An Object-Space Method for Calculating the Minkowski Sums of Simple 3D Objects

Engin Deniz Diktaş and Ali Vahit Şahiner

Department of Computer Engineering, Bogazici University, Turkey

Abstract

This paper presents an easy-to-implement and efficient method to calculate the Minkowski Sums of simple convex objects. The method is based on direct geometrical manipulation of planes in 3D space. The paper also explains the translational and topological invariance properties of Minkowski sums, their use in distance calculations and presents some performance results.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction[†]

In graphic and haptic applications, Minkowski sums are mainly used in the context of collision detection. When checking collision for two objects, one of the objects (the pivot object) is shrunk to a point while the other is grown (dilated) by the amount equal to that of the pivot. The grown object actually is the Minkowski sum of the two objects. The detection of collisions between these objects is achieved simply by checking if the shrunk point is inside the grown object. If the relative position of the objects change, there is no need to recalculate the dilation. A single copy of the dilated object can be tested against any reference point to answer all the collision queries, instead of costly object-object intersection tests. Minkowski sums can also be used to find the penetration depth of one object intersecting the other along a certain direction, to evaluate minimum/maximum distance queries between two convex objects, and in topology simplification and correction.

In the literature, various methods have been proposed to compute the Minkowski sum of two convex polyhedra in R^3. Flato and Halperin [AFH02] present algorithms for robust construction of planar Minkowski sums. Guibas et al. [GRS83] introduced the definition of the convolution, which is a superset of the Minkowski sum, based on the kinetic framework in two dimensions. An algorithm for computing Minkowski sums of polytopes was introduced in [GS87]. Gritzmann and Sturmfels [GS93] obtained a polynomial time algorithm for computing Minkowski sums of k polytopes in R^d for a fixed dimension d, while Fukuda [Fuk04] provided a polynomial algorithm based on linear programming concepts. Ghosh [Gho93] presented a unified algorithm for computing 2D and 3D Minkowski sums of both convex and non-convex polyhedra based on a slope diagram representation. Computing the Minkowski sum amounts to computing the slope diagrams of the two objects, merging them, and extracting the boundary of the Minkowski sum based on the merged diagram. Bekker and Roerdink [BR01] apply a few variations on this idea, where the slope diagram of a 3D convex polyhedron is represented as a 2D object, thus reducing the problem to a lower dimension. Fogel and Halperin [FH06] follow the same approach, where they first construct the dual representations of both 3D convex polyhedra under the Cubical Gaussian Map (CGM). Then they use the overlay of the two planar arrangements of these dual representations to compute the desired Minkowski sum. Their method results in significant performance improvements compared to the other methods they list.

All of these methods involve utilization of complex data structures requiring costly initializations and updates thus have high time-complexities. In this paper, we propose a much simpler geometric method for calculating the

[†] A preliminary version of this paper is accepted to ISCIS-2006

Minkowski sums for simple convex objects and evaluate it within the context of distance computations.

The paper is divided into 4 sections. Section 1 introduces the Minkowski Sum and provides a literature survey. Section 2 presents our method along with a convex hull based method for Minkowski Sum computation. Section 4 discusses invariance properties of Minkowski sums. Section 5 gives some performance evaluation results and the final section presents our conclusions and planned future work.

2. Minkowski Sums

If A and B are two convex polyhedra in R^3, then the Minkowski sum of A and B is defined as the convex polyhedron

$$M = A \oplus B = \{a+b \,|\, a \in A, b \in B\} \tag{1}$$

In other words, the Minkowski sum of two polyhedra geometrically, is nothing but the convex hull of a set of vertices obtained by placing copies of one polyhedron at all vertices of the other. This is shown in Figure 1 for a triangle and a rectangle. The set of vertices $\mathcal{V} = \{v_i\}$, whose convex hull is to be computed, can be given by

$$\mathcal{V} = (\mathcal{V}_T \times \mathcal{V}_R) \cup \mathcal{V}_T \tag{2}$$

where $\mathcal{V}_T$ is the set of all vertices of the triangle, $\mathcal{V}_R$ is the set of all vertices of the rectangle and $\times$ denotes the Cartesian product operation.

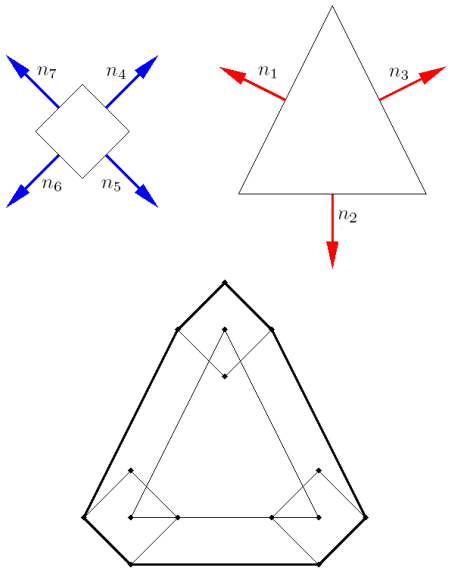

Figure 1: *Sample triangle and rectangle and their Minkowksi sum*

2.1. Proposed Method

The geometric method we describe in this paper generates the convex hull directly by translating lines in 2D space (planes in 3D space), instead of generating the convex hull using conventional convex hull algorithms, like Graham's method in 2D or Incremental Method in 3D [O'R94], which have large computational overheads when applied to simple convex objects.

Our method is based on the fact that every edge normal in the 2D Minkowski sum comes from the set of edge normals of both polygons, i.e. every edge in the Minkowski sum is parallel either to an edge of the triangle or to an edge of the rectangle. The vertices in the Minkowski sum are those that are extremal along these normals. For the example given in Figure 1, the set of normals $\mathcal{N}$ is given by

$$\mathcal{N} = \mathcal{N}_T \cup \mathcal{N}_R \tag{3}$$

where $\mathcal{N}_T$ is the set of normals of the triangle and $\mathcal{N}_R$ is the set of normals of the rectangle. That is, $\mathcal{N}$ consists of the set $\{n_1, \cdots, n_7\}$, where the first 3 normals come from the triangle and, the remaining ones come from the rectangle. Minkowski sum calculation for this example can be analyzed as the two following cases:

(a) Calculating edges parallel to those of the triangle: This is simply achieved by shifting a line passing through the origin with normal $n_i \in \mathcal{N}_T$, in the direction of its normal until the most distant vertex is found. Figure 2 shows this process for the stippled line with normal n_1. Here v_1 and v_2 are the extreme vertices, also the end-points of the edge e_1. The set of extreme vertices for a given normal n_i is thus given by

$$\{\, v_k \,|\, v_k \cdot n_i = \max\{\, v_j \cdot n_i \,,\, v_j \in \mathcal{V} \,\} \,\} \tag{4}$$

where $\mathcal{V}$ is the set of all vertices given by (2). Repeating this procedure for all normals $n_1 \ldots n_3$ coming from the triangle: we get the edges shown in Figure 2 (right).

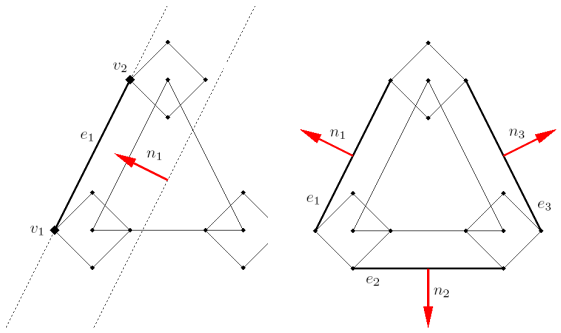

Figure 2: *Edges obtained by shifting lines parallel to triangle-edges*

(b) Calculating the edges parallel to those of the rectangle: Here the same procedure is repeated for the lines with normals coming from the rectangle. Shifting the stippled line in

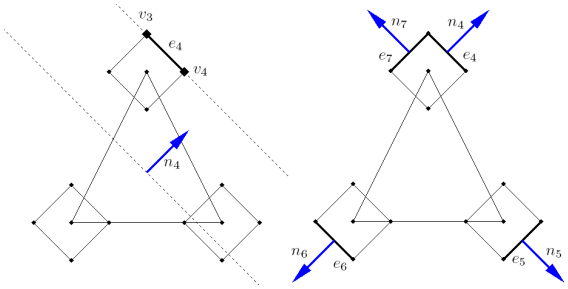

Figure 3: *Edges obtained by shifting lines parallel to rectangle-edges*

Figure 3, which is parallel to one of the edges of the rectangle along the normal n_4 we get extreme vertices v_3 and v_4, and therefore the edge e_4 of the Minkowski sum. Repeating this procedure for all normals $n_4 \dots n_7$ coming from the triangle, we get the edges shown Figure 3(right).

2.2. Minkowski Sum Generation in 3D

The Minkowski sum of two polyhedrons A and B, where B is the pivot polyhedron can be computed by

1. Creating the vertex set $\mathcal{V}$ from which the extreme vertices will be chosen
2. Creating the normal set for all possible facets of the resultant polyhedron
3. Generating the facets by pushing the planes along their normals starting from the origin

These steps will be explained on an example, where the polyhedron A is a cube and B is an octahedron. We can create the vertex set $\mathcal{V}$ simply by placing one copy of B at each vertex of A, as shown in Figure 4. The set of vertices of the resultant object is the set $\mathcal{V}$ given by

$$\mathcal{V} = (\mathcal{V}_A \times \mathcal{V}_B) \cup \mathcal{V}_A \tag{5}$$

where

$$\mathcal{V}_A \times \mathcal{V}_B = \{v_{ij}\} = \{v_{A,i} + v_{B,j}\} \tag{6}$$

for $i = 1 \dots |\mathcal{V}_A|$ and $j = 1 \dots |\mathcal{V}_B|$. Here $v_{A,i}$ and $v_{B,j}$ are the i^{th} and j^{th} vertices of A and B, respectively. In the second step, we have to bear in mind that the facets in the Minkowski sum have normals coming from three possible sources (as opposed to the two sources in 2D case):

- normals coming from the facets of A (Figure 5 top-left)
- normals coming from the facets of B (Figure 5 top-right)
- new normals coming from facets generated by sweeping the edges of B along the edges of A (Figure 5 bottom-left)

The normal set $\mathcal{N}_M$ of the Minkowski sum M is a subset of the normal set $\mathcal{N}$, or $\mathcal{N}_M \subseteq \mathcal{N}$, where $\mathcal{N}$ is given by

$$\mathcal{N} = \mathcal{N}_A \cup \mathcal{N}_B \cup \mathcal{N}_{AB} \tag{7}$$

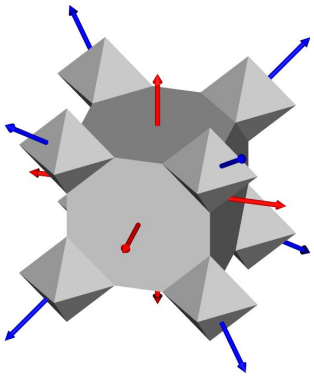

Figure 4: *Multiple Copies of B placed at all vertices of A*

where $\mathcal{N}_{AB}$ consists of normals $\{n_{ij}\}$ with

$$n_{ij} = e_{A,i} \times e_{B,j} \tag{8}$$

for $i = 1 \dots |\mathcal{N}_A|$ and $j = 1 \dots |\mathcal{N}_B|$, and where $e_{A,i}$ is the direction vector of the i^{th} edge of the polyhedron A and $e_{B,j}$ is the direction vector of the j^{th} edge of the polyhedron B.

In step 3, we generate a set of planes each with a normal

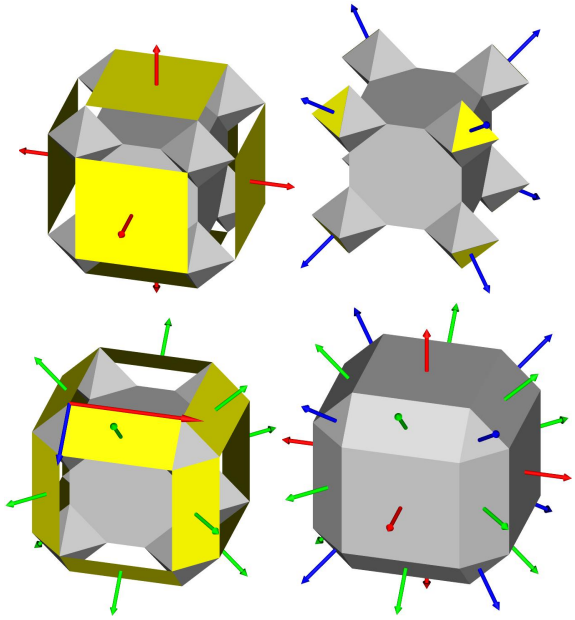

Figure 5: *Three types of faces encountered during Minkowski sum construction and the final Minkowski sum.*

drawn from the set $\mathcal{N}$ and re-position these planes by *pushing* them until extreme vertices from the set $\mathcal{V}$ are encountered. This is done by projecting each vertex onto the normal vector of the corresponding plane and keeping track of all vertices forming the same *maximum* dot-product with the

corresponding plane-normal. During this stage of the algorithm the candidate plane is assigned the number of extreme vertices it contains, i.e. all vertices forming the same extremal dot-product with the candidate plane-normal are assigned to that candidate plane. Once this set contains at least 3 non-collinear vertices, the plane is guaranteed to form a facet of the Minkowski sum. That is,

$$\mathcal{F}_i = \{\, v_k \,|\, v_k \cdot n_i = \max\{\, v_j \cdot n_i \,,\, v_j \in \mathcal{V} \,\} \,\} \qquad (9)$$

where $\mathcal{F}_i$ is the set of vertices belonging to the i^{th} facet of the final Minkowski sum. Since the set N_M contains normals from A and B, the planes having normals drawn from $\mathcal{N}_A$ or $\mathcal{N}_B$ are guaranteed to form facets in the Minkowski sum after being translated to their respective extreme vertices. This can be observed Figure 5 (left, middle), where the normals of all facets of A and B appear as facet-normals of the Minkowski sum. Figure 5 (right) also shows that additional facets are formed by pushing planes having normals drawn from the set $\mathcal{N}_{AB}$.

However, note that not all of the normals in the set $\mathcal{N}_{AB}$ have to form facets in the final Minkowski sum. In fact, a plane having a normal from the set $\mathcal{N}_{AB}$, cannot form a facet if

- A candidate plane is assigned less than 3 non-identical vertices
- If (at least 3) vertices assigned to the candidate plane are collinear

In fact, this is why the normal set of the Minkowski sum is a subset of set of all normals $\mathcal{N}$. The rules above are implemented in [BR01, FH06] using a Gaussian Map. A plane having the normal obtained by taking the cross-product of the direction vectors of two edges will contribute to the final Minkowski sum if their dual images under the Gaussian Map intersect. But in our application, since the number of normals, thus the number of the tests to be performed are very small, constructing the dual representations is a relatively time consuming process compared to our method. If visualization of the Minkowski sum is needed, some vertices need to be eliminated and only those vertices that are assigned to at least 3 distinct incident facets need to be retained. The final Minkowski sum is shown in Figure 5 (bottom-right). Here, red arrows represent the normals coming from the static polyhedron A, blue arrows represent those from the pivot polyhedron B and green arrows represent the normals of facets obtained by sweeping.

2.3. Convex Hull Construction

Since the Minkowski sum geometry is essentially a convex hull, it can be computed using Convex Hull construction methods. One widely used technique is the Incremental Convex Hull construction method, which is $O(n^2)$ [O'R94]. This method initially constructs a tetrahedron from a set of 4 non-coplanar points. It then takes all remaining points one by one and at each step identifies which triangles of the current polyhedron are visible and which are not, generating a shadow boundary between the visible and non-visible triangles. Next it eliminates all faces visible from the current point and constructs new triangles by connecting each edge on the shadow boundary with the current point. It continues with the next point in the list and repeats the same set of operations until no points are left. When computing the Minkowski sum of two convex objects, consisting of n and m vertices respectively, this method will have a computational complexity of $O(n^2m^2)$. Complex update mechanisms also add algorithmic overhead to the method.

Another method for constructing the convex hull is the divide-and-conquer method briefly described in [O'R94], which is an asymptotically faster $O(n\log(n))$ method. This method sorts all points along a coordinate axis and recursively subdivides the point-set into two halves. After the subdivision it constructs the convex hull beginning from the smallest set of points on each side and merging the resulting pair of convex polyhedra at each step. In order for this method to run in $O(n\log(n))$ time the merge operation itself must have this time complexity, which is the main characteristic of the algorithm. This method runs faster for sets consisting of a large number of points compared to the incremental method but is relatively difficult to implement.

3. Robustness Issues

Due to finite precision arithmetic, when calculating the Minkowski sum of two objects at certain relative orientations, the method we propose can produce degenerate results. The degeneracy is due to incomplete assignment of vertices to facets, or vice and versa. This causes some facets to be generated with less vertices than expected or no facets to be generated at all. Numerical errors are introduced when taking dot-products of vertices with the normals: although a certain set of vertices is expected to yield the same maximum dot product for a facet, due to finite precision arithmetic, this is not always the case.

To check if the output of our method results in such an incomplete polyhedron, we evaluate the Euler condition, which requires that for a convex polyhedron $n_v + n_f = n_e + 2$, where n_v is the number of vertices, n_f is the number of facets and n_e is the number of edges of the polyhedron. If this condition fails, we first generate an intermediate Minkowski sum by enclosing the incomplete Minkowski sum through re-triangulation of the holes. Then we use the incremental convex hull method as a local corrector to take the intermediate Minkowski sum to its final form.

Figure 6 shows an example of such a degenerate case for two cubes at a certain relarive orientation. In Figure 6 (top-left) the initial situation is shown where a copy of the smaller cube is placed at each vertex of the bigger cube. In Figure 6 (top-right) the result of our method is shown along

with edges on the hole-boundaries shown in blue. Figure 6 (bottom-left and -right) illustrate the intermediate and final Minkowski sums. Here the enclosing triangles are drawn in yellow to distinguish them from the original ones.

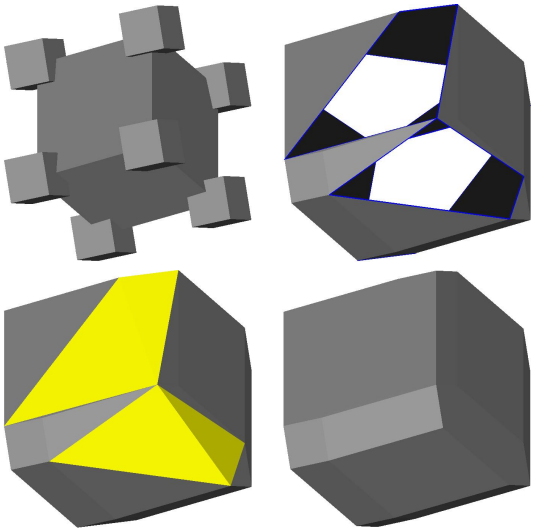

Figure 6: *Degenerate cases, their detection and correction*

4. Translational and Topological Invariance of Minkowski Sums

The Minkowski sum of two geometrical objects has the following two important properties that can prove to be very useful in certain applications:

- Translational Invariance. Minkowski sum of two objects stays the same even if the relative positions of the objects change.
- Topological Invariance. Minkowski sum of two objects has the same topology if the relative scalings of the objects change.

As shown in Figure 7 the Minkowski sums of a triangle and cubes with different sizes (but with the same orientation) have the same topology. That is their facets have the same set of normals and they are essentially the same geometric entity except their facet-sizes, edge-lengths and vertex locations. For example, when calculating the Minkowski sums of the cubes of an octree and a triangle, it is easy to see that once we calculate the Minkowski Sum of a triangle and a single cube *at any depth*, then all Minkowski sums of the cubes at other depths with the same triangle can quickly be calculated by keeping track of the index of the vertex (being the extreme vertex found by shifting the corresponding plane along its normal) selected as the origin of the bounding plane in the final Minkowski Sum. We refer to this method as *index tracking*.

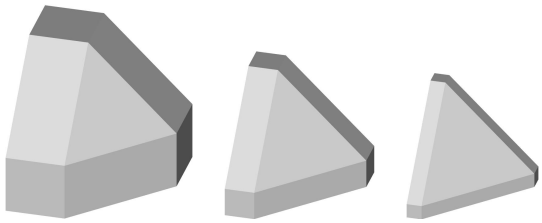

Figure 7: *Minkowski sums of the triangle and cubes at different depths*

The Minkowski sum output of our method is a set of bounding planes, each represented by a normal and a support-point that originates from the set of vertices $\mathcal{V}$. Since at each depth of the octree the vertex set $\mathcal{V}$ is obtained in the same manner, we label each vertex in V with a global vertex index that would indirectly encode the way the vertex is calculated. In case of the Minkowski sum of a triangle and a cube, the vertex set $\mathcal{V} = \{v_k\}$ consists of vertices given by

$$v_k = v_i^c + v_j^t \qquad i = 1\ldots 8,\ j = 1\ldots 3 \tag{10}$$

where $k = i + 8 \cdot j$, v_i^c and v_j^t are the i^{th} and j^{th} vertices of the cube and the triangle, respectively. Therefore k encodes the information which vertex of the cube should be added to which vertex of the triangle. This information can be recovered using $i = k \bmod 8$ and $j = \lfloor k/8 \rfloor$.

5. Performance Evaluations

We have compared the performance of the proposed geometric method with that of the quick-hull routine provided in in CGAL [HS06], which is faster than its incremental version in the same library. The comparison is done for a set of simple geometric objects that are listed in Table 1.

Object	Object	OSM	CHM
cube	rotated cube	6.1	173.4
cube	icosahedron	11.4	150.0
icosahedron	icosahedron	12.5	131.2
cube	dodecahedron	18.7	192.1
dodecahedron	dodecahedron	25.0	276.5
icosahedron	dodecahedron	82.9	203.1

Table 1: *Processing times for calculating the Minkowski sum (*OSM: Object-Space Method, CHM: Convex Hull Method. All results are given in mili-seconds*)*

Note that in case of two cubes (first row in the Table 1), which have very simple geometries, our method provides a very significant improvement. With the increasing geometrical complexity of the objects, the efficiency improvement diminishes. This is expected, since the number of dot-products required to be calculated is directly proportional to the complexity of the respective objects.

More specifically, given two objects with v_1, v_2 number of vertices, e_1, e_2 number of edges and f_1, f_2 number of faces, respectively, the number of dot-products to be calculated is roughly equal to $(v_1 v_2 + v_1) \cdot (f_1 + f_2 + e_1 e_2)$, where the dominating terms are those that come from the vertices and edges. So the overall time complexity of the method is $O(v_1 v_2 e_1 e_2)$.

We have managed to optimize the Minkowski sum calculation for two cubes in relative rotation by implementing the array versions of the geometric data structures and taking advantage of the fact that the Minkowski sum of two cubes is symmetric. As a result, the processing time given in the first row of Table 1 is reduced from 6.1 ms to 1.5 ms.

5.1. Distance Calculations

We have employed the optimized cube-to-cube Minkowski sum calculation in locating the pair of closest nodes of octrees fitted to two separate triangular mesh objects. This allowed us to find the set of closest triangles of the two respective objects. Firstly, a function is called recursively to find the pair closest nodes that have the global minimum distance between them. The maximum distance between this pair provides an upper bound for locating the nodes that contain potentially closest triangles on both objects. The pseudo-code for locating all nodes assigned with potentially closest triangles is given below:

1 Find the pair of closest nodes on both objects based on the minimum distance between the nodes.
2 Calculate the maximum distance between this pair of nodes.
3 Find all nodes whose minimum distance is less than the maximum distance found in the step 2.

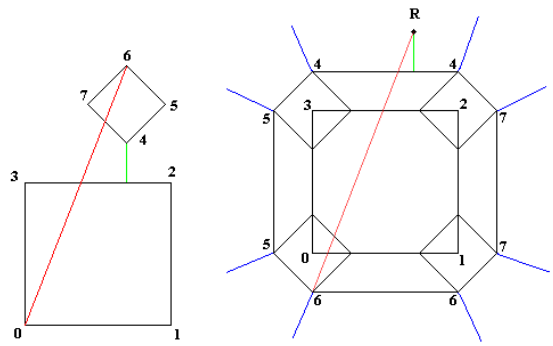

Figure 8: *Minimum (blue) and maximum (red) length line-segments connecting the closest and farthest features of two boxes*

In order to calculate the distance between a pair of cubes in general relative orientation, one of the cubes of the corresponding node is shrunk to a reference point R, which is selected as its centroid for convenience. The 2D version of such a situation is shown in Figure 8 to simplify the discussion. When one of the boxes is shrunk to it a reference point, then finding the minimum distance between them is reduced to the problem of finding the point on the Minkowski sum that is closest to the Minkowski sum. This can be done by locating the reference point R in the Voronoi region of the corresponding edge in 2D (triangle in 3D) and then calculating the minimum distance of the reference point to that edge (triangle in 3D). Note that in Figure 8, the blue lines originating at each vertex designate the boundaries between adjacent Voronoi regions. In 3D space, these boundaries will be planes obtained by sweeping the corresponding edges along a direction, which is obtained by averaging the normals of the triangles adjacent to that edge.

When calculating the distance between a pair of nodes (Figure 8), the algorithm takes advantage of the topological invariance principle: once a prototype Minkowski sum has been constructed, then all Minkowski sums of nodes of the two octrees at all possible depths can quickly be constructed by using the topology information of the prototype Minkowski sum. This results in significant speed-up, especially for deep octrees. As a result we have been able to locate the set of closest node-pairs of octrees fitted to the benchmark model (Stanford Bunny - 16K triangles) and obtain an estimate of the minimum distance between the two bunny-objects at interactive rates, whereas a brute-force approach is on the order of seconds.

Figure 9 shows a snapshot from our application. Here the blue triangles represent the set of closest triangles on both objects. The number of closest triangles on each object is approximately 380 triangles, i.e. about 97.63% of the triangles have been culled effectively. The ability of this method to cull large amount of distant triangles, even without using any normal information, makes it a suitable candidate especially for locating closest features in point-based geometries, where normal information is usually missing.

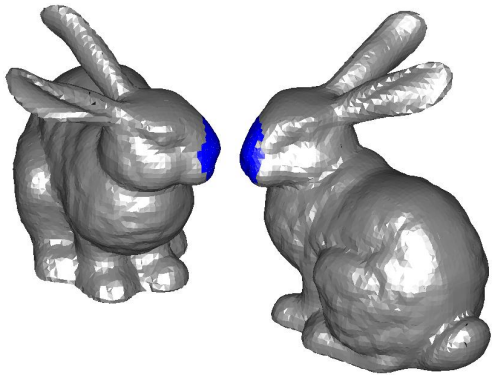

Figure 9: *Set of closest triangles for two Bunny objects*

6. Conclusions

Majority of geometrical representations used in computer graphics applications to represent 3d objects are composed of simple geometrical elements such as triangles (in triangular mesh objects) or cubes (in octrees or in bounding box representations). Minkowski sums can provide efficient and effective means to answer various queries when dealing with such objects. In this paper we have presented a method to calculate the Minkowski sums for such simple geometrical primitives. Our experience has shown that the translational and topological invariance properties of Minkowski sums are very useful in dealing with distance calculations and proximity queries when octrees are used as hierarchial representations.

As future work, we intend to compare our method to dual-space based Minkowski sum calculation methods and also plan to investigate other application areas where we can employ the proposed technique.

References

[AFH02] AGARWAL P. K., FLATO E., HALPERIN D.: Polygon decomposition for efficient construction of minkowski sums. *Comput. Geom. Theory Appl. 21*, 1 (2002), 39–61.

[BR01] BEKKER H., ROERDINK J. B. T. M.: An efficient algorithm to calculate the minkowski sum of convex 3d polyhedra. In *International Conference on Computational Science (1)* (2001), pp. 619–628.

[FH06] FOGEL E., HALPERIN D.: Exact and efficient construction of minkowski sums of convex polyhedra with applications. In *Proc. ALENEX 2006* (2006). To appear.

[Fuk04] FUKUDA K.: From the zonotope construction to the minkowski addition of convex polytopes. *J. Symb. Comput. 38*, 4 (2004), 1261–1272.

[Gho93] GHOSH P. K.: A unified computational framework for minkowski operations. *Computers & Graphics 17*, 4 (1993), 357–378.

[GRS83] GUIBAS L. J., RAMSHAW L., STOLFI J.: A kinetic framework for computational geometry. In *FOCS* (1983), pp. 100–111.

[GS87] GUIBAS L. J., SEIDEL R.: Computing convolutions by reciprocal search. *Discrete & Computational Geometry 2* (1987), 175–193.

[GS93] GRITZMANN P., STURMFELS B.: Minkowski addition of polytopes: Computational complexity and applications to gröbner basis. *SIAM J. Discrete Math. 6*, 2 (1993), 246–269.

[HS06] HERT S., SCHIRRA S.: 3d convex hulls. In *CGAL-3.2 User and Reference Manual*, Board C. E., (Ed.). 2006.

[O'R94] O'ROURKE J.: *Computational geometry in C*. Cambridge University Press, New York, NY, USA, 1994.