings over the one million invocations. *Enhanced GJK* was nearly perfect, generating only 11 warnings, with a maximum distance deviation of 1.6×10^{-4}. These warnings all disappeared when the main numerical tolerance in the implementation was lowered from 10^{-5} to 10^{-7}. *Lin-Canny* performed the worst, generating 208 warnings, with distance deviations as large as 10^{-2}. Several attempts at tweaking the *Lin-Canny* numerical tolerances reduced the warning count to 88.

A second test probed the algorithms' robustness in degenerate situations. Algorithm 10 illustrates the basic test procedure. Each of this test's 100000 trials involved searching for a boundary case while performing collision detection between two cubes. O_1 was kept at the origin, while O_2's pose was interpolated between two poses chosen randomly from a distribution that favored near parallel edges and faces. In steps 13 and 18, the coherence devices returned by the algorithms are saved; these are pairs of closest features in the case of *V-Clip* and *Lin-Canny*, and pairs of simplices in the case of *Enhanced GJK*. A binary search method (steps 14–22) was used to home in on poses were these features or simplices changed. Such boundary cases are the most likely places for anomalous behavior.

This test quickly exposed the degeneracy problems of *Lin-Canny*. With the numerical tolerances set as they had been for minimizing the warning messages in the previous experiments, cycling behavior was observed in 4489 of the trials. Tweaking the tolerances lowered this number to 37, but significantly reduced

Algorithm 10 Degeneracy testing. O_1 and O_2 are cubes, two units per side

1: position O_1 at origin, with no rotation
2: **for** loop = 1 to 100000 **do**
3:   $x \leftarrow$ random(-4,+4)
4:   $y \leftarrow$ random(-4,+4)
5:   $z \leftarrow 4$
6:   $\mathbf{a} \leftarrow$ random vector on unit sphere
7:   $k \leftarrow$ random(0, 20);
8:   $R \leftarrow$ rotation about $\mathbf{a}$ by e^{-k} radians
9:   pose $\underline{P} \leftarrow$ position (x, y, z), orientation R
10:   *choose pose* $\overline{P}$ *in same manner as pose* $\underline{P}$ *(steps 3–9)*
11:   set pose of O_2 to $\underline{P}$
12:   collsionDetect(O_1, O_2)
13:   $\underline{X} \leftarrow$ feature/simplex pair
14:   **while** $\underline{P} \neq \overline{P}$ **do**
15:     $P \leftarrow$ average($\underline{P}$, $\overline{P}$)
16:     set pose of O_2 to P
17:     collsionDetect(O_1, O_2)
18:     $X \leftarrow$ feature/simplex pair
19:     **if** $X = \underline{X}$ **then**
20:       $\underline{P} \leftarrow P$
21:     **else**
22:       $\overline{P} \leftarrow P$

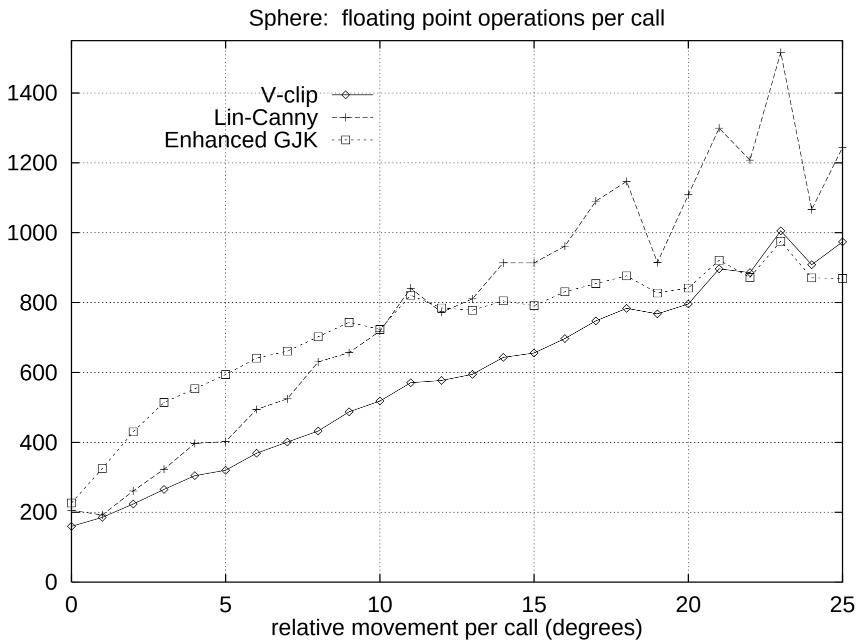


Figure 12: *Total floating-point operations for spheres.*

the accuracy of the reported distances, and caused *Lin-Canny* to return very erroneous distances in 22 trials. *Enhanced GJK* never exhibited any cycling or erroneous distances with its main tolerance set to 10^{-7}. However, with the tolerance set to 10^{-8}, *Enhanced GJK* reported a zero distance (penetration) in 175 of the trials, even though the actual distance between the cubes was never below $4 - 2\sqrt{3}$. *V-Clip*, with no tolerances to tune, did not exhibit any cycling, nor did it report a detectably incorrect distance.

7 Conclusion

V-Clip is a significant improvement over *Lin-Canny* for three reasons: It can efficiently handle penetration, and therefore nonconvex objects, it is easier to code, and it is more robust. The robustness stems from the fact that *V-Clip* does not explicitly construct closest points between features while iterating. Rather, it localizes the closest points to various regions of interest through simple clipping operations and scalar derivative tests. In other words, *V-Clip* uses *topological primitives* [6] while *Lin-Canny* must compute new objects: the closest points between features. Edge-face and face-face are the two most problematic cases in *Lin-Canny*; they account for most of the coding complexity and cycling problems. In *V-Clip*, the edge-face case is much simplified, and the face-face case is eliminated.

The proof that *V-Clip* will always terminate, which was outlined in Section 2.2, is not valid in the face of finite-precision arithmetic. Nonetheless, the simple form of the tests in the algorithm lend it an empirical robustness. *V-Clip* has never cycled nor reported a detectably incorrect answer throughout extensive testing; it alone passed all of the tests. Neither *V-Clip* nor *Enhanced GJK* enjoy a significant robustness advantage over the other, although the absence of numerical parameters to tune is an advantage of the former.

Overall, *V-Clip* had the lowest floating-point operation counts of the three algorithms, however, there were conditions under which each of the other two algorithms performed best. Also, floating-point operation counts are not a direct measure of performance[9]. While the tests were designed to be as fair and general as possible, there is no substitute for actually trying the algorithms in the specific application at hand. Fortunately, implementations of all of the algorithms are now available for doing so.

Acknowledgments

We thank Stephen Cameron for useful discussions and for supplying his *Enhanced GJK* implementation. We thank Jon Cohen, Ming Lin, Dinesh Manocha, and Madhav Ponamgi for their help in debugging the *Lin-Canny* implementation. Paul Beardsley, Bill Freeman, and Joe Marks provided thoughtful reviews of early drafts of this paper.

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. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, August 1990. SIGGRAPH Conference Proceedings, 1990.

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

[4] Kelvin Chung. Quick collision detection library. www.cs.hku.hk/~tlchung/collision_library.html, 1997.

[5] 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.

[6] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, January 1990.

[7] 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.

[8] S. Gottschalk, M. C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH Conference Proceedings*. ACM Press, 1996.

[9] For our *Enhanced GJK* tests, we set the EAGER_HILL_CLIMBING flag in Cameron's implementation, since this lowers the operation counts. However, Cameron has observed that this actually increases the execution time.

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

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

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

[12] David G. Luenberger. *Linear and Nonlinear Programming*. Addison Wesley, Reading, MA, second edition, 1984.

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

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

[15] 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.

[16] H. L. Royden. *Real Analysis*. Macmillan Publishing Company, Ney York, third edition, 1988.

A Proofs

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.*

Denote X's polyhedron by $\mathcal{X}$, Y's polyhedron by $\mathcal{Y}$, and define $\phi \subset \Re^6$ as

$$\phi = \{(\mathbf{u}, \mathbf{v})^T : \mathbf{u} \in \mathcal{X} \text{ and } \mathbf{v} \in \mathcal{Y}\},$$

and the distance function $D : \phi \to \Re$ as

$$D(\mathbf{u}, \mathbf{v}) = ||\mathbf{u} - \mathbf{v}||.$$

Since $\mathcal{X}$ and $\mathcal{Y}$ are convex polyhedra, ϕ is a convex set, and D can be shown to be a differentiable, convex function (the arguments are similar to those used the proof of Theorem 2, below). Let $\mathbf{p} = (\mathbf{x}, \mathbf{y})^T \in \phi$, choose $\mathbf{q}$ to be any other point in ϕ, and let $\Delta\mathbf{p} = (\Delta\mathbf{x}, \Delta\mathbf{y})^T = \mathbf{q} - \mathbf{p}$.

Suppose $\nabla D(\mathbf{p})(\Delta\mathbf{x}, \mathbf{0})^T < 0$. Then for some $\lambda > 0$, $\mathbf{x} + \lambda\Delta\mathbf{x} \in \mathcal{X}$, and the distance from this point to $\mathbf{y}$ is less than the distance from $\mathbf{x}$ to $\mathbf{y}$. But this is impossible since $\mathbf{y}$ is as close to X as to any other feature on $\mathcal{X}$, and $\mathbf{x}$ is the closest point on X to $\mathbf{y}$. Thus $\nabla D(\mathbf{p})(\Delta\mathbf{x}, \mathbf{0})^T \geq 0$. An analogous argument shows $\nabla D(\mathbf{p})(\mathbf{0}, \Delta\mathbf{y})^T \geq 0$, and therefore,

$$\nabla D(\mathbf{p})\Delta\mathbf{p} = \nabla D(\mathbf{p}) \begin{bmatrix} \Delta\mathbf{x} \\ \mathbf{0} \end{bmatrix} + \nabla D(\mathbf{p}) \begin{bmatrix} \mathbf{0} \\ \Delta\mathbf{y} \end{bmatrix} \geq 0.$$

Since for any $\mathbf{q} \in \phi$, $\nabla D(\mathbf{p})(\mathbf{q} - \mathbf{p}) \geq 0$, and D is differentiable and convex over the convex set ϕ, $\mathbf{p}$ is a global minimum of D over ϕ (see [12] §6.5). Thus, $\mathbf{x}$ and $\mathbf{y}$ are a globally closest pair of points between $\mathcal{X}$ and $\mathcal{Y}$. □

Theorem 2 *The edge distance function $D_{E,X}(\lambda)$ is continuous and convex. If $\mathbf{e}(\lambda_0) \notin X$, then $D_{E,X}$ is differentiable at λ_0.*

Proof: Choose two real numbers u and v, and $\rho \in [0,1]$. To show convexity, it suffices to verify

$$D_{E,X}[(1-\rho)u + \rho v] \leq (1-\rho)D_{E,X}(u) + \rho D_{E,X}(v). \tag{4}$$

Recall $\mathbf{e}(\cdot)$ denotes a parameterized point along E. By definition, there exist points $\mathbf{c}$ and $\mathbf{d}$ on X such that

$$\begin{aligned} D_{E,X}(u) &= \|\mathbf{e}(u) - \mathbf{c}\| \\ D_{E,X}(v) &= \|\mathbf{e}(v) - \mathbf{d}\|. \end{aligned}$$

Since X is a convex set, $(1-\rho)\mathbf{c} + \rho\mathbf{d} \in X$ and

$$\begin{aligned} &D_{E,X}[(1-\rho)u + \rho v] \\ &\quad \leq \|(1-\rho)\mathbf{e}(u) + \rho\mathbf{e}(v) - [(1-\rho)\mathbf{c} + \rho\mathbf{d}]\,\|. \end{aligned}$$

By the triangle inequality,

$$D_{E,X}[(1-\rho)u + \rho v] \leq (1-\rho)\|\mathbf{e}(u) - \mathbf{c}\| + \rho\|\mathbf{e}(v) - \mathbf{d}\|,$$

and (4) follows.

Convexity of $D_{E,X}$ implies continuity, and the existence of right- and left-hand derivatives everywhere, and furthermore, at any point λ_0 these derivatives satisfy

$$D'_{E,X}(\lambda_0-) \leq D'_{E,X}(\lambda_0+) \tag{5}$$

(see [16] §5.5). Let $\mathbf{p}$ be the closest point on X to $\mathbf{e}(\lambda_0)$, and define

$$D_{\mathbf{p}}(\lambda) = \|\mathbf{e}(\lambda) - \mathbf{p}\|.$$

By assumption, $\mathbf{e}(\lambda_0) \notin X$, thus $\mathbf{e}(\lambda_0) \neq \mathbf{p}$ and $D_{\mathbf{p}}$ is differentiable at λ_0. Since $D_{E,X} \leq D_{\mathbf{p}}$ everywhere,

$$\begin{aligned} D'_{E,X}(\lambda_0-) &\geq D'_{\mathbf{p}}(\lambda_0) \\ D'_{E,X}(\lambda_0+) &\leq D'_{\mathbf{p}}(\lambda_0). \end{aligned}$$

These inequalities plus (5) imply $D'_{E,X}(\lambda_0-) = D'_{E,X}(\lambda_0+)$, and so $D_{E,X}$ is differentiable at λ_0. $\square$