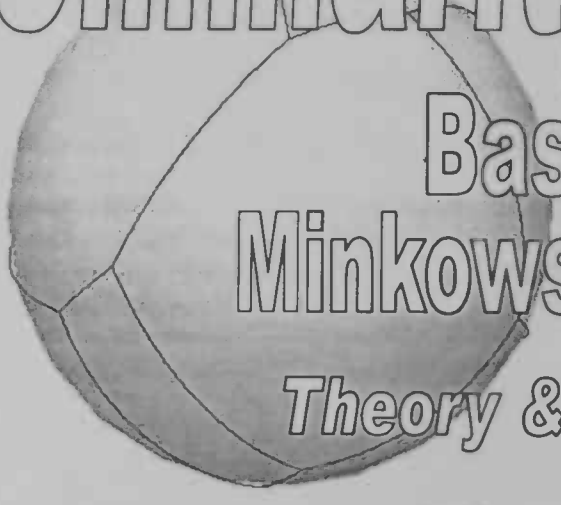
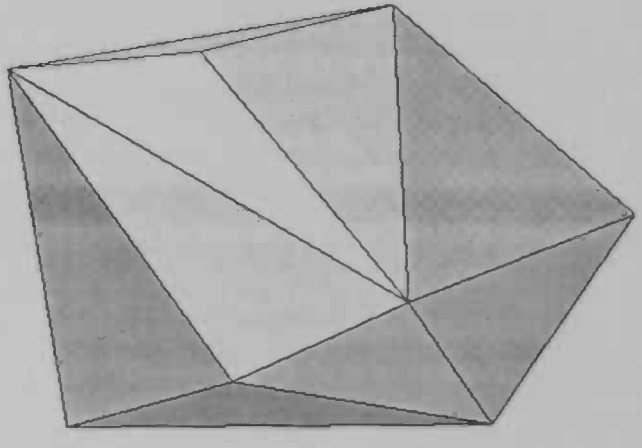
Similarity Measure

Based on Minkowski Addition

Theory & Experiments





Zaher Srour

Supervisor Dr. H. Bekker

Department of mathematics and computer science,
University Of Groningen, The Netherlands

July 24, 2001

Introduction

Given two convex polyhedra, can one tell us how similar they are?

The notion of shape and the notion of similarity of shapes is typical elements of the mathematical morphology, the science that has its roots as a theoretical framework for the quantitative description of shapes and size, with sets. Mathematical morphology is therefore an important formalism in the image processing and computer vision. [For more information about Mathematical Morphology, we refer to the publications of H Heijmans]

In the literature there is, as far as we know, no method that can calculate the similarity measure of two convex polyhedra.

In this thesis we focus on a newly introduced approach based on the Brunn-Minkowski inequality to calculate the similarity of two convex polyhedra. This approach has the advantages that it is purely mathematical, is invariant under translations and possibly under some scaling, rotation and reflection. This approach may be used in any-dimensional space, but we will focus on the 3D-space.

To calculate the similarity measure of two convex polyhedra, following this approach, a function has to be minimized over a finite number of (relative) orientations of these polyhedra. There are two classes of these orientations: $class_1$ and $class_2$. In the literature [1], it has been conjectured that to perform the minimization for that function, it is sufficient to consider $Class_1$ orientations only. An analytical answer on the question "Is it sufficient to consider $Class_1$ orientations only?" could not be found. In this thesis we try to give an answer computationally. Finding a computational answer to this problem is the main goal of this thesis.

Introducing the theory behind this problem, implementing it in a program and testing the program to get an answer for the question form the structure of the thesis. It consists of three parts: theory, implementation and experiment.

Here is a description of these parts:

I. Theory

We introduce the theories behind the similarity measures based on minkowski sums and mixed volumes. These theories are the base for the implementation. After introducing all the theories in the last chapter of this part the question we have to answer is introduced. Answering this question is the goal of this thesis.

II. Implementation

The implementation of the theories introduced in part I is discussed. Some information is given about the approach and technical decisions. Almost all the headings of the chapters in this part are identical to those in part I.

III. Experiments and results

We use the program to perform some experiments. We show the results of these experiments and draw conclusions from these results.

I. Theory

I.0 Introduction

In this part of the thesis we will introduce the theory behind the similarity measures for two convex polyhedra based on minkowski sums and mixed volumes.

All the theory behind this approach can be summarised in one equation, a similarity measure function- equation. Such equation includes:

- Two polyhedra that have to be checked for similarity.
- A minkowski addition (mixed) volume functional.
- The volume of a polyhedron.
- The critical orientation set.

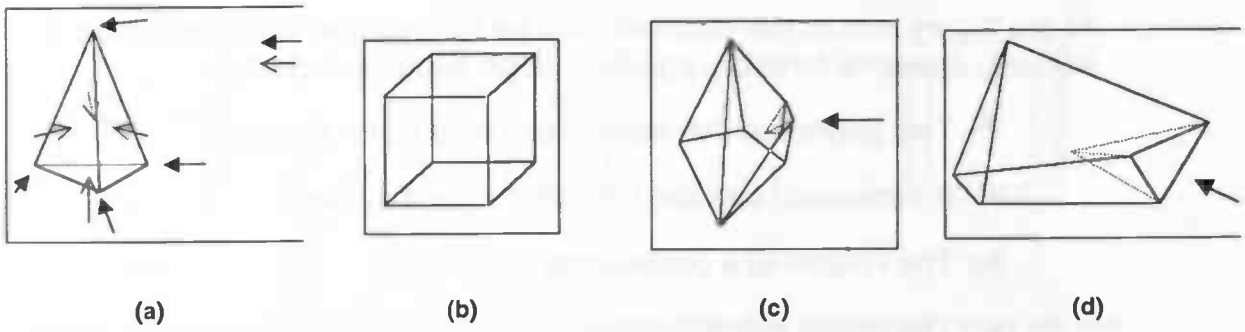
These are the keywords of this thesis and they will be introduced and discussed in the following chapters.

- In the first chapter we will talk about the convex polyhedra. What are they, and what are their properties?
- In the second chapter a representation of polyhedra is introduced: the slope diagram representation. This representation will give us more insight in the relative orientations of the polyhedra.
- The minkowski addition of two polyhedra and the theory behind it is the subject of the third chapter. We will give examples about this topic to make the subject clear and understandable.
- The mixed volume will be introduced in chapter four.
- Orientations in general and the critical orientations in particular are introduced in chapter five. The two classes of critical orientations and their properties are introduced. With this chapter the list of elements needed to introduce the similarity measures is completed.
- In chapter six the similarity measures based on minkowski sums and mixed volumes are introduced.
- The question-goal that has to be answered in this thesis is formulated in chapter seven, and our approach, how we think to solve the problem, is given. This results in a to-do-list, which is the guideline of the Implementation part.

I.1 What is a Polyhedron?

A *polyhedron* is a region of 3D-space whose boundaries are flat faces, where a *face* is a polygon. Two faces of a polyhedron possibly meet in a line segment called an *edge*. Three faces possibly meet in a point called a *vertex*.

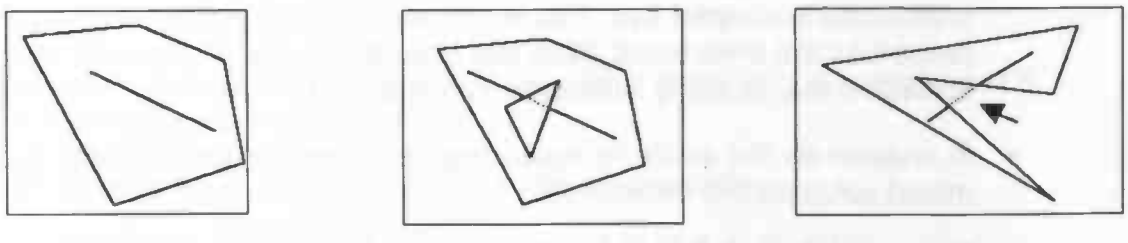
Some example polyhedra are shown in figure1. In figure 1.1(a) the parts of the polyhedron referred by arrows are the faces, and the vertices.



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

Figure1.1: Some examples of polyhedra. (a) shows a tetrahedron. Here the filled-head-arrows represent the corners (vertices) of the polyhedron, the edges are the line segments between the vertices. The not-filled-head-arrows point to the four faces. (b) shows a cube with six faces, eight vertices and twelve edges. (c) and (d) show two non-convex polyhedra. These polyhedra are not convex because they have dents. An arrow points these dents.

For us, only *convex* polyhedra are of interest. In figure1.1 only the first two polyhedra are convex. The last two polyhedra are not convex. A set S is *convex* when every line l whose endpoints are in S, is in S. See figure1.2-a. More formal, if ($x \in S$) and ($y \in S$) then segment xy is included in S, where the segment xy is the set of all points on the straight line between x and y. In plain language this means that a convex polyhedron has no dents. So, in figure1.1 only the first two polyhedra are convex. In figure1.2 three sets are given, the first is convex and the second and third not.



(a) (b) (c)

Figure1.2: three sets in R^2 (a), (b) and (c). (a) is convex, there are no dents in it. The line segment drawn in (a) stays inside the set. This is not the situation in (b) and (c), here we clearly see that there are line segments with there endpoints inside the two sets but smaller segments of these line segments, namely the dotted line segments, fall outside the set.

In the following we assume that we only have to do with non-degenerate convex polyhedra. This means that our polyhedra do not consist of a single point, a single line or a plane, so the volume is greater than zero.

It is easily verified that the smallest possible polyhedron is the tetrahedron, a pyramid like polyhedron with four faces and vertices, and six edges. See figure1.1(a).

The tetrahedron is the first member of the five well-known special polyhedra group, the Tetrahedron, Cube, Octahedron, Dodecahedron and Icosahedron. These polyhedra have the property that their faces are regular polygons. In the following table we give the five polyhedra with their properties.

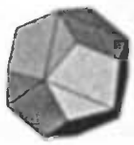
				
Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
4 faces (reg. triangles)	6 faces (squares)	8 faces (reg. triangles)	12 faces (pentagons)	20 faces, reg. Triangles
6 edges	12 edges	12 edges	30 edges	30 edges
4 vertices	8 vertices	6 vertices	20 vertices	12

Table1: The five Platonic polyhedra: Tetrahedron, cube, octahedron, dodecahedron and the Icosahedron. The polyhedra are given with the number of faces, edges and vertices.

A convex polyhedron can also be defined by its support function:

Theorem

Every element A of the set of all compact sets ***C*** (in 2D and 3D-space) is uniquely determined by its *support function*:

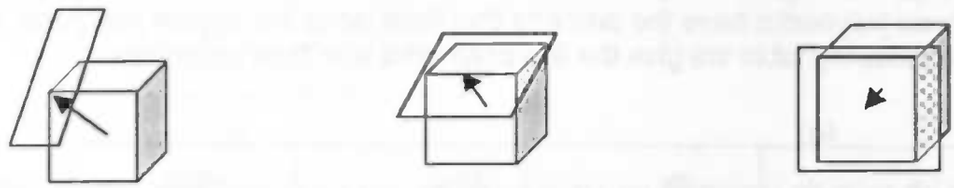
$$h(A,u) = \sup\{\langle a,u\rangle \mid a \in A\}, \quad u \in S^2 \text{ (or } u \in S^1)$$

Here is $\langle a,u\rangle$ the inner product of vectors a and u. S^2 denotes the unit sphere in R^3 (S^1 denotes the unit circle in R^2).

Definition: The plane having **u** as normal vector and ***h*(A,u)** as distance from the origin, is called the *support plane* of A in direction u.

Definition: The *support set*, denoted by F(***A***,**u**), of polyhedron ***A*** at $\mathbf{u} \in \mathbf{S}^2$ consists of all points $\mathbf{a} \in \boldsymbol{A}$ for which $\langle \mathbf{a},\mathbf{u}\rangle = h(\boldsymbol{A},\mathbf{u})$.

Roughly speaking, the support set of polyhedron ***A*** in direction **u** consists of all boundary points of ***A*** which touch the support plane of ***A*** in direction **u**. If the support plane touches one point, then the support set is equal to the vertex with that location, see figure1.3(a). If the plane touches two vertices, then the support set is equal to the edge connecting these vertices, see figure1.3(b). The support set is equal to a face when the support plane touches three or more vertices, see figure1.3(c).



(a) (b) (c)

Figure1.3 the support plane can touch, at the same time, one element → vertex (a), two elements → edge (b) or three or more elements → face (c). The support plane is defined with a normal vector u. See the arrows in (a), (b) and (c).

Representation

There are many different representations of polyhedra. We will introduce two representations.

The *List* representation

The ***List*** representation of a polyhedron is a list consisting of two lists. The first is the list of vertices. The second is the list of faces. A face, in its turn, is also represented as the list of the indices of vertices bounding this face. The sequence of these indices defines, using the right-hand-rule, a normal vector on this face (plane). This normal vector should point outward in the resulting polyhedron. See figure1.4(b).

Example1.1

A tetrahedron is a pyramid-like polyhedron, with 4 faces and 4 vertices and 6 edges. See figure1.4 and Table1.

We can represent the tetrahedron in the following notation:

tetrahedron = {{v1,v2,v3,v4},{{1,2,3},{1,4,2},{1,3,4},{2,4,3}}}

So we have a list of two lists:

$List_1$ = list of the vertices

$List_2$ = list of lists of the indices of the vertices representing the faces of the tetrahedron.

Here the sequence of these indices defines the normal vectors of the faces, as we mentioned before. See figure1.4-b.

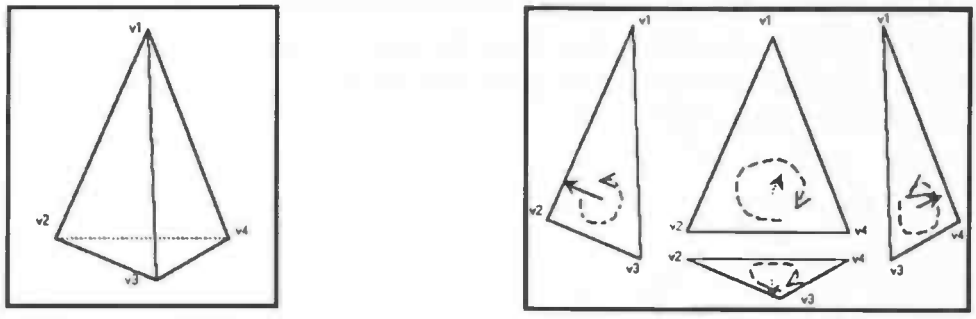


Figure 1.4. The list representation of a tetrahedron. (a) the tetrahedron with vertices $v_1,..,v_4$, (b) the faces are drawn separately. The round arrows give the order of the vertices in the face. This order determines the direction of the normal vector of the face. The normal vectors pointing backwards are given with dotted arrows.

The *Graph* representation

The ***Graph*** representation of a polyhedron is a plane-graph where the nodes, edges and faces of this graph correspondent respectively with the vertices, edges and faces of the polyhedron. The co-ordinates of the vertices are attributed to the corresponding nodes in the graph as attributes.

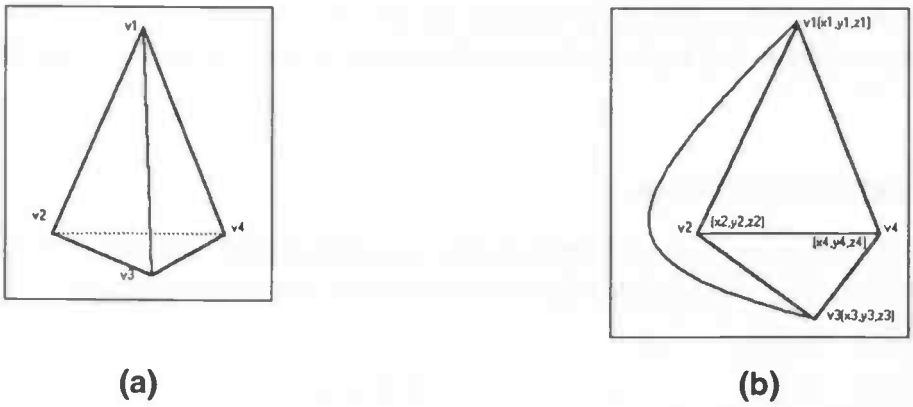


Figure 1.5: an example of a Graph-representation of the tetrahedron. (a) shows the tetrahedron in the R^3-space. (b) the graph-representation of the tetrahedron. In the graph faces, edges and vertices are represented in the graph by respectively faces, edges and nodes. The nodes have as attributes the coordinates of the corresponding vertices.

Classes of polyhedra

There are two extreme classes of polyhedra. One consists of polyhedra with only three vertices per face, and the other consists of polyhedra where every vertex is the intersection of three faces. See figure1.6.

(a) (b)

Figure1.6. Two examples of the classes of polyhedra. (a) is a class1 situation. (b) is a class2 situation. (a) Three planes (faces) intersect in one intersection point (vertex). The probability that a fourth face pass through the same intersection-point is very small. (b) a plane (face) can intersect three points (vertices). The probability that a fourth point lies in the same plane (face) is very small.

In terms of the graph representation, the first class is a graph with three-node cycles, only. The second class is a graph with nodes (vertices) having only three adjacent faces (and edges) for each node (vertex).

In terms of the list representation, the list of faces (second list of the list-representation) of the first class consists of face-lists with only three elements in each face-list. The list of faces of the second class has the property that for each vertex-index i of a vertex V_i in $list_1$ (the vertex-list), the number of lists element of $list_2$ (the list of faces), where i is a member, equals three. We are interested in the both classes, but also in a "mix" of these classes.

Transformations

The Polyhedra are *invariant* under translation and rotation. Polyhedron A and its translate over a line *l* (rotation around a line *l*) are equal in our point of view.

$$\mathbf{R}\,A \equiv A,$$

where **R** is a rotation matrix around line l.

By scaling, the volume of a polyhedron changes. A scaled polyhedron is not equal to its original.

Summarizing

Our polyhedra are always convex. We shall use the graph-representation in the implementation, and both the list- and graph-representation of other places. The polyhedra are member of the two classes (See figure 1.6) or a mix of them. That means that we will work with polyhedra with only three vertices per face, with polyhedra with only three adjacent faces for each vertex and with a mix.

I.2 The Slope Diagram Representation (SDR)

Introduction

As we will see later, we would like be able to rotate a polyhedron so that some of its faces and edges become parallel to faces and edges of another polyhedron. The **S**lope **D**iagram **R**epresentation (**SDR**) of a polyhedron makes easy to see whether this is the case. What the SDR is and what its properties are, is the main subject of this chapter

Definition

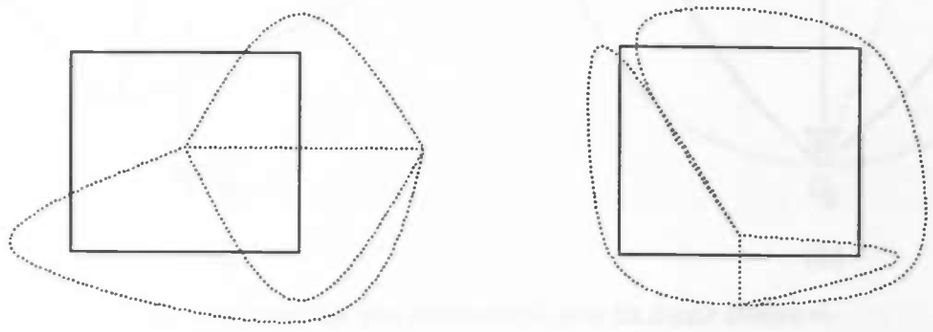
A polyhedron can be represented as a graph (See Ch1), let P be the graph representing a polyhedron. Then the *Slope Diagram* of a polyhedron P, SD(P) or SDP, is the dual graph of P embedded on the unit sphere S^2.

In the following paragraphs we introduce the dual-graph and we show how to embed it on the unit sphere.

The *dual graph* of P, dual(P) or DP, is the transformed polygonal graph of P (P is also polynomial, see previous chapter) where

- For each face *f* of P, DP has a node dual(*f*), and
- For each edge *e* adjacent to two faces f_i and f_j in P, DP has an edge connecting the nodes dual(f_i) and dual(f_j).

It can be seen that the regions between the edges in DP represent the vertices in P.



(a) (b)

Figure 2.1: The dual graph of a square. A square has two faces, the region closed by the four edges and the region outside. That means we get two Dual-nodes. These two nodes are connected by four edges, in the dual graph, representing the edges in the square. In (a) a square is given with solid lines and its dual-graph with dotted ones. In

(b) the same square is given but with an other dual-graph. These two dual-graphs are equal in our view point, if no attributes are taken into account.

The dual graph has the following propriety:

$$\text{Dual}(\text{Dual}(P)) = P,$$

where P is a graph.

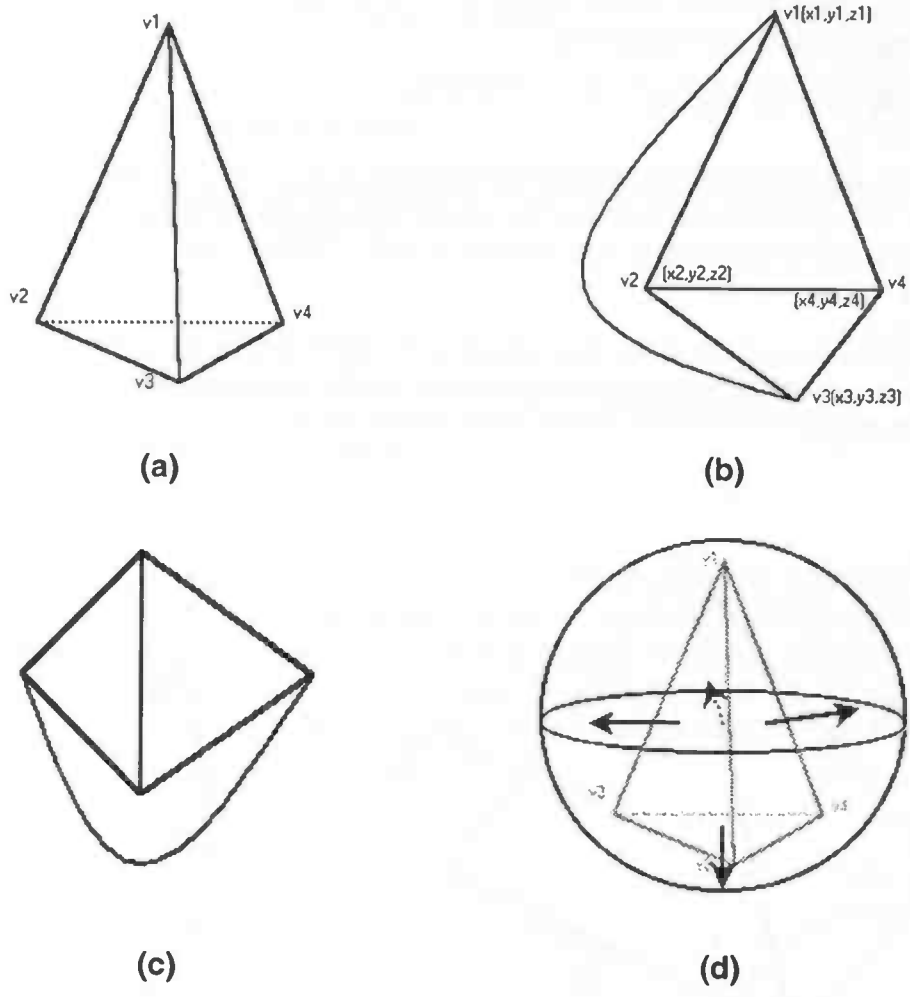


Figure 2.2: A tetrahedron (a) and its graph representation (b). (c) shows the dual graph, and (d) shows the unit sphere with the tetrahedron inside it. The normal vectors of the faces of the tetrahedron are represented by arrows. The ends of these arrows show where the four nodes of dual(tetrahedron) have to be imbedded on the sphere.

Embedding a graph on a sphere means drawing that graph on the surface of the sphere in such a way that two edges do not intersect.

By embedding DP on the unit sphere S^2, we add the outward unit normal *n* of every face *f*, in P, as an attribute to the corresponding node, Dual(*f*). The node is then embedded on the intersection point of *n* and the sphere. An edge *e* in DP is embedded as the small arc of the great circle connecting the two points corresponding to the two adjacent faces of *e*. A vertex of P, represented by

the closed region by edges in DP, is represented now as the region between the spherical arcs representing these edges.

The slope diagram of a polyhedron can also be defined in an purely geometrical way:

Definition:

Given a polyhedron P with n faces $F_1,\ldots,F_{np}$ then the **S**lope **D**iagram **R**epresentation of P (**SDP***) is:

SDR(P) = (V, A), where

$V = \{u_1,\ldots,u_{np}\}$ is the set of spherical points. $A \subseteq V \times V$ is the set of spherical arcs. The *spherical points* are the endpoints (on the unit sphere S^2) of the unit vectors u_i's orthogonal to the faces F_i's in P. These spherical points represent these faces.

Spherical arcs represents the edges of P. More precise, for every edge in P there is an arc connecting the spherical points representing the faces adjacent to this edge.

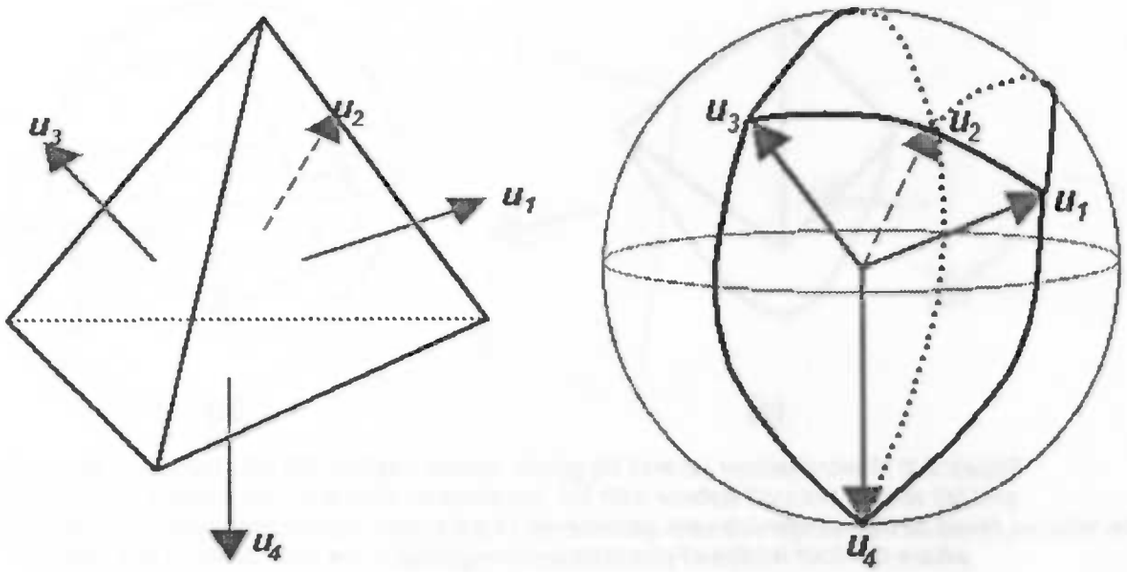


Figure 2.3: the tetrahedron and its Slope Diagram.

A vertex is represented by the interior of the *Spherical polygon* bounded by the arcs corresponding to the edges meeting at that vertex, in P.

---

* means SDR of polyhedron P. We will use later SDA and SDB for SDR of A and SDR of B respectively.

Properties of the slope diagram representation

Rotation

Given a rotation in the 3D space about a line l, say *R*. P is a polyhedron with SDP its slope diagram. Then the following is true:

$$SD(R.P) = R.SDP.$$

Here, the slope diagram of a rotated polyhedron can be found by rotating the slope diagram of that polyhedron itself.

Rotating a polyhedron with respect to another is important for us. We will see the reason in the following section.

Embedding two SDR's

The slope diagram representation gives more insight into the relative orientation of two polyhedra A and B.

Two parallel faces

Consider two polyhedra A and B and their slope diagrams SDA and SDB respectively. SDC is the slope diagram formed by embedding SDA and SDB on the same (unit-) sphere. If one face of A is parallel to a face of B then their (outward) normal vectors have to be parallel. This means that the spherical point representing the face in A coincides with the spherical point representing the face in B, in SDC (See figure2.4c).

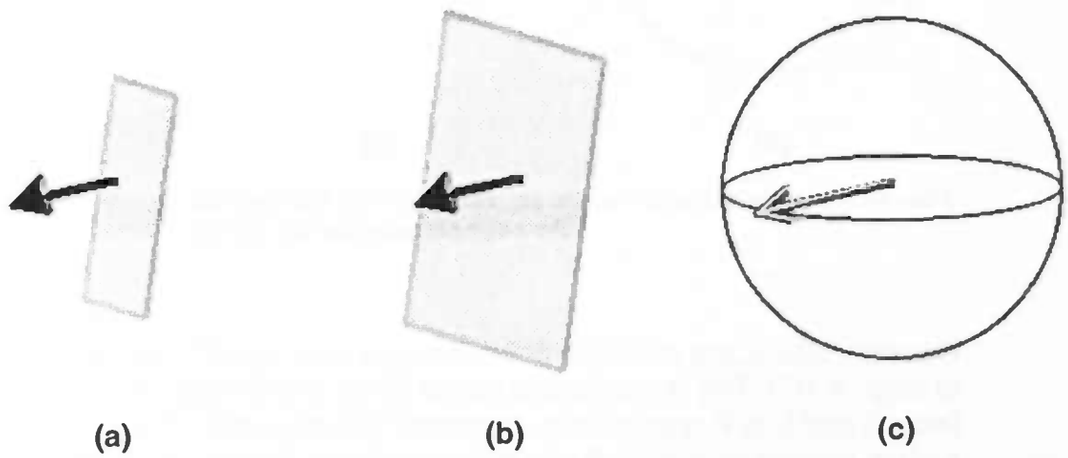


Figure2.4: The face in (a) is parallel with the face in (b) in (c) we can see that their spherical points are coinciding. One outward normal vector is drawn is gray and the other in black.

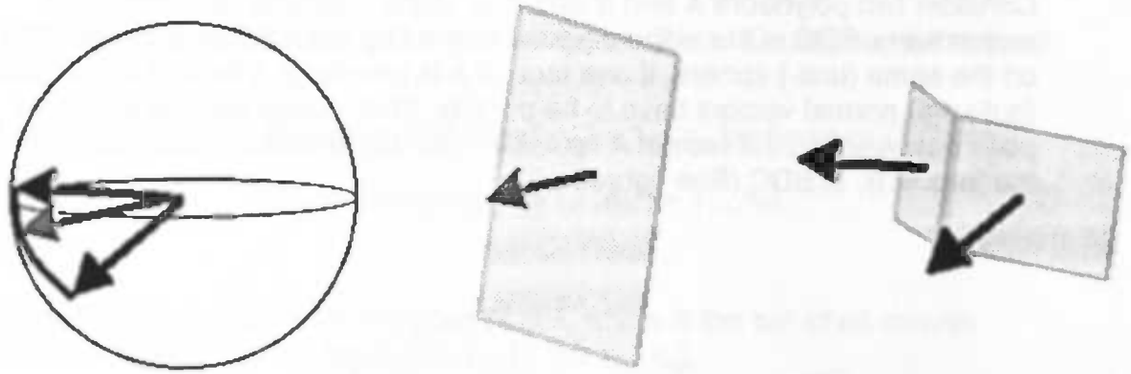
Consider f_i a face in A and f_j a face in B (See figure 2.4). SD(f_i) and SD(f_j) are then their spherical points in SDA and SDB respectively. If $f_i // f_j$ then $n_i // n_j$, where n_i and n_j represent the outward normal vectors of f_i and f_j respectively. Following the geometrical definition of the slope diagram representation the unit vectors u_i and u_j collinear to n_i and n_j respectively, have the same endpoint on the unit sphere in SDC. This means that the spherical points representing the two faces in A and B coincide.

If two spherical points coincide then their unit vectors are collinear and n_i and n_j are parallel. That means that the faces f_i and f_j with n_i and n_j as normal vectors are parallel too.

Conclusion: two spherical points in SDC coincide if and only if the corresponding faces in A and B are parallel, and vice versa.

A face and an edge are parallel

Consider now an edge *e* in A and a face *f* in B. *e* is parallel to *f* if point SDB(*f*) lies on the arc SDA(*e*) (See figure2.5). Here the three outward normal vectors (See figure2.5(c)) are coplanar.



(a) (b) (c)

Figure2.5: if a point lies on an arc (a), then the face represented by that point (b) is parallel with the edge represented by arc (c).

Consider SD(f_i), the spherical point of face f_i in A, and SD(e), the spherical arc of edge *e* in B. The two spherical points SD(f_j) and SD(f_k), representing the faces f_j and f_k in B respectively, represent this arc. Let n_i, n_j and n_k be the normal vectors of faces f_i, fj and fk, respectively. If point SD(f_i) lies on arc SD(e), then n_i becomes coplanar with n_j and n_k. In other words, n_i is now perpendicular to the intersection line of faces f_j and f_k. It is clear that e is a line

segment of that intersection line. So n_i is perpendicular to e. That is, face f_i is parallel with edge e.

We may now conclude that if a spherical point lies on a spherical arc, then the face represented by that spherical points is parallel with the edge represented by that arc (See figure2.5).

I.3 The minkowski addition

The minkowski addition of two sets A, B in R^n is defined by

$$A \oplus B = \{a + b \mid a \in A, b \in B\}. \quad (1)$$

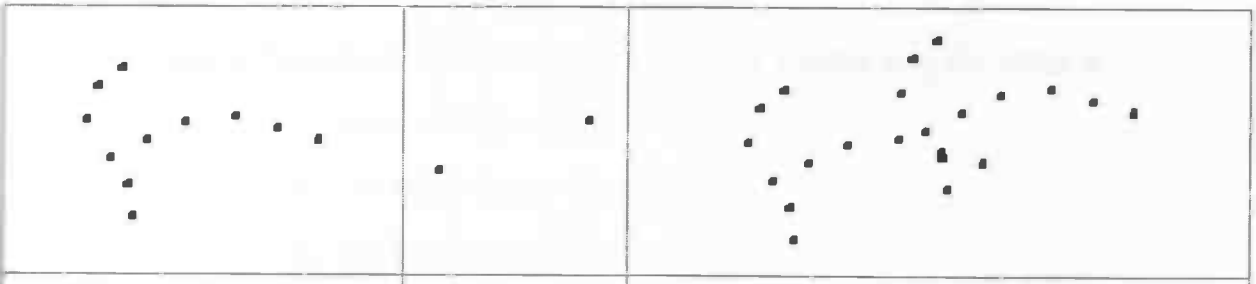


Figure3.1: Two sets in the 2D space (a, b). And their minkowski sum (c).

The minkowski addition of two convex polygons P and Q can then be defined as follows:

Let p_i, $i = 1,..,n_p$ be the vertices of P and q_j, $j = 1,..,n_q$ be those of Q. Then

$$P \oplus Q = \text{conv}\{p_i+q_j \mid i = 1,..,n_p, j = 1,..,n_q\}. \quad (2)$$

Here the conv{ . } denotes the convex hull, which is the set of all convex combinations of the set $\{p_i+q_j \mid i = 1,..,n_p, j = 1,..,n_q\}$.

This is also valid for two convex polyhedra.

To make it clearer for the reader, the minkowski addition of two convex polygons, A and B, is described with some examples. After that we generalize to the 3D-space and give some examples about the minkowski addition of two polyhedra.

Examples

Consider two convex sets in 2D-space: S_1 and S_2. S_1 can be a vertex, an edge or a polygon. The same for S_2. Then the minkowski sum of S_1 and S_2 using (2), can be calculated as follows:

- S_1 is a vertex and contains one point, say v_1. S_2 is a vertex too, with point u_1.See figure 3.2(a). The minkowski sum following (2), is the convex hull of the sum of all vertices (in fact, position vectors of these vertices) of S_1 and all vertices of S_2. Because both sets contain only one point for each, v_1 and u_1,

  $$S_1 \oplus S_2 = \text{conv}\{v_1+u_1\} = \{v_1+u_1\}.$$

  That is point v_1 translated/shifted over u_1, or vice versa.

- S_1 is a vertex (v_1) and S_2 is an edge represented by the two vertices u_1 and u_2. See figure 3.2(b). The minkowski sum following (2) is the convex hull of the sum of all vertices of S_1 and all vertices of S_2. Then

  $$S_1 \oplus S_2 = \text{conv}\{v_1+u_1,\ v_1+u_2\} = \text{edge}\{v_1+u_1,\ v_1+u_2\},$$

  because the convex hull of two vertices that do not coincide is always an edge. This edge can be interpreted as the edge $\{u_1, u_2\}$ translated over (the position vector of) v_1. Or as the edge represented by the two (vertex-) copies of v_1, the one translated over u_1 and the other over u_2.

- S_1 is a vertex (v_1) and S_2 is a triangle-polygon represented by the vertices $\{u_1, u_2, u_3\}$, then the minkowski sum of these two sets is

  $$S_1 \oplus S_2 = \text{conv}\{v_1+u_1,\ v_1+u_2,\ v_1+u_3\} = \text{triangle}\{v_1+u_1,\ v_1+u_2,\ v_1+u_3\},$$

  is the shifted copy of the triangle-polygon represented by the vertices $\{u_1, u_2, u_3\}$, over v_1, or is the triangle represented by three shifted copies of v_1 over the position vectors of the vertices u_1, u_2 and u_3, respectively.

In general, if Set S_1 is represented by the vertices $\{v_1,..,v_{n1}\}$ and S_2 by the vertices $\{u_1,..,u_{n2}\}$, with n_1, n_2 the number of vertices representing S_1, respectively S_2, then the minkowski sum of these two sets can be considered in two ways:

1. n_2 copies of the vertices of S_1 are made and shifted over the position vectors of $u_1,\ldots,u_{n2}$ respectively. Then the convex hull of these n_2 copies of the vertices of S_1 is taken.

2. n_1 copies of the vertices of S_2 are made and shifted over the position vectors of $v_1,\ldots,v_{n1}$ respectively. Then the convex hull of these n_1 copies of the vertices of S_2 is taken.

This is because of the symmetric property of the minkowski addition.

Figure3.2 shows all the possible combinations of the minkowski addition between vertices, edges and faces (polygons are also faces in the 2D space). In the first two columns, S_1 and S_2 are drawn. In the third column, the vertex-wise sum of the vertices of the two sets is shown. The convex hull of this sum results in the minkowski sum of S_1 and S_2. See the fourth column.

Remark

One can conclude from figure 3.2 that the minkowski sum is a face if and only if one of the following cases occurs:

- One of the sets is a face (a polygon is also a face) and the other is
    - A vertex figure 3.2(c)
    - An edge figure 3.2(e)
    - A face figure 3.2(g)
- Both sets represent an edge, and the edges are not collinear. See figure 3.2(f).

This remark is important, as we will see later.

Example

Consider two polyhedra, a cube figure 3.3(a) and a tetrahedron figure 3.3(b). The minkowski sum of them is then the convex hull of the four copies (number of the vertices of a tetrahedron) of the (set representing the) cube located at the four vertex-positions of the tetrahedron. See figure 3.3(c).

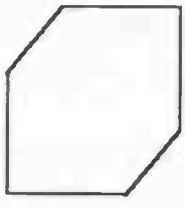
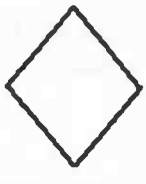
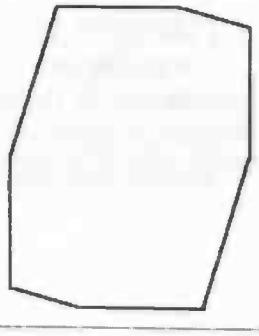
	S_1	S_2	{u+v | u vertex of S_1, v vertex of S_2}	$S_1 \oplus S_2$ =conv{vertex-wise sum}
(a)				
(b)				
(c)				
(d)				
(e)				
(f)				
(g)				

Figure3.2: Vertices, edges and faces are combined in all possible ways to get some experience with the minkowski addition in the 2D space.

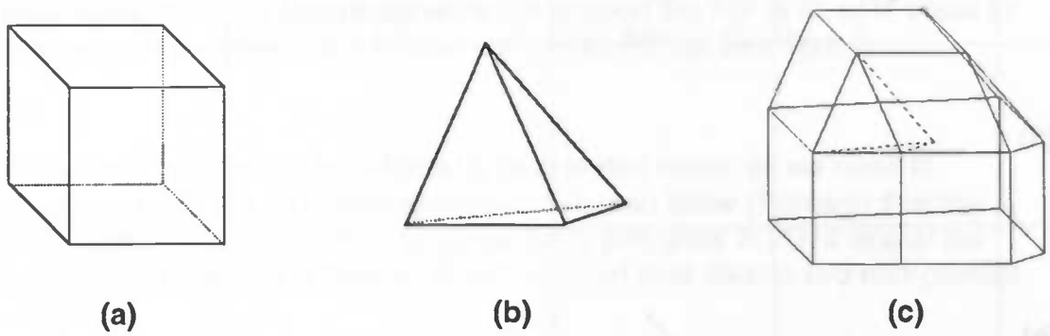


Figure3.3 a cube (a), a tetrahedron (b) and their minkowski sum (c).

We can summarize the method described by (2) as follows:

All vertices of S_1 are combined with all vertices of S_2. By computing the convex hull the vertex-, edge- and face structure is calculated. This is an expensive (time consuming) method, and surely when we use it in the 3D space.

For the interested reader, in the following part we introduce an efficient method to calculate the minkowski sum of two convex polyhedra. For our research, this is not really needed.

An efficient method

At the end of this chapter, we will return to the theme *efficiency*. But let us first introduce the following method.

Till now we used the vertices only to calculate the minkowski sum of two polyhedra. Instead of working with the vertices only we can use the edges (for the polygons in the 2D space) of the faces (in the 3D space). We know that a polygon is defined by the location of its oriented edges. It is sufficient for the computation of the minkowski sum of two polygons, say $P_1 \oplus P_2$, to find its edges. see figure.3.4.

Example

Consider the two polygons and their minkowski sum in figure 3.2(g). One can recognise the edges of the minkowski sum in the two polygons. See figure 3.4(f).

Similarly to polygons, a polyhedron is defined by its oriented faces. It is sufficient for the computation of the minkowski sum of two polyhedra to find only the faces of it. The support set theorem in first chapter will clear the job. The support set has a very important property:

Let P and Q be two convex polyhedra in R^3 then for every $u \in S^2$,

$$F(P \oplus Q, u) = F(P,u) \oplus F(Q,u).$$

In other words, every vertex, edge or face of P⊕Q, defined by the support set at the direction u, can be written as the minkowski addition of the elements (vertices, edges or faces) defined by the support sets in the same direction u, in both polyhedra.

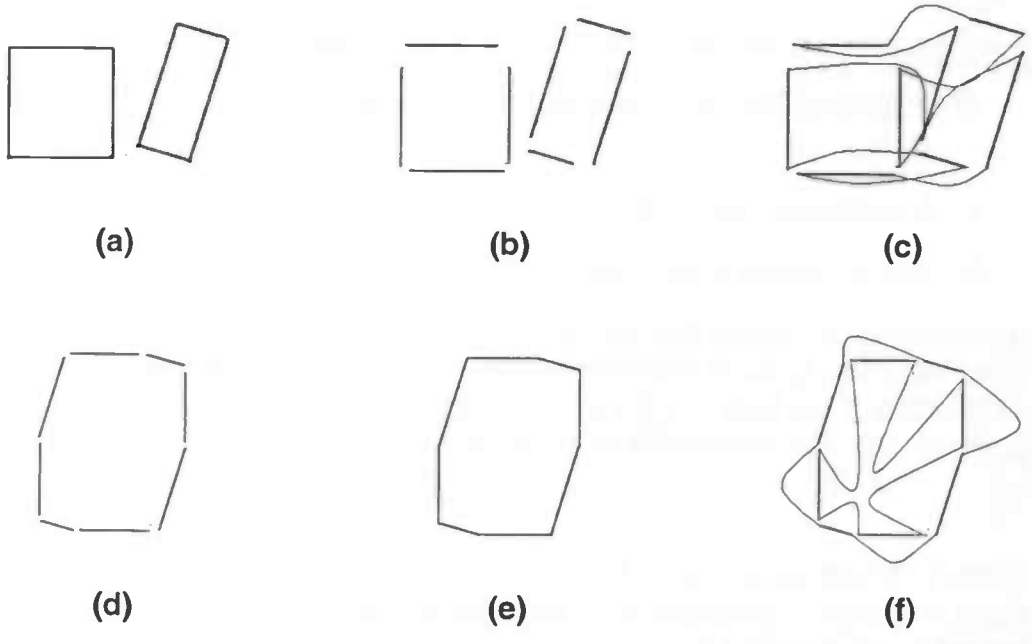


Figure3.4: The steps to make the minkowski sum of a square and a rectangle (a). The two polygons are considered first as connected oriented edges. These edges are disconnected (b). By light-shaded arcs, the edges are connected again (c). The edges are now displaced to get a situation like in (b). The connected edges represent the minkowski sum of the two polygons(e). In (f) the light-shaded arcs shows where they originally come from if they are back-connected.

Example3.3

If the support set F(P,u) is a vertex in P and that of Q, F(Q,u), is one of the faces of Q, then the face defined by the support set F(P $\oplus$ Q, u) is equal to the minkowski sum of face F(Q,u) and vertex F(P,u). See fig.3.3.

Because a polyhedron is defined by its oriented faces, all we need to construct P$\oplus$Q are its (oriented-) faces. We also know (Remark) that the minkowski addition results in faces (actually polygons in 2D) if one of the support sets denotes a face or, if both support sets denote two non-parallel edges.

The slope diagram representation, introduced in the previous chapter, has the following important property:

$$SD(P \oplus Q) = Overlay(SD(P),SD(Q)).$$

In other words, the slope diagram of the minkowski sum P$\oplus$Q, denoted by SDP$\oplus$Q, is equal to the slope diagram constructed by the embedding of the slope diagrams of P and Q, SDP and SDQ, on the same sphere.(see previous chapter about SDR).

By embedding the two slope diagrams on the same sphere, new spherical polygons, arcs and points are formed by those of SDP and SDQ. We are interested in all the spherical points, because they denote the faces of P$\oplus$Q.

A spherical point can be formed in one of the following three cases:

1. A spherical arc of polyhedron P intersects a spherical arc of Q. Or vice versa.
2. A spherical point of one polyhedron lies on a spherical arc of the other.
3. Two spherical points coincide.

Here we like to remark that we are not taking into account the intersection of a spherical point (a face) with a spherical polygon (a vertex), because the calculation of the face is then trivial. That is translating that face over the position vector of that vertex.(see figure 3.2(c)).

Case 1: Spherical arc (u,u') intersects (v,v') at point w $\in S^2$. See figure 3.5(c). Point w is then a spherical point representing a face in P$\oplus$Q. The direction of the edge represented by an arc can be found by using the points defining this arc (normal vectors of two faces). If we do this for all the edges bounding the face represented by point w, the new face can be constructed.

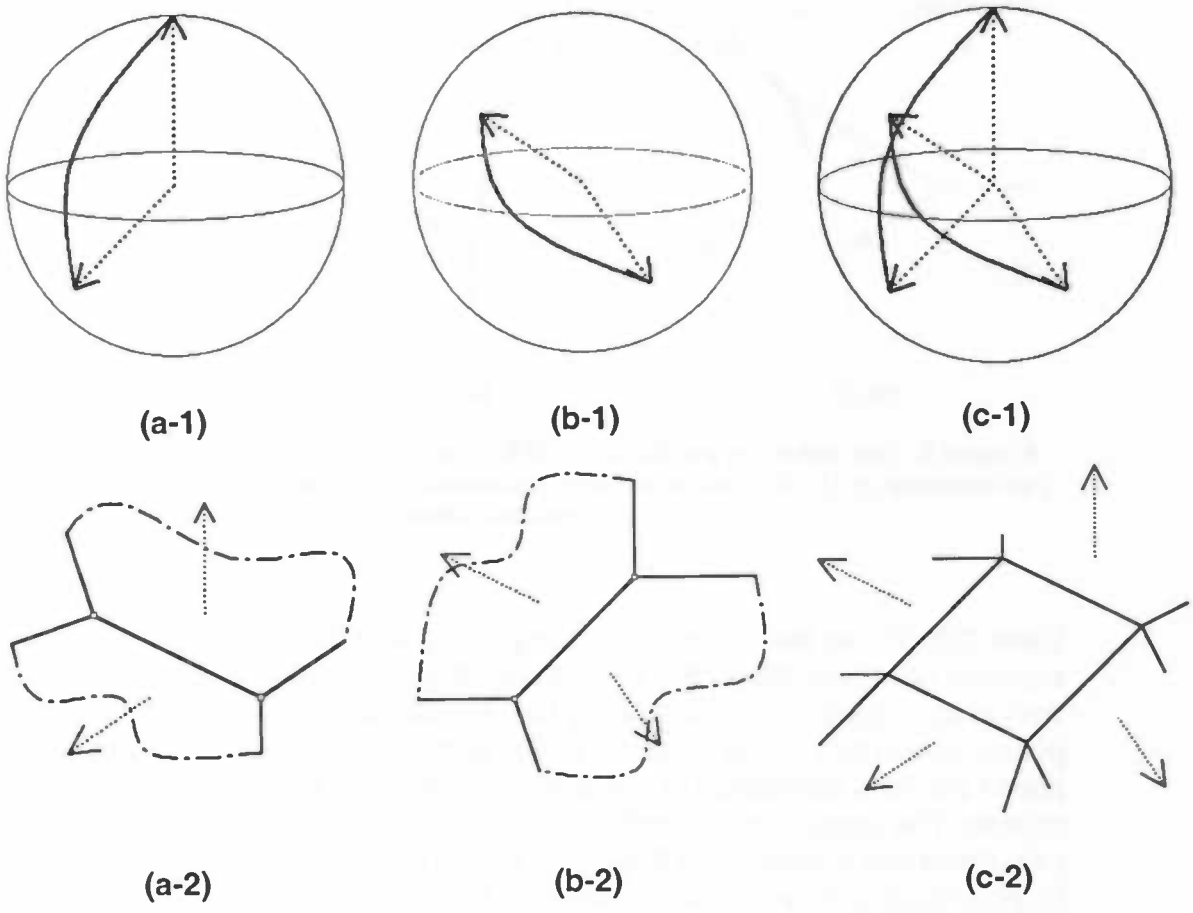
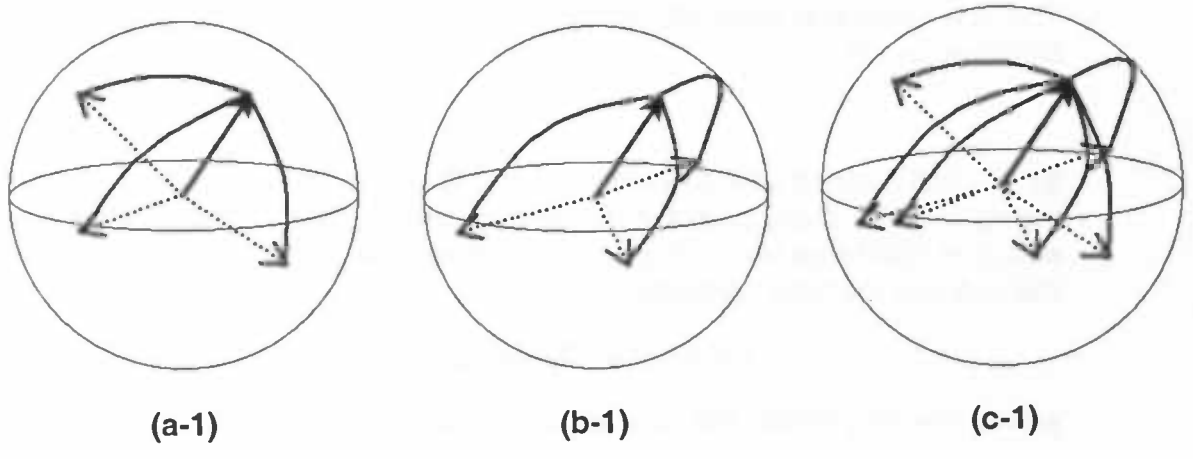
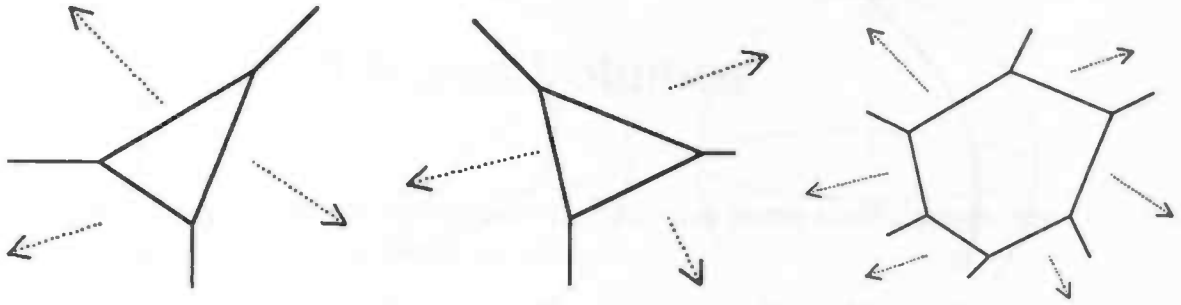


Figure3.5: Two acrs (a-1 and b-1), denoting two edges (a-2 and b-2) can intersect in a spherical point (c-1). This spherical point represents the face shown in (c-2).





(a-2) (b-2) (c-2)

Figure3.6: Two spherical points (a-1 and b-1), representing the faces shown in (a-2 and b-2), can coincide (c-1). This spherical point represents a face with formed by the edges of the two previous faces (c-2).

Case 2,3: These two cases can be explained together, as case2 is a special instance of case3. When the two points, $SD(f_i)$ and $SD(f_j)$ coincide, then $SD(f_i)$ lies on the edges connecting $SD(f_j)$ too. So we now discuss two coinciding points. When two points coincide all the arcs which are connected to those points are now connected to one point on the same location on the unit sphere. The lengths and directions of the edges defined by those arcs are calculated like in case1. By finding all edges bounding the face represented by point w, this face can be constructed too.

Efficiency

This last method is more efficient than the one we used at the beginning of this chapter (2).

By the first method all the points of P and Q had to be combined. By applying the convex hull many points were deleted and the face-, edge- and vertex structure had to be reconstructed. All of these are in the 3D-space. The first method has the time complexity [3]

$$O(nv_P nv_Q)+O(nv_P nv_Q \log(nv_P nv_Q)),$$

where nv_P, nv_Q denote the number of vertices of P, respectively Q.

The first method belongs to the class of [3] MSR (Minkowski **S**um calculated in the $\mathbf{R}^3$-space). The second method belongs to the [3] MSD (Minkowski **S**um in the Slope **D**iagram space) class. Methods belonging to this class can calculate the minkowski sum with a linear time complexity in the number of

edges or faces of the polyhedra [3]. Dr. H. Bekker introduced a new MSR-method to calculate the minkowski sum of two polyhedra [3].

I.4 Mixed Volumes.

In this chapter we introduce the mixed volumes, give some useful properties and theorems we need to achieve our goal.

The volume of a polyhedron P, equal to the minkowski sum of two other polyhedra, A and B, can be divided into four different parts:

1. A part related to the (volume) of A.
2. A part related to the (volume) of B.
3. Two parts related to both polyhedra, A and B.

Each of these two parts is called a *Mixed Volume*.

In figure4.1 we can see which part could be related to A and which to B. The volume of the rest part is equal to the mixed volumes.

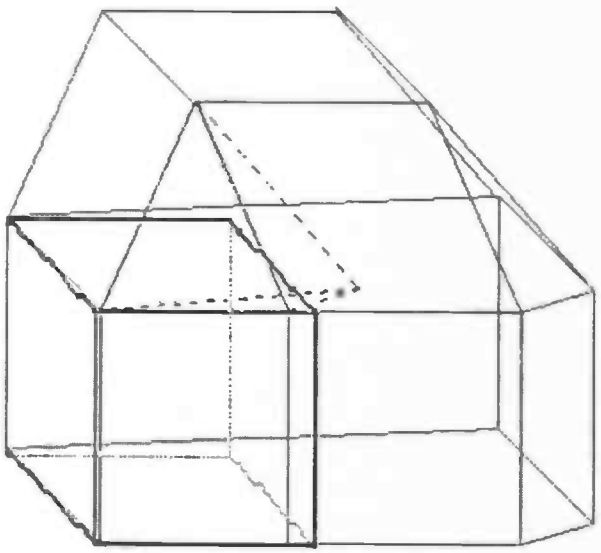


Figure 4.1: In the minkowski sum of the cube and the tetrahedron (from the previous chapter), we can recognize the volumes of the cube and the tetrahedron.

Clearly, it is difficult to indicate in a drawing of a minkowski sum, where these parts (mixed volumes) are. The reader does not have to consider the mixed volumes as a geometrical part of the minkowski sum, but needs only to understand that the sum of the mixed volumes plus the volumes of the original two polyhedra returns the volume of the minkowski sum.

The informal definition of the Mixed Volumes is repeated in a formal way in the following definition.

Definition

The volume of the Minkowski sum $P = \lambda A \oplus \mu B$ of the polyhedra A and B, where $\lambda, \mu \geq 0$, is equal to:

$$V(P) = V(\lambda A \oplus \mu B) = \lambda^3 V(A) + 3\lambda^2 \mu V(A,A,B) + 3\lambda \mu^2 V(A,B,B) + \mu^3 V(B),$$

where the terms V(, ,) (with three arguments) are invariant under permutations of their arguments. These terms are called the *Mixed Volumes* of the polyhedra A and B.

Definition: Let A be two convex polyhedron, then the mixed volume V(A,A,A) is defined as the volume of polyhedron A.

$$V(A,A,A) = V(A)$$

Further, the mixed volume can be calculated as follows:

If P is a convex polyhedron with faces F_i and corresponding outward unit normal vectors u_i, i=1,...,k, then following [1]

$$V(A,P,P) = 1/3 \sum^{k}_{i=1} h(A,u_i) S(F_i),$$

where $S(F_i)$ is the area of the face F_i of P and $h(A,u_i)$ is the value of the support function of A for the normal vector u_i.

Brunn-Minkowski inequality

For two arbitrary convex polyhedra A,B the following inequality holds:

$$V(A \oplus B)^{1/3} \geq V(A)^{1/3} + V(B)^{1/3},$$

with equality if and only if A and B are convex and homothetic modulo translation, i.e., $B \equiv \lambda A$ for some $\lambda > 0$.

The following (Brunn-Minkowski-like) inequality holds [1]

$$V(A \oplus B) \geq 8V(A)^{1/2}\, V(B)^{1/2},$$

with equality if and only if $A \equiv B$ and both polyhedra are convex.

I.5 The Critical Orientations

The volume of the minkowski sum of two polyhedra A and B depends on the orientation of B with respect to A (or vice versa) (See ChapterI.3). In other words:

> The volume of the minkowski sum of A and B, denoted by V(A⊕B), can be defined as a function of the orientation of B w.r.t. A.

The orientation of a polyhedron can be given as a function of three variables: α, ϕ, θ. We can then conclude that the volume of the minkowski sum of A and B is a function of these three angles.

$$V_{A\oplus B}(\alpha, \phi, \theta) \text{ is a } :R^3 \rightarrow R^1 \text{ function.}$$

While rotating B, by changing the values of one or more of the three angles, situations arise where V(A⊕B) reaches its (local) minima. V(A⊕B) is discontinuous in this points. The orientations corresponding to these points of the curve are called *critical orientations*.

Definition

Critical Orientations are relative configurations of B w.r.t. A where the value of V(A⊕B) reach its (local) minima.

In terms of the SDR, these orientations are characterized by the fact that the spherical points of SDB intersect spherical arcs and points of SDA.

Remark

Critical orientations are orientations where spherical points of the slope diagram representation (SDR) of the one polyhedron intersect **as much as possible** spherical arcs and points of the SDR of the other polyhedron.

In the literature [1] a method is given to find these orientations. Here the critical orientations are defined as the set of all orientations where:

- A face of B is parallel to a face of A and,

- An edge of A is parallel to another face of B.

In terms of the SDR, the set of critical orientations, according to this method, can be defined as follows:

Definition

The set of critical orientations is the set of all orientations where a spherical point of SDB coincides with a spherical point of SDA and an other spherical point of SDB lies on a spherical arc of SDA.

Mr Tozikov [1] conjectures that this set of orientations includes all critical orientations as denoted in the Remark. In other words, the orientation of B w.r.t. A where V(A⊕B) is minimal (the SMO-orientation. See chapter), is always included in this set of orientations. Mr Tozikov really does not give a proof of his conjecture.

We had the feeling that not all the critical orientation, according to Remark, including the SMO-orientation, are included in the Tozikov-set of critical orientations.

We do not rule out that SMO is sometimes included in Tozikov-set, but we assert that there exists another set of critical orientations which always includes the SMO-orientation, let us call this set $class_2$ and that of Tozikov $class_1$. The $Class_2$-orientations is defined as the set of all orientations of B w.r.t. A where three different faces of B are parallel to three different edges of A. In terms of the SDR the definition of $class_2$ is:

Definition

$Class_2$ is the class of orientations, of B w.r.t. A, where three different spherical points of SDB lie on three different spherical arcs of SDA.

In the figure 5.1 the properties of these two classes are given in the slope diagram representation of two polyhedra:

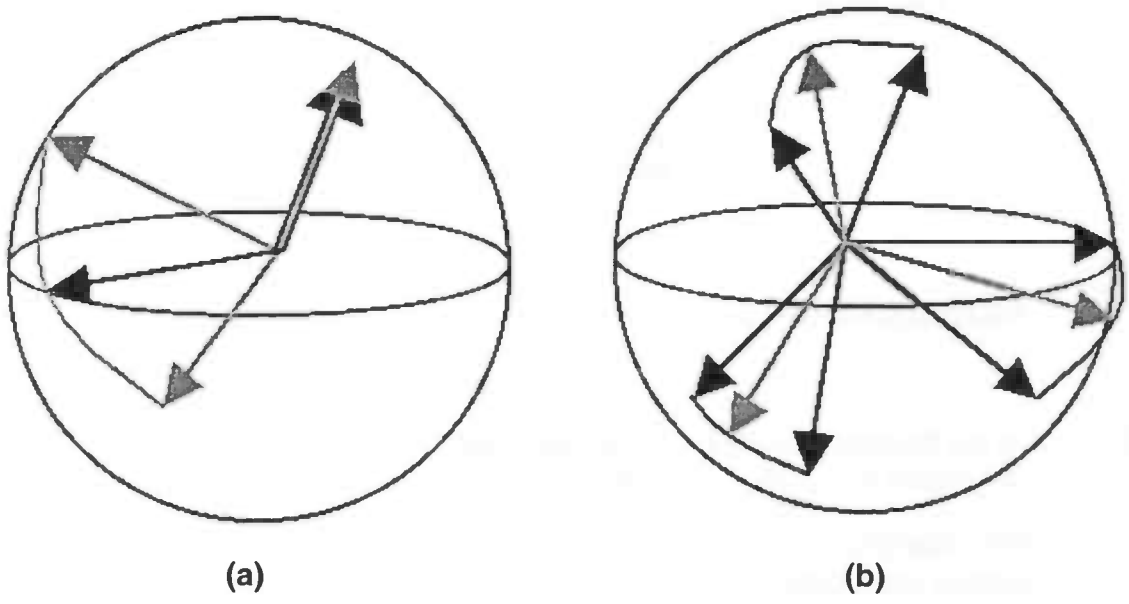


Figure 5.1 two classes of orientations are given here. (a) class$_1$: two spherical points, one of SDA and the other of SDB coincide, and a spherical point of SDB lies on a spherical arc of SDA. (b) class$_2$: three different spherical points of SDB lie on three different spherical arcs of SDA. We used here solid-black and grey to give mark the spherical points and arcs of each SDR.

I.6 Similarity measure

Introduction

In the literature some methods are described to see whether two (convex-) polyhedra are similar or correlated.

For example, two polyhedra are said to be similar when all corresponding angles are equal. A requirement here is that the two polyhedra have to have the same number of faces, edges and vertices. Clearly two polyhedra can be almost similar even if they do not have the same number of faces or edges and vertices.

Another example approach is to calculate the correlation of two polyhedra by matching them, as much as possible, in each other.

Nothing was told about the measurement of the similarity. Roughly speaking, there are no measurement rules to tell us, exactly, how similar two polyhedra are. The main goal of this chapter is to introduce a new method based on a pure mathematical function. The subjects of the previous chapters are the elements to 'build' this function.

This chapter has the following structure: first we give the definition of the similarity measure function by introducing its properties. In the second section two (instance-) functions are introduced. At the end of the chapter a summary of this chapter is given.

DEFINITION

Let R be the set of all rotations in R^3 and Υ the set of convex polyhedra. A function $\sigma: \Upsilon \times \Upsilon \rightarrow [0,1]$ is called a ***similarity measure*** on Υ if

1. $\sigma(A,B) = \sigma(B,A)$;
2. $\sigma(A,B) = \sigma(A',B')$ if $A \equiv A'$ and $B \equiv B'$;
3. $\sigma(A,B) = \sigma(r(A),B)$, $r \in R$;
4. $\sigma(A,B) = 1 \Leftrightarrow B \equiv r(A)$ for some $r \in R$;
5. σ is continuous in both arguments (with respect to the Hausdorff metric).

It is also required that Υ is invariant under R, that is, $r(A) \in \Upsilon$ if $A \in \Upsilon$ and $r \in R$. This remark was also made in the first chapter.

EXAMPLES OF SIMILARITY MEASURE FUNCTIONS

In chapter four (mixed volumes) the following inequalities were introduced:

$$V(A \oplus B) \geq 8V(A)^{1/2}V(B)^{1/2}, \tag{1}$$

With equality if and only if $A \equiv B$ and both polyhedra are convex.

And,

$$V(A,A,B) \geq V(A)^{2/3}V(B)^{1/3}, \tag{2}$$

Again with equality if and only if $A \equiv B$ and both polyhedra are convex.

Consider (1), the volume of polyhedron $A \oplus B$, $V(A \oplus B)$ is equal to $8V(A)^{1/2}V(B)^{1/2}$ if and only if $A \equiv B$. So if

$$\frac{8V(A)^{1/2}\,V(B)^{1/2}}{V(A \oplus B)} = 1, \tag{3}$$

then $A \equiv B$.

It is known (Chapter I.3) that even if $A \equiv B$, $A \oplus B$ can get another shape (and volume) when A and B get different orientations. See figure 5.1. This means that $V(A \oplus B)$ in (3) can increase by rotating B (or A) (with zero as minimum because in ChapterI.1 we required that the volume of a polyhedron is always ≥ 0 is). This is shown in the following example.

Example 6.1

A is a cube with edge length equal to one. B is an exact copy of A. The volume of V(A) = V(B) = 1x1x1 = 1. The minkowski sum of A and B with A and B having the same orientation is again a cube (See figure 5.1(a) and Ch. 3) with edge length equal to two. The volume of $A \oplus B$, $V(A \oplus B)$ = 2x2x2 = 8.

If we rotate B about the z-axis with an angle equal to $^1/_4\pi$, the shape of $A \oplus B$ becomes different and $V(A \oplus B)$ gets bigger. With some calculations we get

$$V(A \oplus B) = 4 + 4\ \mathrm{Sqrt}(2)$$

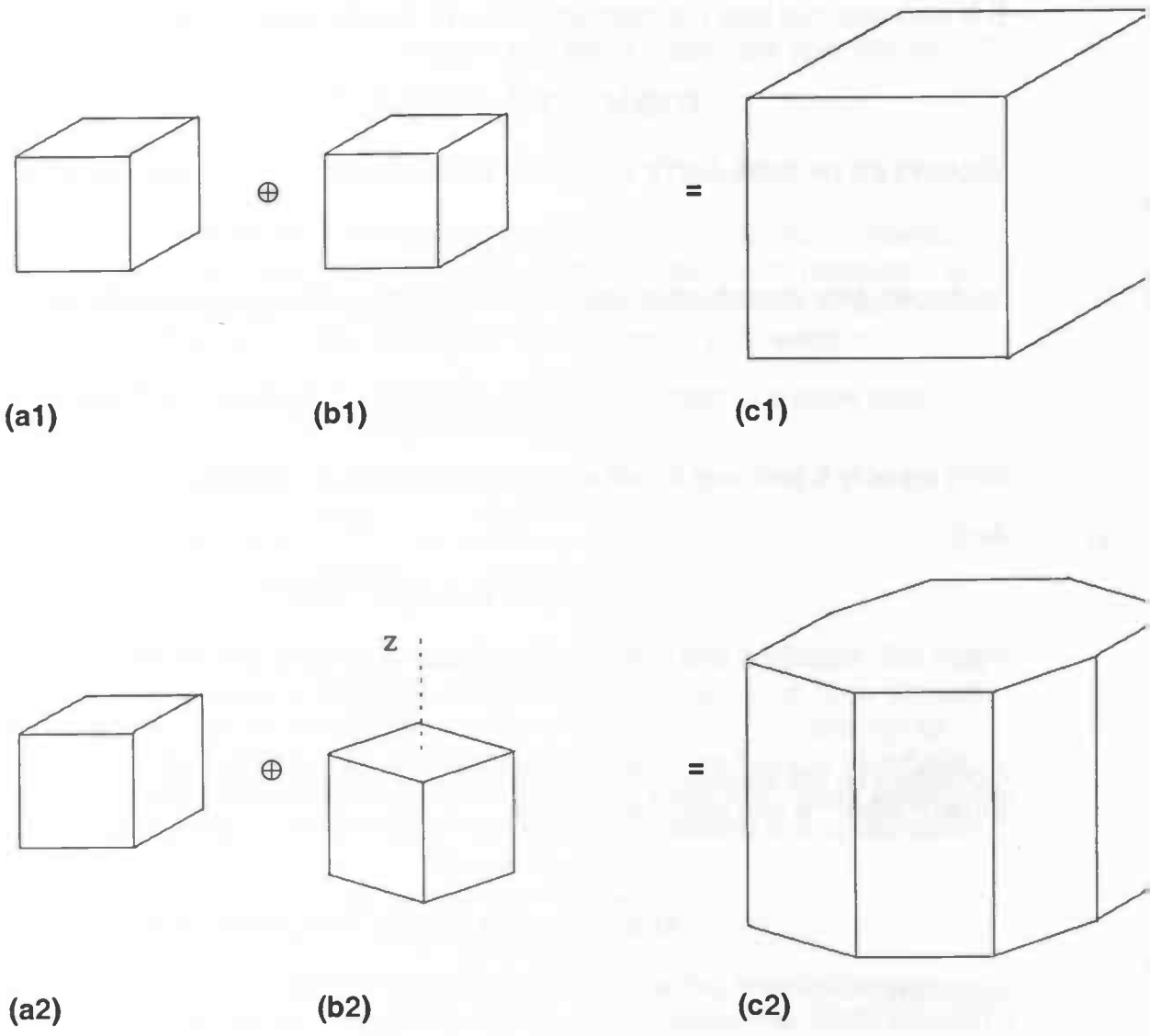


Figure5.1: (a1) and (b1) show two identical cubes. The minkowski sum of them is shown in (c1). (a is again the same cube like in figure (a1) and (b2) is a rotated copy of (b1) about the z-axis with an angle ½π. (c2) shows the minkowski sum of (a2) and (b2).

Definition

$r \in R$ is called *the similarity measure orientation* (SMO) of B when (3) gets its maximal value after rotating B by *r*.

Definition

the similarity measure function of two polyhedra based on the minkowski sum is then defined by:

$$\sigma_1(A,B) = \underset{r \in R}{\text{Sup}} \frac{8V(A)^{1/2}\, V(B)^{1/2}}{V(A \oplus r(B))} \tag{4}$$

where r is the similarity measure orientation (SMO) of B.

Similar to (1), the following similarity measure function can be derived from (2):

Definition

the similarity measure function based on the mixed volume is then defined by

$$\sigma_2(A,B) = \underset{r \in R}{\text{Sup}} \frac{V(A)^{2/3}\, V(B)^{1/3}}{V(A,A,r(B))} \tag{5}$$

where r is the similarity measure orientation of B.

As we will see later (5) is easier to implement than (4). That is the reason that in sequel of this thesis, only (5) is used. σ_2 is then denoted by σ.

I.7 The Problem

Introduction

The similarity measure of two convex polyhedra based on the minkowski addition and the mixed volume is a very important approach because it is the only **pure mathematical approach** to calculate the similarity of two convex polyhedra (this can be later expanded to non-convex polyhedra too).

Consider the formula of the similarity measure function one more time:

$$\sigma(A,B) = \underset{(r \in R)}{\text{Sup}} \frac{V(A)^{2/3}\, V(B)^{1/3}}{V(A,A,r(B))} \qquad (1)$$

Here we are looking for the best orientation, SMO-orientation [See similarity measure chapter for the definition], where the quotient in (1) reaches its maximum value (V(A,A,*r*(B)) minimum-value). Because there is an infinite number of orientations and it is impracticable to check them all till we find the SMO-orientations, a method with finite steps was needed to find the SMO-orientation.

Class$_1$ and Class$_2$ of Critical Orientations

In the literature [1] a method was given to calculate the orientations where V(A,A,*r*(B)) reaches its local minima: The critical orientations. SMO should be one of these orientations. This class of orientations is called *class$_1$*-orientations. See figure 7.1(a).

We assume that the SMO-orientation is not always an element of *class$_1$*-orientations but there is another class of critical orientations where SMO is always included in. The *class$_2$*-orientations. See figure 7.2(b).

The Questions

The Questions are now:

1. Does SMO always Belong to the *class$_1$*-orientations ? (See figure7.1)
2. Or, does SMO always belong to the *class$_2$*-orientations ? (See figure7.2)
3. Or, does SMO Belong to both classes of orientations? (See figure7.3)

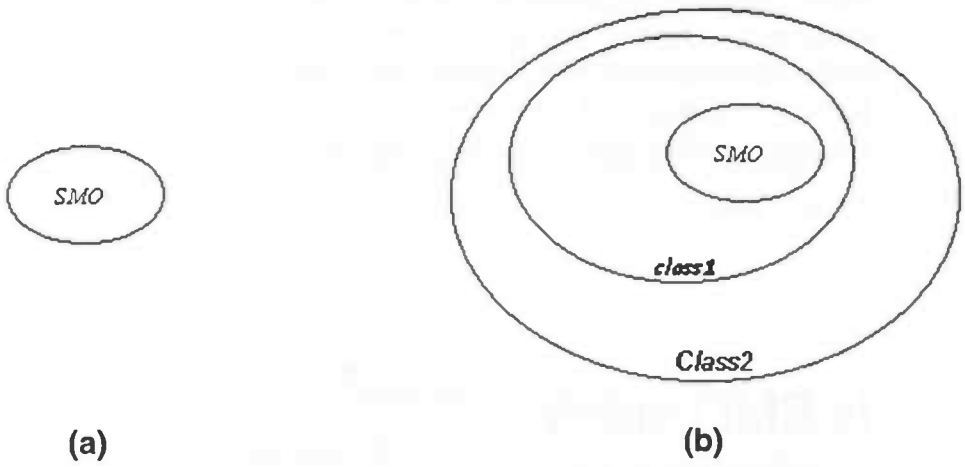


Figure7.1: Let (a) be the set of SMO's then Tozikov conjectures (b) that $class_1$ always includes the SMO-orientation. $Class_1$ is always a sub-set of $Class_2$

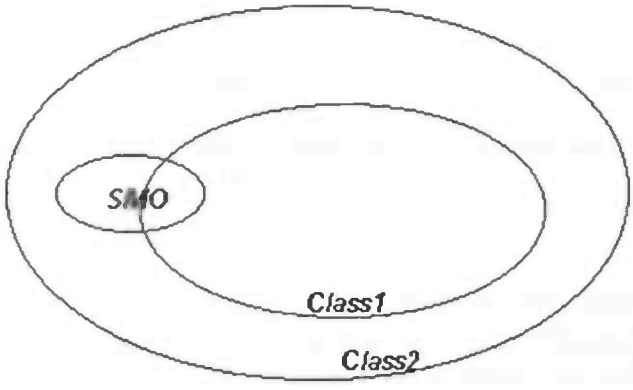


Figure7.2: We think that it is possible that the SMO-orientation is always an element of $class_2$ but we do not ignore that SMO can be an element of $class_1$

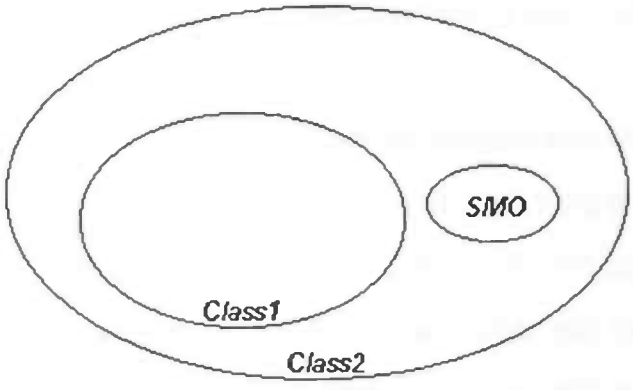


Figure7.3: It is also possible that the SMO-orientation is always an element of $Class_2$ and never an element of $Class_1$.

If the answer to question one is true that means that Mr. Tozikov is right. If the answer to question two is true that means that we are right. And if the answer to question three is true, that means that the SMO-orientation does not always belong to $Class_1$ but still always to $Class_2$ (because $Class_1$ in a sub class of $Class_2$), in other words we are right too.

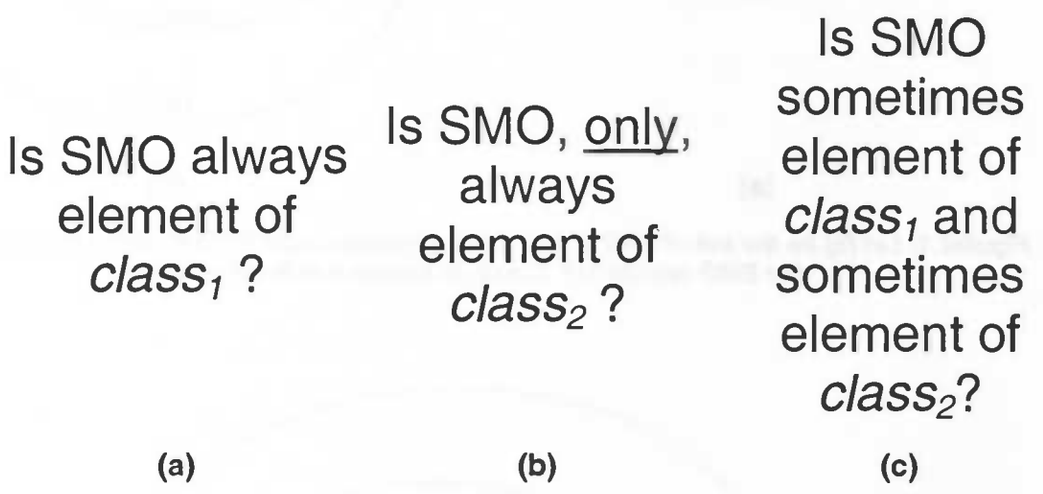


Figure7.4: three questions (a) Is Tozikov right ? (b) are we right ? (c) and here we are right too.

Because there was no analytical solution found for this question, that is, Mr Tozikov [1] could not find an analytical proof for his conjecture and we also ca not proof that our class of critical orientations, $class_2$, is the class that always include the SMO-orientation is, we choose for the experimental mathematical-method to answer this question.

The experimental mathematical method (also computational geometry method) we are going to use to give answers on this question, can be summarised in the following steps:

- Generate two random polyhedra.
- Calculate their SDR's.
- Calculate the $class_1$- and $class_2$-lists of orientations.
- Calculate the SMO-Vaab following $class_1$ and that following $class_2$.
- Compare the two values of the mixed volume.
- Repeat the previous steps x-times.
- Make a table of the results.

From these algorithm steps we derive the following list to-do-things:

1. We like to make **polyhedra** randomly, to get the biggest collection we can have.
2. We like to make the **slope diagram** of a polyhedron. Because rotations and parallel faces are easier to check and make in a SDR.
3. The **minkowski addition** of two polyhedra.
4. The **mixed volume**.
5. Calculating the two sets of **critical orientations**.
6. **Comparing** the two classes.

The main subject of the second part of this thesis is how to implement this list.

II Implementation

II.0 Introduction

Our goal is try to find a concrete case where the minimum value of the mixed volume VAAB, of A and B, belonging to the $class_2$-critical orientations, is smaller than the minimum value of the same functional, E.I. VAAB, when we use the $class_1$-critical orientations.

The programming tools we may use have to have the following properties:

- **Easy to use.** That because we like us to concentrate on the problem and not on the tools. We are *not* evaluating software here. We are testing some propositions and trying to make statements about them.
- **Widely used.** We are working on a new research field. This means that there will be more and more researches about this topic. Reusability of the program is of great benefit for further researches. This is possible when the programming language is widely in use.
- **OOP-support.** The programming language has to support the Object Oriented Programming-approach for reasons we give later in this chapter.

Led by this list of requirements, we decided to use the C++ programming language. C++ has an easy syntax, which has become a kind of standard in the programming languages-world and is widely used. C++ also has the advantage that there are many compilers available running under different operating systems. We chose the MS-VDS© (Microsoft Visual Development Studio) because the ease of use and the many possibilities it offers. MSVDS© only works under MS-Windows©. This operating system also met our requirements.

Strategy

In the following two sections we introduce two strategies and their properties. The first strategy failed. We will give the reasons why we choose for the second strategy.

1. The stone-by-stone strategy

From the beginning we wanted to implement everything ourselves. All the procedures we were going to use. Begin at the roots and end at the leaves. If we worked like this we could work elementary and all what we made would be well understood. This is important because the reliability of the results is the kernel of this research. This program is replacing an analytical proof of fail or succeeds of some propositions. One weak point in this strategy is that we are not working efficiently enough, as there are many libraries that can support our implementation work.

A polyhedron is composed of faces, edges and vertices plus some relation between these elements. This feature of a polyhedron guides us to think about ***objects,*** the key-idea behind OOP-programming. We have made the object-classes `polyhedron`, `face`, `edge` and `vertex`. A polyhedron includes at least four faces, each face includes at least three edges (and at least three vertices) and an edge is always connecting two vertices. These four classes of objects and the relations between them can be represented in the so-called whole-part-diagram following the *Fusion Notation* [4] (See figure0.1).

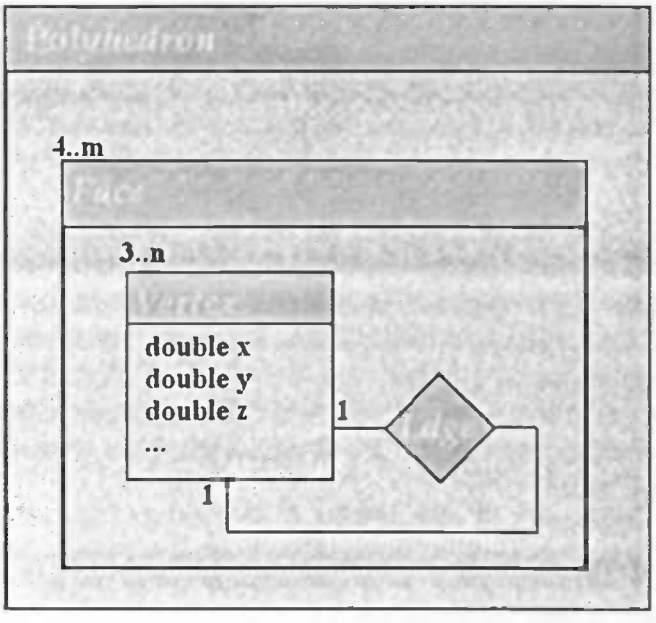


Figure 0.1: The whole-part-diagram of the four classes: `polyhedron`, `face`, `edge` and `vertex`. A polyhedron has at least four faces. A face has three or more vertices. The edge-class is a relation-class between two vertices. The edge is also a member-object of the face-class.

The OOP-approach has three important parts [4] Object Oriented Analyses OOA, Object Oriented Design OOD and Object Oriented Programming OOP. The first two parts are independent of which programming language we are going to use. We followed the outline given in [4] to analyse, design and implement the program.

The disadvantages of this approach

This approach gave some problems:

- An object is a kind of pointer, all the objects in this implementation (polyhedra, faces, edges and vertices) had to be connected with each

other in a very complicated network (faces with adjacent faces and vertices and edges, and double pointers, etc.) (See figure0.1) we had lots of problems with loop-pointers. Besides that, it took too much time to be able to generate the polyhedra themselves.

- A problem that stands in the way of any geometric algorithm-implementation is precision. Because our program was working with floating points, we had problems with the precision of the geometrical entities. For example, given a point v and a plane f. The point is not coplanar with the plane, but is very close to it. Some precision problems can occur and the program can see the point, as a point coplanar with the plane. This is in contradiction with the reliability of the program, which is very important to accept or refuse the proposals made in chapterI.7.

In LEDA© we found the solution for both problems.

2. The Leda© Strategy

What is LEDA©

LEDA stands for Library of Efficient Data types and Algorithms [5].

Apart from the usual built-in types such as integers, reals, vectors, and matrices, LEDA includes complex data types like stacks, queues, dictionaries, sequences, sorted sequences, priority queues, graphs, points, segments... These data types make the combinatorial and geometrical and computations easier.

Some features of LEDA

LEDA contains a convenient data type *graph*. It offers the standard iterations such as ``for all nodes v of a graph G do'' or ``for all neighbours w of v do'', it allows adding and deleting vertices and edges and it offers arrays and matrices indexed by nodes and edges... The data type graph allows writing programs for graph problems in a form close to the typical textbook presentation.

LEDA is implemented by a C++ class library. It can be used with almost any C++ compiler that supports templates.

Iteration

For many (container) types LEDA provides iteration macros. These macros can be used to iterate over the elements of lists, sets and dictionaries or the nodes and edges of a graph. Iteration macros can be used similarly to the C++ **for** statement. Examples for graphs:

- **forall_nodes**(*v*, *G*) { the nodes of *G* are successively assigned to *v*}
- **forall_edges**(*e*, *G*) { the edges of *G* are successively assigned to *e*}
- **forall_adj_edges**(*e*, *v*) { all edges adjacent to *v* are successively assigned to *e*}

After studying the possibilities of Leda we decided to rewrite our program with this library. Because LEDA has its own design for a polyhedron, the OOP-design we made before (See figure0.1) was superfluous. A polyhedron is defined in LEDA as a graph. The nodes are the vertices and the graph faces are the faces of the polyhedron.

What kinds of algorithms

The algorithms we used to implement our program are from different sources:

1. Geometrical algorithms, these are in general trivial algorithms and are introduced in most of the geometrical algorithms books. For example vector mathematics could be used from *the Computational Geometry with C* [the c-geometrical...]
2. [1] algorithms, the theories introduced in the paper *Similarity Measures for Convex Polyhedra Based on Minkowski Addition* [1], served as algorithms to implement some parts of the program. For example the definition of the mixed volume V(A,A,B) and that of the support function *h*(A,u) gave us the guideline to implement the routines `vaab()` and `support()`.
3. [2] algorithms. The algorithm of the most important procedure in this research, the `tvt()` routine, is introduced in the paper of Dr. Bekker *Calculating critical orientations of polyhedra for similarity measure evaluation* [2]. Here I like to point out that the `tvt()` routine was totally implemented and delivered to me by Dr. Bekker.
4. LEDA© algorithms. Many of the geometrical algorithms we used were implemented by LEDA©. One can have a look to the manual of LEDA on the World Wide Web [5].

In the sequel of this part we will give same explanations for the algorithms we used.

The list to-do-steps given in chapter 'The Problem' in the theoretical-part is the skeleton of the following part.

II.1 Making initial polyhedra

We use three methods to make polyhedra:

1. Manually.
2. Randomly with Leda routines.
3. Randomly with a self-made random generator.

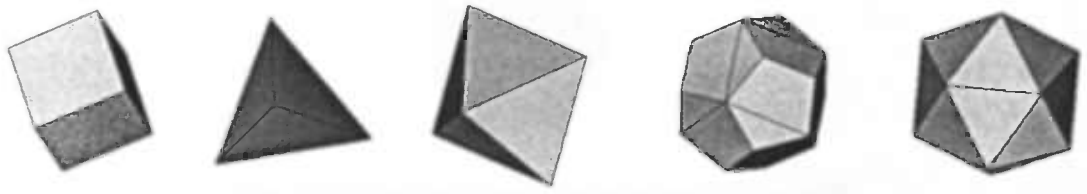
Manually

Here we write the co-ordinates of the vertices by hand. We take the Convex Hull of them to get a Leda-Graph. Many routines are available in LEDA© for graphs. For example, routines for editing and visualisation graph.

Here an example of a Manually generated tetrahedron:

```
{d3_rat_point p1(100,100,100,1);L.append(p1);
d3_rat_point p2(100,-100,-100,1);L.append(p2);
d3_rat_point p3(-100,100,-100,1);L.append(p3);
d3_rat_point p4(-100,-100,100,1);L.append(p4);}
CONVEX_HULL(L,G);
```

The result of this example is shown in figure1.1(b).



(a) Cube **(b) Tetrahedron** **(c) Octahedron** **(d) Dodecahedron** **(e) Icosahedron**

Figure1.1: the five Platonic polyhedra made manually.

Randomly with Leda routines

Leda has many routines to generate points (also 3D rational points) randomly. For example one can use one of the following routines:

```
random_d3_rat_point_in_ball()
random_d3_rat_point_in_cube()
random_d3_rat_point_in_disc()
random_d3_rat_point_in_square()
random_d3_rat_point_in_unit_ball()
random_d3_rat_point_in_unit_cube()
random_d3_rat_point_on_circle()
random_d3_rat_point_on_paraboloid()
random_d3_rat_point_on_sphere()
```

In the following listing we used the first routine to generate the polyhedron shown in figure1.2:

```
random_d3_rat_point_in_ball(6,100,L);
CONVEX_HULL(L,G);
```

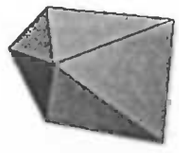


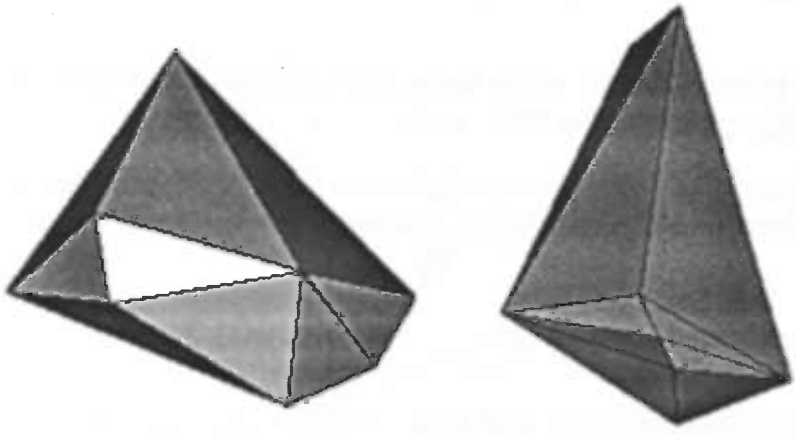
Figure1.2: a polyhedron made by the LEDA© routine **`random_d3_rat_points_in_ball(6,100,L)`**

Randomly with a self-made random generator

One problem with the Leda-random generators is that it is not possible to re-generate the same polyhedron when one gets an interesting result. Therefore we made a generator that can be reset to regenerate the same polyhedra generated before. This random generator is a kind of table of random polyhedra with entries. Calling the random generator with the same entry means generating the same 'random' polyhedron.

The following listing shows the code used to generate the two polyhedra shown in figure1.3.

```
for(j=0;j<8;j++){
      double a[3]; randvec(a); svmul(200, a, a);
      d3_rat_point p(trunc(a[0]), trunc(a[1]), trunc(a[2]));
      L.append(p);}
CONVEX_HULL(L,G);
```



(a) (b)

Figure1.2: (a) Randvec1 and (b) randvec2

Polyhedra can also be made by editing existing polyhedra, by cutting a corner for example, but also by [see following chapter] adding two polyhedra using the minkowski addition. These polyhedra are not initial polyhedra. Therefore they are not included in this chapter.

II.2 The slope Diagram Representation

All we need from the slope diagrams represented in the Theoretical part, of this thesis, are the normal vectors of the faces of the polyhedron (the spherical points) and the list of adjacent faces (the spherical arcs).

We implemented the slope diagram representation of a polyhedron P in two lists:

1. list of normal vectors of the faces of P. These vectors represent the spherical points of the SDP, and
2. list of couples of the normal vectors of adjacent faces from the previous list. These couples of vectors represent the spherical arcs of SDP.

Data types

The list of spherical points is of type: `face_array <d3_rat_point>.`

The list of spherical arcs is of type: `edge_array <d3_rat_segment>.`

We chose for the array of segments because LEDA has type `segment` (and not type `arc`) and because a segment has properties similar to an arc. Like a segment two spherical points define an arc, and two arcs intersect when the line-segments, defined by the vertices of these arcs cross.

Calculation of the list of spherical points of SDP

The following algorithm shows how we implemented the list of spherical points.

```
1. Forall faces fp of P do
2.     a = first_vertex(P)
3.     b = second_vertex(P)
4.     c = third_vertex(P)
5.     line1 = b - a
6.     line2 = c - a
7.     fn[fp] = outer_product(line1, line2)
8. od
```

Explanation of the algorithm.

Check all the faces of polyhedron `P (1.)`. Determine the position of the first three vertices of this face `(2., 3., 4.)`. As we know, the normal vector of a face (plane) is perpendicular to any two crossing lines on this face (plane). From these three vertices (points) we calculate the direction vectors of two crossing lines on the plane, `line1 (5.)` and `line2 (6.)`. In `(7.)` the perpendicular vector on these two lines is calculated by taking the outer product of the direction vectors of these two lines. The result, the normal vector of face `fp` is stored in the list of spherical points `fn` (from face normal vectors) with `fp` as index.

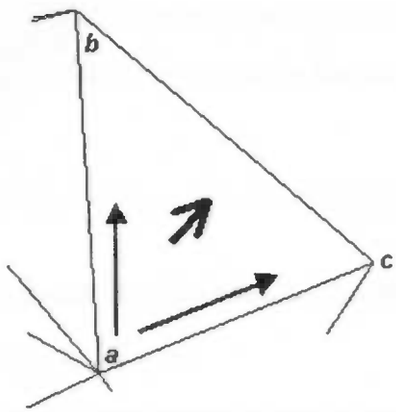


Figure2.1: The normal vector of a face is defined by the vector perpendicular to two crossing lines in that face.

Calculation of the list of spherical arcs of SDP

```
1. forall faces f_p,1 of P do
2.     forall face_edges(e_p, f_p, 1) do
3.             f_p,2 = adj_face(e_p, f_p,1)
4.             if f_p,1 < f_p,2
                   then arc[e_p] = segment( fn[f_p,1] , fn[f_p,2] )
5.             fi
6.     od
7. od
```

Explanation

Check all faces of P `(1.)`. Check all edges of this face `(2.)`. Find the adjacent face of $f_{p,1}$ with e_p as adjacent edge for both faces `(3.)`. The segment defined by the spherical points of $f_{p,1}$ and $f_{p,2}$ represents the spherical arc between

these two spherical points. This segment is stored in `arc`, the list of spherical arcs with e_p as index `(4.)` (See figure2.2).

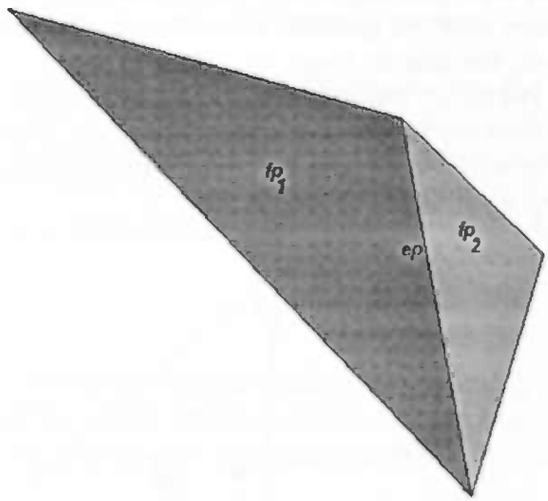


Figure2.2: face $f_{p,1}$ and the adjacent face $f_{p,2}$, with e_p adjacent to both faces.

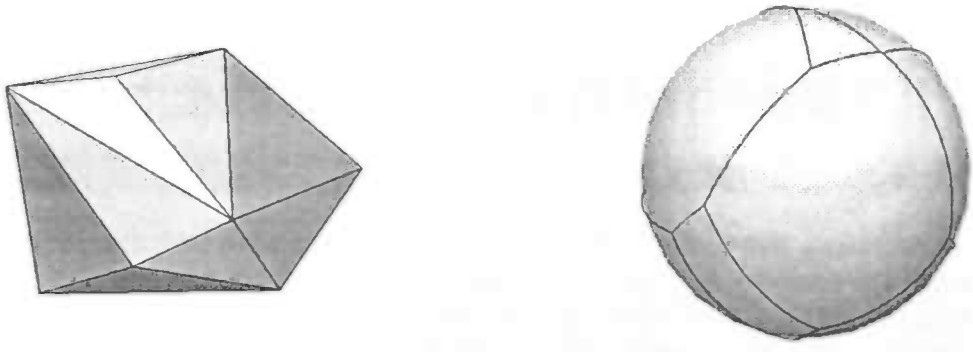
Visualisation

To visualise the slope diagram representation of a polyhedron P (SDP) we used the computer algebra program Maple©.

Example of an input file representing an SDR

```
sldg1 := {
arc( 1870.000000, 15038.000000, -20866.000000, -9855.000000, -24349.000000, 26128.000000),
arc( 1870.000000, 15038.000000, -20866.000000, -1170.000000, -16828.000000, 5636.000000),
arc( 1870.000000, 15038.000000, -20866.000000, 9155.000000, 26139.000000, -10898.000000),
arc( -9855.000000, -24349.000000, 26128.000000, -1170.000000, -16828.000000, 5636.000000),
arc( -1170.000000, -16828.000000, 5636.000000, 9155.000000, 26139.000000, -10898.000000),
arc( -9855.000000, -24349.000000, 26128.000000, 9155.000000, 26139.000000, -10898.000000)
 } ;
```

In figure2.3 a more complicated polyhedron is drawn with its slope diagram.



(a) (b)

Figure2.3: a polyhedron (a) and its slope diagram representation (b).

The SDR is visualised in Maple©

Remark

LEDA has two routines which combined should be able to calculate the normal vector of a face:

- `d3_rat_plane p(a,b,c), and`
- `d3_rat_point normal(plane p).`

The first routine (constructor) makes a plane and the second calculates the normal vector of a plane. The algorithm could look like this:

```
1. Forall faces fp of P do
2.     a = first_vertex(P)
3.     b = second_vertex(P)
4.     c = third_vertex(P)
5.     d3_rat_plane plane(a,b,c)
6.     fn[fp] = normal(plane).
7. od
```

When we implemented this, LEDA did not return the correct answer. It took us a lot of time to discover this bug. This is one more reason not to believe that all software tools are correct!

II.3 Minkowski addition of two polyhedra

How to calculate

To calculate the minkowski sum of two polyhedra we use the method derived from the general definition [ChapterI.2] of the minkowski addition given in the theoretical part. This method uses the vertices of the two polyhedra to calculate the minkowski sum. Here we will give the method in the same form it was introduced in the previous part and after that we will derive the algorithm from it.

> The minkowski sum of two convex polyhedra A and B is again a convex polyhedron, say C, defined as the convex hull of the points formed by adding all possible couples of vertices (V_a, V_b) where $V_a \in A$ and $V_b \in B$.
>
> More formal: $C = A \oplus B = Conv\{V_a + V_b \mid V_a \in A, V_b \in B\}$.
>
> Where Conv{.} to be read as "*the convex hull of . .* ".

The algorithm

The algorithm to calculate the minkowski sum following this definition is trivial:

Given two convex polyhedra A and B, the minkowski sum of these polyhedra is again a convex polyhedron C and can be calculated as follows:

```
1. forall vertices Va of A do
2.     forall vertices Vb in B do
3.             Vc = Va + Vb
4.             L.append(Vc)
5.     od
6. od
7. C = Convex_hull(L)
```

Here

- V_a, V_b and V_c are respectively the vertices of A, B and C.
- L is a list of vertices, initially empty.
- L.Append(V_c) adds a vertex V_c to list L.

This algorithm is $O(nv_A * nv_B)$, where nv_A and nv_B are the numbers of vertices of A and B respectively.

Example

make two random polyhedra with the self-made generator (see ChapterII-1) and calculate the minkowski sum of the two polyhedra.

```
make_polyhedron(A,6);
draw_polyhedron(A);
make_polyhedron(B,6);
draw_polyhedron(B);
minkowski_sum(A,B,C);
draw_polyhedron(C);
```

Figure3.1 shows the result of the previous listing.

(a) randvec1 *(b) randvec2* *(c) randvec1⊕randvec2*

Figure3.1: Two random polyhedra made with the random generator randvec (a), (b). The minkowski sum of the two polyhedra (c).

II.4 Mixed Volumes

Calculation of the mixed volumes

For the calculation of the mixed volume of two polyhedra, A and B, we will use the theorem given in the chapter *mixed volumes* in the theoretical part of this thesis:

Theorem

> The mixed volume V(A,A,B) can be calculated as follows:
>
> If A is a convex polyhedron with faces F_i and corresponding outward unit normal vectors u_i, i=1,...,k, then following [1]
>
> $$V(A,A,B) = {}^1/_3 \sum^k{}_{i=1} h(B,u_i)S(F_i),$$
>
> where $S(F_i)$ is the area of the face F_i of A and $h(B,u_i)$ is the value of the support function of A for the normal vector u_i.

We like to remark here that the calculation of V(A,B,B) is similar to that of V(A,A,B). Also, we note that V(..,..,..) is invariant in the order of its arguments. In other words, V(A,B,B) = V(B,A,B) = V(B,B,A).

The method

From this theorem we derive the following method:

```
1. forall faces fa of A do
2.     tmp_vaab = tmp_vaab + Area(fa) * support(B,ui)
3. od
4. vaab = tmp_vaab/3
```

Hence, we need to calculate the area of each face of A and the support function of polyhedron B in the direction of the outward unit normal of that face.

Calculation of the area of a face f ∈ A

Given a face f, the following routine shows how to calculate the area of this face:

```
1. nodes = adj_nodes(f)
2. pop(a,b,c,nodes)
3. area += area_triangle(a,b,c)
4. while(not_empty(nodes)) do
5.       b = c
6.       pop(c,nodes)
7.       area += area_triangle(a,b,c)
8. fa[f] = area
```

Here:

`nodes` denotes the list adjacent nodes (vertices) of face `f`.

`adj_nodes(f)` returns the adjacent nodes of face `f`.

`pop(a,b,c,nodes)` stores the first three elements of nodes in `a,b` and `c` respectively and delete these elements from `nodes`. `pop(c,nodes)` does the same but for one element at once.

`area_triangle(a,b,c)` calculate the area of the triangle defined by the three points `a,b,c`.

`not_empty(nodes)` returns *true* when `nodes` is not empty and *false* when `nodes` is empty.

`fa[f] = area` the area of face `f` is stored in the face area array `fa` with `f` as index.

Calculate of the support function h(A,u$_l$)

Again, from the following theorem introduced before [ChapterI.4], the calculation of the support function is derived:

Theorem

> Every element A of the set of all compact sets ***C*** (in 2D and 3D-space) is uniquely determined by its *support function*:
>
> $h(A,u) = \sup\{\langle a,u\rangle \mid a \in A\}, \quad u \in S^2$ (or $u \in S^1$)
>
> Here is the $\langle a,u\rangle$ is the inner product of vectors a and u. S^2 denotes the unit sphere in R^3 (S^1 denotes the unit circle in R^2)

Here the way to calculate the support function:

From the previous theorem we derived the following method:

```
1. forall vertices Va of A do
2.      ip = ⟨u, Va⟩
3.      support = MAX(ip, support)
4. od
```

Here:

`Va` is an iterator for the list of all vertices of polyhedron A.

`ip` is the inner product.

and `support` has the value of *h*(A,u). Initially Support = -1e11.

Finally we like to note that we will make use of V(A,A,B) only because polyhedra A and B are both random and it does not matter which we use, V(A,A,B) or V(A,B,B).

II.5 The Critical Orientations

Introduction

In the theoretical part of this thesis [ChapterI.5] we introduced two classes of critical orientations: $Class_1$ and $Class_2$.

$Class_1$ is the class of relative orientations of B w.r.t.* A where a spherical point of SDB coincides a spherical point of SDA and another spherical point of SDB lies on a spherical arc of SDA.

$Class_2$ has the property that three spherical points of SDB lie on three spherical arcs of SDA. Here the spherical points and the arcs may be degenerated. In other words two (or three) spherical points or arcs may be the same.

In this chapter we will give an implementation of an orientation in general. Then the algorithms behind the routines that will deliver the two classes of critical orientations, are discussed.

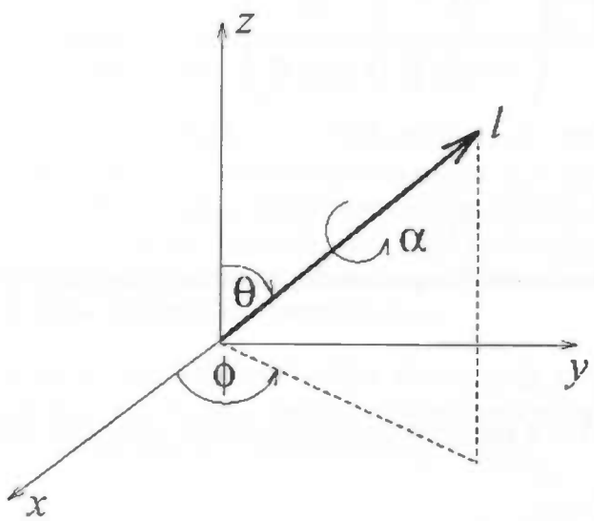


Figure5.1: three angles represent an orientation. ϕ and θ represent the rotation line l and α the rotation-angle about this line.

* Remark: "B w.r.t. A" is the same like "SDB w.r.t. SDA".

Orientation

Any orientation of SDB w.r.t. SDA can be expressed as a rotation with an angle α about a line l. The line can also be represented by two angles: ϕ and θ. So, any orientation can be expressed by these three angles. See figure5.1. Also any rotation about a line l with an angle α, denoted by $r_{l,\alpha}$, can be expressed as a product of five rotations:

$$r_{l,\alpha} = r_{z,\phi}\, r_{y,\theta}\, r_{z,\alpha}\, r_{y,-\theta}\, r_{z,-\phi} \qquad (5.1)$$

where z denotes the z-axis and y the y-axis and $-\theta$ means the rotation-angle θ in the opposite direction.

As we can see in figure5.1 any orientation can be expressed with $r_{l,\alpha}$. What we need here are the rotation matrices about the z- and y-axis. These matrices are given in figure5.2.

$$R_{y,\theta} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \qquad R_{z,\phi} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(a) **(b)**

Figure5.2: two matrices represent (a) a rotation about the y-axis with angle θ and (b) a rotation about the z-axis by angle ϕ.

The rotation of a polyhedron `P` by a matrix `r` is done vertex-wise:

```
1. forall vertices v ∈ P do
2.     mvmul(r, v, v)
3. od
```

Here:

`mvmul()` means the matrix-vector multiplication.

`r` is the rotation matrix, and

`v` is a vertex of polyhedron P.

The Two Vector Triples algorithm (tvt)

The two vector-triples algorithm can be described as follows:

Given two lists, the list of spherical points of SDB, we will call it `fn`, and

the list of spherical arcs of SDA, call it `arcs` (we used the same notation for the two lists, `fn` and `arcs`, like in chapterII-2).

The Problem

For each three elements of list `fn`, say `a,b,c` and each three elements of `arcs`, say `k,l,m` find the orientations that can rotate the triple `k,l,m` in such a way that `a` lies on `k`, `b` lies on `l`, and `c` lies on `m`.

The solution

During our research about the similarity measures, Dr. Bekker could find a solution for this problem: the two vector triples algorithm, shortly the tvt-algorithm [2]. I am not going to explain this algorithm extensively, I refer the interested reader to the paper written by Dr. Bekker about his algorithm. Only, some explanation about the pre- and post-conditions and some properties of this algorithm are given to understand what we are going to use.

The tvt-algorithm is not a direct method to solve the problem. Two formulations of the problem are made:

1. Find the orientations of SDB w.r.t. SDA where `a,b` and `c` lie on the three great circles containing the three arcs `k,l` and `m`, respectively. This because a direct method for solving this problem for spherical arcs was not found [2], but only for the great circles containing these arcs. From those solutions, those ones can be selected with the points located on the arcs.
2. Find the orientations of SDB w.r.t. SDA where `a,b` and `c` are perpendicular with the normal vectors of the three great circles containing the three arcs `k,l` and `m`, respectively.

In general tvt() is a routine that takes two vector triples as input and returns an array of, at most, eight rotation matrices as output. The header of the routine is:

```
tvt(a,b,c,k,l,m,nr_sols,sols)
```

where:

- `a,b,c` are the three spherical points of SDB representing the first vector triple.
- `k,l,m` are the three spherical arcs of SDA representing the second vector triple.
- `nr_sols` is an integer between zero and eight and represents the number of solutions that could be found.
- `Sols` is an array with eight entries. Every entry represents a rotation. A rotation is a 3x3-matrix. Only the first `nr_sols` entries are filled.

The working of the algorithm can be expressed in the following steps:

```
1. if(ill_posed(a, b, c) || ill_posed(k, l, m))
2.     then
3.       nr_sols=0
4.     else
5.     precondition( a, b, c, k, l, m)
6.           fillcoeffs()
7.           calcuvw()
8.           calcsols()
9. fi
```

where :

- `ill_posed(a, b, c)` checks if the vector triple `a, b, c` is ill posed. A vector triple is, for example, ill posed if the three vectors `a, b` and `c` are all collinear. This means that `tvt()` will enter an infinite loop because there is an infinite number of solutions. Identical for `ill_posed(k,l,m)`.
- `precondition( a, b, c, k, l, m)` preconditions the two vector triples such that `a` coincides with the x-axis, and `k` coincides with the z-axis and `b` lies in the xy-plane and `l` lies in the yz-plane
- With `fillcoeffs()` and `calcuvw()` the coefficients and the variables `u, v, w,` needed for the calculation of the eight degree equation, are calculated and filled.

- At the end the at most eight possible solutions of the eight degree equation are calculated by `calcsols()` and copied to `sols`. The number of solutions is stored in `nr_sols`.

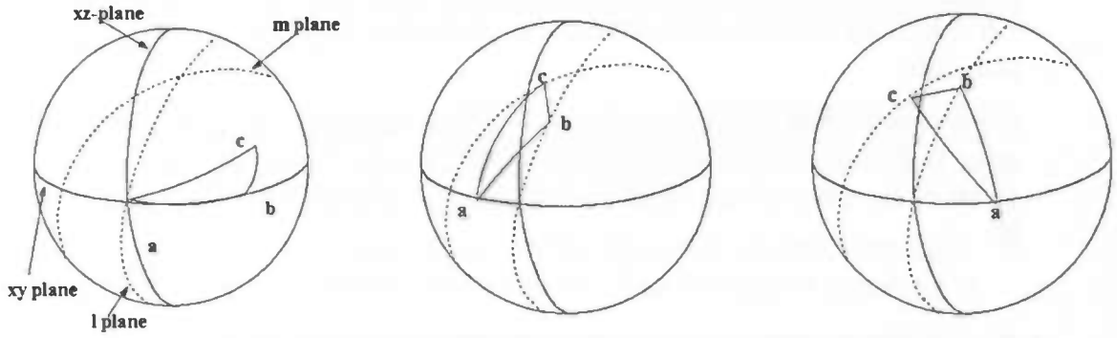


Figure5.3: (a) The situation after preconditioning the vectors a, b, c and k,l,m. The l plane and the m plane are the planes normal to the vectors l and m (not shown). The k plane coincides with the xy plane. (b) shows one of the solutions of tvt. (c) shows another solution. (these figures are taken from [2])

Applying this algorithm on all possible pairs of vector triples of the spherical points-list of SDB and the spherical arcs-list of SDA will give the $Class_2$ critical orientations of SDB w.r.t. SDA. $Class_2$ is not complete like this, because the critical orientations of SDA w.r.t. SDB have to be calculated too.

`tvt()` has the property that:

```
tvt(a,b,c,k,l,m,nr_sols,sols)=
tvt(b,c,a,l,m,k,nr_sols,sols)=
tvt(c,b,a,m,l,k,nr_sols,sols)=
tvt(a,c,b,k,m,l,nr_sols,sols = ….
```

In other words, tvt is invariant in its arguments, when the corresponding vectors take the positions shown above.

The Two Vector pair algorithm (tvp)

The $Class_1$ critical orientations are easier to calculate than the $Class_2$ of critical orientations.

The problem

Given two non-coinciding spherical points of SDB and a spherical point and a spherical arc of SDA. Find the orientations of SDB w.r.t. SDA where one spherical point of SDB coincides with the spherical point of SDA and the other spherical point of SDB lies on the spherical arc of SDA.

The solution

Rotate SDB such that SDB(fi), a spherical point of SDB, coincides with SDA(fj). Now only one rotational degree of freedom is left, namely the rotation about the axis line through SDB(fi) (or SDA(fj) as they are coinciding now) and the origin. By solving an equation, the orientation is found, if there is a solution possible.

In general tvp() is a routine that takes two spherical points of SDB and a spherical point and a spherical arc of SDA, as input and returns, if there is a solution, an array of two rotation matrices as output. The header of the routine is:

```
tvp(a,b,k,l,nr_sols,sols)
```

where:

- `a,b` are the two spherical points of SDB.
- `k` is the spherical point and `l` is the spherical arc of SDA.
- `nr_sols` is an integer and can have zero or two as value and denotes the number of solutions that could be found.
- `Sols` is an array with two entries. Every entry denotes a rotation. Here a rotation is a 3x3-matrix.

The steps of the algorithm are similar to the tvt-algorithm:

```
1. if(ill_posed(a, b, k, l))
2.     then
3.       nr_sols=0
4.     else
5.           nr_sols=2
6.     precondition(a, b, k, l)
7.           fillcoeffs()
8.           calcuvw()
9.           calcsols()
10.fi
```

where :

- `ill_posed(a, b, c, d)` checks, similar to tvt, for degenerated vectors.
- `precondition( a, b, k, l)` preconditions `a` and `k` such that `a` coincides with `k`, and rotate `b` and `l` with the same rotation matrix used for `a` and `k`.
- With `fillcoeffs()` and `calcuvw()` the coefficients and the variables `u, v, w,` needed for the calculation of the second degree equation are calculated and filled.
- At the end the two solutions of the equation are calculated by `calcsols()` and copied to `sols`. The number of solutions is stored in `nr_sols`.

Similar to tvt, the critical orientations of SDA w.r.t. SDB have to be calculated too.

II.6 Similarity Measure

Introduction

Two similarity measure functions were introduced in part-I [ChapterI.6]:

σ_1 is the similarity measure function based on the minkowski addition:

$$\sigma_1(A,B) = \sup_{r \in R} \frac{8V(A)^{1/2}\, V(B)^{1/2}}{V(A \oplus r(B))} \qquad (6.1)$$

and σ_2 is the similarity measure function based on the mixed volume:

$$\sigma_2(A,B) = \sup_{r \in R} \frac{V(A)^{2/3}\, V(B)^{1/3}}{V(A,A,r(B))} \qquad (6.2)$$

where r is the similarity measure orientation (SMO) of B [ChapterI.6].

Because V(A,A,B) is linear proportional to V(A⊕B) as seen in the following equation [part-I]:

$$V(\lambda A \oplus \mu B) = \lambda^3 V(A) + 3\lambda^2 \mu V(A,A,B) + 3\lambda \mu^2 V(A,B,B) + \mu^3 V(B).$$

We decided to work with σ_2, as the answers will have similar effect. With σ_2 we do not need to calculate the minkowski sum for every orientation but the mixed volume, which is much more simpler to implement.

Problem

Given two polyhedra A and B, calculate the similarity measure of them.

Solution

To calculate σ_2, (6.2) has to be solved as follows:

1. Calculate the volume of A and B,

2. Make the slope diagram's SDA and SDB,
3. Calculate the critical orientations
4. For all $r \in$ critical orientations do
    - Rotate SDB with r
    - Calculate the mixed volume
    - If V(A,A,r(B)) is the smallest mixed volume found till now then store it.
5. The stored mixed volume has the smallest value, and r(B) is the SMO-orientation. Fill the mixed volume and the V(A) and V(B) in the quotient in (6.2). The answer is the similarity measure of A and B.

III Experiments and Results

III.1 Experiments and results

We did 1000 experiments. Every experiment is a minimization of the mixed volume V(A,A,*R*(B)) using the two classes of critical orientation, $class_1$ and $class_2$. In these experiments, A and B are both tetrahedra with random edge length.

The following table shows an arbitrary selection of these results.

Vaab	
$Class_1$ minimization value	**$Class_2$ minimization value**
1.509831e+006	1.261319e+006
1.610662e+006	1.166644e+006
1.683144e+006	1.624035e+006
4.261788e+006	3.620793e+006
3.937615e+006	1.810356e+006
5.051870e+005	4.604507e+005
5.519317e+005	4.891879e+005
2.904028e+006	1.010602e+006

The table shows that not all SMO orientations are an element of $class_1$ orientations. But they are all, at the opposite side, an element of $class_2$ orientations.

When we calculated the percentage in all the 1000 experiments, about 90% of the SMO orientations where an element of $class_2$ critical orientations.

The best result we found was

Vaab	
$Class_1$ minimization value	**$Class_2$ minimization value**
3.937615e+006	1.810356e+006

Here V(A,A,B) minimization using $class_1$ orientations is too large with respect to the minimization value using $class_2$ orientations.

III.2 Conclusion and discussion

We did 1000 experiments using 2000 random generated tetrahedra. The mixed volume functional V(A,A,*R*(B)) was minimized using the two classes of critical orientations, $class_1$ and $class_2$. In contrast with what has been conjectured [1], to calculate the similarity measure of two convex polyhedra A and B **both**[1] the classes of critical orientations, class1 and class2 have to be taken into account.

[1] This for the case that both classes are disjunct as we defined $class_2$ before.

References

[1] H. J. A. M. Heijmans, A. V. Tuzikov and J. B. T. M Roerdink. ***Similarity Measures for Convex Polyhedra Based on Minkowski Addition***. Technical report, Department of Computer Science, University of Groningen, 1997. Pattern Recongnition 33 (2000) 979-995.

[2] H. Bekker, J B T M Roerdink. ***Calculating critical orientations of polyhedra for similarity measure evaluation***. Department of Computer Science, University of Groningen. Proc. of the IASTED Int. Conf., Computer Graphics And Imaging, Palm Springs, California, USA, Oct. 25-27 1999.

[3] H. Bekker, J B T M Roerdink. ***An efficient algorithm to calcukate the minkowski sum of convex 3D polyhedra***. Department of Computer Science, University of Groningen.

[4] R. Smedinga. ***Object Georienteerd programmeren***. Lecture about OO-Programming. Department of Computer Science, University of Groningen, 1997.

[5] K. Mehlhorn, S. Näher, M. Seel and C. Uhrig. ***The LEDA User Manual***, Version 4.2. http://www.mpi-sb.mpg.de/LEDA. LEDA research project 2000-10-12