Reducing the time complexity of Minkowski sum based similarity calculations by using geometric inequalities

Henk Bekker[1], Axel Brink[1]

[1]Institute for Mathematics and Computing Science, University of Groningen, P.O.B. 800 9700 AV Groningen, The Netherlands. bekker@cs.rug.nl, a.a.brink@student.rug.nl

Abstract. The similarity of two convex polyhedra A and B may be calculated by evaluating the volume or mixed volume of their Minkowski sum over a specific set of relative orientations. The relative orientations are characterized by the fact that faces and edges of A and B are parallel as much as possible. For one of these relative orientations the similarity measure is optimal. In this article we propose and test a method to reduce the number of relative orientations to be considered by using geometric inequalities in the slope diagrams of A and B. In this way the time complexity of $O(n^6)$ is reduced to $O(n^{4.5})$. This is derived, and verified experimentally.

1 Introduction: Minkowski-sum based similarity measures

Because shape comparison is of fundamental importance in many fields of computer vision, in the past many families of methods to calculate the similarity of two shapes have been proposed. Well-known families are based on the Hausdorff metric, on contour descriptors and on moments of the object, see [1] for an overview. Recently, a new family of methods has been introduced, based on the Brunn-Minkowki inequality and its descendants. The central operation of this method is the minimization of a volume or mixed volume functional over a set of relative orientations [2]. It is defined for convex objects, and can be used to calculate many types of similarity measures. Moreover, it is invariant under translation and rotation, and when desired, under scaling and reflection. The methods may be used in any-dimensional space, but we will concentrate on the 3D case. Experiments with these methods have been performed on 2D polygons and 3D polyhedra [3,4], and show that for polygons the the time consumption is low. However, already for 3D polyhedra of moderate complexity in terms of the number of faces, edges and vertices the time consumption is prohibitive. In this article we present a method to reduce the time complexity of these calculations by reducing the number of relative orientations to be considered.

The structure of this article is as follows. In this section we introduce the Minkowski sum, the notion of mixed volume, the Brunn-Minkowski inequalities, and derive some example similarity measures. In section two we introduce the slope diagram representation of convex polyhedra, define the set of critical orientations to be considered, present the current algorithm to calculate a similarity measure, and discuss its time complexity. In section three we introduce and test the new and more efficient algorithm, and we derive its theoretical time complexity.

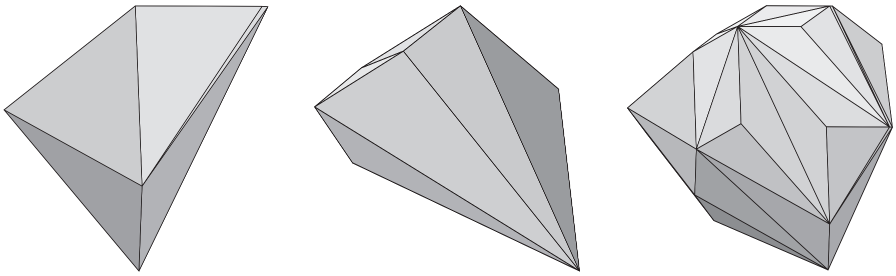

Fig. 1. Two polyhedra A and B and their Minkowski sum C. C is drawn on half the scale of A and B.

Let us consider two convex polyhedra A and B in 3D. The Minkowski sum C of two polyhedra A and B is another polyhedron, generally with more faces, edges and vertices than A and B, see figure 1. It is defined as

$$C \equiv A \oplus B \equiv \{a + b \mid a \in A, b \in B\}. \tag{1}$$

This definition does not give much geometrical insight how C is formed from A and B. To get some feeling for that, we separately look at two properties of C, namely its shape and its position. The shape of C may be defined by a sweep process as follows. Choose some point p in A, and sweep space with translates of A such that p is in B. C consists of all points that are swept by translates of A. The same shape C results when A and B are interchanged. The position of C is roughly speaking the vectorial sum of the positions of A and B. More precise, the rightmost coordinate of C is the sum of the rightmost coordinates of A and B, and analogously the leftmost, uppermost and lowermost coordinates of C. In this article only the shape of C plays a role, not its position. Obviously, the shape and volume of C depend on the relative orientation of A and B.

The volume of C may be written as

$$V(C) = V(A \oplus B) = V(A) + 3V(A, A, B) + 3V(A, B, B) + V(B). \tag{2}$$

Here, $V(A)$ and $V(B)$ are the volumes of A and B, and $V(A, A, B)$ and $V(A, B, B)$ are the *mixed volumes*, introduced by Minkowski [6]. Geometrically it is not obvious how the volume of A and B and the mixed volumes add up to the volume of C. However, it can be shown that $V(A, A, B)$ is proportional to the area of A and the linear dimension of B, and $V(A, B, B)$ is proportional to the linear dimension of A and the area of B.

As an example we derive two typical similarity measure expressions, based on the following two theorems [3,6]:

Theorem1: For two arbitrary convex polyhedra A and B in R^3,

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

with equality if and only if $A = B$.

Theorem2: For two arbitrary convex polyhedra A and B in R^3,

$$V(A \oplus B) \geq 8V(A)^{\frac{1}{2}} V(B)^{\frac{1}{2}} \tag{4}$$

with equality if and only if $A = B$.

From these theorems the similarity measures σ_1 and σ_2 respectively may be derived in a straightforward way,

$$\sigma_1(A,B) \equiv \max_{R \in \mathcal{R}} \frac{V(A)^{2/3} V(B)^{1/3}}{V(R(A), R(A), B)} \tag{5}$$

$$\sigma_2(A,B) \equiv \max_{R \in \mathcal{R}} \frac{8V(A)^{\frac{1}{2}} V(B)^{\frac{1}{2}}}{V(R(A) \oplus B)}. \tag{6}$$

Here $\mathcal{R}$ denotes the set of all spatial rotations, and $R(A)$ denotes a rotation of A by R. Because the volumes in these equations are always positive, σ_1 and σ_2 are always positive and ≤ 1, with equality if and only if $A = B$. Besides the inequalities in theorem1 and theorem2 many other inequalities exist, some based on the volume of the Minkowski sum, some on the mixed volume, some on the area of the Minkowski sum or the mixed area. From every of these inequalities a similarity measure may be derived. In this article we concentrate on computing σ_1 because the technique presented in this article to speed up this computation may be applied to other Minkowski sum based similarity calculations as well.

2 Calculating the similarity measure straightforward

To find the maximum in (5), in principle an infinite number of orientations of A have to be checked. That would make this similarity measure useless for practical purposes. Fortunately, as is shown in [3], to find the maximum value only a finite number of relative orientations of A and B have to be checked. Roughly speaking these orientations are characterized by the fact that edges of B are as much as possible parallel to faces of A. To formulate this more precise we use the slope diagram representation (SDR) of polyhedra.

We denote face i of polyhedron A by $F_i(A)$, edge j by $E_j(A)$, and vertex k by $V_k(A)$. The SDR of a polyhedron A, denoted by SDR(A), is a subdivision on the unit sphere. A vertex of A is represented in SDR(A) by the interior of a spherical polygon, an edge by a spherical arc of a great circle, and a face by a vertex of a spherical polygon, see figure 2. To be more precise:

- *Face representation.* $F_i(A)$ is represented on the sphere by a point $SDR(F_i(A))$, located at the intersection of the outward unit normal vector u_i on $F_i(A)$ with the unit sphere.

– *Edge representation.* An edge $E_j(A)$ is represented by the arc of the great circle connecting the two points corresponding to the two adjacent faces of $E_j(A)$.
– *Vertex representation.* A vertex $V_k(A)$ is represented by the interior of the polygon bounded by the arcs corresponding to the edges of A meeting at $V_k(A)$.

Some remarks. From this description it can be seen that the graph representing SDR(A) is the dual of the graph representing A. SDR(A) is not a complete description of A, it only contains angle information about A. Obviously, when A is rotated by a rotation R, the slope diagram representation rotates in the same way, i.e., $SDR(R(A)) = R(SDR(A))$. In the following, when speaking about distance in an SDR we mean spherical distance, i.e. the length of an arc on the unit sphere. Because the angle between two adjacent faces of a polyhedron is always $< \pi$, the length of the arcs in a SDR is always $< \pi$.

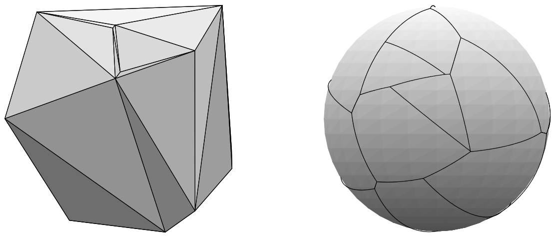

Fig. 2. (a): A polyhedron A. (b): The slope diagram representation of A. The orientations of A and $SDR(A)$ are the same, so with some patience it should be possible to see how they are related.

The slope diagram representation is useful to represent situations where faces and edges of A are parallel to faces and edges of B. It is easily verified that the faces $F_i(A)$ and $F_j(B)$ are parallel when in the overlay of $SDR(A)$ and $SDR(B)$ the point $SDR(F_i(A))$ coincides with the point $SDR(F_j(B))$. Also, an edge $E_i(B)$ is parallel to $F_j(A)$ when the point $SDR(F_j(A))$ lies on the arc $SDR(E_i(B))$. The description given earlier, stating that (5) obtains its maximum value when edges of B are as much as possible parallel to faces of A can now be made more precise in terms of their slope diagrams:

Theorem3: When σ_1 is maximal then three points of $SDR(R(A))$ lie on three arcs of $SDR(B)$.

This theorem is derived in [3]. Unfortunately, this theorem does not tell for which three points in $SDR(R(A))$ and which three arcs in $SDR(B)$ σ_1 is maximal, thus to find the maximum, all rotations R have to be considered for which three points of SDR(R(A)) lie on three arcs of $SDR(B)$. So, for three given points p_1, p_2, p_3 in $SDR(A)$ and three arcs a_1, a_2, a_3 in $SDR(B)$, an algorithm is needed that calculates a spatial rotation R for which holds that $R(p_1)$ lies on

a_1, $R(p_2)$ lies on a_2 and $R(p_3)$ lies on a_3. We developed such an algorithm [5], and implemented it in the function tvt(). It takes as argument three points and three arcs and calculates a rotation R. It is called as tvt(p1, p2, p3, a1, a2, a3, R). The function tvt() first calculates a rotation R with the property that $R(p_1)$ lies on c_1, $R(p_2)$ lies on c_2 and $R(p_3)$ lies on c_3, where c_1, c_2, c_3 is the great circle carrying the arc a_1, a_2, a_3 respectively. When $R(p_1)$ lies on a_1, $R(p_2)$ lies on a_2 and $R(p_3)$ lies on a_3, tvt() returns "true", else it returns "false". The time complexity of tvt() is constant. Notice that the rotation returned by the call tvt(p1, p2, p3, a1, a2, a3, R), is the same as the rotation returned by the calls tvt(p1, p3, p2, a1, a3, a2, R), tvt(p2, p1, p3, a2, a1, a3, R), tvt(p3, p1, p2, a3, a1, a2, R), tvt(p3, p2, p1, a3, a2, a1, R) and tvt(p2, p3, p1, a2, a3, a1, R). That is because the the order of the statements "$R(p_1)$ lies on a_1, $R(p_2)$ lies on a_2, $R(p_3)$ lies on a_3" is irrelevant. In the implementation this observation may be used to gain a factor of six. Now calculating $\sigma_1(A, B)$ consists of running through all triples of points in $SDR(A)$ and all triples of arcs in $SDR(B)$, to calculate for every combination the rotation R, and to evaluate σ_1 for every valid R. The maximum value is the similarity measure $\sigma_1(A, B)$. Assuming that $SDR(A)$ and $SDR(B)$ have been calculated, this results in the following algorithm outline, called algorithm1.

```
for all points p1    // of SDR(A)
  for all points p2 > p1
    for all points p3 >  p2
      for all arcs a1    // of SDR(B)
        for all arcs a2
          for all arcs a3
            if (tvt(p1, p2, p3, a1, a2, a3, R)){
               sigma1=Vol(A)^{2/3} Vol(B)^{1/3}/Vol(R(A),R(A),B)
               if(sigma1>sigma1_max){sigma1_max=sigma1}
            }
return sigma1_max;
```

In the implementation it is assumed that the arcs and points are stored in a linearly ordered data structure. In this data structure, the variable p1 runs through all points, the variable p2 runs through all points greater than p1, and the variable p3 runs through all points greater than p2. In this way irrelevant permutation evaluations are avoided.

The time complexity of algorithm1 is easily derived. We assume that A and B are approximately of the same complexity, i.e. have approximately the same number of vertices, edges and faces. We denote the number of faces of A and B as f, the number of edges of A and B as e. So, the number of points in $SDR(A)$ equals f, and the number of arcs in $SDR(B)$ equals e. Because e is proportional to f, the inner loop is evaluated $O(f^6)$ times. For polyhedra of small and medium complexity the time consumption of tvt() by far exceeds the timeconsumtion of calculating the mixed volume, so the time complexity of the complete algorithm is $O(f^6)$.

3 Using geometric inequalities to skip orientations

As explained before, the function tvt() calculates a rotation R with the property that $R(p_1)$ lies on a_1, $R(p_2)$ lies on a_2 and $R(p_3)$ lies on a_3. However, without calling tvt(), it is possible to detect cases where no such R exists. As an example, let us look at two points p_1 and p_2 with a spherical distance d(p_1, p_2), and at two arcs a_1 and a_2, where dmin(a_1, a_2) and dmax(a_1, a_2) are the minimal and maximal distance between the arcs. Here, dmin(a_1, a_2) is defined as the minimum distance of the points q_1 and q_2 where q_1 lies on a_1 and q_2 lies on a_2, i.e., dmin(a_1, a_2) $\equiv \{\min(d(q_1, q_2))|q_1 \text{ on } a_1, q_2 \text{ on } a_2\}$. Dmax($a_1$,$a_2$) is defined analogously. Obviously, only when dmin(a_1, a_2) $\leq$ d(p_1, p_2) $\leq$ dmax(a_1, a_2), p_1 can lie on a_1 while at the same time p_2 lies on a_2, see figure 3. This observation may be used to skip calls of tvt(). Of course, the same principle may be used for the other two pairs of points and arcs, i.e, tvt() should only be called when

$$\text{dmin}(a_1, a_2) \leq d(p_1, p_2) \leq \text{dmax}(a_1, a_2) \text{ and} \tag{7}$$

$$\text{dmin}(a_2, a_3) \leq d(p_2, p_3) \leq \text{dmax}(a_2, a_3) \text{ and} \tag{8}$$

$$\text{dmin}(a_3, a_1) \leq d(p_3, p_1) \leq \text{dmax}(a_3, a_1). \tag{9}$$

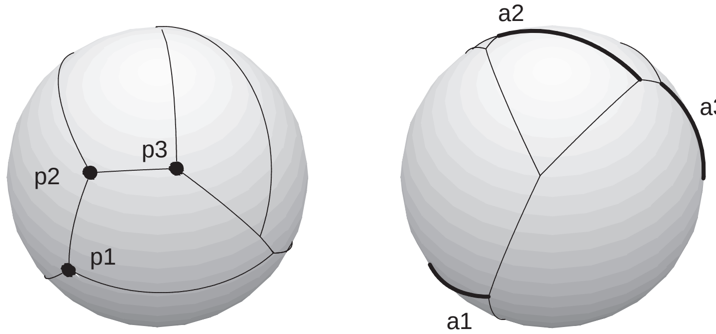

Fig. 3. (a): $SDR(A)$ with three marked points p_1, p_2, p_3. (b): $SDR(B)$ with three marked arcs a_1, a_2, a_3. $SDR(A)$ may be rotated so that in the overlay $R(p_2)$ lies on a_2 and $R(p_3)$ lies on a_3, but clearly then $R(p_1)$ can not lie on a_1.

In the implementation we calculate the distance between all pairs of points of $SDR(A)$ in a preprocessing phase, and store these distances in a table indexed by two points. In the same way we store the minimal and maximal distance between all arcs in $SDR(B)$ in tables indexed by two arcs. Now we can give algorithm2.

```
fill_distance_tables()
for all points p1        // of SDR(A)
  for all points p2 > p1
```

```
    for all points p3 > p2
     for all arcs a1      // of SDR(B)
       for all arcs a2
         for all arcs a3
          if   (dmin(a1, a2)  <= d(p1,p2) <= dmax(a1, a2) and
                dmin(a2, a3)  <= d(p2,p3) <= dmax(a2, a3) and
                dmin(a3, a1)  <= d(p3,p1) <= dmax(a3, a1)){
                if (tvt(p1, p2, p3, a1, a2, a3, R)){
                  sigma1=Vol(A)^{2/3} Vol(B)^{1/3}/Vol(R(A),R(A),B)
                  if(sigma1>sigma1_max){sigma1_max=sigma1}
                }
          }
return sigma1_max;
```

Obviously, the number of calls of tvt() in algorithm2 is less than the number of calls in algorithm1. Moreover, as the complexity of B increases, the arcs in $SDR(B)$ get smaller. The smaller the arcs, the smaller the range of distances between them and thus the smaller the probability that a pair of points will fit between them, resulting in a higher probability that combinations are skipped. I.e. the improvement is not simply a constant factor but is stronger for more complex polyhedra. In the following section we present the experimental time complexity of algorithm1 and algorithm2 , and we derive a first order approximation of the time complexity of algorithm2.

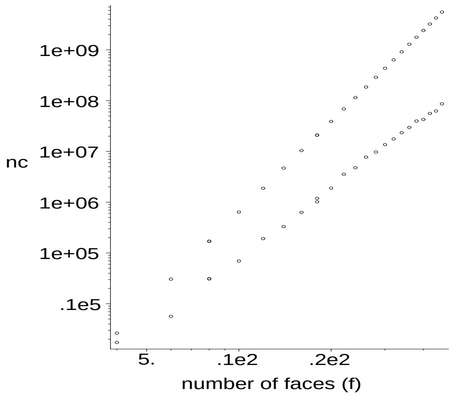

Fig. 4. The results of the experiments with algorithm1 (upper dots) and algorithm2 (lower dots), plotted logarithmically on both axes. In this plot, the results of both algorithms are linear, indicating that the time complexity is of the form $nc = f^e$, where nc is the number of calls of tvt(), f is the number of faces of A and B, and e the exponent. A least square fit in this plot gives $e = 6.0$ for algorithm1, and $e = 4.57$ for algorithm2. So, the experimental time complexity of algorithm1 is $O(f^6)$ and of algorithm2 $O(f^{4.57})$.

4 Results and complexity analysis

We tested algorithm1 and algorithm2 on randomly generated polyhedra, ranging in complexity from 4 to 46 faces. The polyhedra A and B were of the same complexity in terms of the number of faces, edges and vertices. To generate polyhedra of the same complexity, for every test we generated a random polyhedron A, and generated random polyhedra until a polyhedron was found with the same number of faces, edges and vertices as A. This polyhedron was assigned to B. For the pair A, B we used algorithm1 and algorithm2 to determine the number of calls of tvt(). In figure 4 the logarithm of the number of calls nc of tvt() is plotted as a function of the logarithm of the number of faces. From this plot it can be seen that algorithm2 is significantly faster than algorithm1. For polyhedra with 10 faces algorithm2 is $\approx$ 10 times faster than algorithm1, and for polyhedra with 46 faces it is $\approx$ 60 times faster, see figure 5. In the log-log plot, the results of algorithm1 and algorithm2 are both linear, indicating that both algorithms have a time complexity of the form $nc = a.f^e$ where nc is the number of calls of tvt(), f the number faces, and a and e constants. Fitting a line through these points with a least squares method gives for algorithm1 e=6, and for algorithm2 e=4.57. So, the experimental time complexity of algorithm1 is $O(f^6)$ and of algorithm2 $O(f^{4.57})$.

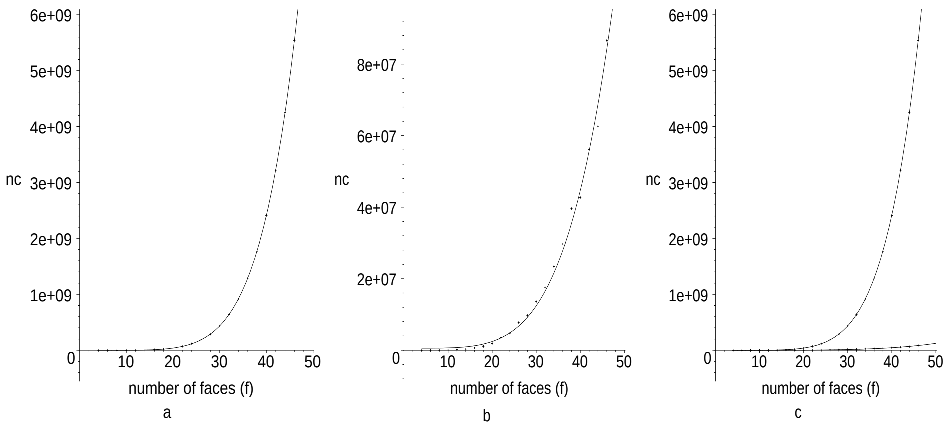

Fig. 5. a: The results of algorithm1. The measured performance is represented by points, and the function f^6 is given as a curve. b: The results of algorithm2. The measured performance is represented by points, and the function $f^{4.57}$ is given as a curve. c: The graphs from a and b in one figure, showing the difference in performance of algorithm1 and algorithm2 on a linear scale.

Now we will derive a first order approximation of the time complexity of algorithm2. The polyhedra used in our experiments are random polyhedra. For these polyhedra it holds that the number of edges is proportional to the number of vertices, and the number of faces is proportional to the number of vertices. In the slope diagram a similar property holds: the number of points, arcs and faces are proportional to each other. In the derivation we use the easily verified fact that the average arc length in the slope diagram is proportional to $\frac{1}{\sqrt{(f)}}$.

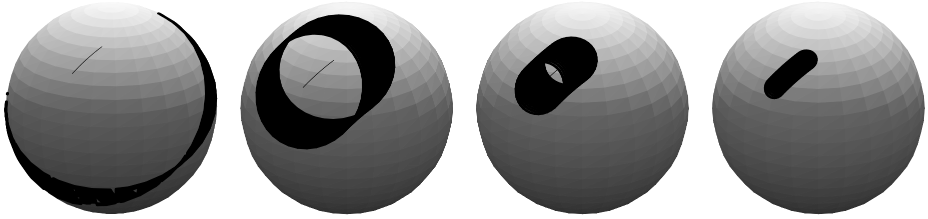

Fig. 6. Four spheres with an arc a1, and regions (black) consisting of all points with a spherical distance d of 1.5, 0.58, 0.25, 0.1 respectively to a1. In the last two figures the arc is (partially) covered by the black area. When a second arc a2 (not shown) has no point in common with the black region then there are no points q1∈ a1 and q2∈ a2 with the property that d(q1, q2)=d. I.e., for two points p1 and p2 with d(p1, p2)=d, the call tvt(p1,p2,..,a1,a2,..) will return false. So, for this situation the call may be skipped.

Let us now look at the following situation. On the unit sphere we draw a small arc $a1$ with length $|a1|$, and we draw the region consisting of all points with a distance d to $a1$. See figure 6. The region consists of a belt with an average width proportional to $|a1|$. So, the area of the belt is proportional to $|a1|$, which is proportional to $\frac{1}{\sqrt{(f)}}$. Now we draw an arc $a2$ on the sphere at a random place. Because A and B are approximately of the same complexity it holds that $|a1| \approx |a2|$. When $a2$ has no point in the black region there are no points $q1 \in a1$ and $q2 \in a2$ with the property that d($q1, q2$)=d. I.e., for two points p1 and p2 with d(p1, p2)=d, the call tvt(p1,p2,..,a1,a2,..,R) will return false. So, for this situation the call of tvt() may be skipped. The number of arcs in a slope diagram is $\propto f$, and the area of the belt is $\frac{1}{\sqrt{(f)}}$, so, for a given arc $a1$ the number of arcs lying (partially) in the belt is $\propto \sqrt{(f)}$. Therefore, summed over all arcs $a1$, the number of calls of tvt() $\propto f^{1.5}$. More precise, when a1 and a2 run through all arcs of $SDR(B)$, and p1 and p2 run through all points in $SDR(B)$, then the number of times that it holds that $dmin(a1, a2) \leq d(p1, p2) \leq dmax(a1, a2)$ is proportional to $f^{1.5}$. This holds for one pair of points and one pair of arcs. Algorithm2 runs through three pairs of arcs and points, so the the number of calls of tvt() will be proportional to $(f^{1.5})^3 = f^{4.5}$ This agrees reasonably well with our experimental result of $f^{4.57}$. That our experimental result differs slightly from the theoretical result may be caused by the fact that in our experiments we also used some polyhedra with few faces. In figure 4 it can be seen that the slope of the curve of algorithm2 decreases slightly for more complex polyhedra. Leaving out in this figure the first ten data points and fitting a line to the remaining points gives an exponent of 4.52. So, for polyhedra of medium complexity, our experimental complexity corresponds very well with the theoretical complexity.

5 Discussion and conclusion

The method presented in this article reduces the number of relative orientations to be considered by using geometric distance inequalities. When, for a given set of arcs a_1, a_2, a_3 and points p_1, p_2, p_3, the distance inequalities are not fulfilled then there is no rotation R such that $R(p_1)$ lies on a_1, $R(p_2)$ lies on a_2 and $R(p_3)$ lies on a_3. However, when all three inequalities are fulfilled that does not mean such a rotation exists. That is because, for example, the angle defined by the points $p1, p2, p3$ does not correspond with the range of angle defined by the arcs $a1, a2, a3$. Analogous to the distance inequalities, we can give three angle inequalities. We expect that, by combining distance inequalities with angle inequalities, we can reduce the time complexity of Minkowski-sum based similarity calculations even further.

In this article we presented a method to speed up the search for the relative orientation that minimizes the mixed volume of two convex polyhedra. However, because the volume of the Minkowski sum consists of two mixed volumes (and the orientation independent volumes V(A) and V(B)), the method may also be used to speed up the search for the relative orientation that minimizes the volume of the Minkowski sum. I.e. it may be used for speeding up Minkowski sum based similarity calculations in general.

Literature

[1] Veltkamp, R.C. Shape Matching: Similarity Measures and Algorithms. *Shape Modeling International 2001: 188-196*

[2] Heijmans, H. J. A. M., and Tuzikov, A. Similarity and symmetry measures for convex shapes using Minkowski addition. *IEEE Trans. Patt. Anal. Mach. Intell. 20*, 9 (1998), 980–993.

[3] Tuzikov, A. V., Roerdink, J. B. T. M., and Heijmans, H. J. A. M. Similarity measures for convex polyhedra based on Minkowski addition. *Pattern Recognition 33*, 6 (2000), 979–995.

[4] Roerdink J.B.T.M. ,Bekker H. Similarity measure computation of convex polyhedra revisited. *LNCS vol. 2243, (2001) Springer Verlag.*

[5] Bekker, H., and Roerdink, J. B. T. M. Calculating critical orientations of polyhedra for similarity measure evaluation. In *Proc. 2nd Annual IASTED International Conference on Computer Graphics and Imaging, Palm Springs, California USA, Oct. 25-27* (1999), pp. 106–111.

[6] Sangwine-Yager, J.R. Mixed volumes. Chapter 1.2 of: *Handbook of convex geometry.* (1993) Eds. Gruber, P.M., Wills, J.M. Elsevier science publishers B.V.