An Efficient Algorithm to Calculate the Minkowski Sum of Convex 3D Polyhedra

Henk Bekker, Jos B. T. M. Roerdink

Institute for Mathematics and Computing Science, University of Groningen,
P.O.B. 800 9700 AV Groningen, The Netherlands,
{bekker,roe}@cs.rug.nl

Abstract. A new method is presented to calculate the Minkowski sum of two convex polyhedra A and B in 3D. The method works as follows. The slope diagrams of A and B are considered as graphs. These graphs are given edge attributes. From these attributed graphs the attributed graph of the Minkowski sum is constructed. This graph is then transformed into the Minkowski sum of A and B. The running time of the algorithm is linear in the number of edges of the Minkowski sum.

1 Introduction: the Minkowski sum and the slope diagram

The Minkowski sum of two sets $A, B \subseteq R^3$ is defined as

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

In this article A and B are convex polyhedra in R^3, and we represent their Minkowski sum by C, so, $C = A \oplus B$. It can be easily shown that C is also a convex polyhedron, but in general C is more complex then A and B. E.g. the faces of C consist of all the faces of A and B, and some additional faces. See fig. 1a, 1b, 1f, 3b. The Minkowski sum can be defined in a space of any dimension. Amongst others, it is used in computational geometry, computer vision and imaging, robot motion planning and in pattern recognition. Our motivation for designing an efficient Minkowski sum algorithm comes from mathematical morphology. In this field we are experimenting with a method to compare the shape of two convex polyhedra, based on Minkowski addition [1, 2]. In this method, to calculate the similarity of two convex polyhedra, their Minkowski sum has to be calculated for many relative orientations (hundreds) of the polyhedra.

In 2D space, algorithms are known [4] to compute the Minkowski sum of two convex polygons A and B in linear time $O(nv_A + nv_B)$, where nv_A, nv_B are the number of vertices of A, B respectively. In R^3 two classes of algorithms exist to compute the Minkowski sum of two convex polyhedra; the ones working in R^3, and the ones working in *slope diagram space.* We will denote these two classes by MSR and MSD. In essence, MSD methods work in two dimensional space.

As we will see later, in MSD methods the polyhedra A and B are transformed to a 2D space. There the transformed polyhedra A and B are added in some way, and the result is backtransformed, giving C. In general, MSR algorithms are simpler to implement than MSD algorithms, but are less efficient.

In the literature much is said about MSR algorithms but hardly any integral and concrete discussion of MSD algorithms is available. In [3] it is shown that it is in principle possible to calculate C in linear time (Later we explain the meaning of linear in more detail). However, no concrete method or algorithm is given. In this article we discuss briefly three known algorithms, called method1..method3, and present our own algorithm, method4. Method1 is a simple and expensive MSR method. Method2 is a mixed MSR-MSD method. It is complex and not efficient. Method3 is a generic MSD method but we think it has a time complexity that is worse than the one derived in [3]. Method4 is an MSD method with a linear time complexity, and is easy to implement. Before discussing these methods we introduce our representation of polyhedra, and introduce the slope diagram.

We represent a convex polyhedron, say A, by an attributed graph. Nodes, edges and faces of this graph represent vertices, edges and faces resp. of A. Every node of the graph has an attribute representing the position of the corresponding vertex. In this paper, a polyhedron and its graph are equivalent, so, calculating the Minkowski sum of two convex polyhedra A and B is equivalent to transforming the attributed graphs A and B into an attributed graph C.

The graphs representing polyhedra are so-called *polygonal* graphs. They have the property that they are plane, and that every edge bounds two different faces. (The outer region of the graph is also a face.) A polygonal graph A may be transformed into another polygonal graph, its *dual* graph, denoted by $dual(A)$ or DA. DA is calculated as follows:

– DA has one node for each face f of A, denoted by $dual(f)$.
– DA has one edge for each edge of A. Let e be a common edge of the faces f_i and f_j of A. Then in DA the nodes $dual(f_i)$ and $dual(f_j)$ are connected by an edge, called the dual of e.

It can be easily checked that in this way the nodes of A give faces of DA. Clearly, by computing $dual(A)$ only the graph structure of DA is defined, not its attributes. For a polygonal graph A it holds that $dual(DA) = A$

A drawing of a graph on some surface (e.g. the plane or a sphere) such that no two edges cross is called an embedding of the graph. We now introduce the embedding on the unit sphere of the graphs DA and DB. We call these embeddings SDA and SDB, or the *slope diagrams* of A and B. To compute SDA, (and similarly SDB) we have to define where every node and edge of DA is mapped on the sphere. First consider the nodes. Every node n of DA is the image of some face f of A. To n is assigned as attribute the outward unit normal on the face f. The node n is mapped on the sphere to the end point of this unit vector. Second consider the edges. An edge e connecting in DA the nodes n_1 and n_2 is mapped to the arc of the unit circle on the sphere connecting the images of n_1 and n_2. For an example of two polyhedra and their slope diagrams, see fig. 1a, 1b, 1c, 1d.

A few words about designating the elements of the slope diagram. A slope diagram consists of spherical faces, spherical edges and points on a sphere. In the rest of this article we omit the word "spherical", so we will speak about the faces, edges and points of a slope diagram. So, a face, edge or point of a slope diagram is the image of a vertex, edge or face resp. of the corresponding polyhedron.

From SDA and SDB a new slope diagram may be created by *overlaying* SDA and SDB. Overlaying two embedded graphs amounts roughly speaking to superimposing the two graphs and merging them into one graph [6, 7]. An important well known property of the Minkowski sum is that the slope diagram of C is identical to the overlay of the slope diagrams of A and B [2], so,

$$SDC = overlay(SDA, SDB).$$

The node positions of SDC consist of (i) the node positions of SDA and SDB, and (ii) the node positions defined by intersecting edges of SDA and SDB. The first ones may be copied from SDA and SDB to SDC, the latter ones are obtained during the overlay calculation. It is important to note that by calculating the dual of SDC the graph structure of C is obtained, but because this graph has no node attributes, no complete description of C is available yet. In a later section we show how in method3 and method4 the attributes of SDC are calculated.

2 Some common methods to calculate the Minkowski sum

Method1 is a pure MSR method; it is simple but time consuming [5]. It is a two step process. In the first step the position vectors of all the vertices of A are added to the position vectors of all the vertices of B. This results in a total of $nv_A nv_B$ points where nv_A and nv_B are the number of vertices of A and B resp. In the second step the convex hull of these points is computed, giving C. Obviously, the first step has time complexity $O(nv_A nv_B)$. Using some standard convex hull algorithm [4] the second step has time complexity $O(nv_A nv_B \log(nv_A nv_B))$. This method of computing C is expensive because it works entirely in R^3, whereas using SDA and SDB implies working in R^2. Another disadvantage is that the result is not a graph but a set of points. Yet, this method is often used when efficiency is not crucial.

Method2 is a mixed *MSR-MSD* method. The key idea of this method is to compute all planes bounding C, i.e. all planes that contain a face of C. By calculating the intersections of these planes, the edges and vertices of C are computed. The method works as follows.

1. For every face f of A it is determined in which face of SDB the slope diagram image of f is located. This face of SDB is the image of some vertex of B,

say v. Now the plane containing the face f is translated over the position vector of v. The resulting plane is a bounding plane of C.
2. The same as 1 with A and B interchanged.
3. In the superimposed slope diagrams of A and B it is determined which edges of SDA intersect edges of SDB. Assume that the edges se_i and se_j intersect. Assume that the corresponding edges in A and B are e_i and e_j. Now we construct a plane containing e_i that is parallel with e_j. This plane is shifted over a vector ending somewhere on e_j (say one of its endpoints). The resulting plane is a bounding plane of C.

The intersection of the half spaces defined by the planes described above is C. The faces of C contained in the planes as constructed in step 1 and 2, have the same shape and size as the faces of A resp. B, i.e. are shifted instances of the faces of A resp. B. The faces of C contained in the planes constructed in 3 are new faces, i.e. are not copies of the faces of A or B. These faces are parallelograms with edges e_i and e_j. See figure 1f, 3b for examples. Method2 is more efficient than method1 because it uses slope diagrams. Yet, it contains much redundant work: C contains many faces identical with faces of A and B, but this fact is not used in this method. Most faces are completely reconstructed.

Concluding: in both methods too much geometrical computations are done. The method we propose aims at minimizing these geometrical computations.

3 The Minkowski sum by merging attributed graphs

Method3 is a straightforward MSD method, but in the literature we could not find an integral description of it. It consists of the following four steps.

1. Calculate the slope diagram SDA. Besides the earlier mentioned node attributes (unit vectors), the slope diagram is given face attributes. Every face f of SDA is given an attribute $attr(f)$, namely the position vector of the corresponding vertex in A. The attributed slope diagram SDB is calculated similarly.
2. Calculate the overlay of SDA and SDB, that is, calculate the graph of SDC. This graph has no attributes yet.
3. Calculate the face attributes of SDC. This is done as follows. When SDA, SDB and SDC are superimposed, every face f of SDC is located in precisely one face f_i of SDA, and in precisely one face f_j of SDB. Face f gets as attribute the sum of the attributes of f_i and f_j.
4. Calculate the dual graph C of SDC as follows. Copy from SDC the face attributes to the corresponding nodes of C. The graph C, with its node attributes, represents the Minkowski sum of A and B.

In the following, the process of determining for every face of SDC in which face of SDA and SDB it is located (see point 3), will be called *face location*. It is instructive to compare method3 with method1. In method1 all vertices of A are combined with all vertices of B. Afterwards, during the convex hull computation,

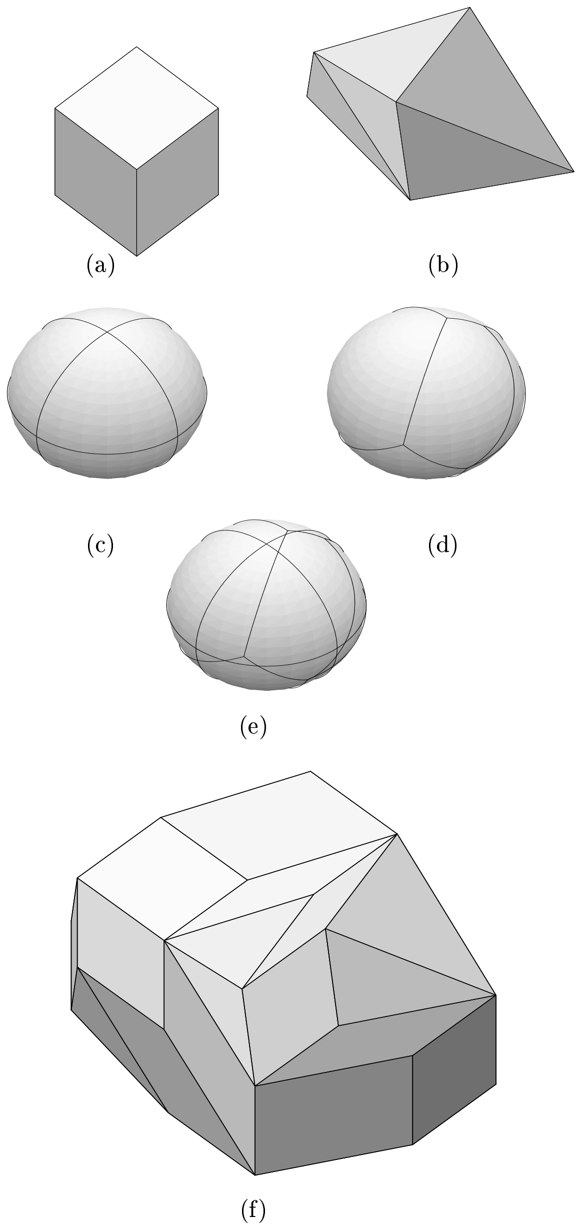

Fig. 1. Two polyhedra A and B (a) (b), their slope diagrams SDA and SDB (c), (d), the overlay of these slope diagrams SDC (e), and the Minkowski sum C of the polyhedra A and B, (f). It may take some time to see the relation between (b) and (d). It can be seen that the (f) consists of the faces of (a) and (b), and additional parallelepiped faces (See also figure 3b). MSR methods calculate (f) directly from (a) and (b). MSD methods use the slope diagrams in (c), (d) and (e) to calculate (f).

it is decided which of these points are vertices of C. In method3, by face location, it is decided which vertices of A and B have to be combined to give a vertex of C.

Using a standard graph method [8], the time complexity of calculating SDA and SDB is $O(ne_A + ne_B)$ where ne_A and ne_B are the number of edges of A and B. In the next section we show that calculating the overlay of SDA and SDB can be done in time $O(ne_A + ne_B + k)$ where k is the number of intersecting edges of SDA and SDB. Method3 may be summarized as follows. Step 1 and 4 are transformations to and from the slope diagram domain. In step 2 the overlay is constructed, and in step 3 the face attributes of SDC are calculated. In the following sections we will take a closer look at calculating the overlay and face location.

4 Overlaying and face location

Overlaying two subdivisions of the plane is a standard problem of computational geometry. Unfortunately, for our problem, i.e. calculating the overlay of two subdivision of the sphere, no implementations are available, so we had to develop our own implementation. For this we adapted an existing implementation in the plane [6, 7] that runs in linear time $O(ne_A + ne_B + k)$, where k is the number of intersecting edges of SDA and SDB. An additional feature of our implementation is that the edges in the overlay SDC get an attribute indicating from which edge of SDA or SDB the edge stems. Let us explain. When SDA, SDB and SDC are superimposed, every edge e of SDC coincides with part of or a whole edge of SDA or SDB (that is roughly speaking the definition of an overlay). During face location, it is necessary to know which set of edges of SDC bound a given face of SDA or SDB. Therefore, during the overlay construction, every edge of SDC is given two attributes, one referring to an edge of SDA and one referring to an edge of SDB. In general, only one of these references is non-nil. Only when an edge of SDA (partially) coincides with an edge of SDB, both references are non-nil. This situation occurs for example in the extreme case when the Minkowski sum of two identical polyhedra is calculated, i.e. when $C = A \oplus A$.

Now consider face location. The edge attributes of SDC as described above make it possible to find in SDC the edges that coincide with the edges of SDA or SDB, and that bound a given face of SDA or SDB. We call such a set of edges an A-cycle or a B-cycle. Let us look at some A-cycle a. See fig. 2(c). We want to find all faces of SDC that are within a. First we collect those nodes of SDC that are on or within a. Let us call the set of nodes on and inside a, $a.nodes$. Nodes on a are found by going through the edges of a. Nodes within a are found when, starting from every node on a, inward edges are followed recursively. Using the $a.nodes$, we collect all faces of SDC that have one or more nodes of $a.nodes$ as vertex. The faces collected in this way are within or directly adjacent to a. From these faces those ones are selected that only have vertices from $a.nodes$. These faces are inside a, and get an attribute referring to the face of SDA corresponding to a. This is done for all A-cycles and B-cycles. In this way every face of SDC gets two attributes telling in which face of SDA and SDB it is located. Using these attributes, every face of SDC is given a vector valued attribute in the following way. If face f of SDC is in face f_i of SDA and in face F_j of SDB then face f gets a vector attribute $attr(f) = attr(f_i) + attr(f_j)$. In LEDA [8], the Computational Geometry platform we use, all operations in the face location algorithm above are available as standard methods.

We will not discuss the time complexity of face location. In the first place because it is not trivial, and second because in the in the next section we present our MSD method, i.e. method4, that works without face location. Because the time complexity of method4 is dominated by calculating the overlay, method4 is superior to method3, whatever the time complexity of face location in method3 may be.

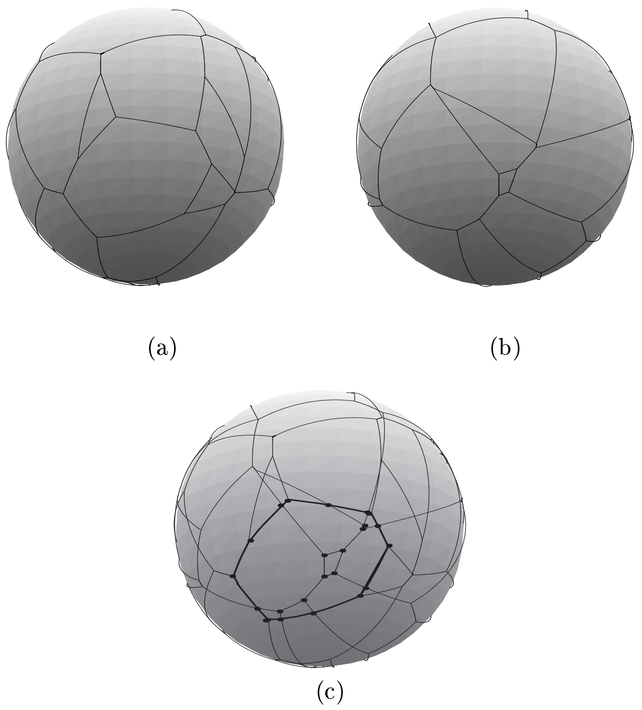

Fig. 2. The slope diagrams SDA (a) and SDB (b) of two randomly generated polyhedra, and the overlay SDC (c) of these slope diagrams. In SDC an A-cycle a is shown. The nodes on and inside a are marked.

5 A method without face location

In method3 face location was essential for calculating the face attributes of SDC, i.e. for calculating the vertex positions of C. We now present method4, which works with edge attributes instead of face attributes, and thus avoids face location. As a side effect, the absolute position of C is lost. However, in a final step this position is recovered. To describe and implement method4 we use bidirected graphs. This means that every edge e of A, B, C, SDA, SDB and SDC has a *source node* and a *target node*, designated by $source(e)$ resp. $target(e)$. Moreover, for every edge there is a reversal edge, i.e. when there exists an edge e starting at $source(e)$ and ending at $target(e)$ then there is also an edge starting at $target(e)$ and ending at $source(e)$.

Method4 is a six step process and works as follows.

1. Switch to relative coordinates of A and B. More precisely, instead of using node attributes representing absolute node positions, we switch to edge attributes. Each edge of A and B is attributed with a 3D vector. The vector is the relative position of the target of the edge w.r.t. the source of the edge, so, for edge e the attribute $attr(e)$ is given by $attr(e) = position(target(e)) - position(source(e))$.
2. Calculate SDA and SDB. Copy the edge attributes described in step 1 to the corresponding edges of SDA and SDB.
3. This is the crucial step. First compute the overlay SDC. As described in method3, during the overlay construction, every edge e of SDC gets two attributes indicating from which edge of SDA or SDB e stems. Using these attributes, calculate the attributes for every edge e of SDC in the following way. When e stems from only one edge, so an edge of SDA (exclusive)-or SDB, e is given the vector attribute of this edge. When e stems from an edge of SDA and an edge of SDB, e gets as attribute the vector sum of the attributes of these edges.
4. Calculate the dual of SDC, called C. Of every edge of SDC the edge attribute is copied to the corresponding edge of C.
5. Calculate node attributes of C, representing vertex positions of C as follows. Choose some node n_0 of C and assign to it some freely chosen position $pos(n_0)$, for example $(0, 0, 0)$. Then for every edge e which has n_0 as source, visit the node target(e), and assign to it the attribute $pos(n_0) + attr(e)$, i.e. to all nodes directly connected with n_0 are assigned the position of n_0 plus the edge vector of the connecting edge. This process is continued until every node has been visited.
6. Shift C to the correct position. This is done with three similar operations, one for the x direction, one for the y direction and one for the z direction. We explain the x direction shift.
   Let A_max_x be the x position of the most extreme point(s) of A in the positive x direction, and similarly for B_max_x and C_max_x. It can be easily checked that it should hold that

$$C\_max\_x = A\_max\_x + B\_max\_x. \tag{2}$$

In the previous step C was placed at a provisional position in space. Let $prov\_C\_max\_x$ be the maximal x position of C at this provisional position. Then, by shifting C over

$$A\_max\_x + B\_max\_x - prov\_C\_max\_x \tag{3}$$

C gets its correct position in the x direction. By a similar shift in the y and z direction C gets its correct position.

6 Discussion

Method3 works because a convex polyhedron is defined by its vertices. Method4 works because a polyhedron is defined, up to its absolute position, by its edge vectors. Method4 works without face location. Instead, edge location is done. The advantage of method4 over method3 is that edge location has already been done during the overlay phase without overhead. In terms of the number of computations, the advantage of not having to do face location outweighs the need to restore the absolute position of C.

As mentioned before, the overlay may be computed in time $O(ne_A + ne_B + k)$. The time complexity of computing SDA and SDB, of attribute manipulation, and of shifting C to the correct position, is inferior to the time complexity of computing the overlay, so, method4 has a time complexity of $O(ne_A + ne_B + k)$. Obviously, this is better than method1 and method2. Probably it is also better than method3 because we think that the time complexity of face location is greater than $O(ne_A + ne_B + k)$.

Why does method4 work? We can prove that it is correct by using the support function [9, 10], but because of limited space it can not be given here. We only make a few remarks which may serve as a starting point of a proof.

As remarked earlier, most faces of C are identical to the faces of A and B, plus additional paralellogram faces with one edge from A and one edge from B. So, the edges of C only consist of edges of A and B, and occasionally an edge that is the sum of an edge of A and an edge of B. The last type of edge occurs when an edge in SDA (partially) coincides with an edge in SDB. In step 3 of method4 these three kinds of edges of C are created, that is, every edge of C is an edge of A or of B or the sum of an edge of A and of B.

Another remark. The edges of SDC (partially) coincide with edges of SDA or SDB. To cover completely an edge e of say SDB with edges of SDC may require two or more edges of SDC. In step 3 of method4 each of these edges of SDC gets the same attribute, indicating that C gets some parallel edge vectors. More precisely: When an edge of SDB is subdivided in SDC in n edges this indicates that C will have n parallel edges, parallel to the corresponding edge of B. See fig. 3.

7 Conclusion

We have shown that the Minkowski sum of two convex polyhedra may be computed almost entirely in the slope diagram domain, and that the usual face

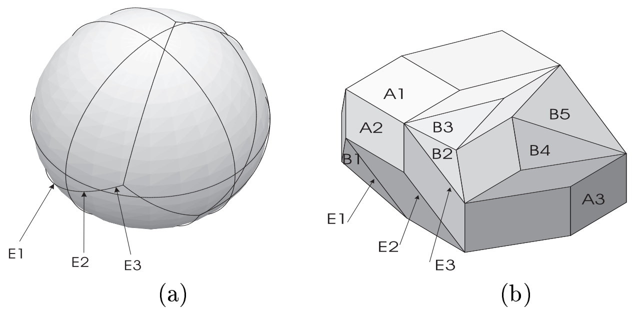

Fig. 3. The slope diagram SDC and C from figure 1. In SDC an edge of SDB is subdivided in three edges of SDC (See three arrows $E1..E3$). This results in three parallel edges in C (See three arrows $E1..E3$) between $B1$ and $B2$. In B (see figure 1) there was only one edge between $B1$ and $B2$. In C the faces from A and B are marked with $A1..A3$ and $B1..B5$.

location can be avoided, leading to a more efficient algorithm. The crucial part of the method is the construction of the attributed overlay of the slope diagrams of A and B. Further, only simple attribute manipulations and simple geometrical computations are used. The time complexity of the method is linear in the size of the input plus output.

Literature

[1] H. Bekker, J. B. T. M. Roerdink: Calculating critical orientations of polyhedra for similarity measure evaluation. Proc. of the IASTED Int. Conf. Computer Graphics and Imaging, 1999, Palm Springs, USA. p. 106-111
[2] A. V. Tuzikov, J. B. T. M. Roerdink, H. J. A. M. Heijmans: Similarity Measures for Convex Polyhedra Based on Minkowski Addition. Pattern Recognition 33 (2000) 979-995
[3] L. J. Guibas, R. Seidel: Computing convolutions by reciprocal search. Discrete and Computational Geometry. Vol. 2, p. 175-193, 1987.
[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: Computational Geometry, Algorithms and Applications. Springer Verlag, Berlin. (1997)
[5] P. K. Ghosh: A unified computational framework for Minkowski operations. Comput. & Graphics, Vol. 17, No. 4, 1993.
[6] U. Finke, K. H. Hinrichs: Overlaying simply connected planar subdivisions in linear time. Proc. of the 11th Int. symposium on computational geometry, 1995.
[7] A. M. Brinkmann: Entwicklung und robuste Implementierung eines laufzeitoptimalen Verschneidungsoperators für Trapezoidzerlegungen von thematishen Karten. PhD. Thesis, University of Münster, 1998.
[8] K. Melhorn, S. Näher: LEDA A Platform for Combinatorial and Geometric Computing. Cambridge University press,Cambridge. 1999
[9] H. G. Eggleston: Convexity. Cambridge University Press, Cambridge. 1958
[10] H. Busemann: Convex Surfaces. Interscience, Inc., New York. 1958