Exact and Efficient Construction of Minkowski Sums of Convex Polyhedra with Applications*

Efi Fogel† Dan Halperin†

Abstract

We present an exact implementation of an efficient algorithm that computes Minkowski sums of convex polyhedra in $\mathbb{R}^3$. Our implementation is complete in the sense that it does not assume general position. Namely, it can handle degenerate input, and it produces exact results. We also present applications of the Minkowski-sum computation to answer collision and proximity queries about the relative placement of two convex polyhedra in $\mathbb{R}^3$. The algorithms use a dual representation of convex polyhedra, and their implementation is mainly based on the Arrangement package of CGAL, the Computational Geometry Algorithm Library. We compare our Minkowski-sum construction with the only three other methods that produce exact results we are aware of. One is a simple approach that computes the convex hull of the pairwise sums of vertices of two convex polyhedra. The second is based on Nef polyhedra embedded on the sphere, and the third is an output sensitive approach based on linear programming. Our method is significantly faster. The results of experimentation with a broad family of convex polyhedra are reported. The relevant programs, source code, data sets, and documentation are available at `http://www.cs.tau.ac.il/~efif/CD`, and a short movie [16] that describes some of the concepts portrayed in this paper can be downloaded from `http://www.cs.tau.ac.il/~efif/CD/Mink3d.avi`.

1 Introduction

Let P and Q be two closed convex polyhedra in $\mathbb{R}^d$. The Minkowski sum of P and Q is the convex polyhedron $M = P \oplus Q = \{p + q \,|\, p \in P, q \in Q\}$. A polyhedron P translated by a vector t is denoted by P^t. *Collision Detection* is a procedure that determines whether P and Q overlap. The *Separation Distance* $\pi(P,Q)$ and *the Penetration Depth* $\delta(P,Q)$ defined as

$$\pi(P,Q) = \min\{\|t\| \,|\, P^t \cap Q \neq \emptyset, t \in \mathbb{R}^d\} ,$$
$$\delta(P,Q) = \inf\{\|t\| \,|\, P^t \cap Q = \emptyset, t \in \mathbb{R}^d\}$$

are the minimum distances by which P has to be translated so that P and Q intersect or become interior disjoint respectively. The problems above can also be posed given a normalized direction d, in which case the minimum distance sought is in direction d. The *Directional Penetration Depth*, for example, is defined as

$$\pi_d(P,Q) = \inf\{a \,|\, P^{\vec{da}} \cap Q = \emptyset\} .$$

We present an exact, complete, and robust implementation of efficient algorithms to compute the Minkowski sum of two convex polyhedra, detect collision, and compute the Euclidean separation distance between, and the directional penetration-depth of, two convex polyhedra in $\mathbb{R}^3$. The algorithms use a dual representation of convex polyhedra, polytopes for short, named *Cubical Gaussian Map*. They are implemented on top of the CGAL library [1], and are mainly based on the Arrangement package of the library [17], although other parts, such as the Polyhedral-Surface package produced by L. Kettner [28], are used as well. The results obtained by this implementation are exact as long as the underlying number type supports the arithmetic operations $+$, $-$, $*$, and $/$ in unlimited precision over the rationals,[1] such as the rational number type `Gmpq` provided by GMP — Gnu's Multi Precision library [2]. The implementation is complete and robust, as it handles all degenerate cases, and guarantees exact results. We also report on the performance of our methods compared to other.

Minkowski sums are closely related to proximity queries. For example, the minimum separation

*This work has been supported in part by the IST Programmers of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE — Motion Planning in Virtual Environments), by the IST Programmers of the EU as Shared-cost RTD (FET Open) Project under Contract No IST-006413 (ACS — Algorithms for Complex Shapes), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

†School of Computer Science, Tel-Aviv University, 69978, Israel. {efif,danha}@post.tau.ac.il

[1] Commonly referred to as a *field* number type.

distance between two polytopes P and Q is the same as the minimum distance between the origin and the boundary of the Minkowski sum of P and the reflection of Q through the origin [12]. Computing Minkowski sums, collision detection and proximity computation comprise fundamental tasks in computational geometry [26, 32, 35]. These operations are ubiquitous in robotics, solid modeling, design automation, manufacturing, assembly planning, virtual prototyping, and many more domains; see, e.g., [10, 27, 29]. The wide spectrum of ideas expressed in the massive amount of literature published about the subject during the last three decades has inspired the development of quite a few useful solutions. For a full list of packages and overview about the subject see [32]. However, only recent advances in the implementation of computational-geometry algorithms and data structures made our exact, complete, and efficient implementation possible.

Various methods to compute the Minkowski sum of two polyhedra in $\mathbb{R}^3$ have been proposed. The goal is typically to compute the boundary of the sum and provide some representation of it. The combinatorial complexity of the Minkowski sum of two polyhedra of m and n features respectively can be as high as $\Theta(m^3n^3)$. One common approach to compute it, is to decompose each polyhedron into convex pieces, compute pairwise Minkowski sums of pieces of the two, and finally the union of the pairwise sums. Computing the exact Minkowski sum of non-convex polyhedra is naturally expensive. Therefore, researchers have focused on computing an approximation that satisfies some criteria, such as the algorithm presented by Varadhan and Manocha [36]. They guarantee a two-sides Hausdorff distance bound on the approximation, and ensure that it has the same number of connected components as the exact Minkowski sum. Computing the Minkowski sum of two convex polyhedra remains a key operation, and this is what we focus on. The combinatorial complexity of the sum can be as high as $\Theta(mn)$ when both polyhedra are convex.

Convex decomposition is not always possible, as in the presence of non-convex curved objects. In these cases other techniques must be applied, such as approximations using polynomial/rational curves in 2D [30]. Seong at al. [34] proposed an algorithm to compute Minkowski sums of a subclass of objects; that is, surfaces generated by slope-monotone closed curves. Flato and Halperin [7] presented algorithms for robust construction of planar Minkowski sums based on CGAL. While the citations in this paragraph refer to computations of Minkowski sums of non-convex polyhedra, and we concentrate on the convex cases, the latter is of particular interest, as our method makes heavy use of the same software components, in particular the CGAL Arrangement package [17], which went through a few phases of improvements since its usage in [7] and recently was redesigned and re-implemented [38].

A particular accomplishment of the *kinetic framework* in two dimensions introduced by Guibas et al. [24] was the definition of the *convolution* operation in two dimensions, a superset of the Minkowski sum operation, and its exploitation in a variety of algorithmic problems. Basch et al. extended its predecessor concepts and presented an algorithm to compute the convolution in three dimensions [8]. An output-sensitive algorithm for computing Minkowski sums of polytopes was introduced in [25]. Gritzmann and Sturmfels [22] obtained a polynomial time algorithm in the input and output sizes for computing Minkowski sums of k polytopes in $\mathbb{R}^d$ for a fixed dimension d, and Fukuda [18] provided an output sensitive polynomial algorithm for variables d and k. Ghosh [19] 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 from the merged diagram. Bekker and Roerdink [9] provided a few variations on the same idea. The slope diagram of a 3D convex polyhedron can be represented as a 2D object, essentially reducing the problem to a lower dimension. We follow the same approach.

A simple method to compute the Minkowski sum of two polytopes is to compute the convex hull of the pairwise sum of the vertices of the two polytopes. While there are many implementations of various algorithms to compute Minkowski sums and answer proximity queries, we are unaware of the existence of complete implementations of methods to compute exact Minkowski sums other than (i) the naïve method above, (ii) a method based on Nef polyhedra embedded on the sphere [21], and (iii) an implementation of Fukuda's algorithm by Weibel [37]. Our method exhibits much better performance than the other methods in all cases, as demonstrated by the experiments listed in Table 4. Our method well handles degenerate cases that require special treatment when alternative representations are used. For example, the case of two parallel facets facing the same direction, one from each polytope, does not bear any burden on our method,

and neither does the extreme case of two polytopes with identical sets of normals.

In some cases it is sufficient to build only portions of the boundary of the Minkowski sum of two given polytopes to answer collision and proximity queries efficiently. This requires locating the corresponding features that contribute to the sought portion of the boundary. The *Cubical Gaussian Map*, a dual representation of polytopes in 3D used in our implementations, consists of six planar maps that correspond to the six faces of the unit cube — the parallel-axis cube circumscribing the unit sphere. We use the CGAL Arrangement package to maintain these data structures, and harness the ability to answer point-location queries efficiently that comes along, to locate corresponding features of two given polytopes.

The rest of this paper is organized as follows. The *Cubical Gaussian Map* dual representation of polytopes in $\mathbb{R}^3$ is described in Section 2 along with some of its properties. In Section 3 we show how 3D Minkowski sums can be computed efficiently, when the input polytopes are represented by cubical Gaussian maps. Section 4 presents an exact implementation of an efficient collision-detection algorithm under translation based on the dual representation, and provides suggestions for future directions. In Section 5 we examine the complexity of Minkowski sums, as a preparation for the following section, dedicated to experimental results. In this last section we highlight the performance of our method on various benchmarks. The software access-information along with some further design details are provided in the Appendix.

2 The Cubical Gaussian Map

The *Gaussian Map* G of a compact convex polyhedron P in Euclidean three-dimensional space $\mathbb{R}^3$ is a set-valued function from P to the unit sphere $\mathbb{S}^2$, which assigns to each point p the set of outward unit normals to support planes to P at p. Thus, the whole of a facet f of P is mapped under G to a single point — the outward unit normal to f. An edge e of P is mapped to a (geodesic) segment $G(e)$ on $\mathbb{S}^2$, whose length is easily seen to be the exterior dihedral angle at e. A vertex v of P is mapped by G to a spherical polygon $G(v)$, whose sides are the images under G of edges incident to v, and whose angles are the angles supplementary to the planar angles of the facets incident to v; that is, $G(e_1)$ and $G(e_2)$ meet at angle $\pi - \alpha$ whenever e_1 and e_2 meet at angle α. In other words, $G(v)$ is exactly the "spherical polar" of the link of v in P. (The link of a vertex is the intersection of an infinitesimal sphere centered at v with P, rescaled, so that the radius is 1.) The above implies that $G(P)$ is combinatorially dual to P, and metrically it is the unit sphere $\mathbb{S}^2$.

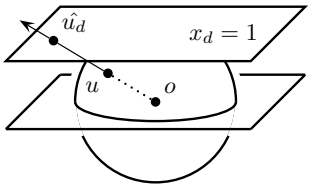


Figure 1: Central projection

An alternative and practical definition follows. A direction in $\mathbb{R}^3$ can be represented by a point $u \in \mathbb{S}^2$. Let P be a polytope in $\mathbb{R}^3$, and let V denote the set of its boundary vertices. For a direction u, we define the *extremal point* in direction u to be $\lambda_V(u) = \arg\max_{p \in V} \langle u, p \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the inner product. The decomposition of $\mathbb{S}^2$ into maximal connected regions, so that the extremal point is the same for all directions within any region forms the Gaussian map of P. For some $u \in \mathbb{S}^2$ the intersection point of the ray $\vec{ou}$ emanating from the origin with one of the hyperplanes listed below is a *central projection* of u denoted as $\hat{u_d}$. The relevant hyperplanes are $x_d = 1$, $d = 1, 2, 3$, if u lies in the positive respective hemisphere, and $x_d = -1$, $d = 1, 2, 3$ otherwise.

Similarly, the *Cubical Gaussian Map* (CGM) C of a polytope P in $\mathbb{R}^3$ is a set-valued function from P to the six faces of the unit cube whose edges are parallel to the major axes and are of length two. A facet f of P is mapped under C to a central projection of the outward unit normal to f onto one of the cube faces. Observe that, a single edge e of P is mapped to a chain of at most three connected segments that lie in three adjacent cube-faces respectively, and a vertex v of P is mapped to at most five abutting convex dual faces that lie in five adjacent cube-faces respectively. The decomposition of the unit-cube faces into maximal connected regions, so that the extremal point is the same for all directions within any region forms the CGM of P. Likewise, the inverse CGM, denoted by C^{-1}, maps the six faces of the unit cube to the polytope boundary. Each planar face f is extended with the coordinates of its dual vertex $v = C^{-1}(f)$ among the other attributes (detailed below), resulting with a unique representation. Figure 2 shows the CGM of a tetrahedron.

While using the CGM increases the overhead of some operations sixfold, and introduces degeneracies that are not present in the case of alternative representations, it simplifies the construction and manipulation of the representation, as the partition of each cube face is a planar map of segments, a well known concept that has been intensively experimented with in recent years. We use the CGAL

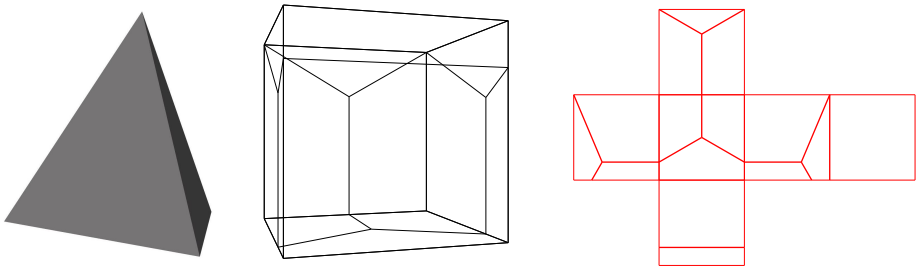


Figure 2: (a) A tetrahedron, (b) the CGM of the tetrahedron, and (c) the CGM unfolded. Thick lines indicate real edges.

`Arrangement_2`[2] data structure to maintain the planar maps. The construction of the six planar maps from the polytope features and their incident relations amounts to the insertion of segments that are pairwise disjoint in their interiors into the planar maps, an operation that can be carried out efficiently, especially when one or both endpoints are known, and we take advantage of it. The construction of the Minkowski sum, described in the next section, amounts to the computation of the overlay of six pairs of planar maps, an operation well supported by the data structure as well.

A related dual representation had been considered and discarded before the CGM representation was chosen. It uses only two planar maps that partition two parallel planes respectively instead of six, but each planar map partitions the entire plane.[3] In this representation facets that are near orthogonal to the parallel planes are mapped to points that are far away from the origin. The exact representation of such points requires coordinates with large bit-lengths, which increases significantly the time it takes to perform exact arithmetic operations on them. Moreover, facets exactly orthogonal to the parallel planes are mapped to points at infinity, and require special handling all together.

Features that are not in general position, such as two parallel facets facing the same direction, one from each polytope, or worse yet, two identical polytopes, typically require special treatment. Still, the handling of many of these problematic cases falls under the "generic" case, and becomes transparent to the CGM layer. Consider for example the case of two neighboring facets in one polytope that have parallel neighboring facets in the other. This translates to overlapping segments, one from each CGM of the two polytopes,[4] that appear during the Minkowski sum computation. The algorithm that computes it is oblivious to this condition, as the underlying `Arrangement_2` data structure is perfectly capable of handling overlapping segments. However, as mentioned above, other degeneracies do emerge, and are handled successfully. One example is a facet f mapped to a point that lies on an edge of the unit cube, or even worse, coincides with one of the eight corners of the cube. Figure 8(a,b,c) depicts an extreme degenerate case of an octahedron oriented in such a way that its eight facet-normals are mapped to the eight vertices of the unit cube respectively.

The dual representation is extended further, in order to handle all these degeneracies and perform all the necessary operations as efficiently as possible. Each planar map is initialized with four edges and four vertices that define the unit square — the parallel-axis square circumscribing the unit circle. During construction, some of these edges or portions of them along with some of these vertices may turn into real elements of the CGM. The introduction of these artificial elements not only expedites the traversals below, but is also necessary for handling degenerate cases, such as an empty cube face that appears in the representation of the tetrahedron and depicted in Figure 2(c). The global data consists of the six planar maps and 24 references to the vertices that coincide with the unit-cube corners.

The exact mapping from a facet normal in the 3D coordinate-system to a pair that consists of a planar map and a planar point in the 2D coordinate-

[2] CGAL prescribes the suffix _2 (resp. _3) for all data structures of planar objects (resp. 3D objects) as a convention.

[3] Each planar map that corresponds to one of the six unit-cube faces in the CGM representation also partitions the entire plane, but only the $[-1,-1] \times [1,1]$ square is relevant. The unbounded face, which comprises all the rest, is irrelevant.

[4] Other conditions translate to overlapping segments as well.

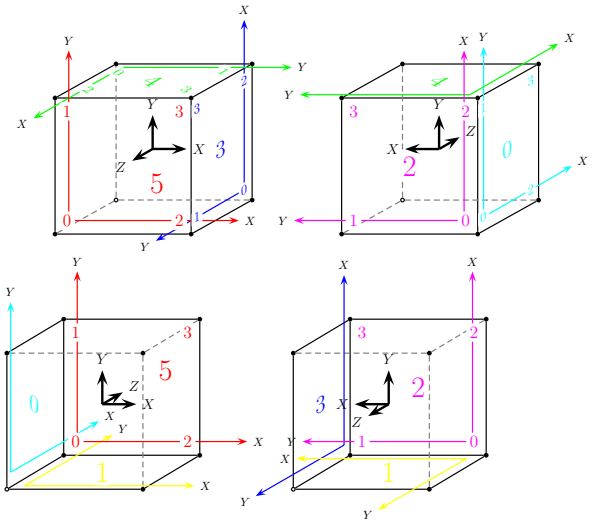
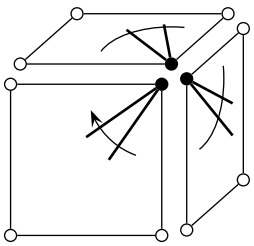


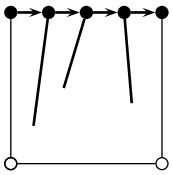
Figure 3: The data structure. Large numbers indicate plane ids. Small numbers indicate corner ids. X and Y axes in different 2D coordinate systems are rendered in different colors.

system is defined precisely through the indexing and ordering system, illustrated in Figure 3. Now before your eyes cross permanently, we advise you to keep reading the next few lines, as they reveal the meaning of some of the enigmatic numbers that appear in the figure. The six planar maps are given unique ids from 0 through 5. Ids 0, 1, and 2 are associated with planes contained in negative half spaces, and ids 3, 4, and 5 are associated with planes contained in positive half spaces. The major axes in the 2D Cartesian coordinate-system of each planar map are determined by the 3D coordinate-system. The four corner vertices of each planar map are also given unique ids from 0 through 3 in lexicographic order in their respective 2D coordinate-system, see Table 1 columns titled **Underlying Plane** and **2D Axes**.

A *doubly-connected edge list* (DCEL) data structure is used by the `Arrangement_2` data structure to maintain the incidence relations on its features. Each topological edge of the subdivision is represented by two halfedges with opposite orientation, and each halfedge is associated with the face to its left. Each feature type of the `Arrangement_2` data structure is extended to hold additional attributes. Some of the attributes are introduced only in order to expedite the computation of certain operations, but most of them are necessary to handle degenerate cases such as a planar vertex lying on the unit-square boundary. Each planar-map vertex v is extended with (i) the coefficients of the plane containing the polygonal facet $C^{-1}(v)$, (ii) the location of the vertex — an enumeration indicating whether the vertex coincides with a cube corner, or lies on a cube edge, or contained in a cube face, (iii) a boolean flag indicating whether it is non-artificial (there exists a facet that maps to it), and (iv) a pointer to a vertex of a planar map associated with an adjacent cube-face that represents the same central projection for vertices that coincide with a cube corner or lie on a cube edge. Each planar-map halfedge e is extended with a boolean flag indicating whether it is non-artificial (there exists a polytope edge that maps to it). Each planar-map face f is extended with the polytope vertex that maps to it $v = C^{-1}(f)$.



Each vertex that coincides with a unit-cube corner or lies on a unit-cube edge contains a pointer to a vertex of a planar map associated with an adjacent cube face that represents the same central projection. Vertices that lie on a unit-cube edge (but do not coincide with unit-cube corners) come in pairs. Two vertices that form such a pair lie on the unit-square boundary of planar maps associated with adjacent cube faces, and they point to each other. Vertices that coincide with unit-cube corners come in triplets and form cyclic chains ordered clockwise around the respective vertices. The specific connections are listed in Table 1. As a convention, edges incident to a vertex are ordered clockwise around the vertex, and edges that form the boundary of a face are ordered counterclockwise. The `Polyhedron_3` and `Arrangement_2` data structures for example, both use a DCEL data structure that follows the convention above. We provide a fast clockwise traversal of the faces incident to any given vertex v. Clockwise traversals around internal vertices are immediately available by the DCEL. Clockwise traversals around boundary vertices are enabled by the cyclic chains above. This traversal is used to calculate the normal to the (primary) polytope-facet $f = C^{-1}(v)$ and to render the facet. Fortunately, rendering systems are capable of handling a sequence of vertices that define a polygon in clockwise order as well, an order opposite to the conventional ordering above.



The data structure also supports a fast traversal over the planar-map halfedges that form each one of the four unit-square edges. This traversal is used during construction to quickly locate a vertex that coincides with a cube corner or lies on a cube edge. It is also used to update the cyclic chains of pointers mentioned above; see Section 3.

Underlying Plane		2D Axes		Corner							
				0 (0,0)		**1** (0,1)		**2** (1,0)		**3** (1,1)	
Id	**Eq**	***X***	***Y***	**PM**	**Cr**	**PM**	**Cr**	**PM**	**Cr**	**PM**	**Cr**
0	$x = -1$	Z	Y	1	0	2	2	5	0	4	2
1	$y = -1$	X	Z	2	0	0	2	3	0	5	2
2	$z = -1$	Y	X	0	0	1	2	4	0	3	2
3	$x = 1$	Y	Z	2	1	1	3	4	1	5	3
4	$y = 1$	Z	X	0	1	2	3	5	1	3	3
5	$z = 1$	X	Y	1	1	0	3	3	1	4	3

Table 1: The coordinate systems, and the cyclic chains of corner vertices. **PM** stands for **Planar Map**, and **Cr** stands for **Corner**.

We maintain a flag that indicates whether a planar vertex coincides with a cube corner, a cube edge, or a cube face. At first glance this looks redundant. After all, this information could be derived by comparing the x and y coordinates to -1 and $+1$. However, it has a good reason as explained next. Using exact number-types often leads to representations of the geometric objects with large bit-lengths. Even though we use various techniques to prevent the length from growing exponentially [17], we cannot avoid the length from growing at all. Even the computation of a single intersection requires a few multiplications and additions. Cached information computed once and stored at the features of the planar map avoids unnecessary processing of potentially-long representations.

3 Exact Minkowski Sums

The overlay of two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$ is a planar subdivision $\mathcal{S}$ such that there is a face f in $\mathcal{S}$ if and only if there are faces f_1 and f_2 in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively such that f is a maximal connected subset of $f_1 \cap f_2$. The overlay of the Gaussian maps of two polytopes P and Q identifies all the pairs of features of P and Q respectively that have common supporting planes, as they occupy the same space on the unit sphere, thus, identifying all the pairwise features that contribute to the boundary of the Minkowski sum of P and Q. A facet of the Minkowski sum is either a facet f of Q translated by a vertex of P supported by a plane parallel to f, or vice versa, or it is a facet parallel to two parallel planes supporting an edge of P and an edge of Q respectively. A vertex of the Minkowski sum is the sum of two vertices of P and Q respectively supported by parallel planes. A similar argument holds for the cubical Gaussian map with the unit cube replacing the unit sphere. More precisely, a single map that subdivides the unit sphere is replaced by six planar maps, and the computation of a single overlay is replaced by the computation of six overlays of corresponding pairs of planar maps. Recall that each (primal) vertex is associated with a planar-map face, and is the sum of two vertices associated with the two overlapping faces of the two CGM's of the two input polytopes respectively.

Each planar map in a CGM is a convex subdivision. Finke and Hinrichs [15] describe how to compute the overlay of such special subdivisions optimally in linear time. However, a preliminary investigation shows that a large constant governs the linear complexity, which renders this choice less attractive. Instead, we resort to a sweep-line based algorithm that exhibits good practical performance. In particular we use the overlay operation supported by the `Arrangement_2` package. It requires the provision of a complementary component that is responsible for updating the attributes of the DCEL features of the resulting six planar maps.

The overlay operates on two instances of `Arrangement_2`. In the description below v_1, e_1, and f_1 denote a vertex, a halfedge, and a face of the first operand respectively, and v_2, e_2, and f_2 denote the same feature types of the second operand respectively. When the overlay operation progresses, new vertices, halfedges, and faces of the resulting planar map are created based on features of the two operands. There are ten cases described below that must be handled. When a new feature is created its attributes are updated. The updates performed in all cases except for case (1) are simple and require constant time. We omit their details due to lack of space.

1. A new vertex v is induced by coinciding vertices v_1 and v_2.
   The location of the vertex v is set to be the same as the location of the vertex v_1 (the locations of v_2 and v_1 must be identical). The induced vertex is not artificial if (i) at least one of the vertices v_1 or v_2 is not artificial, or

(ii) the vertex lies on a cube edge or coincides with a cube corner, and both vertices v_1 and v_2 have non-artificial incident halfedges that do not overlap.

2. A new vertex is induced by a vertex v_1 that lies on an edge e_2.

3. A new vertex is induced by a vertex v_2 that lies on an edge e_1.

4. A new vertex is induced by a vertex v_1 that is contained in a face f_2.

5. A new vertex is induced by a vertex v_2 that is contained in a face f_1.

6. A new vertex is induced by the intersection of two edges e_1 and e_2.

7. A new edge is induced by the overlap of two edges e_1 and e_2.

8. A new edge is induced by the an edge e_1 that is contained in a face f_2.

9. A new edge is induced by the an edge e_2 that is contained in a face f_1.

10. A new face is induced by the overlap of two faces f_1 and f_2.

After the six map overlays are computed, some maintenance operations must be performed to obtain a valid CGM representation. As mentioned above, the global data consists of the six planar maps and 24 references to vertices that coincide with the unit-cube corners. For each planar map we traverse its vertices, obtain the four vertices that coincide with the unit-cube corners, and initialize the global data. We also update the cyclic chains of pointers to vertices that represent identical central projections. To this end, we exploit the fast traversal over the halfedges that coincide with the unit-cube edges mentioned in Section 2.

The complexity of a single overlay operation is $O(k \log n)$, where n is the total number of vertices in the input planar maps, and k is the number of vertices in the resulting planar map. The total number of vertices in all the six planar maps in a CGM that represents a polytope P is of the same order as the number of facets in the primary polytope P. Thus, the complexity of the entire overlay operation is $O(F \log(F_1 + F_2))$, where F_1 and F_2 are the number of facets in the input polytopes respectively, and F is the number of facets in the Minkowski sum.

4 Exact Collision Detection

Computing the separation distance between two polytopes with m and n features respectively can be done in $O(\log m \log n)$ time, after an investment of at most linear time in preprocessing [13]. Many practical algorithms that exploit spatial and temporal coherence between successive queries have been developed, some of which became classic, such as the GJK algorithm [20] and its improvement [11], and the LC algorithm [31] and its optimized variations [14, 23, 33]. Several general-purpose software libraries that offer practical solutions are available today, such as the SOLID library [4] based on the improved GJK algorithm, the SWIFT library [5] based on an advanced version of the LC algorithm, the QuickCD library [3], and more. For an extensive review of methods and libraries see the recent survey [32].

Given two polytopes P and Q, detecting collision between them and computing their relative placement can be conveniently done in the configuration space, where their Minkowski sum $M = P \oplus (-Q)$ resides. These problems can be solved in many ways, and not all require the explicit representation of the Minkowski sum M. However, having it available is attractive, especially when the polytopes are restricted to translations only, as the combinatorial structure of the Minkowski sum M is invariant to translations of P or Q. The algorithms described below are based on the following well known observations:

$$P^u \cap Q^w \neq \emptyset \Leftrightarrow w - u \in M = P \oplus (-Q) \ ,$$
$$\delta(P^u, Q^w) = \min\{\|t\| \,|\, (w - u + t) \in M, t \in \mathbb{R}^3\} \ ,$$
$$\pi_d(P^u, Q^w) = \inf\{\alpha \,|\, (w - u + \vec{d}\alpha) \notin M\} \ .$$

Given two polytopes P and Q in the CGM representation, we reflect Q through the origin to obtain $-Q$, compute the Minkowski sum M, and retain it in the CGM representation. Then, each time P or Q or both translate by two vectors u and w in $\mathbb{R}^3$ respectively, we apply a procedure that determines whether the query point $s = w - u$ is inside, on the boundary of, or outside M. In addition to an enumeration of one of the three conditions above, the procedure returns a witness of the respective relative placement in form of a pair that consists of a vertex $v = C(f)$ — a mapping of a facet f of M embedded in a unit cube face, and the planar map $\mathcal{P}$ containing v. This information is used as a hint in consecutive invocations. The facet f is the one stabbed by the ray r emanating from an internal point $c \in M$, and going through s. The internal point could be the average of all vertices

of M computed once and retained along M, or just the midpoint of two vertices that have supporting planes with opposite normals easily extracted from the CGM. Once f is obtained, determining whether P^u and Q^w collide is trivial, according to the first formula (of the three) above.

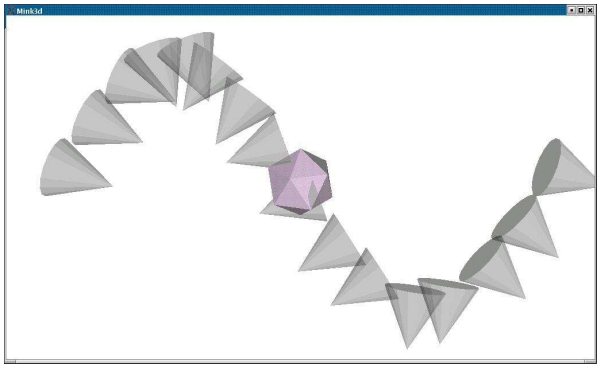


Figure 4: Simulation of motion.

The procedure applies a local walk on the cube faces. It starts with some vertex v_s, and then performs a loop moving from the current vertex to a neighboring vertex, until it reaches the final vertex, perhaps jumping from a planar map associated with one cube-face to a different one associated with an adjacent cube-face. The first time the procedure is invoked, v_s is chosen to be a vertex that lies on the central projection of the normal directed in the same direction as the ray r. In consecutive calls, v_s is chosen to be the final vertex of the previous call exploiting spatial and temporal coherence. Figure 4 is a snapshot of a simulation program that detects collision between a static obstacle and a moving robot, and draws the obstacle and the trail of the robot. The Minkowski sum is recomputed only when the robot is rotated, which occurs every other frame. The program is able to identify the case where the robot grazes the obstacle, but does not penetrate it. The computation takes just a fraction of a second on a Pentium PC clocked at 1.7 GHz. Similar procedures that compute the directional penetration-depth and minimum distance are available as well.

We intend to develop a complete integrated framework that answers proximity queries about the relative placement of polytopes that undergo rigid motions including *rotation* using the cubical Gaussian-map in the follow-up project. Some of the methods we foresee compute only those portions of the Minkowski sum that are absolutely necessary, making our approach even more competitive. Briefly, instead of computing the Minkowski sum of P and $-Q$, we walk simultaneously on the two respective CGM's, producing one feature of the Minkowski sum at each step of the walk. Such a strategy could be adapted to the case of rotation by rotating the trajectory of the walk, keeping the CGM of $-Q$ intact, instead of rotating the CGM itself.

5 Minkowski Sum Complexity

The number of facets of the Minkowski sum of two polytopes in $\mathbb{R}^3$ with m and n facets respectively is bounded from above by $\Theta(mn)$. Before reporting on our experiments, we give an example of a Minkowski sum with complexity $\Omega(mn)$. The example depicted in Figure 6 gives rise to a number as high as $\frac{(m+1)(n+1)}{2}$ when mn is odd, and $\frac{(m+1)(n+1)+1}{2}$ when mn is even. The example consists of two identical squashed dioctagonal pyramids, each containing n faces ($n = 17$ in Figure 6), but one is rotated about the Z axis approximately[5] 90° compared to the other. This is perhaps best seen when the spherical Gaussian map is examined, see Figure 5. The pyramid must be squashed to ensure that the spherical edges that are the mappings of the pyramid-base edges are sufficiently long. (A similar configuration, where the polytopes are non-squashed is depicted in Figure 8(d,e,f,g,h,i). A careful counting reveals that the number of vertices in the dual representation excluding the artificial vertices reaches $\frac{(m+1)(n+1)}{2} = 162$, which is the number of facets of the Minkowski sum. We are still investigating the problem of bounding the *exact* maximum complexity of the Minkowski sum of two polytopes. Our preliminary results imply that the coefficient of the mn component is higher than in the example illustrated here.

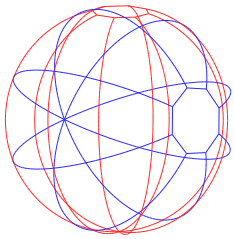
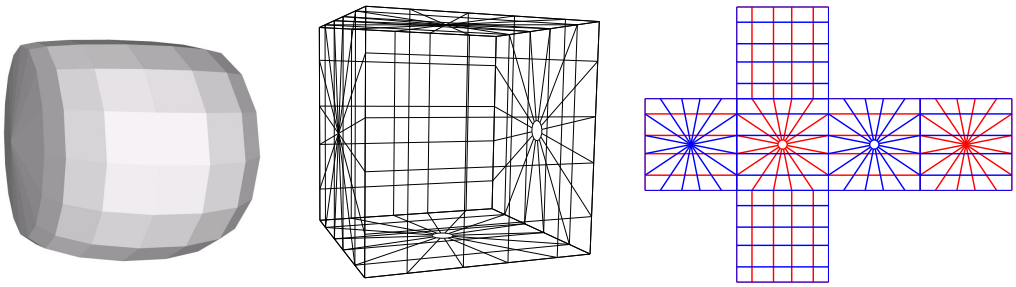


Figure 5: $m = n = 9$

Not every pair of polytopes yields a Minkowski sum proportional to mn. As a matter of fact, it can be as low as n in the extremely-degenerate case of two identical polytopes variant under scaling. Even if no degeneracies exist, the complexity can be proportional to only $m + n$, as in the case of two geodesic spheres[6] level $l = 2$ slightly rotated

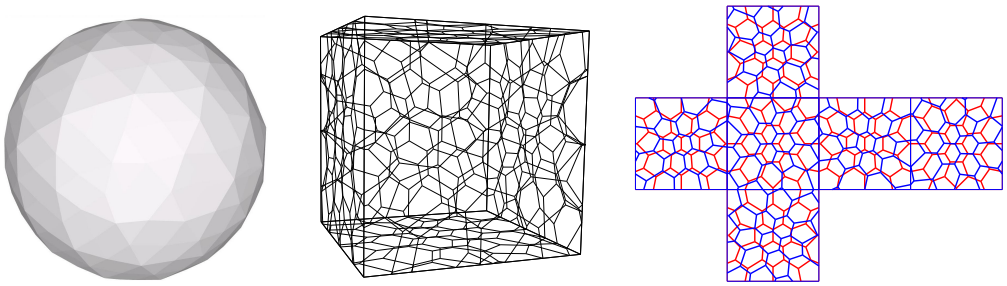
[5] The results of all rotations are approximate, as we have not yet dealt with exact rotation. One of our immediate future goals is the handling of exact rotations.

[6] An icosahedron, every triangle of which is divided into $(l+1)^2$ triangles, whose vertices are elevated to the circumscribing sphere.



(a) (b) (c)

Figure 6: (a) The Minkowski sum of two approximately orthogonal squashed dioctagonal pyramids, (b) the CGM, and (c) the CGM unfolded, where red lines are graphs of edges that originate from one polytope and blue lines are graphs of edges that originate from the other.



(a) (b) (c)

Figure 7: (a) The Minkowski sum of two geodesic spheres level 2 slightly rotated with respect to each other, (b) the CGM of the Minkowski sum, and (c) the CGM unfolded.

with respect to each other, depicted in Figure 7. Naturally, an algorithm that accounts for all pairs of vertices, one from each polytope, is rendered inferior compared to an output sensitive algorithm such as ours in such cases, as we demonstrate in the next section.

6 Experimental Results

We have created a large database of convex polyhedra in polygonal representation stored in an extended VRML format [6]. In particular, each model is provided in a representation that consists of the array of boundary vertices and the set of boundary polygons, where each polygon is described by an array of indices into the vertex array. (Identical to the *IndexedFaceSet* representation.) Constructing the CGM of a model given in this representation is done indirectly. First, the CGAL `Polyhedron_3` data structure that represents the model is constructed [28]. This data structure consists of vertices, edges, and facets and incidence relations on them. Then, the CGM is constructed using the accessible incidence relations provided by `Polyhedron_3`. Once the construction of the CGM is complete, the intermediate representation is discarded.

Table 2 shows the number of vertices, halfedges, and faces of the six planar maps that comprise the CGM of our squashed dioctagonal pyramid. The number of faces of each planar map include the unbounded face. Table 3 shows the number of features in the primal and dual representations of a small subset of our polytopes collection. The number of planar features is the total number of features of the six planar maps.

Planar map	**V**	**HE**	**F**
0, $(x = -1)$	12	32	6
1, $(y = -1)$	36	104	18
2, $(z = -1)$	12	32	6
3, $(x = 1)$	12	32	6
4, $(y = 1)$	21	72	17
5, $(z = 1)$	12	32	6
Total	105	304	59

Table 2: The number of features of the six planar maps of the CGM of the dioctagonal pyramid object.

As mentioned above, the Minkowski sum of two polytopes is the convex hull of the pairwise sum of the vertices of the two polytopes. We have implemented this straightforward method using the CGAL *convex_hull_3* function, which uses the `Polyhedron_3` data structure to represent the resulting polytope, and used it to verify the correctness of our method. We compared the time it took to compute exact Minkowski sums using these two

methods, a third method implemented by Hachenberger based on Nef polyhedra embedded on the sphere [21], and a fourth method implemented by Weibel [37], based on an output sensitive algorithm designed by Fukuda [18].

The Nef-based method is not specialized for Minkowski sums. It can compute the overlay of two arbitrary Nef polyhedra embedded on the sphere, which can have open and closed boundaries, facets with holes, and lower dimensional features. The overlay is computed by two separate hemisphere-sweeps.

Fukuda's algorithm relies on linear programming. Its complexity is $O(\delta LP(3,\delta)V)$, where $\delta = \delta_1+\delta_2$ is the sum of the maximal degrees of vertices, δ_1 and δ_2, in the two input polytopes respectively, V is the number of vertices of the resulting Minkowski sum, and $LP(d,m)$ is the time required to solve a linear programming in d variables and m inequalities. Note, that Fukuda's algorithm is more general, as it can be used to compute the Minkowski sum of polytopes in an arbitrary dimension d, and as far as we know, it has not been optimized specifically for $d = 3$.

The results listed in Table 4, produced by experiments conducted on a Pentium PC clocked at 1.7 GHz, show that our method is much more efficient in all cases, and more than three hundred times faster than the convex-hull method in one case. The last column of the table indicates the ratio $\frac{F_1F_2}{F}$, where F_1 and F_2 are the number of facets of the input polytopes respectively, and F is the number of facets of the Minkowski sum. As this ratio increases, the relative performance of the output-sensitive algorithms compared to the convex-hull method, increases as expected.

Object	Primal			Dual		
Type	V	E	F	V	HE	F
Tetrahedron	4	6	4	38	94	21
Octahedron	6	12	6	24	48	12
Icosahedron	12	30	20	72	192	36
DP	17	32	17	105	304	59
PH	92	150	60	196	684	158
TI	120	180	62	230	840	202
GS4	252	750	500	708	2124	366

Table 3: Complexity of the primal and dual representations. DP — Dioctagonal Pyramid, PH — Pentagonal Hexecontahedron, TI — Truncated Icosidodecahedron, GS4 — Geodesic Sphere level 4.

References

[1] The CGAL project homepage. http://www.cgal.org/.

[2] The GNU MP bignum library. http://www.swox.com/gmp/.

[3] The QUICKCD library homepage. http://www.ams.sunysb.edu/~jklosow/quickcd/QuickCD.html.

[4] The SOLID library homepage. http://www.win.tue.nl/cs/tt/gino/solid/.

[5] The SWIFT++ library homepage. http://gamma.cs.unc.edu/SWIFT++/.

[6] The web3D homepage. http://www.web3d.org/.

[7] P. K. Agarwal, E. Flato, and D. Halperin. Polygon decomposition for efficient construction of Minkowski sums. *Comput. Geom. Theory Appl.*, 21:39–61, 2002.

[8] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proc. 4th Annu. Euro. Sympos. Alg.*, volume 1136 of *LNCS*, pages 302–319. Springer-Verlag, 1996.

[9] H. Bekker and J. B. T. M. Roerdink. An efficient algorithm to calculate the Minkowski sum of convex 3d polyhedra. In *Proc. of the Int. Conf. on Comput. Sci.-Part I*, pages 619–628. Springer-Verlag, 2001.

[10] J.-D. Boissonnat, E. de Lange, and M. Teillaud. Minkowski operations for satellite antenna layout. In *Proc. 13th Annu. ACM Sympos. on Comput. Geom.*, pages 67–76, 1997.

[11] S. A. Cameron. Enhancing GJK: computing minimum and penetration distances between convex polyhedra. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 3112–3117, Apr. 1997.

[12] S. A. Cameron and R. K. Culley. Determining the minimum translational distance between two convex polyhedra. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 591–596, 1986.

[13] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Int. Colloq. Auto. Lang. Prog.*, volume 443 of *LNCS*, pages 400–413. Springer-Verlag, 1990.

[14] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra by multi-level Voronoi marching. In *Proc. IEEE/RST Int.. Conf. Intell. Robot. Sys.*, pages 2101–2106, 2000.

[15] U. Finke and K. H. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *Proc. 11th Annu. ACM Sympos. on Comput. Geom.*, pages 119–126. ACM Press, 1995.

[16] E. Fogel and D. Halperin. Video: Exact minkowski sums of convex polyhedra. In *Proc. ACM Sympos. on Comput. Geom.*, pages 382–383, 2005.

[17] E. Fogel, R. Wein, and D. Halperin. Code flexibility and program efficiency by genericity: Improving CGAL's arrangements. In *Proc. 12th Annu. Euro. Sympos. Alg.*, volume 3221 of *LNCS*, pages 664–676. Springer-Verlag, 2004.

[18] K. Fukuda. From the zonotope construction to the Minkowski addition of convex polytopes. *Journal*

Object 1	Object 2	Minkowski Sum						CGM	NGM	Fuk	CH	$\frac{F_1F_2}{F}$
		Primal			Dual							
		V	E	F	V	HE	F					
Icos	Icos	12	30	20	72	192	36	**0.01**	0.36	0.04	0.1	20
DP	ODP	131	261	132	242	832	186	**0.02**	1.08	0.35	0.31	2.2
PH	TI	248	586	340	514	1670	333	**0.05**	2.94	1.55	3.85	10.9
GS4	RGS4	1048	2568	1531	1906	6288	1250	**0.31**	14.33	5.80	107.35	163.3

Table 4: Time consumption (in seconds) of the Minkowski-sum computation. Icos — Icosahedron, DP — Dioctagonal Pyramid, ODP — Orthogonal Dioctagonal Pyramid, PH — Pentagonal Hexecontahedron, TI — Truncated Icosidodecahedron, GS4 — Geodesic Sphere level 4, RGS4 — Rotated Geodesic Sphere level 4, **C**H — the Convex Hull method, **C**GM — the Cubical Gaussian Map based method, **N**GM — the Nef based method, **Fuk**— Fukuda's Linear Programming based algorithm, $\frac{F_1F_2}{F}$ — the ratio between the product of the number of input facets and the number of output facets.

of Symbolic Computation, 38(4):1261–1272, 2004.

[19] P. K. Ghosh. A unified computational framework for Minkowski operations. *Comp. Graph.*, 17(4):357–378, 1993.

[20] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects. *Proc. of IEEE Int. J. Robot. Auto.*, 4(2):193–203, 1988.

[21] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3d selective nef complexes: Data structure, algorithms, and implementation. In *Proc. 11th Annu. Euro. Sympos. Alg.*, volume 2832 of *LNCS*, pages 174–186. Springer-Verlag, 2003.

[22] P. Gritzmann and B. Sturmfels. Minkowski addition of polytopes: Computational complexity and applications to Gröbner bases. *SIAM J. Disc. Math*, 6(2):246–269, 1993.

[23] L. Guibas, D. Hsu, and L. Zhang. H-walk: Hierarchical distance computation for moving convex bodies. In *ACM Sympos. on Comput. Geom.*, pages 265–273, 1999.

[24] L. J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[25] L. J. Guibas and R. Seidel. Computing convolutions by reciprocal search. *Disc. Comp. Geom.*, 2:175–193, 1987.

[26] D. Halperin, L. Kavraki, and J.-C. Latombe. Robotics. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 48, pages 1065–1093. CRC, 2004.

[27] A. Kaul and J. Rossignac. Solid-interpolation deformations: Construction and animation of PIPs. In *Eurographics'91*, pages 493–505, 1991.

[28] L. Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Comput. Geom. Theory Appl.*, 13:65–90, 1999.

[29] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[30] I. K. Lee, M. S. Kim, and G. Elber. Polynomial/rational approximation of Minkowski sum boundary curves. *Graphical Models and Image Processing*, 60(2):136–165, 1998.

[31] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proc. of IEEE Int. Conf. Robot. Auto.*, pages 1008–1014, 1991.

[32] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 35, pages 787–807. CRC, 2004.

[33] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, 1998.

[34] J.-K. Seong, M.-S. Kim, and K. Sugihara. The Minkowski sum of two simple surfaces generated by slope-monotone closed curves. In *Geom. Model. Proc.: Theory and Appl.*, pages 33–42. IEEE Comput. Sci., 2002.

[35] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Edition*, chapter 47, pages 1037–1064. CRC, 2004.

[36] G. Varadhan and D. Manocha. Accurate Minkowski sum approximation of polyhedral models. In *Proc. Comput. Graph, and Appl., 12th Pacific Conf. on (PG'04)*, pages 392–401. IEEE Comput. Sci., 2004.

[37] C. Weibel. Minkowski sums. `http://roso.epfl.ch/cw/poly/public.php`.

[38] R. Wein, E. Fogel, B. Zukerman, and D. Halperin. Advanced programming techniques applied to cgal's arrangement package. In *Library-Centric Software Design Workshop (LCSD'05)*, 2005. Available online at `http://lcsd05.cs.tamu.edu/#program`.

A Software Components, Libraries and Packages

We have developed the `Cubical_gaussian_map_3` data structure, which can be used to construct and maintain cubical Gaussian-maps, and compute Minkowski sums of pairs of polytopes represented by the `Cubical_ gaussian_map_3` data structure.[7] We have developed two interactive 3D applications; a player of 3D objects stored in an extended VRML format, and an interactive application that detects collisions and answers proximity queries for polytopes that undergo translation and rotation. The format was extended with two geometry nodes: the `ExactPolyhedron` node represents models using the CGAL `Polyhedron_3` data structure, and the `CubicalGaussianMap` node represents models using the `Cubical_gaussian_map_3` data structure. Inability to provide exact coordinates impairs the entire process. To this end, the format was further extended with a node called `ExactCoordinate` that represents exact coordinates. It has a field member called `ratPoint` that specifies triple rational-coordinates, where each coordinates is specified by two integers, the numerator and the denominator of a coordinate in $\mathbb{R}^3$. Both applications are linked with (i) CGAL, (ii) a library that provides the exact rational number-type, and (iii) internal libraries that construct and maintain 3D scene-graphs, written in C++, and built on top of `OpenGL`. We experimented with two different exact number types: one provided by LEDA 4.4.1, namely `leda_rat`, and one by GMP 4.1.2, namely `Gmpq`. The former does not normalize the rational numbers automatically. Therefore, we had to initiate normalization operations to contain their bit-length growth. We chose to do it right after the central projections of the facet-normals are calculated, and before the chains of segments, which are the mapping of facet-edges, are inserted into the planar maps. Our experience shows that indiscriminate normalization considerably slows down the planar-map construction, and the choice of number type may have a drastic impact on the performance of the code overall. The internal code was divided into three libraries; (i) `SGAL` — The main 3D scene-graph library, (ii) `SCGAL` — Extensions that depend on CGAL, and (iii) `SGLUT` — Miscellaneous windowing and main-event loop utilities that depend on the `glut` library.

The 3D programs, source code, data sets, and documentation can be downloaded from `http://www.cs.tau.ac.il/~efif/CD/3d`.

[7] We intend to introduce a package by the same name, `Cubical_gaussian_map_3`, to a prospective future-release of CGAL.

Unfortunately, compiling and executing the programs require an unpublished fairly recent version of CGAL. Thus, until the upcoming public release of CGAL (version 3.2) becomes available, the programs are useful only for those who have access to the internal release. Precompiled executables, compiled with `g++` 3.3.2 on Linux Debian, are available as well.

B Additional Models

See next page.

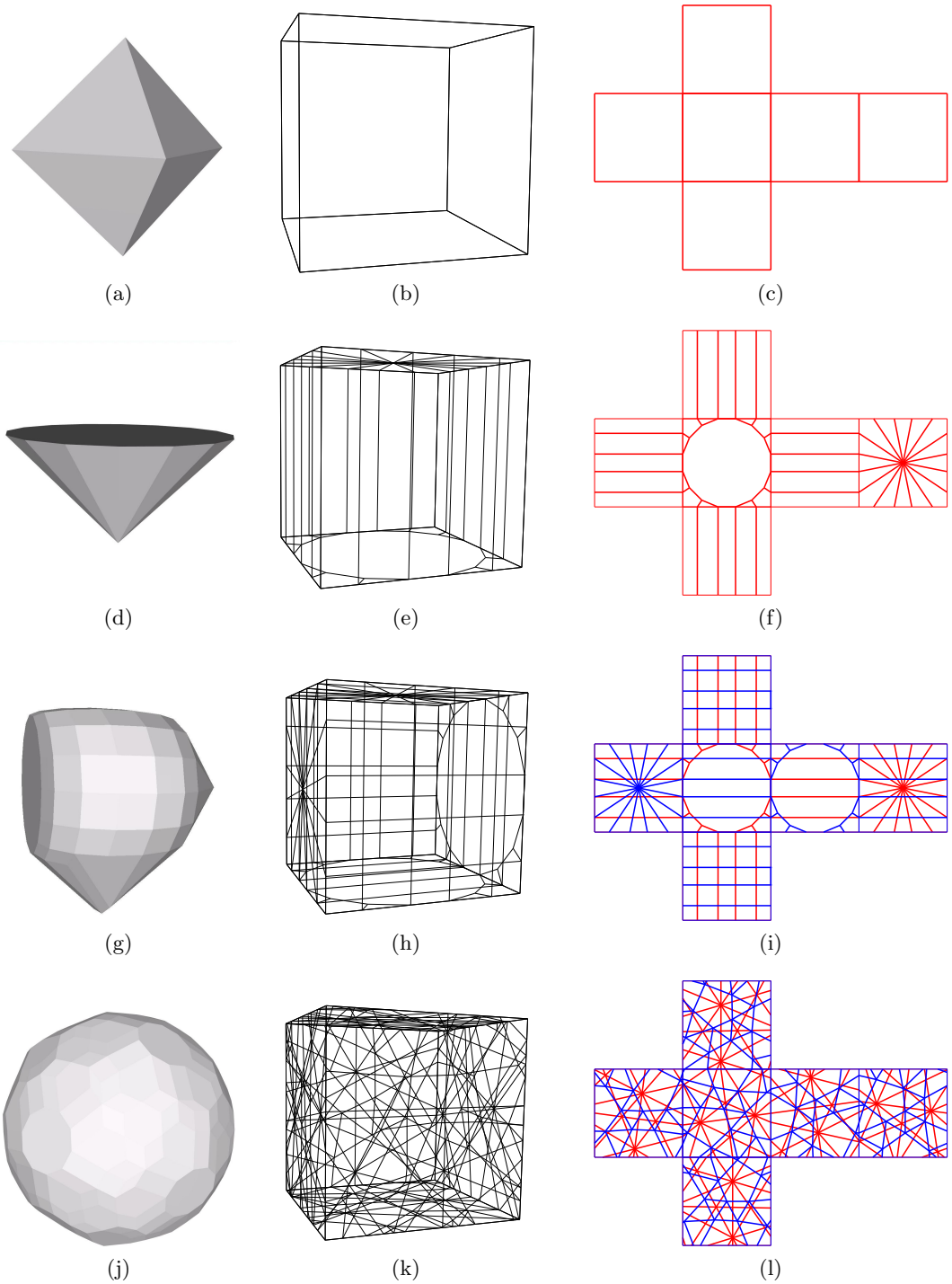


Figure 8: (a) An octahedron, (d) a dioctagonal pyramid, (g) the Minkowski sum of two approximately orthogonal dioctagonal pyramids, (j) the Minkowski sum of a Pentagonal Hexecontahedron and a Truncated Icosidodecahedron, (b,e,h,k) the CGM of the respective polytope, and (c,f,i,l) the CGM unfolded.