Fast Approximative Data Structures for Applications in the Automotive Industry

Dissertation
zur Erlangung des Grades
„Doktor der Naturwissenschaften“

am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

vorgelegt von
Dominik Will
geboren in Limburg a. d. Lahn

Mainz, Juli 2016

Tag der mündlichen Prüfung: 28. September 2016

D77

Abstract

In the automotive industry digital prototyping becomes more and more important. By digital prototypes many tests can be done virtually, the digital design process can be adapted dynamically and thus costs can be reduced. The basic algorithms used in this context are always the same and mainly concern the calculation of distances and penetrations.

In this work an approach will be presented how distances and penetrations can be calculated approximately very fast. This approach will be applied to pack arbitrarily shaped objects into a trunk, which is one of many problems arising in the digital design process. Because the geometry describing a trunk is usually a "soup" of triangles which might even contain large holes, at first a robust heuristic will be presented how for such input data a voxel representation of the interior of a trunk can be computed.

Another field where distance calculations play an important role are checks concerning the engine. By data generated during a test drive and consisting of rigid transformations describing the vibration movement of an engine it can be checked for the digital model of the engine that a certain safety distance to other parts is maintained.

A new approach will be presented how the concept of bounding volumes, which is only applied to 3-dimensional geometry so far, can be transferred to rigid transformations. By this approach the minimum distance between the engine and the surrounding parts, that occurs throughout a test drive, can be calculated very fast. Furthermore, by this approach the course of the distance between engine and surrounding parts during the test drive can be calculated approximately very fast as well.

The motion data of an engine during a test drive is collected by motion tracking systems and several types of such systems have become available at low costs in the last years. Therefore, finally an approach will be shown how data collected by several motion tracking systems fixed to the same object can be combined and put into geometric relation to each other allowing a comparison of the recorded data.

Zusammenfassung

In der Automobilindustrie erlangt das Digital Prototyping eine immer größere Bedeutung. Anhand digitaler Modelle können viele Tests virtuell durchgeführt, der Designprozess dynamisch angepasst und dadurch die Kosten reduziert werden. In diesem Kontext kommen immer wieder die gleichen grundlegenden Algorithmen zum Einsatz und betreffen hauptsächlich die Berechnung von Abständen und Durchdringungen.
In dieser Arbeit wird ein Ansatz vorgestellt, wie Abstände und Durchdringungen sehr schnell approximativ berechnet werden können. Dieser Ansatz wird für das Packen beliebig geformter Objekte in einen Kofferraum angewendet, was eine von vielen Problemstellungen ist, die im digitalen Designprozess auftritt. Da die Kofferraum-Geometrie für gewöhnlich eine „Dreieckssuppe“ ist, die sogar große Löcher aufweisen kann, wird zunächst eine robuste Heuristik vorgestellt, mit der für solche Eingabedaten eine Voxel-Repräsentation des Inneren eines Kofferraums berechnet werden kann.
Ein anderes Gebiet, in dem Abstandsberechnungen eine große Rolle spielen, sind Tests, die den Motor betreffen. Anhand von Daten, die während einer Testfahrt aufgezeichnet werden und aus starren Transformationen bestehen, die die Vibrationsbewegung eines Motors beschreiben, kann für das digitale Modell des Motors überprüft werden, dass bestimmte Sicherheitsabstände zu anderen Bauteilen eingehalten werden.
Ein neuartiger Ansatz wird vorgestellt, wie das Konzept von Hüllkörpern, das bisher nur für 3-dimensionale Geometrie verwendet wurde, auf starre Transformationen übertragen werden kann. Mit Hilfe dieses Ansatzes kann der minimale Abstand des Motors zu umgebenden Bauteilen, der während einer Testfahrt auftritt, sehr schnell berechnet werden. Außerdem kann durch diesen Ansatz der Verlauf des Abstandes zwischen Motor und umgebenden Bauteilen während der Testfahrt sehr schnell approximativ berechnet werden.
Die Bewegungsdaten eines Motors während einer Testfahrt werden mit Hilfe von Motion Tracking Systemen aufgezeichnet und in den letzten Jahren sind verschiedene Typen solcher Systeme zu niedrigen Preisen verfügbar geworden. Deshalb wird abschließend eine Vorgehensweise vorgestellt, wie Bewegungsdaten, die von mehreren verschiedenen am gleichen Objekt befestigten Motion Tracking Systemen aufgezeichnet wurden, kombiniert und geometrisch in Beziehung zueinander gesetzt werden können, um einen Vergleich der aufgezeichneten Daten zu ermöglichen.

Acknowledgement

At first I want to thank my advisor for the long-time support and for initiating and managing the industrial cooperations which allowed me to write this work.

Furthermore, I want to thank the second reader of this work.

My special thanks goes to the long-time office mate and friend who proofread this work.

Finally, I want to thank my family for their support making this work possible.

Contents

Introduction

In the context of computer aided design (CAD) and digital mock-up (DMU) in the automotive industry several geometrical problems arise. This covers packing problems, layout problems, checks for safety distance and motion analysis, to name but a few. Because of the increasing number of components and thus short running installation space in new car models, it becomes more and more important to solve these problems during the DMU phase. The algorithms to solve these problems usually use a basic set of fundamental algorithms as workhorses that do most of the work. One of these workhorses is the calculation of distances and penetrations. In this work an industrial strength algorithm to do that and its application to a packing problem will be presented, though it might be applied to many other problems, such as the above-mentioned. As might be known, the size of a trunk is usually not defined by its continuous volume, but there are two standards that define it as the result of packing problems using boxes as packing objects. Firstly, this is the German standard DIN 70020, which is common in Europe and uses equally sized boxes with a volume of one liter, and secondly, the American standard SAE J1100, which is common in the USA and uses a set of seven different boxes with volumes in the range from 5.63 to 67.47 liters.
In the scope of a long-time cooperation with a German car manufacturer a software has been developed which allows the generation of box packings according to DIN 70020 and SAE J1100 by different algorithms, mainly based on Monte-Carlo techniques and contact simulation. However, our industrial partner also expressed the need to not only generate packings of boxes according to DIN 70020 and SAE J1100 but to also pack arbitrarily shaped objects like suitcases, golf bags etc. In fact, the standard SAE J1100 already contains, besides the seven box types, a stylized golf bag which has been left away in the algorithms implemented so far.
In Chapter 2 the above-mentioned algorithm will be presented, together with the necessary data structures, which calculates distances and penetrations approximately very fast. The penetration calculation will be used to pack arbitrarily shaped objects into a trunk by simulated annealing, which is a probabilistic meta-heuristic and can be considered as a standard technique for solving global optimization problems. The main input for simulated annealing is an objective function that is minimized, and it is an obvious idea to use the sum of penetrations of the packing objects and the trunk geometry for that. Because simulated annealing is a Monte

Carlo approach making a random walk through the configuration space, it works better, the more configurations are calculated. Thus, it is much more important for the objective function to be quickly evaluable than to deliver exact values.
The base of the presented data structures and algorithms is a voxelization of the packing objects and the trunk geometry. However, the data delivered by our industrial partner consists of "soups" of triangles which are usually no watertight 2-manifold meshes and thus no volume representation that can be easily voxelized. Therefore in chapter 1 at first a heuristic for generating voxelizations for such input data will be presented. By using flow techniques this heuristic is robust against even large holes in the input geometry and by using an octree approach it is fast and memory efficient enough to reach millimeter resolution and below for even large trunks within a reasonable running time.

Another large field in the automotive industry besides packing problems are geometrical tests related to the engine. Thus, our industrial partner makes test drives with new car models and measures the movement of the engine in its compartment during extreme driving maneuvers. By this data it is checked for the digital model of the engine and the compartment that fixed and moving parts keep a certain safety distance. For this application a software has already been developed to calculate the distance between fixed and moving parts for every sampling point of the movement measurement by using the standard technique of bounding volumes.
By transferring the idea of bounding volumes for 3-dimensional objects to rigid transformations in chapter 3 an approach will be shown how the sampling point of the movement measurement can be found extremely fast for which the minimum distance throughout all measured sampling points occurs. Furthermore, we will show how this approach can be used to calculate an approximate course of the distance with conservative error bounds which is much faster than calculating the exact distance for every sampling point separately.
To generate the motion data of the engine our industrial partner uses an optical tracking system. Since different types of tracking systems have become cheaper and cheaper and more and more precise in the last few years, it is an obvious idea to combine or to compare the data of several tracking systems. Therefore in chapter 4 an iterative algorithm to combine the values of three or more independent single point trackers will be presented, that are fixed to one object, in order to robustly calculate the pose of this object. Furthermore, an approach will be presented how for poses that are measured by two independent tracking systems and "live" in different coordinate systems, the transformations between the two coordinate systems fixed to the tracking sensors and the two coordinate systems fixed to the tracking markers can be calculated. This approach is mathematically equivalent to an existing approach but needs less computational effort and gives an amazing geometric view of the problem beyond the formulas. Additionally, by means of a RANSAC approach an ambiguity concerning rotation quaternions will be overcome which seems to have not been treated properly in previous publications using rotation quaternions to solve this problem.

Chapter 1

Inner Space Detection

The CAD data provided by our industrial cooperation partner consists of triangle sets which usually neither form a connected surface nor have a consistent orientation regarding what is inside and what is outside of a car trunk or component part. Therefore we want to speak about "triangle soups" in the following. Because we need a volume representation of the interior of a trunk and packing objects like suitcases for the packing algorithm that will be presented in chapter 2 of this work, we need a robust heuristic to determine the interior and exterior of a surface geometry described by a triangle soup.
For that a technique already presented in a former work [1] has been refined. Besides this, there seems to be no other work on this problem. Many algorithms for mesh reconstruction from point clouds exist, like the Poisson surface reconstruction [2], but such algorithms could be applied in our case only for closing very small holes and not such big ones as shown in figure 1.1.
The basic idea of the former work [1] is to find enclosed cells in a coarse voxelization of the geometry. These cells are defined as inside and then in a finer voxelization these inner cells are separated from the outside by a minimum cut resp. maximum flow calculation, hereinafter briefly denoted as min cut and max flow.
For that the cells are considered as a network and the inner cells from the coarser voxelization are considered as source and the cells on the outer surface of the voxelization as sink. Naturally, in this network every cell is a vertex and every cell is connected to its six neighbors by an edge with capacity 1. However, this approach only works for quite coarse target resolutions, which was 50 mm in that work. For higher resolutions targeted in the scope of this work, this approach would lead to a too long running time for the min cut calcuation and a too high memory consumption for storing a number of cells which is cubic in the resolution.
Therefore this technique has been enhanced. Since this enhancement involves some nested max flow min cut calculation, some graph theoretical basics are necessary.

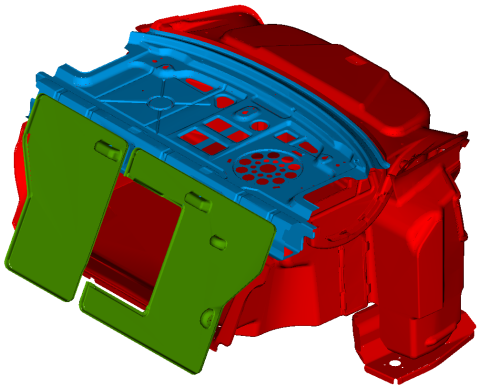


Figure 1.1: The trunk geometry described by the CAD data provided by our industrial partner can contain large holes, as in the rear bench (green) of the depicted model, or many small holes, as in the trunk's ceiling sheet metal (blue).

1.1 Graph Theoretical Preliminaries

In the following always a network $N = (V, E)$ with a capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$ is considered. To meet our requirements we want to take not only single vertices but whole subsets S and T of the vertex set V as source and sink, which shall always imply $S \cap T = \emptyset$. This can be reduced to the usual case of one element source and sink by contracting S and T to single vertices. Contracting a subset $A \subset V$ to a single vertex a is done by considering the network $\tilde{N} = (\tilde{V}, \tilde{E})$ with

$$\tilde{V} := V' \cup \{a\} \quad , \quad V' := V \setminus A$$

$$(v_1, v_2) \in \tilde{E} :\Leftrightarrow \begin{cases} (v_1, v_2) \in E & \text{if } v_1 \neq a \,,\ v_2 \neq a \\ (A \times V') \cap E \neq \emptyset & \text{if } v_1 = a \,,\ v_2 \neq a \\ (V' \times A) \cap E \neq \emptyset & \text{if } v_1 \neq a \,,\ v_2 = a \end{cases}$$

and the capacity mapping $\tilde{c} : \tilde{E} \mapsto \mathbb{R}_{\geq 0}$ with

$$\tilde{c}(v_1, v_2) := \begin{cases} c(v_1, v_2) & \text{if } v_1 \neq a \,,\ v_2 \neq a \\ \sum_{a' \in A} c(a', v_2) & \text{if } v_1 = a \,,\ v_2 \neq a \\ \sum_{a' \in A} c(v_1, a') & \text{if } v_1 \neq a \,,\ v_2 = a \end{cases}$$

and the usual convention $c(v_1, v_2) = 0$ if $(v_1, v_2) \notin E$.
Furthermore we want to use the following notations for a network (V, E) with capacity mapping c:

$$\begin{array}{ll} \overline{A} := V \setminus A & \text{for all } A \subset V \\ c(A, B) := \sum_{a \in A, b \in B} c(a, b) & \text{for all } A, B \subset V \\ c(A) := c(A, \overline{A}) & \text{for all } A \subset V \end{array}$$

At first we want to show the following fundamental lemma:

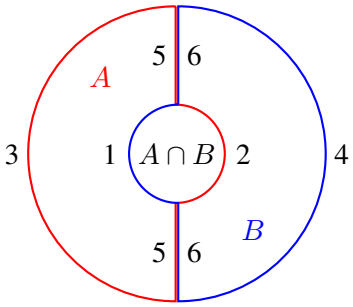


Figure 1.2: The boundary between two sets A and B, their intersection and the complement can be divided in 5 elementary parts if $A \cap B \neq \emptyset$, $A \not\subset B$ and $B \not\subset A$.

Lemma 1.1 *Let $S_A, T_A \subset V$ be source and sink in a network (V, E) with capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$ and let $(A, \overline{A})$ be a corresponding S_A-T_A min cut. That means $S_A \subset A$, $T_A \subset \overline{A}$ and $c(A, \overline{A})$ is minimum. Furthermore let $S_B, T_B \subset V$ be another source and sink and $(B, \overline{B})$ a S_B-T_B min cut with $S_A \subset S_B$ and $A \cap T_B = \emptyset$. Besides, A shall be minimum concerning set inclusion. That means $(A', \overline{A'})$ is not a S_A-T_A min cut for any true subset $A' \subsetneq A$. Then A must be a subset of B.*

Proof:
Let us assume that $A \not\subset B$. In this case B cannot be a subset of A either because otherwise $(B, \overline{B})$ would be obviously also a S_A-T_A min cut in contradiction to the minimality of A concerning subsets. Because of $S_A \subset A$ and $S_A \subset S_B \subset B$ the intersection of A and B cannot be empty. Thus the situation is described by figure 1.2. The boundary between A and B, their intersection and the complement can be divided in 5 elementary parts. We define the following six sums of capacities of edges crossing these parts (for part 5/6 in both directions):

$$\begin{array}{ll} c_1 := c(A \cap B, A \setminus B) & c_2 := c(A \cap B, B \setminus A) \\ c_3 := c(A, \overline{A \cup B}) & c_4 := c(B, \overline{A \cup B}) \\ c_5 := c(B \setminus A, A \setminus B) & c_6 := c(A \setminus B, B \setminus A) \end{array}$$

Because $(A, \overline{A})$ is a S_A-T_A min cut and because of $S_A \subset A \cap B$ it is

$$c_2 + c_3 + c_6 = c(A) < c(A \cap B) = c_1 + c_2 \quad \Rightarrow \quad c_3 + c_6 < c_1 \,. \tag{1.1}$$

The inequality must be strict because of the minimality of A concerning subsets since in the equality case $(A \cap B, \overline{A \cap B})$ would also be a S_A-T_A min cut.
Because of $A \cap T_B = \emptyset$ and $T_B \subset \overline{B} \Rightarrow B \cap T_B = \emptyset$ it must be $(A \cup B) \cap T_B = \emptyset$. Since $(B, \overline{B})$ is a S_B-T_B min cut and $S_B \subset B \subset A \cup B$, it follows in a similar way as in (1.1):

$$c_1 + c_4 + c_5 = c(B) \leq c(A \cup B) = c_3 + c_4 \quad \Rightarrow \quad c_1 + c_5 \leq c_3 \tag{1.2}$$

Adding inequations (1.1) and (1.2) leads to $c_5 + c_6 < 0$ what is obviously a contradiction because of $c_5, c_6 \geq 0$.

□

Definition 1.2 *In a network (V, E) with capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$ we define as the min cut component $\mathrm{MCC}(S, T)$ for source $S \subset V$ and sink $T \subset V$ the smallest superset U of S so that $(U, \overline{U})$ is a S-T min cut:*

$$\mathrm{MCC}(S, T) := \arg\min_{U \subset V} \left\{ |U| \,\middle|\, (U, \overline{U}) \text{ is } S\text{-}T \text{ min cut} \right\}$$

We want to call $\mathrm{MCC}(S, T)$ the min cut component of the source.

Proof:
$\mathrm{MCC}(S, T)$ is well-defined. Suppose that there are two smallest supersets U and U' of S with $|U| = |U'|$. Then lemma 1.1 can be applied twice for $S_A = S_B = S$ and $A = U$, $B = U'$ resp. $A = U'$, $B = U$ and it follows $U \subset U'$ and $U' \subset U \Rightarrow U = U'$.

□

Lemma 1.3 *Let (V, E) be a network with capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$, source $S \subset V$ and sink $T \subset V$ and let $f : E \mapsto \mathbb{R}_{\geq 0}$ be a max flow from S to T. Then $\mathrm{MCC}(S, T)$ is S extended by the from S reachable vertices in the residual network for f.*

Proof:
According to the max-flow min-cut theorem the set U of from S reachable vertices in the residual network and its complement are a S-T min cut. According to lemma 1.1 U must be a superset of $\mathrm{MCC}(S, T)$. But according to the max-flow min-cut theorem $\mathrm{MCC}(S, T)$ cannot have outgoing edges as S-T min cut in the residual network. Hence $\mathrm{MCC}(S, T)$ cannot be a subset of U and thus $\mathrm{MCC}(S, T) = U$ must hold.

□

Annotation 1.4 *If (V, E) is an undirected graph with capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$, source $S \subset V$, sink $T \subset V$ and max flow $f : E \mapsto \mathbb{R}_{\geq 0}$ from S to T, then $\mathrm{MCC}(T, S)$ is T extended by the from T reachable vertices in the residual network for f with flipped edges because the max flow problem is symmetric for undirected graphs. Hence we want to call $\mathrm{MCC}(T, S)$ in this case briefly the min cut component of the sink.*

Lemma 1.5 *Let (V, E) be a network with capacity mapping $c : E \mapsto \mathbb{R}_{\geq 0}$ and T_1, T_2 and T_3 three pairwise disjoint subsets of V. Then it is*

$$\mathrm{MCC}(T_1, T_2 \cup T_3) \subset \mathrm{MCC}(T_1 \cup T_2, T_3) \,.$$

Proof:
Because it is obviously $\mathrm{MCC}(T_1, T_2 \cup T_3) \cap T_3 = \emptyset$, we can apply lemma 1.1 with $A = \mathrm{MCC}(T_1, T_2 \cup T_3)$, $B = \mathrm{MCC}(T_1 \cup T_2, T_3)$, $S_A = T_1$ and $S_B = T_1 \cup T_2$.

□

1.2 Enhanced Inner Space Detection

For inner space detection it is started at first with the bounding box of the geometry as "1-cell-voxelization" of the geometry which is then refined by doubling the resolution resp. halving the edge length of voxels. After each refinement cells on the boundary of the bounding box containing no geometry are flagged as outside as well as all cells connected to these cells, which can be found by a depth first search (DFS) or a breadth first search (BFS). Cells containing no geometry we want to call *empty cells* and cells containing geometry *geometry cells* in the following. Cells flagged as outside are not refined in the next refinement step and refined empty cells, that are connected to outside cells or the boundary, are flagged as outside as well. This refinement is repeated until empty cells are found which are not connected to outside cells. The connected components of these cells are determined, to each component an ID is assigned and the assigned IDs are stored in the according cells.
In the second part of this heuristic geometry cells are again iteratively refined. The basic idea is now again to separate the outer cells, which we want to refer to as outer component, from all other components identified in the first part of the heuristic by a min cut calculation. But it is not clear that all those other components are really some kind of "inner" component inside the car trunk since there might be some concave regions on the surface of the trunk with empty cells completely enclosed by geometry cells on a coarse refinement level, which will be connected to the outer component on a finer voxelization level. For that reason all components are treated equally and each component is to be separated from any other component including the outer component.
That is done by taking one component as source and all others as sink, calculating a min cut resp. a max flow and assigning all cells that are reachable from the source component in the residual network to this source component. That is done for all components. More formally we want to calculate $\mathrm{MCC}(S_i, T_i)$, $T_i := \bigcup_{j \neq i} S_j$, $0 \leq i \leq n$, if we denote the outer component as S_0 and the inner components as $S_1, \ldots, S_n$, and then extend S_i to $\mathrm{MCC}(S_i, T_i)$ for the next refinement step. Because our graph is undirected and because of annotation (1.4), $\mathrm{MCC}(T_i, S_i)$ is obviously disjoint from $\mathrm{MCC}(S_i, T_i)$. But because of lemma 1.5 $\mathrm{MCC}(T_i, S_i)$ is a superset of $\mathrm{MCC}(S_j, T_j)$ for $j \neq i$. Hence all $\mathrm{MCC}(S_i, T_i)$ are pairwise disjoint showing that the procedure described at the beginning of this paragraph makes sense at all.

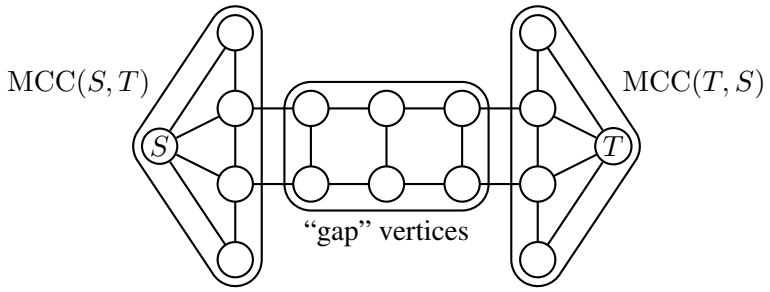


Figure 1.3: Besides $\mathrm{MCC}(S,T)$ and $\mathrm{MCC}(T,S)$ in an undirected graph there usually remain "gap" vertices.

The next question that arises is if S_i can be extended to $\mathrm{MCC}(S_i,T_i)$ right after each flow calculation or if this can be done only after all flow calculations are finished. The former would be preferable since memory for storing every $\mathrm{MCC}(S_i,T_i)$ could be saved and the following flow calculation would become faster because extending S_i to $\mathrm{MCC}(S_i,T_i)$ means that a bigger number of cells is contracted to one vertex for the flow calculation, i.e. the flow network becomes simpler. And in fact this can be done since according to annotation (1.4) $\mathrm{MCC}(S_i,T_i)$ is a subset of the cells reachable from T_j, $j \neq i$, in the residual network after calculating a max flow from S_j to T_j if all edges are flipped. But it should be clear that contracting such a subset to a single vertex does not really affect the max flow min cut calculation.
In general the graph is not completely partitioned into $\mathrm{MCC}(S_i,T_i)$, $0 \leq i \leq n$, but there remain some kind of "gap" cells in between as illustrated in figure 1.3 for the two component case. Actually such gap cells are welcome because we do not want neighbored cells to belong to different components until the finest voxelization level. Especially at slant boundaries this would be an undesirable behavior since then these slant boundaries would stay voxelized at a coarse voxelization level.
However, after calculating $\mathrm{MCC}(S_i,T_i)$ and assigning them to the i-th component there might be neighbored cells belonging to different components. For that reason the current voxelization level is scanned for such cells and they are changed to gap cells. Furthermore this "gap component" is extended by a 1-voxel layer of cells on the level of currently finest voxels in order to reduce the already mentioned effect that slant boundaries might stay voxelized at a coarse voxelization level.
When all the flow calculations to assign new cells on the finest voxelization level to one of the components or to flag them as gap cells are finished, there might still be unassigned cells enclosed by geometry cells. The connected components of such cells are considered to be new inner components and therefore appropriate IDs are assigned to their cells. When reaching very fine voxelization levels of 1 mm and below the number of these new components tends to explode to thousands because there are usually many small enclosed cavities in common geometries. Because thousands of components would lead to an unacceptable running time for the flow

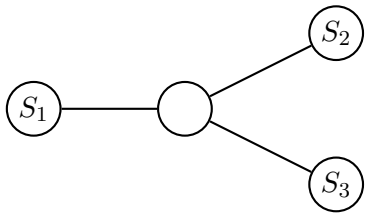


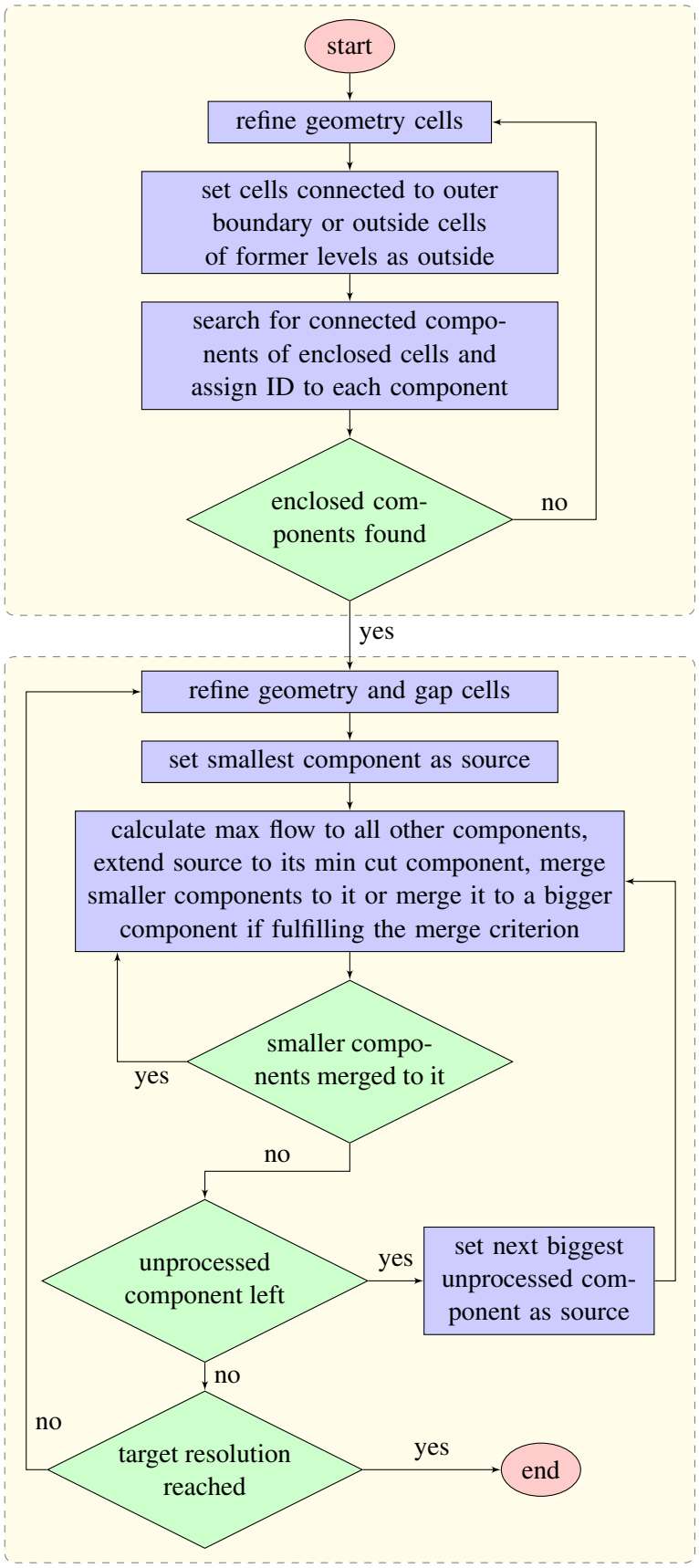
Figure 1.4: A max flow from S_1 to S_2 and S_3 in this graph could run completely to S_2 or S_3 if all edges have the same capacity. Thus the max flow distribution over several sinks is not unique in general.

calculation and we are usually not interested in small cavities, below a certain voxelization level only new components are accepted if their volume exceeds an adjustable threshold. Otherwise these cells are considered to be geometry cells. Components of former voxelization levels that are also under this threshold are switched to geometry cells, too.
After a voxelization level has been processed in this way, besides geometry cells all gap cells are refined. Hence the flow calculation at the next voxelization level is done on refined gap subcells as well as refined geometry subcells that do not contain geometry. Only sources and sinks consist of bigger cells from former voxelization levels.
As already mentioned, it is a common scenario that there are concave regions on the geometry surface with enclosed cells on a coarse voxelization level leading to separate components. In order to avoid treating these cells as separate components until the very finest voxelization level a merging step for components after every max flow min cut calculation is introduced. To decide if the source component is to be merged with a sink component, the flow to this component is used, which is recorded during the max flow min cut calculation. A sink component might be merged with the source component if it fulfills the criterion

$$\text{flow}^3 \geq \text{threshold} \cdot \min\{\text{source component size}, \text{sink component size}\}^2 \ . \quad (1.3)$$

The exponents are inspired by the fact that the size can be considered as volume and the flow can be interpreted as an area the flow runs through. As threshold 1 is used, inspired by the simple situation that a cube shaped component should be merged if there is a flow corresponding to the area of one of its sides. However, it should be noticed that it is ambiguous how the total max flow is distributed over the sink components as illustrated by figure 1.4, even if nearer sinks are preferred. But as the whole inner space detection is a heuristic, we accept this ambiguity.
Besides, there is another ambiguity anyway when starting to merge components just after each flow calculation because now the order in which the components are processed makes a difference. By “process a component” we want to denote the process of calculating a min cut with this component as source and merging its min cut component to this component. The components are processed in an order sorted by their size starting with the smallest one. After processing a component

it is merged with all smaller and thus already processed components that fulfill the flow criterion (1.3) and then it is processed again. This might happen later if this component has become bigger than another unprocessed component by merging.
If there are no smaller components fulfilling the flow criterion, it is searched for a bigger and thus unprocessed component fulfilling the flow criterion. If there are several ones, that one with the biggest flow to it is merged with the just processed component. The new bigger resulting component will be processed later again according to its size.
The biggest advantage of processing smaller components at first is that in most cases smaller components will be merged with bigger ones an thus a reprocessing is avoided. And if it occurs, it is mostly for smaller components, for which the min cut calculation is much faster than for bigger components.
When the target voxel resolution is reached, one might deviate from the handling of former coarser voxelization levels depending on what is to be detected. If all of the components are of interest, for example in order to detect other inner spaces like wheel recesses in a trunk, one will process components in the same way as for the voxelization levels before. But if one is only interesting in the real packing space of a trunk, then usually only the biggest inner component is of importance. Therefore one can stop after processing the biggest inner component, merge all other inner components with the outer component and save the running time for processing the outer component since the min cut problem is symmetric in an undirected network. Accordingly if one is only interested in the outer space, for example to generate a voxelization of a suitcase given as a holey hull geometry, one can skip the biggest inner component, process the outer component and merge all inner components. However, processing all components except the outer and the biggest inner component is necessary in any case because these components might be merged with the outer or biggest inner component depending on the calculated flow. Furthermore flagging adjacent cells of different components as gap can be omitted on the targeted voxelization level since there is no finer voxelization level, on which the boundary could be better voxelized.
The whole enhanced inner space detection is illustrated in a simplified manner in figure 1.5 by a flow chart. In figure 1.6 it is shown for a test geometry that the inner space detection works in the expected way. In figure 1.7 the inner space detection for a trunk model with a big hole in the rear bench is illustrated by showing different voxelization levels. As it can be seen, even such big holes do not disturb the inner space detection strongly.



Part 1: Search for enclosed components on coarsest voxelization level

Part 2: Iteratively refine and separate components by min cut calculation

Figure 1.5: Heuristic for detecting inner space illustrated as flow chart in a simplified manner.

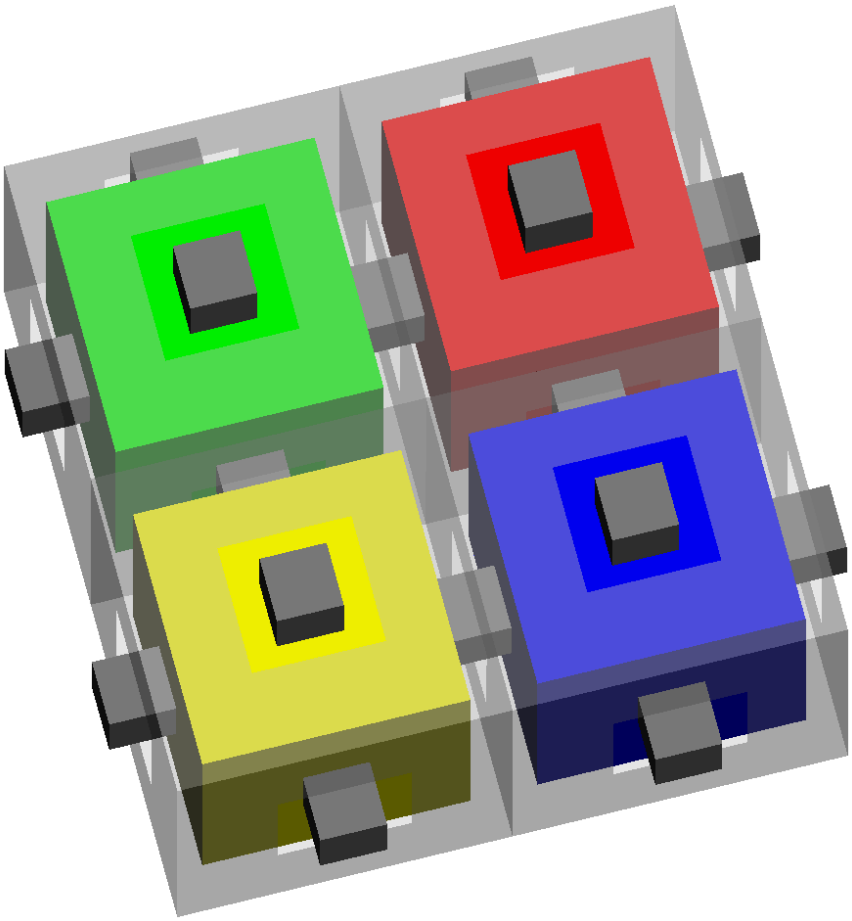


Figure 1.6: Inner space detection showed for a test geometry that is drawn transparent. Gap cells are colored gray. The test geometry consists of four cubes with a an edge length of 10 voxel units and a 4×4 sized hole in the center of each of their faces. The four found inner components are differently colored. In the last step of the inner space detection, that is not yet done, these four components would be merged together including the gap cells in between.

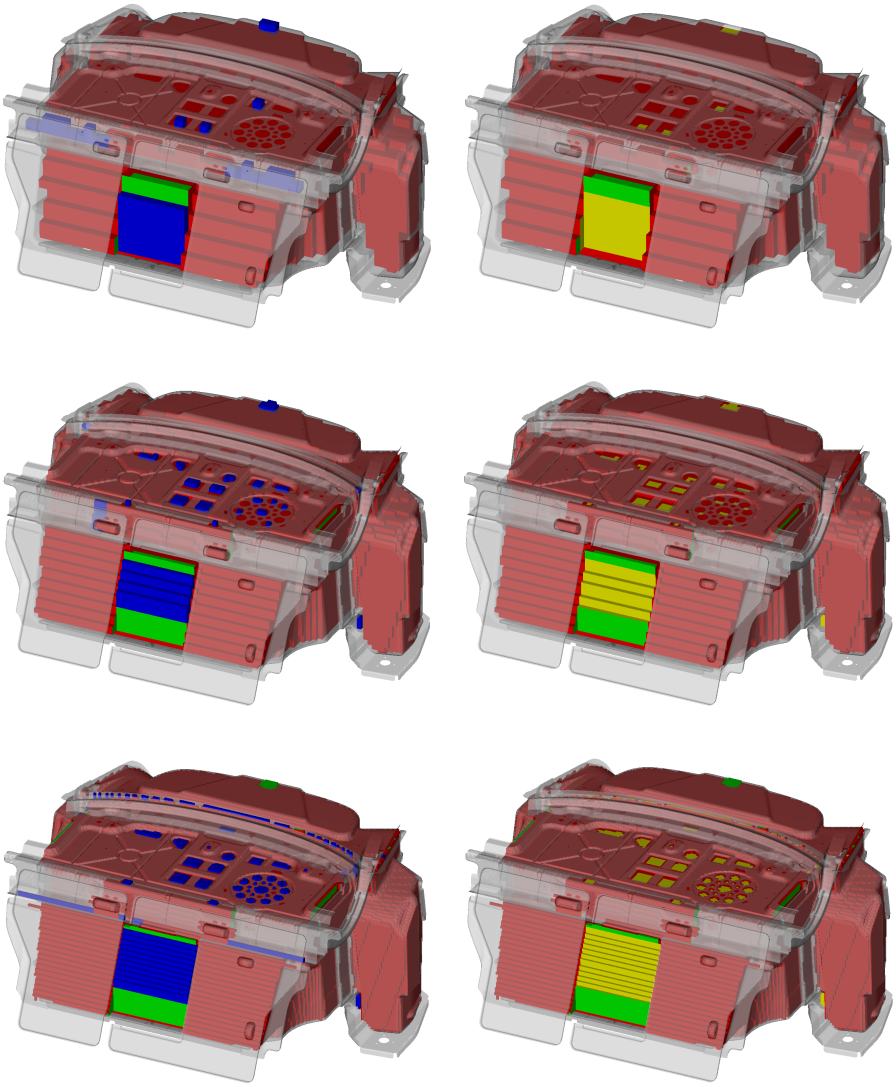


Figure 1.7: Inner space detection for a trunk with a big hole in the rear bench and many small holes in the top covering in 3 resolutions (32 mm top, 16 mm middle, 8 mm bottom). The trunk geometry is drawn transparent and inner component cells red. On the left hand side gap cells are drawn blue and the boundary of inner cells to the outside is drawn green. On the right hand side they are omitted, but the boundary faces touching them are drawn yellow.

1.3 Implementation Details

The first step for the inner space detection is to voxelize the bounding geometry, which is a soup of triangles. The voxelization is represented by an octree [3] whose root is the axis aligned bounding box of the geometry, which is enlarged while keeping its center so that its edge length is a power of two of the targeted voxel resolution. At first only the root is instanced and flagged as empty respectively not filled. Then all triangles are added one after another to the voxelization by propagating them through the octree hierarchy. On every level a triangle is checked for collision with the current octree cell. If there is a collision the triangle is propagated to the eight octree child cells, which have to be instanced in the case that they have not been touched so far. If the leaf level is reached and there is a collision, the leaf cell is flagged as filled and this is propagated back to its parent. If now all eight children of the parent are filled, the parent is flagged as filled as well, its children are deleted and the filling of the parent is propagated to the grandparent and so on. If one encounters a filled cell during propagation of another triangle later, the propagation can be immediately stopped for that cell without checking for collision because this cell has already been completely filled by former triangles. Thus the propagation of triangles becomes faster with increasing filling degree.
This voxelization algorithm can easily be parallelized by starting the propagation of triangles not with the root but with the 8^d cells in depth d to get 8^d independent jobs. However, more jobs should be generated than parallel threads to do them since some subtrees might be processed much faster than others because of differing filling degrees.
The other interesting and most running time consuming aspect of the implementation of inner space detection is the max flow resp. min cut calculation. Usual algorithms for solving max flow min cut problems are the Edmonds-Karp algorithm [4], Dinic's algorithm [5] and the push-relabel algorithm [6]. The former algorithm has a $O(VE^2)$ running time and the two latter $O(V^2E)$, V denoting the number of vertices and E the number of edges. The push-relabel algorithm can even achieve a $O(V^3)$ running time by proper implementation [6]. Though in our application every vertex has at most 6 or rather 12 edges if we distinct between incoming and outgoing edges, that means $E \in O(V)$. Thus the asymptotic running time of all those algorithms is in fact the same for our application. By introducing additional data structures, some of those algorithm can be accelerated, as Dinic's algorithm to $O(EV \log V)$ [7] by dynamic trees and the push-relabel algorithm to $O(V^2\sqrt{E})$ by keeping lists of vertices with the same label which is a natural number that is assigned to each vertex during the algorithm. However, all these accelerations introduce additional per vertex data and hence are no option for our implementation since memory consumption is critical for high voxel resolutions.
Yet, $E \in O(V)$ is not the only special property of our graph. Besides, all of its edges have capacity 1. For such graphs it can be shown that Dinic's algorithm even runs in $O(E^{3/2})$ [8]. Therefore it is first choice for our application.

Furthermore since max flow calculation is the bottleneck of inner space detection, a parallelized implementation is desirable. Dinic's algorithm can be broken apart in two general operations, that alternate during the algorithm. One is the calculation of a so called layered network, which is in fact removing all edges in the residual network not belonging to a shortest path from the source by a BFS, and the other is the calculation of a so called blocking flow in this layered network, by which the total flow in the network is subsequently increased. This changes the residual network and both steps are repeated until there is no path from source to sink in the layered network anymore.
Since the first operation is in fact only a BFS, the ways to parallelize that are quite limited. A BFS can be divided in steps, each step consisting of collecting the unvisited neighbors of a set of recently visited vertices. These collected neighbors then become the recently visited vertices for the next step and for the first step the recently visited vertices are the source of the BFS. When the collecting of unvisited neighbors is done by several threads in parallel, it is clear that all threads must have finished the collecting before the unvisited neighbors of the just collected vertices can be collected in the next BFS step. Thus a thread synchronization is necessary after every BFS step.
Hence the only question when parallelizing a BFS is how the collecting of unvisited neighbors is allotted to the threads. In our implementation every thread has a queue of recently visited vertices and a queue for the neighbors it collects. At first thread i processes its own recently visited vertices and after that it supports thread $(i+1)\%p$, $(i+2)\%p$, ... by processing its vertices (with p the total number of threads). The queues are implemented by arrays and the vertices are taken from it for processing in chunks. Where the next chunk to process starts, is saved by atomic integers. Of course if an array holding the collected neighbors runs out of capacity, there must be a reallocation, but since every thread has its own queue of collected neighbors, no synchronization is needed. By this implemenation a data structure requiring locks for synchronization can be completely avoided.
The parallelization of the other operation of Dinic's algorithm, the calculation of a blocking flow in a layered network, is much more complicated to parallelize in general. Some solutions have been suggested [9, 10, 11]. Usually they combine preflow techniques, first introduced by Karzanov [12], with some quite sophisticated data structures for managing the parallelization. However, in our special case of an unit capacity network with a limited number of edges per vertex things are much simpler.
Let us at first consider the standard implementation for the blocking flow calculation in a layered network. Usually the paths from source to sink are found recursively by a DFS. If a path is found, the edge with the smallest residual capacity on this path is determined and the flow is updated by this value. Thus this edge is saturated and deleted from the network. Furthermore to avoid running into "dead ends" with no path to the sink several times one also deletes completely processed vertices during backtracking and all of their incoming edges as long as no path to the sink has been found. This search for paths from source to sink is repeated until

all outgoing edges of the source have been deleted, either because they are saturated or there is no path to the sink using them left.
In an unit capacity network things are much simpler. An augmenting path from source to sink has always capacity one. Although there might be edges with a residual capacity of 2 because of flow over the backward edge, such edges can never start at the source or end in the sink since for Dinic's algorithm flow never runs back to source or out of sink. Hence there can never be a complete path of such 2-capacity edges. The idea is now to reduce the residual capacity of every edge processed during DFS by 1 even if a path from source to sink is not yet found. The reason why this can be done is that either a path from sink to source using this edge will be found, and then everything is fine because every path has capacity 1, or there is no such path, but then the edge is removed during backtracking anyway. The advantage of this procedure is that the search for paths can be done in parallel by several threads because the capacity needed for an augmenting path is taken away at once and cannot be consumed twice by another thread running over the same edge. But of course, the residual capacity of an edge has to be stored as an atomic integer so that it can be safely decreased concurrently. Furthermore that is only possible because a layered network never contains edges in both directions so that if the capacity of an edge was decreased, the capacity of the corresonding backward edge would have to be increased.
Actually two threads can only run over the same edge if it has a capacity of 2. Only in this case and when additionally such edges lead to "dead ends", then there is a real overhead of the parallelization in comparison to the sequential version, that is not technical like the usage of atomic integers. Such edges are processed only once by the sequential version as they are deleted during backtracking. But since capacity 2 edges should be quite rare in a layered network, this effect should be neglectable.
Figure 1.8 shows the speed-ups for inner space detection for 12 different trunks for 2, 4, 8 and 16 threads compared to the single threaded version. As it can be seen, for 2 and 4 threads there is quite a reasonable speed-up in the range from 1.4 to 1.7 resp. 2.0 to 3.2. However, for higher thread counts the speed-up becomes worse compared to the thread count because the overhead for thread synchronization becomes a dominating factor. Especially since in our use case about half the running time is consumed by the BFS for calculating the layered network and the BFS obviously suffers very strongly from thread synchronization.

1.4 Running Time Results

Table 1.1 contains the running time results for the inner components detection of 12 different trunks for three voxel resolutions down to 0.5 mm. For each resolution the running time for the voxelization of the input geometry (soup of triangles), for the remaining inner components detection and the part of the latter for the flow calculation is stated. As it can be seen, the flow calculation is the dominating factor

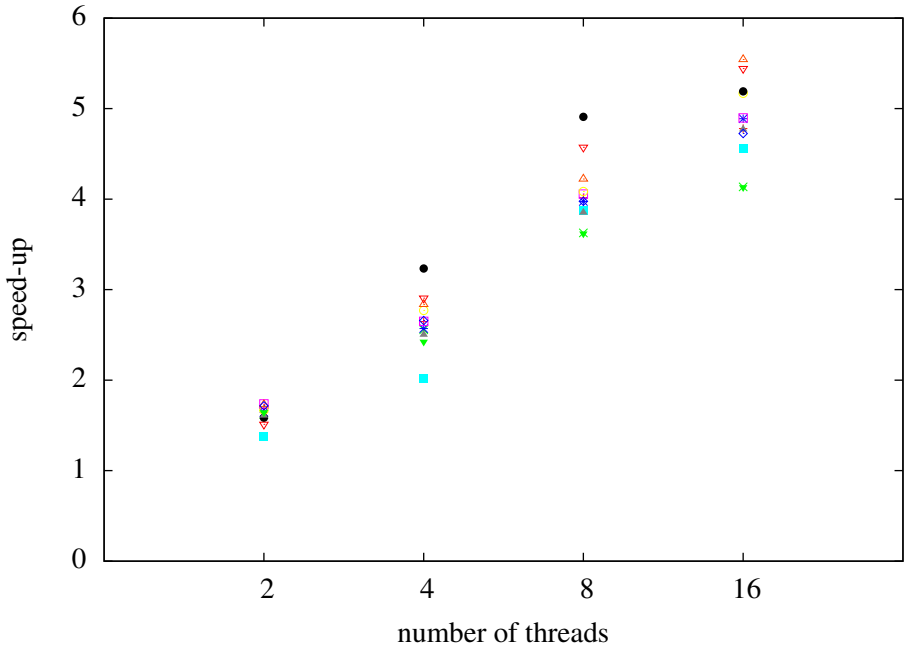


Figure 1.8: Speedups of flow calculation for 2, 4, 8 and 16 threads in relation to single threaded version for inner space calculation of 12 different trunks.

	2 mm			1 mm			0.5 mm		
	t_{geo} [sec]	t_{vox} [sec]	p_{flow} [%]	t_{geo} [sec]	t_{vox} [sec]	p_{flow} [%]	t_{geo} [sec]	t_{vox} [sec]	p_{flow} [%]
Trunk A	0.38	34.5	87.7	1.22	205	87.6	4.18	850	93.5
Trunk B	0.15	11.8	84.7	0.56	94	89.4	2.15	507	94.2
Trunk C	0.69	29.8	84.7	1.78	255	85.8	5.30	1039	93.4
Trunk D	0.55	29.3	87.4	1.49	186	86.9	4.56	884	93.9
Trunk E	1.36	69.2	84.7	3.30	967	89.0	9.44	3088	95.1
Trunk F	0.28	18.6	76.2	0.94	118	80.6	3.74	456	88.3
Trunk G	0.39	6.4	75.1	1.13	573	96.1	3.44	895	95.6
Trunk H	0.65	80.6	79.0	2.02	647	79.2	7.84	2627	92.5
Trunk I	0.31	36.1	88.7	0.99	263	90.5	3.77	1229	95.9
Trunk J	2.16	50.6	84.3	3.66	370	86.5	9.98	2065	95.4
Trunk K	0.41	100.7	96.4	1.18	346	92.8	3.79	1488	96.3
Trunk L	0.35	35.1	91.2	1.00	252	92.9	3.56	691	93.8

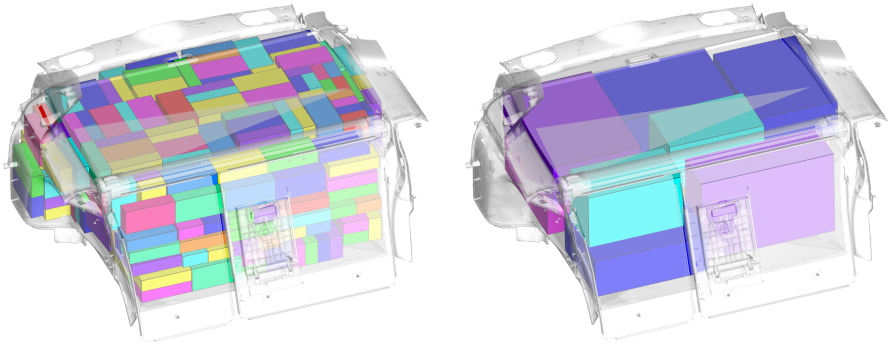
Table 1.1: Running time results for the inner components detection of 12 different trunks with a voxel size of 2 mm, 1 mm and 0.5 mm done with 8 threads on a Intel® Xeon® X5550 CPU at 2.66 GHz. The first column of each resolution contains the time for the voxelization of the geometry (soup of triangles), the second the remaining running time for the inner components detection and the third the part of this running time used for the flow calculation.

and amounts to nearly the whole running time with increasing resolution.
The total running time for 0.5 mm resolution remains under an hour for all trunks. Thus, we want to finally state that by the presented algorithm for calculating the voxelization of the interior of a trunk even submillimeter resolution can be reached within a reasonable running time.

Chapter 2

Packing Arbitrarily Shaped Objects into a Trunk

As already mentioned in the introduction, the continuous volume of a trunk is usually not measured by its continuous volume, but there are two standards that define it as the result of packing problems using boxes as packing objects. Firstly, this is the German standard DIN 70020 [13], which is common in Europe and uses equally sized boxes with a volume of one liter, and secondly, the American standard SAE J1100 [14], which is common in the USA and uses a set of seven differently sized boxes with volumes in the range from 5.63 to 67.47 liters. A DIN and a SAE packing of the same trunk can be seen in figure 2.1.
For both packing problems much work exists. For the DIN packing problem that are combinatorial approaches like integer linear programming [15, 16, 17], Monte-Carlo techniques [18, 19], evolutionary algorithms [20] and approaches that arrange the boxes in layers [1, 21]. For the SAE packing problem less work exists, but that also includes combinatorial approaches [22], evolutionary algorithms [20] and Monte-Carlo techniques [23].
Many of this work has flown into a software that has been developed in a long-time cooperation with a German car manufacturer and allows the generation of DIN 70020 and SAE J1100 packings. However, at a certain point of this cooperation our industrial partner expressed the need for not only generating box packings according to DIN 70020 and SAE J1100, but also packing arbitrarily shaped objects like suitcases, golf bags etc. The solution developed according to this requirement will be presented in this chapter.
It quickly became clear that simulated annealing [24] as standard technique should be applied to solve the problem since it has already been successfully applied to the box packing problem [19, 23]. Simulated annealing is a well-known probabilistic metaheuristic to find a good approximation for the global optimum of a given function. In order to use simulated annealing to pack a trunk, we have to encode packing a trunk with a set of arbitrarily shaped packing objects as optimization of



(a) (b)

Figure 2.1: A DIN 70020 packing (a) with equally sized small one liter boxes and a SAE J1100 packing with the larger differently sized boxes according to the standard.

a scalar function. But at first we want to give a decently formal description of a packing problem.
We are given a container described as an usually connected compact volume $C \subset \mathbb{R}^3$ in 3-dimensional space and a finite set $P_1, \ldots, P_n$, $P_i \subset \mathbb{R}^3$ of packing objects described as connected compact volumes in 3-dimensional space. To each of this objects we can apply a rigid transformation $T : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \mathbf{R}_i\mathbf{x}+\mathbf{t}_i$ with a rotation matrix $R_i \in \mathrm{SO}(3)$ and a translation vector $\mathbf{t}_i \in \mathbb{R}^3$ so that the transformed objects can be described by:

$$P_i(\mathbf{R}_i, \mathbf{t}_i) = T(P_i) = \left\{ \mathbf{x} \in \mathbb{R}^3 \,\middle|\, \mathbf{R}_i^T(\mathbf{x} - \mathbf{t}_i) \in P_i \right\}$$

Obviously a valid packing of the container can now be described as a tuple $(\mathbf{R}_{1,0}, \mathbf{t}_{1,0}), \ldots, (\mathbf{R}_{n,0}, \mathbf{t}_{n,0})$ of rigid transformations so that it is

$$P_i(\mathbf{R}_{i,0}, \mathbf{t}_{i,0}) \cap P_i(\mathbf{R}_{j,0}, \mathbf{t}_{j,0}) = \emptyset \ , \ 1 \leq i, j \leq n$$

$$\text{and} \quad P_i(\mathbf{R}_{i,0}, \mathbf{t}_{i,0}) \subset C \ , \ 1 \leq i \leq n$$

Given a function $p : \mathcal{P}(\mathbb{R}^3) \times \mathcal{P}(\mathbb{R}^3) \to \mathbb{R}_{\geq 0}$ estimating the "penetration" of two objects in some way with the property

$$p(A, B) = 0 \quad \Leftrightarrow \quad A \cap B = \emptyset \ , \ A, B \subset \mathbb{R}^3 \ ,$$

we can formulate the packing problem as a global minimization problem of the objective function

$$f : \ \left(SO(3) \times \mathbb{R}^3\right)^n \ \to \ \mathbb{R} \quad , \quad (\mathbf{R}_1, \mathbf{t}_1, \ldots, \mathbf{R}_n, \mathbf{t}_n) \ \mapsto$$

$$\sum_{i=1}^{n} p(P_i(\mathbf{R}_i, \mathbf{t}_i), \mathbb{R}^3 \setminus C) + \sum_{i=1}^{n} \sum_{j=i+1}^{n} p(P_i(\mathbf{R}_i, \mathbf{t}_i), P_j(\mathbf{R}_j, \mathbf{t}_j)) \ . \tag{2.1}$$

Like many probabilistic heuristics, simulated annealing works by searching for good solutions in the configuration space, in our case $\left(SO(3) \times \mathbb{R}^3\right)^n$, by applying some kind of random changes of the configuration, which are called moves, and evaluating the objective function on the found configurations. Since there is no guarantee for a probabilistic heuristic to find a global optimum, the choice of the objective function is crucial.
But like all other heuristics working in that way simulated annealing works the better, the more random moves can be done in a certain time. In most cases evaluating the objective function is much more expensive than applying a random move. Thus there is usually a trade-off when choosing this function. On the one hand it should be fast to evaluate and on the other hand it should describe the problem very well. That means for "good" configurations it should yield small values and for "bad" configurations it should yield big values.
Besides the objective function the choice of the random moves is crucial as well. On the one hand "small" moves in configuration space should be applied so that the algorithm will end up in a local minimum in any case and not in a configuration which is "close" to better configurations that could not be reached by a few moves. But on the other hand there should also be "big" moves in configuration space because otherwise the random walk might never visit far off regions of the configuration space or it might even be caught up in a local minimum.
However, the focus of this work is mainly on choosing a decent objective function that is very fast to evaluate. Thus the simple sum of "penetrations" in (2.1) is used and as objective function

$$p(A,B) := \max\left\{\max_{\mathbf{x}\in A\cap B}\inf_{\mathbf{y}\in A\setminus B}|\mathbf{x}-\mathbf{y}|, \max_{\mathbf{x}\in A\cap B}\inf_{\mathbf{y}\in B\setminus A}|\mathbf{x}-\mathbf{y}|\right\} \tag{2.2}$$

with $|.|$ the Euclidean norm. The infimum is necessary because $A \setminus B$ is not closed in opposite to A, B and thus $A \cap B$.
In fact, in order to speed up the calculation we are even going a step further and use an approximation of the penetration according to (2.2) which can be evaluated considerably faster after a proper preprocessing of the input geometry. The approach used for doing that is an advancement of the Voxmap Pointshell method [25].
The basic idea of that method is to detect a collision of two objects after applying different rigid transformations by voxelizing the one object and sampling the surface of the other by points and then check if sampling points lie in the voxelization of the former object. The advantage of this is that the voxelization and surface point sampling can be done as preprocessing and then the rigid transformations can be applied very easily and fast to the sampling point set. Checking if two objects A and B transformed by rigid transformations T_A and T_B collide, i.e. check if $T_A(A) \cap T_B(B)$ is non-empty, is the same as checking if $A \cap T_A^{-1}T_B(B)$ is non-empty. Therefore it is sufficient to apply a rigid transformation to the sampling point set, which is much faster than applying a rigid transformation to a voxelization.
However, in this simplest version the Voxmap Pointshell approach is not suitable

for our demands because of two reasons. Firstly, we do not only want to test objects for collision but give an approximation for the penetration according to (2.2) and, secondly, a simple voxelization stored in an array is no option because in the automotive industry precision at least in the magnitude of millimeters is needed while the geometric objects have sizes in the magnitude of meters. Hence a simple voxelization stored in an array would lead to unacceptable memory requirements.
The memory problem is solved by using a hierarchical voxelization, i.e. an octree representation in our case. In fact the packing items are not given by our industrial partner as a solid volume description but as a triangulation of their boundaries. And this triangulation is not necessarily a watertight mesh but might even have holes. Thus the heuristic developed in chapter 1 for detecting the interior of a trunk can be reused for detecting the outside instead of the inside. Since this heuristic treats outer and inner components coequally it can be easily adapted for doing so and we get an octree representation of our packing items by the way.
The other problem to solve is that we do not want to test only for collision but also get an approximation for the penetration according to (2.2). To achieve this the voxelization is extended to some kind of a so-called distance field which will be presented in the following section.
So far there seems to be no comparable approach but only algorithms for calculating the penetration between convex polyhedra [26, 27] or convex quadric primitives [28]. However, that is no option for our application case because packing objects might not be convex and using the triangle representation directly would be too slow for our speed requirements.

2.1 Generating an Adaptive Distance Field

While a voxelization stores only the information whether a voxel belongs to the voxelized object or not, a distance field additionally stores the distance of this voxel to the surface of the voxelized object. In this section we want to show how this idea can be transferred to an octree, which is in fact an adaptive voxelization, and how the resulting data structure, that we want to call an adaptive distance field, can be built up efficiently.
Starting point is an octree which describes a container or a packing object by labelling each of its cells as being inside or outside. The octree originally resulted from triangles describing the object's surface. However, as a simplification this original triangle geometry is discarded and the faces between inner and outer octree cells are considered as the new surface of the object. For every octree cell C the bounds d_{min} and d_{max} for the distance to the surface S are to be calculated so that for all $x \in C$ it is

$$d_{min} \leq d(x, S) \leq d_{max} \quad \text{with} \quad d(x, S) = \inf_{y \in S} |x - y| \,. \tag{2.3}$$

Why it is useful to have such bounds for our Voxmap Pointshell approach will become completely clear later on. (2.3) means that the tightest possible bounds are

$$d_{min,best} = \min_{x\in C}\min_{y\in S}|x-y| \tag{2.4}$$

$$d_{max,best} = \max_{x\in C}\min_{y\in S}|x-y|\ . \tag{2.5}$$

We can write $\min$ and $\max$ here and do not need $\inf$ or $\sup$ because all octree cells as well as the surface are compact sets in $\mathbb{R}^3$. Because the surface is the boundary between inner and outer octree cells, it can be represented as the union of a finite number of square shaped axis aligned faces $F_1,\ldots,F_m$:

$$S=\bigcup_{i=1}^{m}F_i$$

We suppose that all boundary cells are refined down to the voxel resolution which can be achieved by a proper preprocessing. Thus the edge length of the boundary faces F_i is the voxel resolution. Since $d_{min,best}$ for a cell C can be calculated by considering the minimum distance to every boundary face F_i, we want to use it as lower bound d_{min}:

$$d_{min,best}=\min_{x\in C}\min_{y\in S}|x-y|=\min_{i\in\{1,\ldots,m\}}\min_{x\in C}\min_{y\in F_i}|x-y|$$

$$=\min_{i\in\{1,\ldots,m\}}d(C,F_i)\ =:d_{min}\quad\text{with}\quad d(A,B):=\inf_{x\in A}\inf_{y\in B}|x-y| \tag{2.6}$$

Unfortunately, $d_{max,best}$ can not be calculated as easily because for different points in a cell there might be different points on the surface which are nearest to the particular point in the cell. Or mathematically spoken, we used in (2.6) the commutativity of the minimum set operator, which we cannot do for $d_{max,best}$, because its definition according to (2.5) also contains a maximum set operator.
However, obviously we can give an upper bound for $d_{max,best}$ similar to (2.6):

$$d_{max,best}=\max_{x\in C}\min_{y\in S}|x-y|\le\min_{i\in\{1,\ldots,m\}}\max_{x\in C}\min_{y\in F_i}|x-y|$$

$$=\min_{i\in\{1,\ldots,m\}}d_{max}(C,F_i)\ =:d_{max}\quad\text{with}\quad d_{max}(A,B):=\sup_{x\in A}\inf_{y\in B}|x-y| \tag{2.7}$$

The expressions $d(C,F_i)$ and $d_{max}(C,F_i)$ can be evaluated very easily for an octree cell C and a cell facet F_i. Furthermore it can be shown very easily that their values are in fact distances between points of the grid the deepest octree level corresponds to because minimum and maximum distances are reached between the corners of the cells and faces. Hence $d^2(C,F_i)$ and $d^2_{max}(C,F_i)$ can be expressed as multiples of the squared grid distance of the deepest octree level and be calculated completely by integer arithmetics in units of the squared grid distance.
The basic idea for a fast calculation of an adaptive distance field is now to take

the input voxelization represented as an octree, store for every octree cell d_{min}^2 and d_{max}^2 (in units of the squared grid distance), initialize them with the maximum integer, collect all boundary faces and update d_{min}^2 and d_{max}^2 of the cells to the correct values by propagating all faces through the octree hierarchy. This is possible because d_{min} and d_{max} are according to (2.6) and (2.7) minimum values over all boundary faces F_i, $1 \leq i \leq m$. Hence they can be initialized with the maximum value and successively be updated for all boundary faces.
The advantage of the hierarchical propagation is that that an early exit strategy can be used. Obviously the d_{max} value of a cell calculated for a subset of all boundary faces is an upper bound for the d_{min} and d_{max} values calculated for all boundary faces for this cell and all of its subcells. Thus, if a face F_i is propagated through the octree hierarchy and $d(C, F_i)$ is bigger for a cell C than the cell's current d_{max} value, we do not need to propagate this face to the cell's subcells because $d(C, F_i)$ and thereby $d_{max}(C, F_i)$ cannot be smaller than the current d_{min} and d_{max} value of this cell or its subcells.
However, one element is still missing to complete our algorithm to calculate an adaptive distance field based on an octree voxelization. As the input octree of the voxelization will contain big leaf cells in the inside and outside part of the voxelization, the d_{min} and d_{max} bounds for these cells will be far apart. For cells far away from the surface this is acceptable because this is the adaptiveness of the distance field which makes it possible to reach grid resolutions on the deepest level that would exceed acceptable memory limits when using a simple grid. Furthermore, the goal is to get only an estimation of the penetration, which can be used for simulated annealing. And for this heuristic it is satisfactory to roughly rate a configuration as very poor because there is much penetration. An exact quantification is not needed.
As already mentioned, boundary cells sharing facets with the surface are to be refined down to the deepest level. But depending on how the surface between inside and outside is positioned in the octree grid there might be also big leaf octree cells near the surface with their d_{max} value being several times larger than their d_{min} value, which is not desirable. Hence we need a criterion when an octree cell should be refined. For that the ratio of the average and half the distance of d_{min} and d_{max} is used:

$$\frac{\frac{1}{2}\left(d_{max} - d_{min}\right)}{\frac{1}{2}\left(d_{max} + d_{min}\right)} = \frac{d_{max} - d_{min}}{d_{max} + d_{min}} > \text{threshold} \tag{2.8}$$

In fact this criterion is needed for the squared values of d_{min} and d_{max} since the squared values are stored for all octree cells during the update process. But because of $d_{min}, d_{max} \geq 0$ and $0 < \text{threshold} < 1$ criterion (2.8) is equivalent to:

$$\left(d_{min}^2 + d_{max}^2\right)^2 > 4 \left(\frac{1 + \text{threshold}^2}{1 - \text{threshold}^2}\right)^2 d_{max}^2 \, d_{min}^2 \tag{2.9}$$

The threshold 0.5 was used throughout our experiments. Criterion (2.8) and (2.9) are not well-defined for boundary cells with $d_{min} = 0$. But boundary cells are

refined down to the deepest level anyway.
The refinement criterion (2.8) is checked when the boundary faces are propagated through the octree hierarchy and the d_{min} and d_{max} values of the cells are updated. Because for the function

$$t : \mathbb{R}_{>0} \times \mathbb{R}_{\geq 0} \to \mathbb{R} \quad , \quad t(x, y) \mapsto \frac{x - y}{x + y}$$

it is

$$\frac{\partial}{\partial x} t(x, y) = \frac{2y}{(x + y)^2} \geq 0 \text{ and } \frac{\partial}{\partial y} t(x, y) = \frac{-2x}{(x + y)^2} < 0 \text{ for } x > 0, y \geq 0 \tag{2.10}$$

and the d_{min} and d_{max} values are only reduced during face propagation, the refinement criterion can obviously only be fulfilled for a cell after its d_{min} value has been reduced. In this case new children are generated immediately for that cell and the face which caused the refinement is propagated further to these new children. The d_{max} value of a cell is always an upper bound for its d_{min} value as well as for its children's d_{max} and thus d_{min} values. Hence the d_{max} and d_{min} values of refined children are initialized by the parent's d_{max} value. In doing so the octree retains at any time the property that the d_{max} value of a cell is always greater or equal to the d_{max} values of all of its subcells.
When a cell is refined due to criterion (2.8), propagation of previously processed boundary faces to the newly created subcells might be necessary. This is achieved by a second pass of propagation. An important point is that during this second pass the refinement criterion cannot get fulfilled for any leaf cell and thus no further passes are necessary.
This can be easily proven by contradiction. Suppose there is a leaf cell for which the refinement criterion gets fulfilled in the second pass when propagating a certain face. Then this cell must be contained by a parent cell higher in the hierarchy for which the criterion was not fulfilled when the face was propagated during the first pass. But this is not possible because, since this parent cell has been updated by that face in the first pass, its d_{min} value is less or equal to the d_{min} value of the subcell updated in the second pass. Furthermore, its d_{max} value is greater or equal to the d_{max} value of the subcell as stated above. But then because of (2.10) it is clear that the refinement criterion cannot be fulfilled for the subcell while being not fulfilled for the parent cell with a smaller or equal d_{min} and a bigger or equal d_{max} value.
For boundary cells, including cells with only a corner on the boundary, d_{min} and d_{max} are not stored because, if d_v is the length of a voxel's edge on the deepest level, it is for boundary cells always $d_{min} = 0$ and $d_{max} = d_v$ resp. $d_{max} = \sqrt{3} d_v$ for cells with only a corner on the boundary. Instead for such boundary cells it is stored in a bit field which of its $3^3 - 1 = 26$ neighbors is on the other side.
Finally we want to query the adaptive distance field for the distance of points to the surface. And if a query point is in a boundary cell, by this neighbor information the exact distance to the boundary can be quickly calculated, which is better than

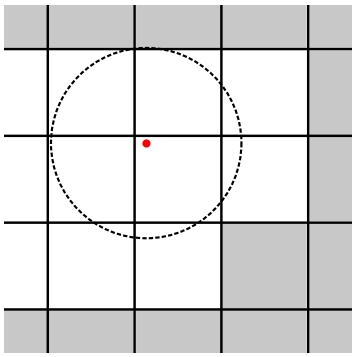


Figure 2.2: If only corners of a cell are on the boundary, the nearest point on the boundary might be a point on the surface of a next but one neighbor instead of the nearest corner.

just giving d_{min} and d_{max} as bounds for that distance.
However, if only corners of such a cell are on the boundary and the distance of a query point to the nearest corner on the boundary is larger than the distance to a next but one neighbor, this distance to the next but one neighbor has to be used as a lower bound and the distance to the corner as an upper bound because the next but one neighbor might be on the other side as illustrated in figure 2.2 for the 2-dimensional case. Obviously the distance of a point to the nearest next but one neighbor can be calculated as d_v plus the distance to the cell surface.
When all boundary faces have been propagated through the octree hierarchy for two times and all non-boundary cells contain the correct squared d_{min} and d_{max} values, in a final postprocessing step the square roots are extracted and for outside cells the d_{min} and d_{max} values are negated and swapped so that they are lower and upper bounds for a "negative penetration", which means in fact a distance. Thus storing in an additional boolean variable whether a cell is inside or outside is avoided. Furthermore using positive and negative values for penetrations resp. distances is more suitable when using the adaptive distance field in conjunction with other data structures, as it will become clear during the following sections. Of course, to store penetrations as positive and distances as negative values is arbitrary. The distances could also be stored as positive and the penetrations as negative values.

2.2 Generating a Proper Pointshell

For the Voxmap Pointshell approach a heuristic is necessary to sample the surface of the container and packing objects with points to generate a Pointshell. Because it is clear that the number of points will be crucial for the performance of the penetration test, the points should be generated only on the object's surface and not in the interior. The generated points should be distributed equally over the surface, but points should be placed preferably on sharp features like edges and spikes.
Because a voxelization of every object is generated anyway, it is obvious to use this voxelization for generating the surface point sampling. The basic idea is to choose

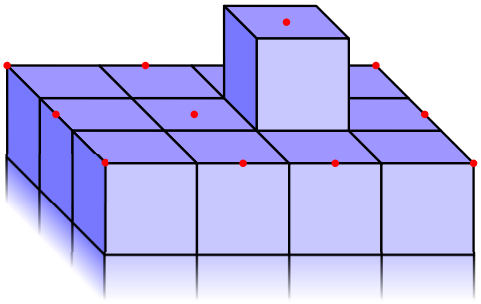
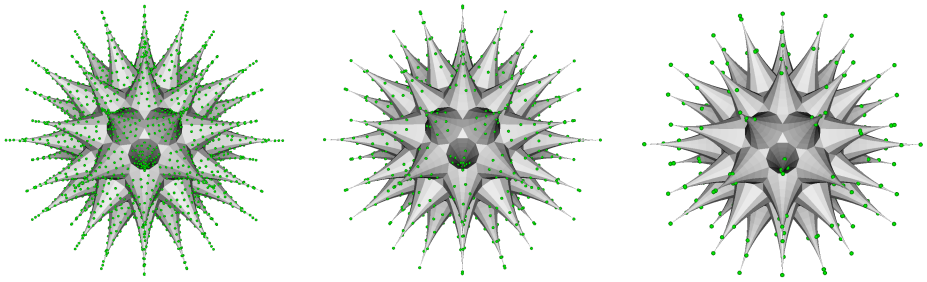


Figure 2.3: Illustration of "ideal" points on the boundary of surface cells to which the nearest point in the cell is chosen for a surface point sampling.

a certain voxelization resolution, which will be usually much lower than for the adaptive distance field, and place one point per geometry cell on the this resolution level neighbored to cells on the other side. In doing so it is achieved that really the surface is sampled and not the interior and that the points are distributed equally over the surface.
To place points preferentially on sharp features the following heuristic is used to decide which point among several points in a cell should be chosen for the surface sampling: For every axis direction of the voxelization it is checked if the cell has a lower and an upper neighbor belonging to the object's interior. If only one of the two neighbors belongs to the interior, the squared distance of the point to the opposite boundary of the cell in this direction is calculated. If both or none of the neighbors belongs to the interior, the squared distance of the point to the mid of the cell in this direction is calculated. These squared distances are added for all of the three axis directions and the point with the minimum value is chosen.
In fact that means that depending on which neighbors belong to the interior some kind of "ideal" point on the cell's boundary is defined, which is either in the mid of a face, in the mid of an edge or in a corner, and the point nearest to that "ideal" point is chosen. In figure 2.3 these "ideal" points are illustrated for some cells of an exemplary voxelization.
To generate a sufficient number of points in a geometry cell, of which one point can be chosen, the original triangles describing the packing object or the container are refined by splitting them recursively in the mid of their longest edge. That is done as long as the length of the longest edge is bigger than a certain threshold, which is small compared to the voxelization resolution. After the triangles have been refined in this way, for each cell at the surface one of the in this cell lying corners of the refined triangles is chosen as surface point according to the previously described heuristic.
In figure 2.4 the results for the described point sampling heuristic are illustrated for a sea-urchin-shaped test geometry with a voxelization resolution of 10 %, 20 % and 40 % of the minimum distance between two neighbored spikes. The total diameter of the test geometry is about 3.7 times this distance. As it can be seen, the



(a) (b) (c)

Figure 2.4: Results of the point sampling heuristic for a sea-urchin-shaped test geometry with a voxelization resolution of 10 % (a), 20 % (b) and 40 % of the minimum distance between two neighbored spikes.

heuristic works quite well in the sense that sampling points are reliably placed on the tips of the spikes. For the coarse voxelization resolution there are no longer sampling points in the concave regions between the spikes. However, that should not be a problem because for the final penetration test the surface point sampling of one object is always checked against the voxelization volume representation of the other object *and* vice versa because the theoretical penetration definition (2.2) is symmetric, too. Additionally, a penetration according to (2.2) between two points in concave regions of both objects' surfaces might not be completely impossible but quite pathological.

The points of a surface sampling are managed in a tree of bounding spheres calculated by the Welzl algorithm [29]. For that the bounding sphere for the points is calculated and then they are partitioned by their median according to their projection to the biggest eigenvector of their covariance matrix in their centroid system. The covariance matrix for a set of points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ is calculated by

$$\sum_{i=1}^{n} (\mathbf{p}_i - \bar{\mathbf{p}}) (\mathbf{p}_i - \bar{\mathbf{p}})^T \quad \text{with} \quad \bar{\mathbf{p}} := \frac{1}{n} \mathbf{p}_i \, .$$

Because the covariance is obviously symmetric it can be diagonalized for example by the Jacobi eigenvalue algorithm [30] for calculating the eigenvector of the biggest eigenvalue. Figure 2.5 shows six consecutive levels of bounding spheres calculated in this way for the surface point sampling of a suitcase. Because the points contained by a bounding sphere are partitioned into two subset for which the bounding spheres are calculated on the next level, the number of bounding spheres doubles in every level. This process is continued until a subset consists only of a certain number of points at most. Then this subset is not partitioned any further and their bounding sphere is a leaf in the tree.

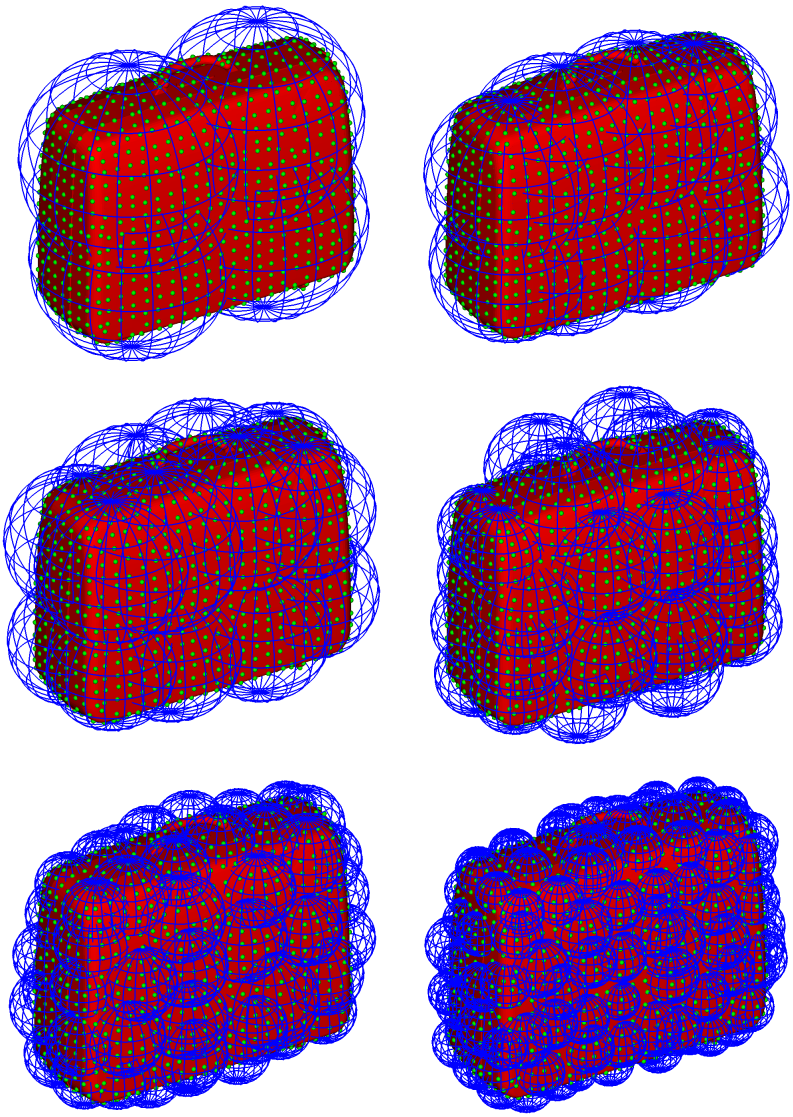


Figure 2.5: Six consecutive bounding sphere levels of the surface point samplings for a suitcase. Every sphere is the smallest enclosing sphere for a subset of the points, which a are divided by their median according to their projection to the biggest eigenvector of their covariance matrix in their centroid system into two subsets for the next deeper level.

2.3 Combining Data Structures for a Fast Penetration Estimation

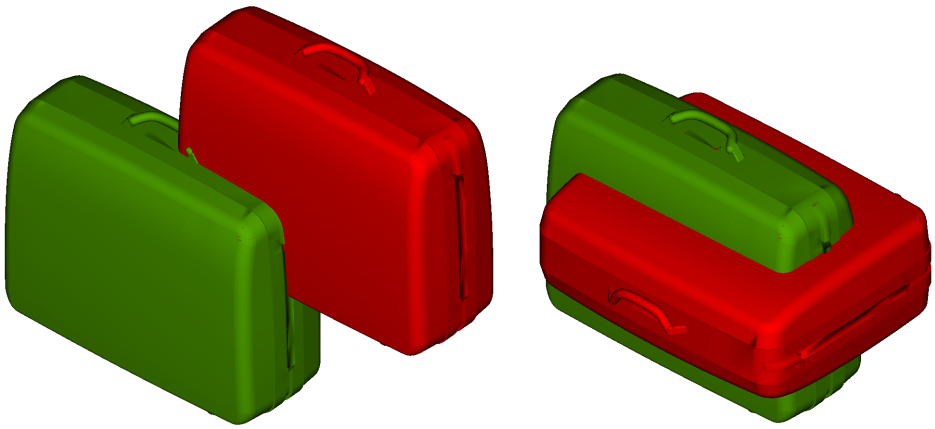
The data structures described so far – the adaptive distance field (ADF) on the one hand and the surface point sampling managed in a bounding sphere tree (BST) on the other hand – can now be combined for a fast penetration estimation. We want to recall that by an ADF lower and upper bounds for a point's distance resp. penetration with the object described by the ADF can be retrieved. Penetration is defined as distance to the object's surface for points lying inside the object and by convention penetrations for points lying inside are positive and distances for point lying outside are negative.
If we want to check a packing object transformed by a rigid transformation $T_1 : \mathbf{x} \mapsto \mathbf{R}_1\mathbf{x} + \mathbf{b}_1$ for penetration with another packing object transformed by $T_2 : \mathbf{x} \mapsto \mathbf{R}_2\mathbf{x} + \mathbf{b}_2$, we can also check the first object transformed by

$$T_2^{-1}T_1 : \mathbf{x} \mapsto \mathbf{R}_2^T(\mathbf{R}_1\mathbf{x} + \mathbf{b}_1 - \mathbf{b}_2) = \underbrace{\mathbf{R}_2^T\mathbf{R}_1}_{=:\mathbf{R}}\mathbf{x} + \underbrace{\mathbf{R}_2^T(\mathbf{b}_1 - \mathbf{b}_2)}_{=:\mathbf{b}}$$

for penetration with the second untransformed object.
Given a BST and an ADF description, this check for penetration can be conducted by traversing the BST hierarchy by the recursive function described in algorithm 2.1. For the initial invocation of the function on the root of the BST the currently minimum penetration d_{min} is initialized by 0. What makes this penetration estimation algorithm fast is the early exit at line 7. However, because of the early exit it is important that d_{min} increases as fast as possible to its final value. Thus the order of the recursive invocations on the children in the lines 18 and 19 is crucial.
The used heuristic to get a reasonable order of both recursive invocations is that this child is processed first whose transformed center is closer to the center of the bounding sphere of the other object's surface point sampling. Because the transformed center of each bounding sphere must be calculated at line 4 anyway, the additional computing costs for this heuristic concern only the distance calculation. When actually implementing the function in algorithm 2.1, the transformed center must be passed as parameter from the preceding recursive function call.
However, the Voxmap Pointshell approach to calculate a penetration by algorithm 2.1 has some quirks that have to be considered. One is that if two equal objects are placed congruently in the same position, the Voxmap Pointshell approach might detect no penetration or only a very small penetration because no points on the surface of the one object are deeply penetrating the voxelization of the other object. To overcome this problem the center of the ADF cell with the biggest penetration, i.e. distance to the object's surface, is taken and checked additionally to the object's surface points for penetration with the ADF of the other object when checking two objects for interpenetration.
Another problem is that the penetration retrieved by this approach might not be in any case a good measure for how hard it is to separate both objects. Figure

Algorithm 2.1 Recursive BST traversal for calculating the penetration d with and object described by an ADF. d is initialized by 0.

```
1:  function TRAVERSE(bounding sphere S)
2:      c ← center of S
3:      r ← radius of S
4:      c_t ← Rc + b                          ▷ calculate transformed center
5:      u ← upper bound for distance of c_t to surface retrieved from ADF
6:      if u + r ≤ d then
7:          return
8:      end if
9:      if S is leaf in BST hierarchy then
10:         for each surface point p contained by S do
11:             p_t ← Rp + b                  ▷ calculate transformed point
12:             l ← lower bound for distance of p_t to surface retrieved from ADF
13:             if l > d then
14:                 d ← l
15:             end if
16:         end for
17:     else                                  ▷ recursive invocation
18:         TRAVERSE(child 1)
19:         TRAVERSE(child 2)
20:     end if
21: end function
```



(a) (b)

Figure 2.6: Although the suitcases have in both positions the same ADF-BST penetration, position (b) is worse in the sense that a bigger move has to be applied to one of the two suitcases in order to separate them.

2.6 shows two suitcases in two positions in which they penetrate each other. The ADF-BST penetration is the biggest possible value in both positions because surface points of the one suitcase are lying in innermost ADF cells of the other suitcase. However, it is clear that the arrangement of the suitcases in position (b) is "worse" in the sense that a bigger move of one of the suitcases is necessary in order to separate them. Obviously this is most problematic for packing objects with an extent along one principal axis which is much smaller than the extent along the two other principal axes.
Usually such objects have roughly the shape of thin boxes just like the suitcases in figure 2.6, and thus it is an obvious idea to additionally "approximate" them by boxes, calculate some kind of "reasonable" penetration between these boxes and finally take as penetration for the simulated annealing the maximum of this penetration and the ADF-BST penetration.
A reasonable penetration between two boxes can be calculated by the so called separating axis theorem (SAT) [31]. This theorem states that two boxes do not penetrate each other resp. are disjoint iff all their projections onto the $3 + 3 = 6$ edge directions of both boxes as well as onto the $3 \times 3 = 9$ cross products of these edge directions are disjoint. If now two boxes penetrate each other, they overlap in all theses 15 projections and as a penetration measure we can simply take the minimum overlap of all these projections since a translation by this length along the corresponding projection direction would separate both objects. The overlap of two intervals $[a_1, b_1]$ and $[a_2, b_2]$ can be calculated by (if w.l.o.g. $a_1 \leq a_2$)

$$\text{overlap}([a_1, b_1], [a_2, b_2]) = \begin{cases} 0 & \text{if } a_2 \geq b_1 \\ b_1 - a_2 & \text{if } a_2 < b_1 \leq b_2 \\ \min\{b_1 - a_2, b_2 - a_1\} & \text{if } b_1 \geq b_2 \end{cases} .$$

But how can an arbitrary packing object be approximated by a box? Because we want to take finally the maximum of the box penetration and the ADF-BST penetration for the simulated annealing, it is clear that the approximating boxes may only penetrate each other if the packing objects penetrate each other, too. Thus the approximating boxes must be contained by the packing objects and hence it is clear that it is best to take some kind of maximum box contained by the packing object. It is obvious to take the box with maximum volume contained by the voxelization because only the voxelization is a solid representation of the packing object.
To keep this problem computationally solvable in reasonable time the box should have the same alignment as the cells of the voxelization. However, the input geometry consisting of a soup of triangles describing the packing object's surface might have an arbitrary orientation in space. In order to find a proper alignment at first the surface point sampling is calculated. Then the alignment for the voxelization is determined by the principal axes of this point set, which are calculated as the eigenvectors of the points' covariance matrix, which must be calculated anyway for dividing the points into two sets for the first BST level (see section 2.2).
In the following sections an algorithm is presented for finding a box with maximum volume contained by a 3D voxelization. The basis of this algorithm is another al-

gorithm for enumerating all rectangles contained by a 2D voxelization which are maximal with respect to inclusion. This algorithm could also be used to solve the 2-dimensional problem – finding a rectangle with maximum area contained by a 2D voxelization – but in fact it is more general and can even be generalized to arbitrary orthogonal polygons instead of 2D voxelizations.

2.4 A Simple Sweep Line Algorithm for Enumerating All Maximal Rectangles in an Orthogonal Polygon

In this section a simple sweep line algorithm for enumerating all rectangles in an orthogonal polygon which are maximal with respect to inclusion is presented. A rectangle A contained by an orthogonal polygon $P \in \mathbb{R}^2$ is maximal with respect to inclusion if no other rectangle B contained by P exists which itself contains A without being equal to A. An orthogonal polygon is defined as a polygon whose edges are parallel to the coordinate axes. More mathematically you can define an orthogonal Polygon P as the closure of an open set $Q \subset \mathbb{R}^2$ so that its boundary $\partial P = P \backslash Q$ is the union of a finite number of vertical and horizontal line segments, the edges of the polygon:

$$\partial P = \bigcup_{i=1}^{n_v} \{(x_{i,v}, y) \,|\, y \in [y_i^-, y_i^+]\} \cup \bigcup_{j=1}^{n_h} \{(x, y_{j,h}) \,|\, x \in [x_j^-, x_j^+]\} \qquad (2.11)$$

Let $x_1 < \ldots < x_{n_x}$ and $y_1 < \ldots < y_{n_y}$ be the sorted x and y coordinates occurring in (2.11). Obviously a maximal rectangle contained by P, i. e. a rectangle which is not contained by an other rectangle contained by P, must be of the form

$$[x_{i_-}, y_{j_-}] \times [x_{i_+}, y_{j_+}] \quad \text{with} \quad 1 \leq i_- < i_+ \leq n_x \text{ and } 1 \leq j_- < j_+ \leq n_y \,.$$

Otherwise the boundary coordinates could be extended to the next smaller resp. larger value in $\{x_1, \ldots, x_{n_x}\}$ resp. $\{y_1, \ldots, y_{n_y}\}$ without leaving the area of the polygon in contradiction to the maximality of the rectangle with respect to inclusion. Hence the number of maximal rectangles is finite and all maximal rectangles can be enumerated.
For a number of sorted real values $r_1 < \ldots < r_n$ obviously a strictly monotone continuous and thus bijective function can be found with

$$f_{r_1,\ldots,r_n} : [r_1, r_n] \to [1, n] \text{ with } f_{r_1,\ldots,r_n}(r_i) = i \text{ for } i \in \{1, \ldots, n\} \,.$$

Thus by applying the function

$$\begin{array}{rccc} f_P : & [x_1, x_{n_x}] \times [y_1, y_{n_y}] & \to & [1, n_x] \times [1, n_y] \,, \\ & (x, y) & \mapsto & \Big(f_{x_1,\ldots,x_{n_x}}(x), f_{y_1,\ldots,y_{n_y}}(y)\Big) \end{array}$$

to the orthogonal polygon it can be transformed so that all of its corners have natural coordinates and it can be considered as a 2D voxelization. Figure 2.7 shows an

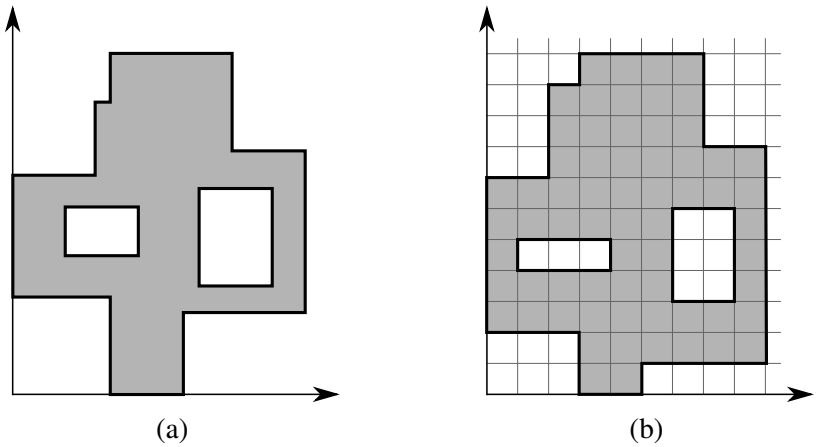


Figure 2.7: When looking for rectangles which are maximal with respect to inclusion in an orthogonal polygon (a) (that means all of its edges are parallel to the coordinate axes) this can be done for an equivalent polygon (b) whose corners lie on an equidistant grid.

orthogonal polygon and the corresponding 2D voxelization.
Obviously set inclusion is compatible with such a mapping, that means rectangles maximal with respect to inclusion are mapped to maximal rectangles in both direction. This means that when we want to enumerate all maximal rectangles contained by an orthogonal polygon it is sufficient to do that for the corresponding 2D voxelization. In fact we want to apply this algorithm only for 2D voxelizations anyway but describing this algorithm for 2D voxelization does not mean any restriction when considering orthogonal polygons.
The sweep line algorithm works by processing an interval representation of the polygon as shown in figure 2.8. The intervals span cells with the same x coordinate and describe stripes parallel to the y axis, but of course the algorithm would also work when choosing the other interval representation perpendicular to that. The intervals are processed along the x axis with ascending x coordinates but of course this is again arbitrary and they could also be processed in descending order. Such an interval representation can obviously be retrieved by sorting the vertices of the polygon according to their x and y coordinates in lexicographical order which requires an effort in $O(n_{vert} \log n_{vert})$ if n_{vert} is the polygon's number of vertices. For our application case the interval representation can even be retrieved directly, which will be explained in the next section.
When now the intervals are processed from left to right with increasing x coordinate, they have to be put into relation with the intervals from the previous step. Seven basic cases can be distinguished when processing an interval in the next step:

1. The interval is not contained by an interval of the previous step.
2. The interval spans the same y range as a previous interval.
3. The interval extends a previous interval.

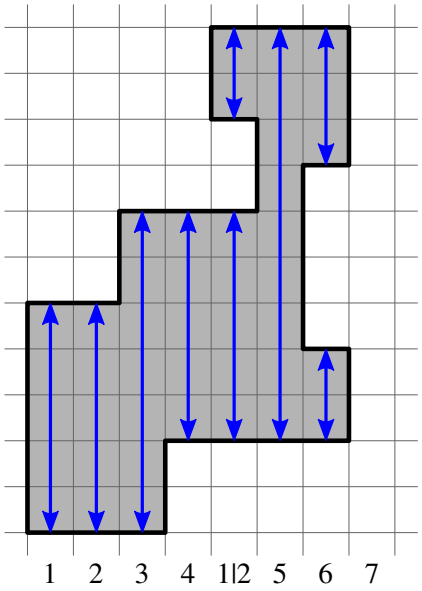


Figure 2.8: For the sweep line algorithm an interval representation consisting of intervals spanning cells with the same x coordinate is processed from left to right. Seven basic cases can occur when doing that, which are numbered at the bottom and explained in the text.

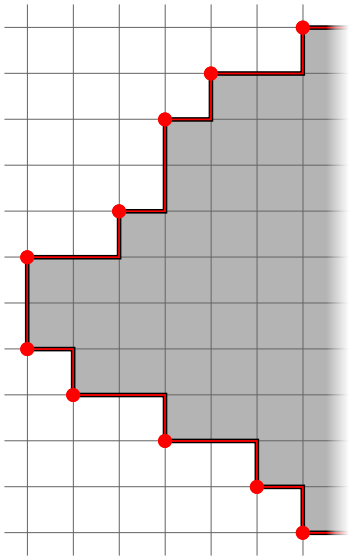


Figure 2.9: As long as the interval spanning the polygon's height only increases when processing from left to right, the coordinates of outer vertices (red dots) of vertical edges are stored in a deque (double ended queue) that actually resembles the polygon's contour.

4. The interval shortens a previous interval.
5. The interval merges two or more previous intervals.
6. There are two or more intervals overlapping with a previous interval and thus splitting it.
7. For a previous interval there is no interval in the next step overlapping with it.

The seven cases are illustrated in figure 2.8. The cases are not mutual exclusive but several cases can occur at once, just like cases 1 and 2 occur simultaneously between cases 4 and 5 in figure 2.8.
In the first step for the smallest x value a new interval is created (case 1). As long as this interval is not changed (case 2) or only extended (case 3), the algorithm simply stores the coordinates of the vertices which are outer endpoints of vertical

Algorithm 2.2 Deque unwinding when processing an edge spanning the interval $[\tilde{y}_{min}, \tilde{y}_{max}]$ at the x coordinate x_{sweep}

```
1:  x_min ← x_sweep
2:  while true do
3:      y_min ← y coordinate of first vertex in deque
4:      y_max ← y coordinate of last vertex in deque
5:      if ỹ_min ≤ y_min and y_max ≤ ỹ_max then
6:          if ỹ_min < y_min then
7:              add vertex (x_min, ỹ_min) to the front of deque
8:          end if
9:          if y_max < ỹ_max then
10:             add vertex (x_min, ỹ_max) to the back of deque
11:         end if
12:         break
13:     end if
14:     x_min ← maximum of x coordinates of first and last vertex in deque
15:     enumerate rectangle [x_min, x_sweep] × [y_min, y_max]
16:     if x_min = x coordinate of first vertex in deque then
17:         remove first vertex from deque
18:     end if
19:     if x_min = x coordinate of last vertex in deque then
20:         remove last vertex from deque
21:     end if
22: end while
```

edges in a deque (double ended queue) as shown in figure 2.9 so that this deque actually resembles the contour of the polygon. The y coordinates of these edges correspond to the boundary y values of the processed intervals.

If now the interval is shortened (case 4) we can unwind this deque from both ends and enumerate a number of rectangles which will no longer be contained by the contour described by the unwound deque. If x_{sweep} is the x coordinate of the sweep line and $[\tilde{y}_{min}, \tilde{y}_{max}]$ is the y range of the processed interval on the sweep line, this can be done by the loop shown in algorithm 2.2.

If the interval bounds $\tilde{y}_{min}$ and $\tilde{y}_{max}$ do not match existing y coordinates in the deque, new vertices must be added to its front resp. back after unwinding (see lines 6 to 11). What rectangles are enumerated and how the contour described by the deque looks like after unwinding the deque is shown in figure 2.10 for an exemplary case.

Obviously the same code can be used when there is no interval on the sweep line overlapping with a previous interval (case 7), and the polygon is closed at the previous interval by an edge spanning it. In this case only the break condition from line 5 to 13 is left away, and the loop runs as long as the deque is not empty.

If an interval is split by two separate intervals on the sweep line overlapping with

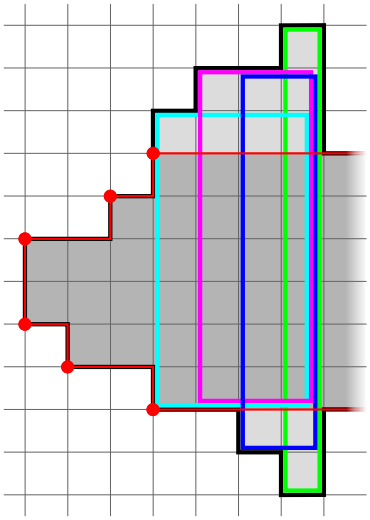


Figure 2.10: Rectangles that are enumerated when the next interval is smaller than the opening range of the deque. The rectangles are enumerated by unwinding the deque of vertices (red). The four rectangles are drawn slightly smaller so that their contours do not overlap.

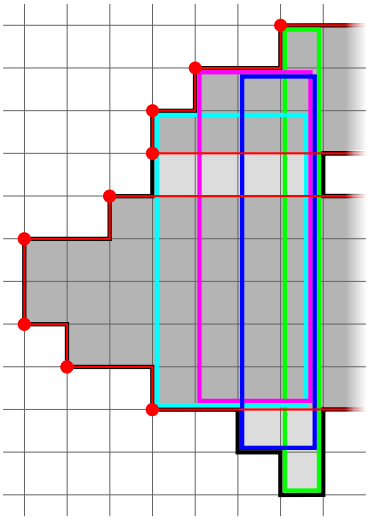


Figure 2.11: When there are two intervals overlapping with a previous interval and thus splitting it, the deque is unwound and rectangles are enumerated in the same way, but the unwound vertices are not thrown away but added two a second deque forming another contour.

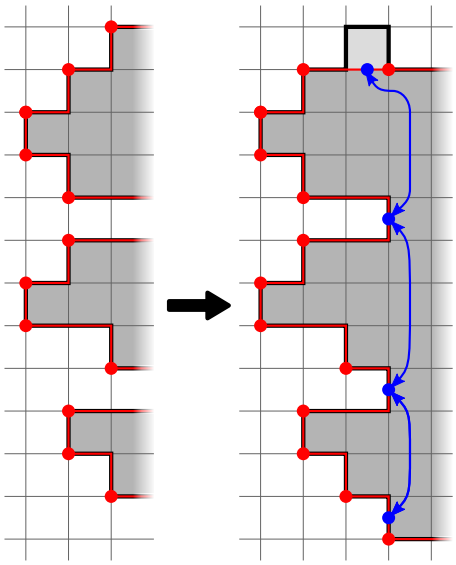


Figure 2.12: When several contours are merged because they are spanned by the next interval, artificial nodes are additionally added at the front and back of the deque as well as between every pair of neighbored contours. In these artificial nodes references to their neighbors are stored so that they form a double linked list in the deque. Behind the artificial nodes at the front and back of the deque normal nodes are added even if the outmost contours are not extended in y direction.
In the artificial nodes no coordinates are stored. Thus the position between their neighbors at which they are drawn in the figure is not crucial.

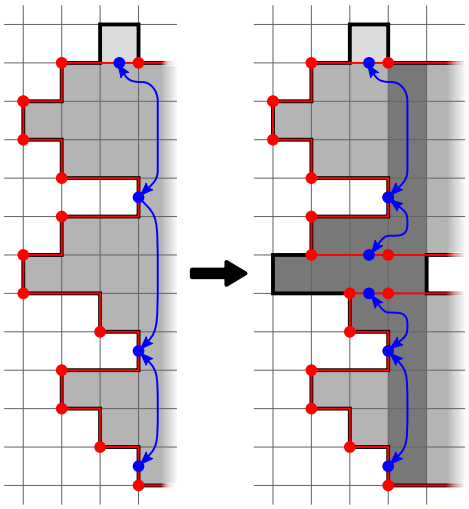


Figure 2.13: When a contour with several bays is split, only the bay(s) in the "split region" are processed. Thus in the depicted case only rectangles in the dark gray region are enumerated.

it (case 6), the procedure is quite similar. The deque is unwound in the same way until the spanned interval is contained by one of the two intervals on the sweep line. But the unwound vertices are not thrown away but added to a second deque forming another contour if their y coordinates are contained by the next interval on the sweep line. This situation is shown in figure 2.11.

If two or more contours are merged (case 5), artificial nodes are added at the front and back of the deque as well as in between of every pair of neighbored contours as shown in figure 2.12. In these artificial nodes references to the two neighbored artificial nodes are saved so that they form a double linked list within the deque. The old contours form some kind of "bays" in the new contour. Behind the artificial nodes at the front and back of the deque normal nodes are added even if the outmost contours are not extended in y direction. These nodes ensure that when unwinding the deque a rectangle spanning the whole extent in y direction is enumerated before processing the bays.

The bays obviously require some special treatment when the new contour is modified. If it is shortened beyond complete bays, the deque unwinding code has to fork for each of these bays and all rectangles in these bays have to be enumerated. But if it is split, only bays lying in the "split region" have to be processed while bays that are spanned completely by one of the new parts have not to be processed. This is illustrated in figure 2.13.

As already mentioned, the seven basic cases are not mutual exclusive, but several cases can occur at once, and this is taken into account by the algorithm's actual implementation. Just like the sweep line moves from left to right, the polygon's intervals at the current sweep line position are processed from bottom to top, and the contours are stored in ascending order according to their y coordinates. If a contour of the last step is shortened at its top, it has to be checked for the cut off part of the contour if it overlaps with the next interval on the sweep line so that there is a split. But this next interval might even span the next contours so that they

are merged with this remaining cut off part and so on.

2.4.1 Running Time Considerations

It is clear that the algorithm 2.2 for unwinding the deque has a running time linear in the number of rectangles it enumerates and that it is also linear in the size of the unwound deque because every normal node or at most pair of nodes at both ends leads to a rectangle that is enumerated when unwinding. There might be additional effort because of artificial nodes but it is obvious that there cannot be more artificial nodes than normal nodes.
Except the unwinding of deques the described algorithm only comprises the processing of intervals on the sweep line. If the 2D voxelization resulted from an orthogonal polygon, it is clear that, when processing the intervals for a certain sweep line position, at least one deque is changed by adding nodes or unwinding the deque. If we assume that the number of intervals on the sweep line is limited by a constant, the additional effort for processing the intervals is also linear in the number of nodes of the contours and thus linear in the number of maximal rectangles. Hence in this case the whole algorithm has a running time only linear in the number of maximal rectangles and thus it can be considered to be optimum.
If the 2D voxelization did not result from an orthogonal polygon, there might be sweep line positions at which none of the existing contours is changed and thus the running time of the algorithm is linear in the horizontal extent of the voxelization if that is bigger than the number of maximal rectangles.
Now the question is if the running time and thus the number of maximal rectangles can also be put in relation to other measures. It can be easily shown that the running time is linearly limited by the number of inner cells of the voxelization. The effort to process the intervals on the sweep line is obviously limited by the number of inner cells these intervals span. When unwinding a deque there will be at least one row of inner cells just in the depth the deque has to be unwound which are no longer contained by any contour after unwinding and which have not to be taken into account any more.
At first glance this upper bound might not to be very good, but in fact one can easily construct examples in which it is tight. Figure 2.14 shows such an example. The shown polygon has n vertices (in the figure for $n = 6$) at its right bottom side and every vertex is the lower right corner of n maximal rectangles. Hence on the whole it contains n^2 maximal rectangles and it has

$$2n - 1 + 2 \sum_{i=n}^{2n-2} i = 3n^2 - 3n + 1$$

inner cells. Thus the number of maximal rectangles is in fact linear in the number of inner cells.

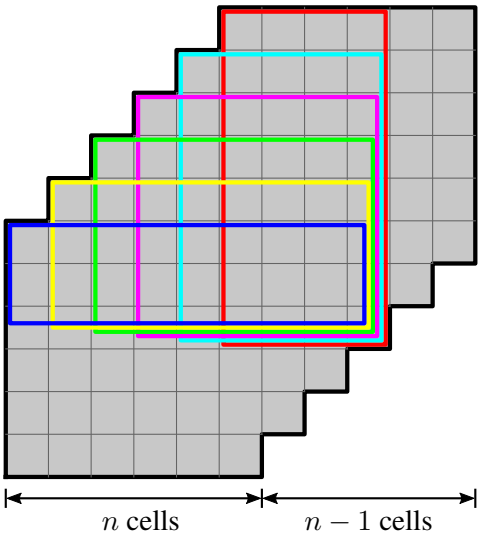


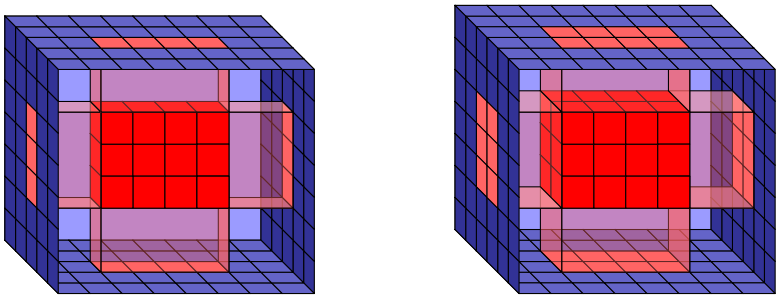
Figure 2.14: A polygon as depicted has n vertices at its right bottom side and every vertex is the lower right corner of n maximal rectangles. In the picture these rectangles are drawn for one vertex, but a little bit smaller at their right and bottom side so that their contours do not overlap. Thus this polygon contains n^2 maximal rectangles.

2.5 Finding a Largest Box in a 3D Voxelization

With the help of the algorithm for enumerating all maximal rectangles in a 2D voxelization described in the preceding section we can now give a quite easy algorithm for finding a largest box (concerning volume) in a 3D voxelization. The basic idea is that every box in a 3D voxelization can be described as the offset of a smaller "inner box" in the voxelization or more formally as the Minkowski sum of an inner box in the voxelization and a cube whose edge length is an integral multiple of the cell length. If the offset is made as large as possible and the inner box as small as possible and the box extends over l grid cells at its shortest edge, the inner box extends only one grid cells in this direction if l is odd and two if l is even. Then for the offset d it is

$$\begin{aligned} d &= \tfrac{l-1}{2} && \text{if } l \text{ is odd and} \\ d &= \tfrac{l}{2} - 1 && \text{if } l \text{ is even.} \end{aligned}$$

The two cases are visualized in figure 2.15(a) and (b).
The idea is to start with the voxelization's surface cells and successively calculate the surface cells of the voxelization's negative offset by 1, 2, 3, ... cells until no cell is left. This is done by a simple BFS which collects in every step the surface cells of the $d+1$ offset as those among the 26 neighbors of every d offset surface cells which do not belong to the surface of the d or $d-1$ offset.
Usually for a BFS a visited flag is stored for every cell. However, the voxelization is stored by an octree representation to save memory and to keep the memory



(a) (b)

Figure 2.15: A box in a 3D voxelization (blue, front face not drawn) can be described as offset of an inner box (red). To make the position of the inner box clear the projections of the inner box's left, right, bottom and top faces onto the surface of the outer box is visualized. If the offset is made as large as possible and the inner box as small as possible and the outer box extends over an odd number of cells in its shortest edge direction, the inner box extends only over one cell in this direction (a). If the outer box extends over an even number of cells in its shortest edge direction, the inner box extends over two cells in this direction (b).

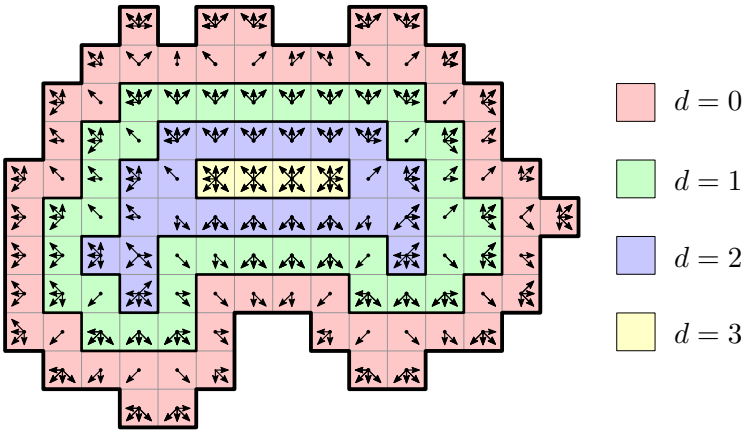


Figure 2.16: The surface cells of all negative offsets of the voxelization are calculated by a BFS until no cells are left. For every cell it is stored in a bit field which of its neighbors belong to the previous layer or are outside in case of the outmost layer. These bit fields are visualized by small arrows.

Algorithm 2.3 BFS for calculating negative voxelization offsets

```
1:  S ← hash map with indices of voxelization's surface cells as keys and bit field
        storing neighbors outside as mapped value
2:  while S is not empty do
3:      nextS ← empty hash map
4:      for all cells in S do
5:          for all neighbors not stored in bit field as belonging to previous layer
            do
6:              if S does NOT contain neighbor then
7:                  retrieve neighbor's bit field from nextS or insert it zero-
                        initialized if nextS does not yet contain neighbor
8:                  set current cell in neighbor's bit field as belonging to
                        previous layer
9:              end if
10:         end for
11:     end for
12:     swap S and nextS
13: end while
```

consumption proportional to the voxelization's surface and not its volume. Storing flags for every cell would be contrary to that. Thus the surface cells of the offsets are stored by their 3-dimensional indices in a hash set. When the neighbors of the d offset's surface cells are iterated for collecting the surface cells of the $d+1$ offset, we can find out by querying this hash set which neighbors are surface cells of the d offset and do not belong to the next surface layer. Additionally these neighbors must be excluded which belong to the surface of the $d-1$ offset. This is done by storing in a bit field for the cells when they are collected which of their 26 neighbors belong to the previous surface layer. For the original surface cells of the voxelization, which can be considered as the $d = 0$ offset surface cells, this bit field stores which neighbors in the grid do not belong to the voxelization. Actually, the hash set is extended to a hash map and the cell index is stored as key and the bit field as value. The different surface layers and these bit fields are visualized in figure 2.16 for the 2D case. The pseudo code for the BFS is shown in algorithm 2.3.

A similar idea for the 2-dimensional case can be found in [32] where it is applied in order to find a rectangle with maximum area contained by a 2D voxelization. However, the algorithm presented in [32] has a running time which is linear in the number of the cells of the 2D voxelization in any case. With the algorithm presented in section 2.4 such a maximum rectangle in a 2D voxelization can be found usually much faster and only with a running time linear in the number of cells in worst case scenarios.

To extend this approach to the 3-dimensional case the idea is to look in every pass of the BFS loop at line 2 in algorithm 2.3 for all 2-dimensional cuts of the

voxelization's d offset for a rectangle with width w and height h which is maximum with respect to its "extended area" $(w+2d)\cdot(h+2d)$. Of course this is to be done for cuts perpendicular to x, y as well as z direction. For example if we find such a maximum rectangle in the $x = x_0$ cut of the voxelization's d offset with (y_0, z_0) as its lower left corner, that means that the corresponding box

$$[x_0 - d, y_0 - d, z_0 - d] \times [x_0 + d, y_0 + w + d - 1, z_0 + h + d - 1]$$

is contained by the original voxelization and there is no other box whose shortest edge extends over $2d + 1$ cells and which has a larger volume than this box. The interval bracket notation is to be understood that cells with the contained integral indices are spanned by the box. Thus an interval $[a, b]$ spans $b - a + 1$ cells.
If we determine such maximum rectangles for all $d = 0, 1, \ldots$ offsets and take the corresponding box with the largest volume, we know that this is a box contained by the voxelization which is maximum concerning volume among the boxes that extend over an odd number of cells in their shortest edge direction.
However, we want to find a maximum box among all boxes contained by the voxelization. Thus not only the 2-dimensional cuts which extend over one cell in their normal direction have to be taken into account, but also the intersection of every pair of neighbored cuts which extend over two cells in normal direction. If we find a rectangle with width w and height h in the intersection of the $x = x_0$ and $x = x_0 + 1$ cut of the voxelization's d offset with (y_0, z_0) as its lower left corner, that means that there is a corresponding box

$$[x_0 - d, y_0 - d, z_0 - d] \times [x_0 + d + 1, y_0 + w + d - 1, z_0 + h + d - 1]$$

contained by the original voxelization.
The search for a maximum rectangle in the 2-dimensional cuts of the voxelization's offsets is done by the algorithm for enumerating all rectangles in a 2D voxelization which are maximal with respect to inclusion described in section 2.4. The big advantage is that the necessary interval representation can be easily retrieved from the offset's surface cell indices and the stored bit fields for neighbors belonging to the previous layer. For that at first the indices have to be sorted in lexicographical order, for the x-cuts sorted according to xyz, for the y-cuts according to yzx and for the z-cuts according to zxy.
Let us say we have sorted according to zxy and want to find a maximum rectangle in the z-cuts. By iterating over all indices with the same z-coordinate z_0 we can now enumerate all y-intervals with the corresponding x-coordinate describing the $z = z_0$ cut in the correct order as input for the rectangle enumeration algorithm (the sweep line proceeds in x direction) as shown in algorithm 2.4
If we want to apply the rectangle enumeration algorithm to the intersection of two neighbored cuts, we do not have to explicitly calculate and store the interval representation of this intersection. We can simply apply again algorithm 2.4 to both cuts, calculate the intersection of the returned intervals on the fly as shown in algorithm 2.5 (exemplary for z-cuts) and use the result as input for the rectangle enumeration

Algorithm 2.4 z-cut interval enumeration

```
1: declare y_min                                  ▷ variable for lower interval bound
2: for all cells with sorted indices (x, y, z_0) do
3:     if lower and upper y neighbor belongs to previous layer then
4:         return (x, [y, y]) as next "single cell" interval
5:     else if only lower y neighbor belongs to previous layer then
6:         y_min ← y
7:     else if only upper neighbor belongs to previous layer then
8:         return (x, [y_min, y]) as next interval
9:     end if
10: end for
```

algorithm. "Valid interval" at line 3 means that there was really an interval left when retrieving it.
However, not for all intersections of neighbored pairs of cuts a maximal rectangle has to be searched. If V_{max} is the volume of the largest box found so far, such a rectangle must have an extended area $(w+2d)\cdot(h+2d)$ larger than $V_{max}/(2d+2)$ so that the corresponding box has a larger volume. But if such a rectangle exists in the intersection, rectangles with at least this extended area must also exist in both cuts. Thus the intersection of two neighbored cuts has only to be checked for a maximal rectangle if both cuts contain maximal rectangles with an extended area larger than $V_{max}/(2d+2)$.

2.5.1 Running Time Considerations

To estimate the running time of the described algorithm let us assume that the voxelization is embedded in a $n \times n \times n$ sized cubic grid so that the number of inner cells is in $O(n^3)$. The algorithm consists of two parts. Firstly, there is the BFS of the inner cells and the sorting of the surface cells according to their indices. Because every inner cell is a surface cell in exactly one offset, actually all inner cells are sorted yielding a running time in

$$O(n^3 \log n^3) = O(n^3 \log n) \,.$$

Secondly, there is the searching for maximal rectangles in all cuts of all offsets of the voxelization. Obviously the number of offsets is in $O(n)$ and every offset has a number of cuts in $O(n)$ as well. As established in the previous section the algorithm for finding maximal rectangles has a running time that can be linear in the number of inner cells and the number of inner cells of a cut is obviously in $O(n^2)$. On the whole that means the running time for finding the maximal rectangles is in

$$O(n^4)$$

Algorithm 2.5 z-cut interval intersection enumeration

```
1: (x_1, [y_min,1, y_max,1]) ← first interval of z-cut 1
2: (x_2, [y_min,2, y_max,2]) ← first interval of z-cut 2
3: while (x_1, [y_min,1, y_max,1]) and (x_2, [y_min,2, y_max,2]) are valid intervals do
4:     if x_1 < x_2 then
5:         (x_1, [y_min,1, y_max,1]) ← next interval of z-cut 1
6:     else if x_2 < x_1 or y_max,2 < y_min,1 then
7:         (x_2, [y_min,2, y_max,2]) ← next interval of z-cut 2
8:     else if y_max,1 < y_min,2 then
9:         (x_1, [y_min,1, y_max,1]) ← next interval of z-cut 1
10:    else                          ▷ it is x_1 = x_2 and both y intervals overlap
11:        y_min ← max{y_min,1, y_min,2}
12:        if y_max,1 < y_max,2 then
13:            y_max ← y_max,1
14:            (x_1, [y_min,1, y_max,1]) ← next interval of z-cut 1
15:            return (x_1, [y_min, y_max]) as next interval
16:        else
17:            y_max ← y_max,2
18:            (x_2, [y_min,2, y_max,2]) ← next interval of z-cut 2
19:            return (x_1, [y_min, y_max]) as next interval
20:        end if
21:    end if
22: end while
```

and thus seems to be the dominating part. However, as experimental results showed that will be presented later, actually the BFS seems to be dominating. Probably this is due to the fact that the running time of the algorithm for finding maximal rectangles is usually much shorter than the worst case scenario of linearity in the number of inner cells. Obviously the running time of this algorithm is correlated to the boundary complexity of the cuts and it can be in $O(n)$ instead of $O(n^2)$ for "good" cases. Furthermore it is clear that the surface complexity of the offsets becomes smaller and smaller until no cells are left and thus the boundary complexity of the cuts becomes smaller, too.

Probably there are also algorithms which could find a rectangle with a maximal extended area in a cut asymptotically faster than enumerating all rectangles maximal with respect to inclusion. For the problem of finding an axis-parallel rectangle with maximal common area an $O(m \log^5 m)$ algorithm exists [33] where m is the the number of vertices. The running time of this algorithm can even be improved to $O(m \log^2 m)$ [34]. But since the search for maximal rectangles was experimentally faster than the BFS, there seemed to be no necessity for spending much effort on trying to improve this part of the algorithm.

To get a comparison for the asymptotic running time of $O(n^4)$ we want to consider the running time of a naive implementation. A naive implementation could

for example check every inner cell as lower left front corner of a maximum box and every other cell with coordinates bigger or equal in every component as upper right back corner. This would already result in a running time of $O(n^5 \log n)$ if a binary search for the last coordinate is used. Additionally it has to be checked for every box if it is contained by the voxelization. When using an octree structure describing the voxelization this can be done in $O(\log^3 n)$ leading to a running time of $O(n^5 \log^4 n)$ on the whole. Compared to that a running time in $O(n^4)$ is a distinct improvement.

2.5.2 Technical Details of Parallelization

Of course when implementing the described algorithm for finding a box in a 3D voxelization with maximum volume an interesting question is how it can be parallelized efficiently. The parallelization of finding maximal rectangles in the cuts is straight forward because it can be simply done for different cuts in parallel just like the sorting of the cells' indices according to xyz, yzx and zxy if they are copied or references on the actual indices are sorted.
However, as already mentioned in experimental results, the BFS seemed to be dominating. BFSes are usually problematic to parallelize because that imposes a quite big effort on synchronization. In a common implementation every thread collects the cells for the next BFS step in its own data structure and after every BFS step these data structures have to be reasonably distributed on the threads again. Especially in our case this would be problematic because not only neighbored cells are to be collected but also bit fields stored in a hash map are to be updated.
But there is an elegant alternative using the fact that the number of surface cells of the offsets usually decreases. There are rare cases in which it can increase but it is clear that in most cases it will decrease. After all it finally reaches 0. The idea is to use a thread-safe lock-free hash map in which the next surface cells can be inserted simultaneously and for which the stored bit fields can be updated simultaneously. The hash table storing buckets of cell indices with the same hash value can be preallocated with a size equal to the number of surface cells of the last offset. Because the number of surface cells usually decreases or only increases slightly in rare cases we know in advance that the hash table will be big enough to hold all surface cells of the next offset so that the number of hash conflicts is limited. That is important because a reallocation of the hash table would not be possible in a thread-safe way. Hence only a thread-safe lock-free data structure is necessary for the buckets of cell indices and the corresponding bit fields which can be searched in linear time for existing elements and to which elements can be added. This can be realized in C++11 as a singly-linked list using so-called atomics. C++11 supports atomic integers which can be used to store the bit fields (for 26 neighbors 32 bit integers are sufficient) as well as atomic pointers which can be used as the elements of the hash table storing the address of each singly-linked list's head node resp. a null pointer if the list is empty.

Algorithm 2.6 Thread-safe lock-free singly-linked list manipulation / insertion

```
1  struct Node {
2      Node *next;
3      Index index;
4      std::atomic<int> bitField;
5  };
6  void setBit(std::atomic<Node*> &listHead,
7              const Index &index, int bit)
8  {
9      Node *oldHead = listHead.load();
10     Node *node = oldHead;
11     while (node) {
12         if (node->index == index) {
13             node->bitField.fetch_and(bit);
14             return;
15         }
16         node = node->next;
17     }
18     // index not found => create new node for inserting
19     Node *newHead = new Node{oldHead, index, bit};
20     // must check if head has changed in meantime
21     while (!listHead.compare_exchange_weak(newHead->next,
22                                            newHead)
23     {
24         node = newHead->next;
25         while (node != oldHead) {
26             if (node->index == index) {
27                 // index has been inserted in meantime
28                 node->bitField.fetch_and(bit);
29                 delete newHead;
30                 return;
31             }
32             node = node->next;
33         }
34         oldHead = newHead->next;
35     }
36 }
```

The code for a function setting a bit for an existing cell index in a list resp. adding this cell index with this bit set to the list is shown in algorithm 2.6. The `load` function simply loads and returns the value hold by an atomic and the `fetch_and` function atomically replaces the hold integer by the result of bitwise `AND` of this integer and the integer given as parameter. The third used atomic function is `compare_exchange_weak`. It takes as first parameter an expected value by reference and as second parameter a new value and only replaces the value hold by the atomic and returns true if its previous value is equal to the expected value. If that is not the case, it copies the atomic's value to the expected value (because of that it must be passed by reference) and returns false. The `weak` means that the function might spuriously fail to exchange the atomic value although it is equal to the expected value. There is also a `strong` version of this function for that this cannot occur but the C++11 standard [35] suggests using the `weak` version if there is a loop calling this function anyway. Because if the function fails, it must be searched in the indices inserted in the meantime and then the function called again, that is true for our case.
That a thread-safe lock-free data structure can be used in such an easy way at this point is really remarkable and due to the fact that only a very limited number of access operations is necessary. Already removing the head node of a list as used in algorithm 2.6 is not easily possible in a thread-safe way. At first glance one might think that this could be done by replacing the pointer to the head node by the pointer to its successor with a `compare_exchange` call in a loop. But when a thread loads the pointer to the head node, a second thread might also load it in the meantime, replace it by the pointer to the successor and free the head node's memory. When then the first thread tries to read the value of the pointer to the successor from the head node's memory, it will access freed memory. There are even several other problems in this scenario like the *ABA problem* [36] and they can only be solved by big efforts like implementing an own memory management.

2.5.3 Experimental Running Time Results

In figure 2.17 the calculated boxes for the voxelizations (1 mm resolution) of six different objects can be seen – 4 suitcases, a golf bag and the well-known bunny scaled so it spans 500 mm in its longest direction. Figure 2.18 shows the average running time for calculating them on an Intel® Core™ i7-970 CPU at 3.20 GHz with 6 cores when using 1, 2, 4 and 6 threads. Every run was repeated ten times. As it can be seen the parallelization works quite well for up to 4 threads while when using 6 threads the speed-up compared to using 4 threads is quite low. That could mean that the synchronization overhead becomes dominating or it might be related to the property of modern processors that they can automatically overclock slightly when not using all cores.

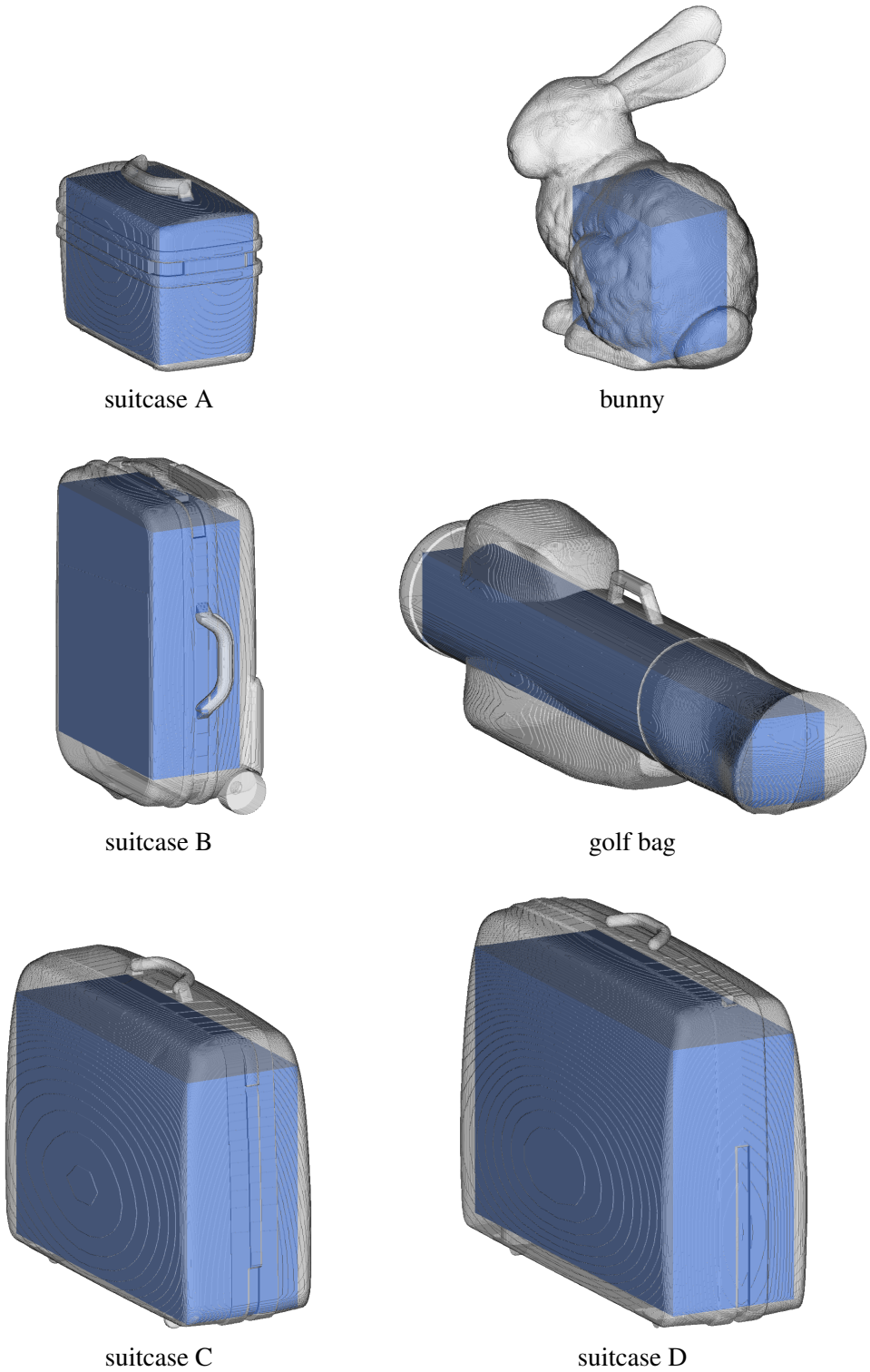


Figure 2.17: Voxelizations of different objects drawn transparently (and smooth-shaded for better visibility) with calculated contained axis aligned boxes with maximum volume for a voxel resolution of 1 mm.

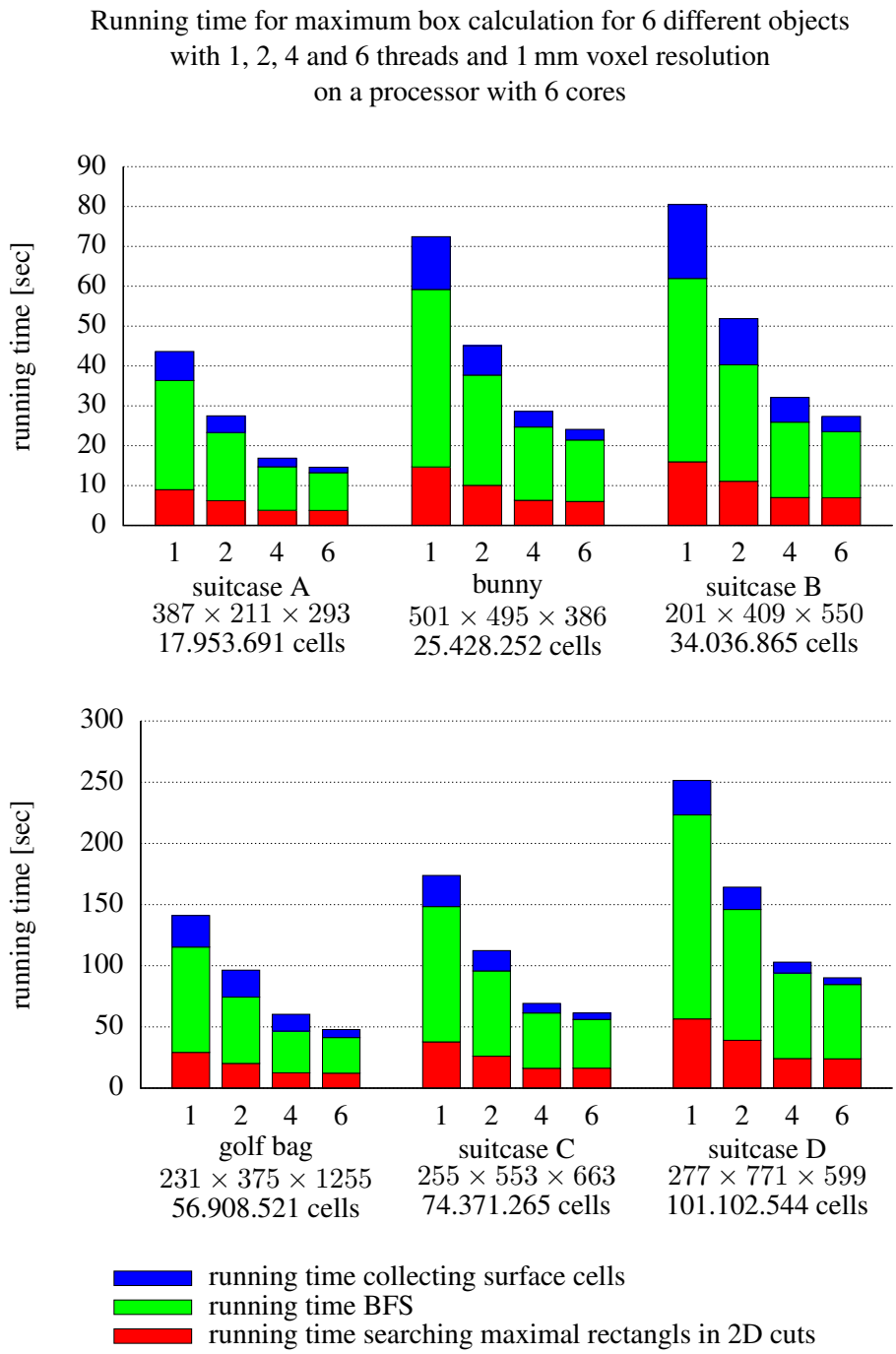


Figure 2.18: Running time for the maximum box calculation for 6 different objects done with 1, 2, 4 and 6 threads. The running time is broken down into the running time for initially collecting the surface cells, the running time for the BFS for calculating the negative offsets (exclusive searching for maximal rectangles in the 2D cuts) and the running time for searching for maximal rectangles in the 2D cuts. Below the name of the objects their bounding box in cell units and their number of inner cells is printed.

2.6 Combined Penetration Estimator

On the whole the penetration of two packing objects A and B is estimated according to the following formula

$$p_{\text{final}}(A, B) := \max\{p_{\text{BoxBox}}(A, B), p_{\text{VMPS}}(A, B), p_{\text{VMPS}}(B, A)\} \tag{2.12}$$

with p_{VMPS} the penetration according to the Voxmap Pointshell approach and p_{BoxBox} the box-box-penetration of the enclosed boxes as described in section 2.3. In figure 2.19 the different penetration (positive values) and distance (negative values) measures for two suitcases are depicted. The red suitcase is moved along one of its principal axes through the other suitcase. For the Voxmap Pointshell penetration resp. distance upper as well as lower bounds are drawn in the graph (same color). Additionally the minimum of the distances between all pairs of triangles is drawn blue in the graph. The final penetration measure according to (2.12) is the maximum of all curves and zero. Figure 2.20 shows the same for two golf bags.
Originally for the Voxmap Pointshell approach only an upper bound for the penetration resp. a lower bound for the distance – which is also an upper bound when considering distances to be negative – is calculated when traversing the bounding sphere hierarchy of the point sampled object (see algorithm 2.1). This is done by taking the maximum upper bound from the adaptive distance field for all sampling points. Additionally a lower bound can obviously be calculated by taking the maximum lower bound from the adaptive distance field for all sampling points and these bounds are additionally printed in figures 2.19 and 2.20 in the same color as the upper bounds.
For larger distances the steps induced by the increasing cells of the voxelization resp. adaptive distance field can be clearly seen. Because the red suitcase resp. golf bag is moved along a principal axis these steps are clearer and bigger when the red suitcase resp. golf bag is voxelized and the green one's surface is sampled by points. When looking at the graph for the suitcases one might think that the box box penetration could be a sufficient penetration measure on its own. But that is only due to the fact that the suitcases have nearly the shape of boxes. For the golf bags the gap between the positions when the golf bags touch each other and the position when the contained boxes touch each other is much bigger.

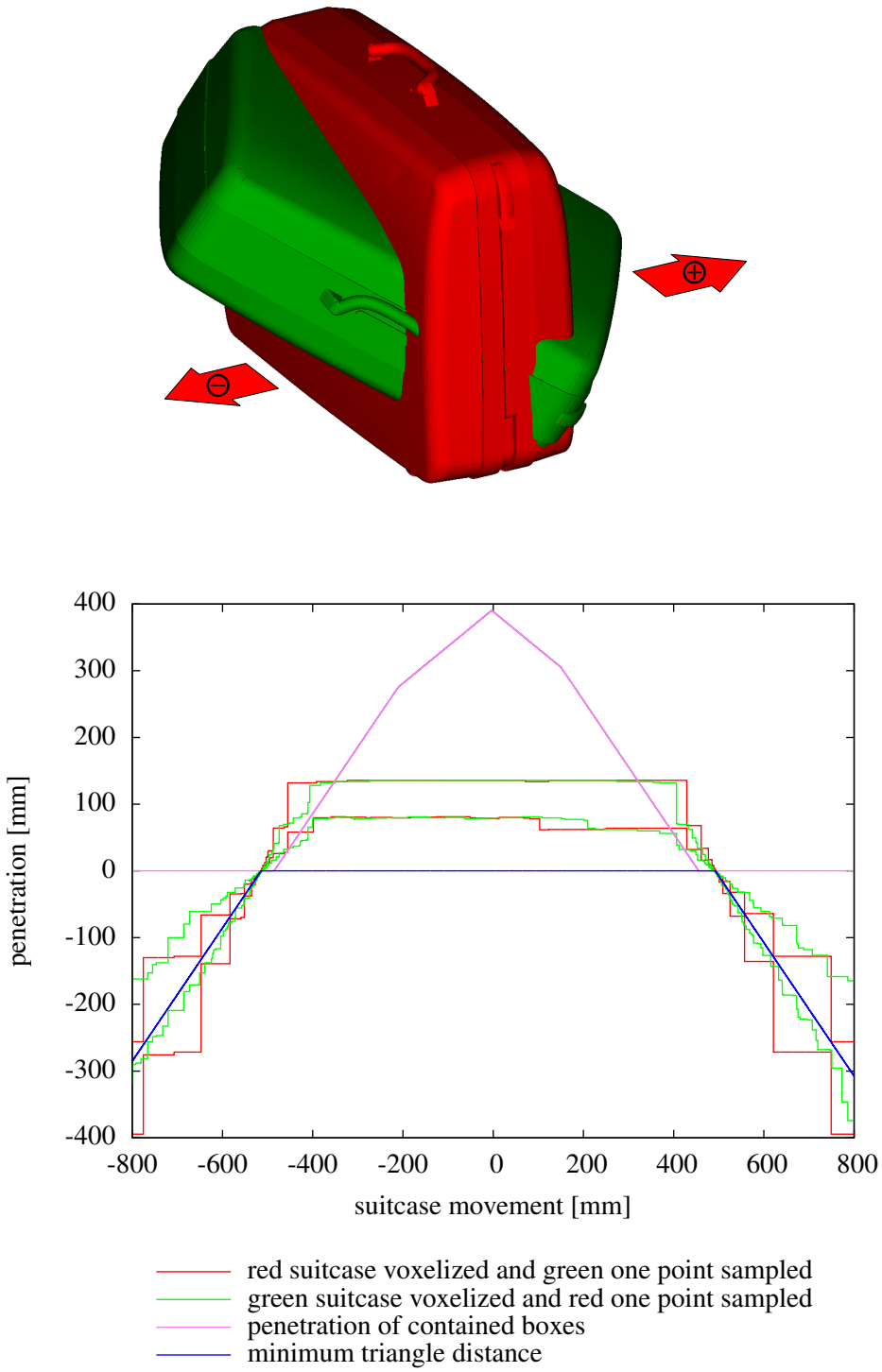


Figure 2.19: Different penetration (positive) and distance (negative) measures of the two depicted suitcases when the red one is moved along the principal axis as shown by the arrows.

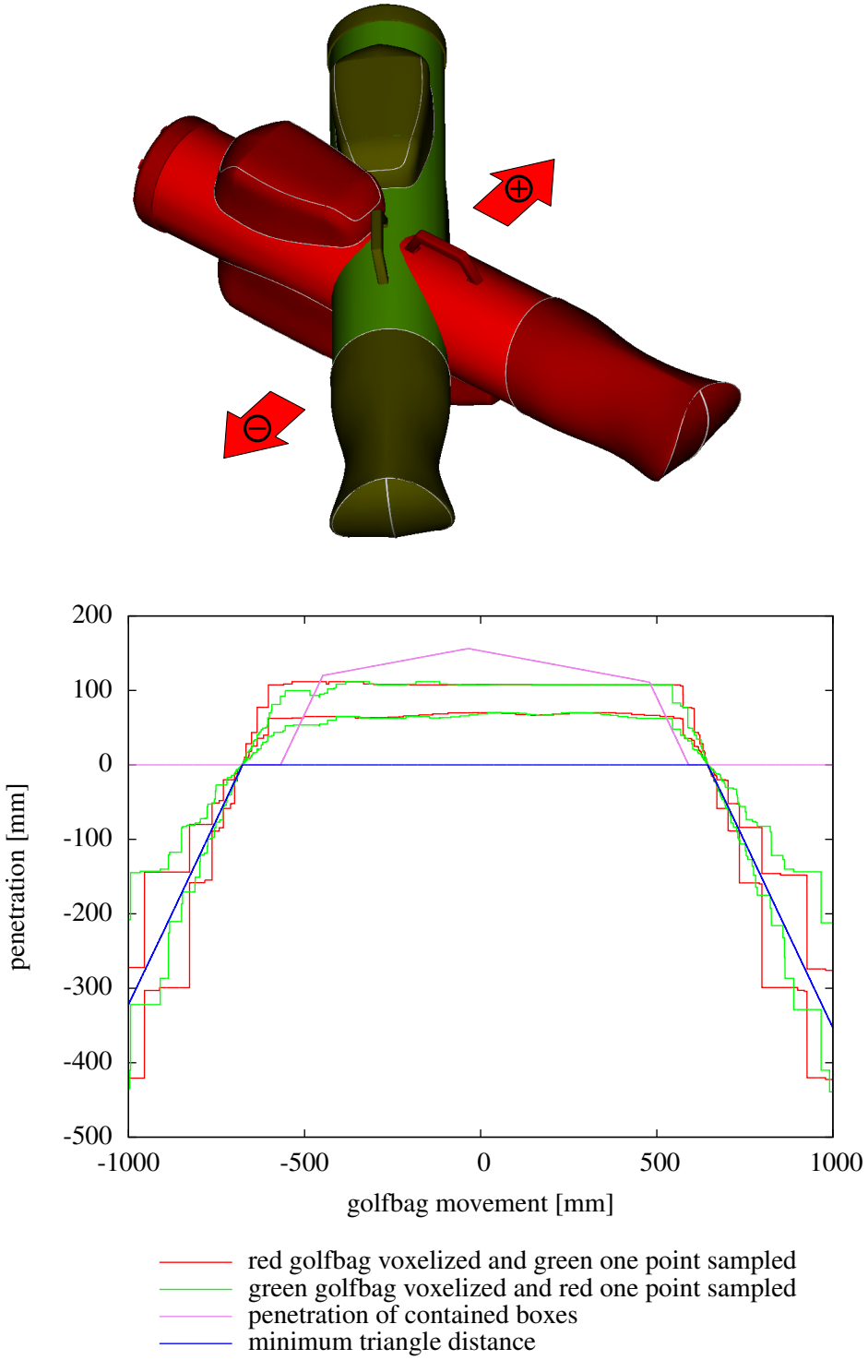


Figure 2.20: Different penetration (positive) and distance (negative) measures of the two depicted golf bags when the red one is moved along the principal axis as shown by the arrows.

2.7 Error Bounds for the Voxmap Pointshell Approach for False Positive and False Negative Collision Tests

When a collision is detected for the Voxmap Pointshell approach, a point of the surface sampling of the one object lies inside the voxelization of the other object. If we assume that the input geometry is described by a closed surface so that its interior is clearly defined, every cell of the voxelization must contain a point belonging to the original input geometry. Thus a point that is contained by the voxelization but not by the original input geometry has a distance of at most $\sqrt{3}d$ to the original input geometry with d the edge length of a voxelization cell on the deepest level. Thus the distance of two objects is bounded by this value for a false positive collision case detected by the Voxmap Pointshell approach.
For the false negative case it has to be considered that in order to keep the running time of a collision check sufficient short for simulated annealing it is not possible to make the surface point sampling so dense that the typical distance of two neighbored surface sampling points has the same magnitude as the cell length of the voxelization. For the false negative case a penetration measure has to be defined that can be bounded. At the beginning of this chapter the following penetration measure has been introduced:

$$p(A,B) := \max\left\{ \sup_{\mathbf{x}\in A\cap B} \inf_{\mathbf{y}\in A\setminus B} |\mathbf{x}-\mathbf{y}|, \sup_{\mathbf{x}\in A\cap B} \inf_{\mathbf{y}\in B\setminus A} |\mathbf{x}-\mathbf{y}| \right\} \tag{2.13}$$

Furthermore we want to define as the r-value of a set of surface sampling points $\mathbf{p}_1, \ldots \mathbf{p}_n \in \mathbb{R}^3$ of an object C the minimum radius so that the surface ∂B of B is completely covered by n balls with the centers $\mathbf{p}_1, \ldots, \mathbf{p}_n$ and the radius r:

$$r := \min\left\{x \in \mathbb{R} \mid \forall \mathbf{q} \in \partial B : \min\left\{|\mathbf{q}-\mathbf{p}_i| \mid i \in \{1,\ldots,n\}\right\} \leq r\right\} . \tag{2.14}$$

With this definition it is clear that a surface point $\mathbf{x}$ of an object A cannot penetrate another object B deeper than r according to the measure

$$p(\mathbf{x},B) := \inf_{\mathbf{y}\in A\setminus B} |\mathbf{x}-\mathbf{y}|$$

without a collision being detected by the Voxmap Pointshell approach.
However, if two objects have sharp features which are thinner than the r-value of the other object, both features can penetrate each other so that no collision is detected by the Voxmap Pointshell approach, the penetration measure according to (2.13) is smaller than r but a translation larger than r is necessary to separate both objects as illustrated in figure 2.21. Besides, the maximum penetration in the false negative case does not have to be achieved for a surface point. For example two icosahedrons with the same center can be rotated so that none of them contains vertices of the other one as shown in figure 2.22. Such cases are excluded by at first checking the center of an innermost voxelization cell of the one object against the voxelization of the other object. But a structure like such an icosahedron might

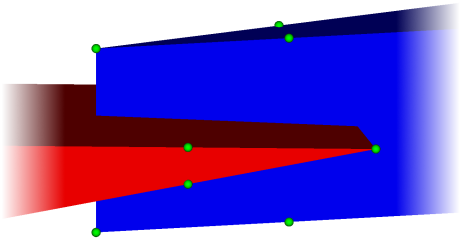


Figure 2.21: Objects with sharp features which are thinner than the maximum distance r between the surface sampling points might have a penetration measure according to (2.13) smaller than r but might require a translation larger than r to get separated. (Consider that the wedges might be extended infinitely to their sides.)

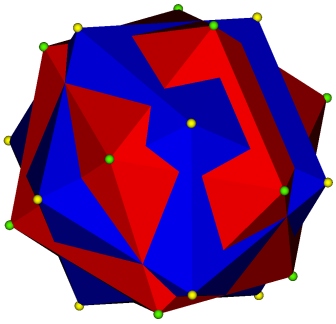


Figure 2.22: Two icosahedron with the same center can be rotated so that none of them contains vertices of the other one. Thus a collision between them might not be detected by the Voxmap Pointshell approach with the vertices as surface sampling points and a voxelization edge length small compared to the icosahedron edge length.

be only a smaller part of the whole object connected by a thin bar which does not contain surface sampling points of the other icosahedron.
Thus it must be stated that for arbitrarily shaped objects which are any kind of compact connected set in $\mathbb{R}^3$ no bounds can be given for the false negative case. To be able to do that at least one of the objects must have a property which means that it does not have any sharp features or necking regions to a certain degree which is related to the density of the surface point sampling of the other object. Let us consider two packing objects A and B which are closed connected sets in $\mathbb{R}^3$. Then we can prevent A from having sharp features or necking regions by demanding that it is the r-offset of an closed connected set A' with r according to (2.14):

$$A = O_r(A') \quad \text{with} \quad O_r(X) := \{\mathbf{p} \in \mathbb{R}^3 \mid \exists\, \mathbf{p}' \in X : |\mathbf{p} - \mathbf{p}'| \leq r\} \tag{2.15}$$

To fulfill such a criterion the sharp features of an arbitrary object can be ground off by applying a negative r-offset and then a positive r-offset again:

$$O_r(O_r^-(A)) \quad \text{with} \quad O_r^-(X) := \{\mathbf{p} \in X \mid |\mathbf{p} - \mathbf{p}'| > r \;\forall\, \mathbf{p}' \in \mathbb{R}^3 \setminus X\}$$

Obviously $O_r(O_r^-(A))$ is a subset of A. However, A must not have necking regions because then its negative offset might not longer be connected.
If A fulfills (2.15), the penetration of both objects is limited in the following way

$$\max_{\mathbf{x} \in A \cap B} \inf_{\mathbf{y} \in A \setminus B} |\mathbf{x} - \mathbf{y}| \leq r \tag{2.16}$$

for the false negative case if we assume that A' is not contained completely by B. In this case the collision should be detected by the reverse Voxmap Pointshell collision check.
This can be proven in the following way: Assume that (2.16) is not fulfilled. This means that there is a point $\mathbf{p}$ in $A \cap B$ so that $|\mathbf{p}-\mathbf{y}| > r$ for all $\mathbf{y} \in A \setminus B$. Because of property (2.15) there is a point $\mathbf{p}' \in A'$ with $|\mathbf{p}-\mathbf{p}'| \leq r$ and hence $\mathbf{p}' \notin A \setminus B$. Thus $\mathbf{p}'$ is in $A \cap B$, too, which means that A' collides with B as well. Because A' is not contained completely by B, there must also be a point $\mathbf{q}$ on the surface of B which is contained by A'. And because of property (2.14) there must be a point $\mathbf{p}_i$, $1 \leq i \leq n$, of the surface sampling of B with $|\mathbf{p}_i - \mathbf{q}| \leq r$. But then $\mathbf{p}_i$ must be contained by A because of (2.15). Thereby the Voxmap Pointshell approach would detect a collision which is a contradiction to that we are in the false negative case. Thus the assumption is wrong and (2.16) must be valid.
However, in the real world application of packing different objects into a trunk such a formal property like (2.15) cannot be assumed to be fulfilled. But objects that are to be packed into a trunk do not have very sharp features because then they were quite likely to damage other objects or get damaged themselves. The same holds for necking regions which were likely to get broken. And thereby the penetration bound (2.16) can be considered not as a proof but as an indication that there will be no severe undetected collision cases when using the Voxmap Pointshell approach for packing common objects into a trunk.
Furthermore the covering of a trunk usually consists of a soft material which can be pushed in by a few millimeters. The same holds for many common packing objects like suitcases made of plastic. Thus for our industrial partner penetrations of a few millimeters are acceptable and in practice it has emerged to be sufficient to use a voxelization resolution of 1 mm on the deepest level for the Voxmap and a voxelization resolution of 8 mm to generate one sampling point per cell for the Pointshell as described in section 2.2.

2.8 Comparison with the PQP Library

To assess the running time potential of our Voxmap Pointshell (VMPS) approach its ability to estimate distances has been compared to the PQP (proximity query package) library [37] which calculates exact distances of the form

$$\text{distance}(A, B) = \min_{\mathbf{a}\in A, \mathbf{b}\in B} |\mathbf{a} - \mathbf{b}| \tag{2.17}$$

for two triangle soups $A, B \subset \mathbb{R}^3$. This is done by PQP by managing the triangles in hierarchies of so-called RSS (rectangle swept sphere) bounding volumes.
Of course, that is in some way comparing apples and oranges because PQP calculates exact distances while the Voxmap Pointshell approach only gives intervals the distance lies within which become larger with increasing distance. Furthermore, the voxelization is a superset of the triangles so that for both triangle soups being

PQP distance range [mm]	number of configurations	VMPS running time [s]	VMPS config. per sec.	PQP running time [s]	PQP config. per sec.	speed-up VMPS / PQP
$0< \ldots \leq 10$	4,767,360	340	14,001	4,345	1,097	12.8
$10< \ldots \leq 20$	4,926,271	351	14,052	4,479	1,100	12.8
$20< \ldots \leq 30$	5,091,176	441	11,542	4,817	1,057	10.9
$30< \ldots \leq 40$	5,250,850	538	9,755	5,325	986	9.9
$40< \ldots \leq 50$	5,425,688	636	8,529	5,273	1,029	8.3
$50< \ldots \leq 60$	5,591,434	736	7,596	5,381	1,039	7.3
$60< \ldots \leq 70$	5,767,866	878	6,571	5,519	1,045	6.3
$70< \ldots \leq 80$	5,943,668	961	6,186	5,876	1,012	6.1
$80< \ldots \leq 90$	6,117,026	1,135	5,389	6,267	976	5.5
$90< \ldots \leq 100$	6,306,170	1,216	5,186	6,308	1,000	5.2

Table 2.1: Average running time results (10 repetitions) for calculating about 55 million configurations of two suitcases with a distance in the range of $0< \ldots \leq 100$ mm by PQP and the VMPS approach.

very close to each other not the same distance results can be expected and for larger distances the distance according to (2.17) might be slightly larger than the VMPS range. However, a short running time is much more important than exact results when using them for a heuristic like simulated annealing and there seemed to be no better alternatives for comparing existing algorithms to the Voxmap Pointshell approach.

How both measures compare to each other could already be seen in figures 2.19 and 2.20 with the blue line describing the distance according to (2.17) and the red and the green line describing the distance ranges of the Voxmap Pointshell approach with the one object voxelized and the other object's surface point sampled and vice versa.

The comparison with PQP was done for 100 random rotations for the two suitcases in figure 2.19, each of which consists of about 15,000 triangles. The translational configuration space was sampled by a grid with a width of 10 mm. For every rotation this grid was scanned for all configurations with a distance between the two suitcases according to (2.17) calculated by PQP in the range $0< \ldots \leq 100$ mm. In overall there were about 55 million configurations.

The configurations were sorted according to the distance calculated by PQP and divided into ten regions. Then the running time for calculating the distance of all configurations by PQP and the VMPS approach for every region was measured ten times on a on an Intel® Core™ i7-970 CPU at 3.20 GHz. The mean value results can be found in table 2.1. The VMPS range was calculated symmetrically just as it is done for the penetration in formula (2.12). That means two ranges were calculated by taking the voxelization of the one suitcase and the surface point sampling of the other *and* vice versa. The bounds of the final range then are the minima of the bounds of the two separate ranges.

The speed-up for the VMPS approach over the PQP library is shown in the last column. Again it must be said that this is no fair comparison as PQP calculates exact distances while the VMPS approach only gives a range the distance lies in and the only purpose of this comparison is to assess the speed of the VMPS approach in some way. As it can be seen for small distances, the speed-up lies in the region of 10 and drops to 5 when the distance increases to 100 mm. The number of configurations calculated per second by the VMPS approach drops in the same way while the number of configurations calculated per second by PQP stays almost constant. The reason for that is probably the decreasing resolution of the adaptive distance field with increasing distance to its surface so that the distance bound given by the adaptive distance field is less sharp and there are fewer early exits when traversing the bounding sphere hierarchy according to algorithm 2.1.
However, for our application actually penetrations are to be calculated which are bounded by the "thickness" of the packing object in some ways. Thereby calculating penetrations by the VMPS approach of objects which are some centimeters "thick" should have a running time behavior similar to calculating small distances in the range of a few centimeters because the resolution of the adaptive distance field depends only on the distance to the voxelization surface and not if it is inside or outside.
Furthermore it has to be considered that the running time of approaches like PQP which use bounding volume hierarchies of the actual triangles also depend on the number of triangles, of course. Thus the running time of PQP would increase for models with more than the 15,000 triangles of the suitcase in this comparison. In opposite to that the VMPS approach does not depend on the input geometry complexity, only the running time for the preprocessing will become longer with increasing input geometry complexity. On the whole by this comparison it should be clear that by the VMPS approach, which accepts a certain degree of inaccuracy, a speed level can be reached which is not attainable by exact approaches using the original triangle geometry.
To get an impression of the degree of inaccuracy figure 2.23 shows the minimum, maximum and average values for the width of the distance ranges calculated by the VMPS approach. To determine these values all configurations were divided into 0.1 mm wide intervals concerning their exact distance. As it can be seen the average width of the VMPS range depends quite linearly on the exact distance which should be a consequence of the refinement criterion (2.8) which is in fact a linear relation between the width of the distance interval and its center.
As already mentioned, because the triangle geometry is replaced by an enclosing voxelization, it cannot be expected that the exact distance calculated by PQP always lies in the VMPS range. Since the voxelization is a superset of the triangle geometry, we have to expect that the VMPS approach detects collisions although there are no original triangles colliding and that the VMPS approach underestimates the distance compared to the exact distance of the original triangles to a certain degree. Furthermore due to the inaccuracy of the VMPS approach depending on the surface point sampling, as explained in section 2.7, the distance might

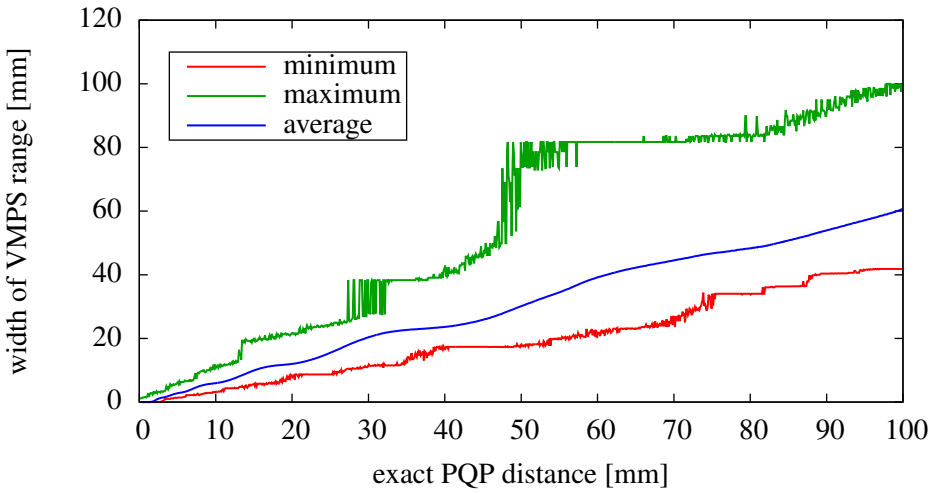


Figure 2.23: Minimum, maximum and average values for the width of the VMPS distance ranges depending on the exact PQP distance. To determine these values all configurations were divided into 0.1 mm wide intervals concerning their exact distance.

also be overestimated in some cases by the VMPS approach.
Just this behavior can be found in figure 2.24(a), which illustrates the number of configurations for the different possible cases up to an exact distance of 20 mm corresponding to about 9.7 million configurations. There are many VMPS collisions (yellow) although no original triangles collide until to an exact distance of 1.52 mm. Up to this distance that accounts for about 44 % of all configurations and 1.52 mm is obviously below the theoretical limit of $\sqrt{3} \approx 1.73$ mm for 1 mm resolution.
As expected, the most frequent deviation is an underestimation of the distance (red). And it is clear that this deviation can occur for any exact distance though it becomes less and less frequent for increasing distance. Below 20 mm it accounts for 27 % of the configurations while above it are only 3.1 ‰. The distribution how much the exact distance lies above the VMPS range can be seen in figure 2.24(b). It falls quite linearly and reaches 1.70 mm at maximum.
As predicted because of the inaccuracy introduced by the surface point sampling, there are also some cases in which the exact distance lies below the VMPS range (blue). This occurs up to a distance of 2.77 mm, but accounts for only 4.5 ‰ of all configurations up to this distance. The distribution how much the exact distance lies below the VMPS range can be seen in figure 2.24(c). It seems to fall exponentially and reaches 1.04 mm at maximum.

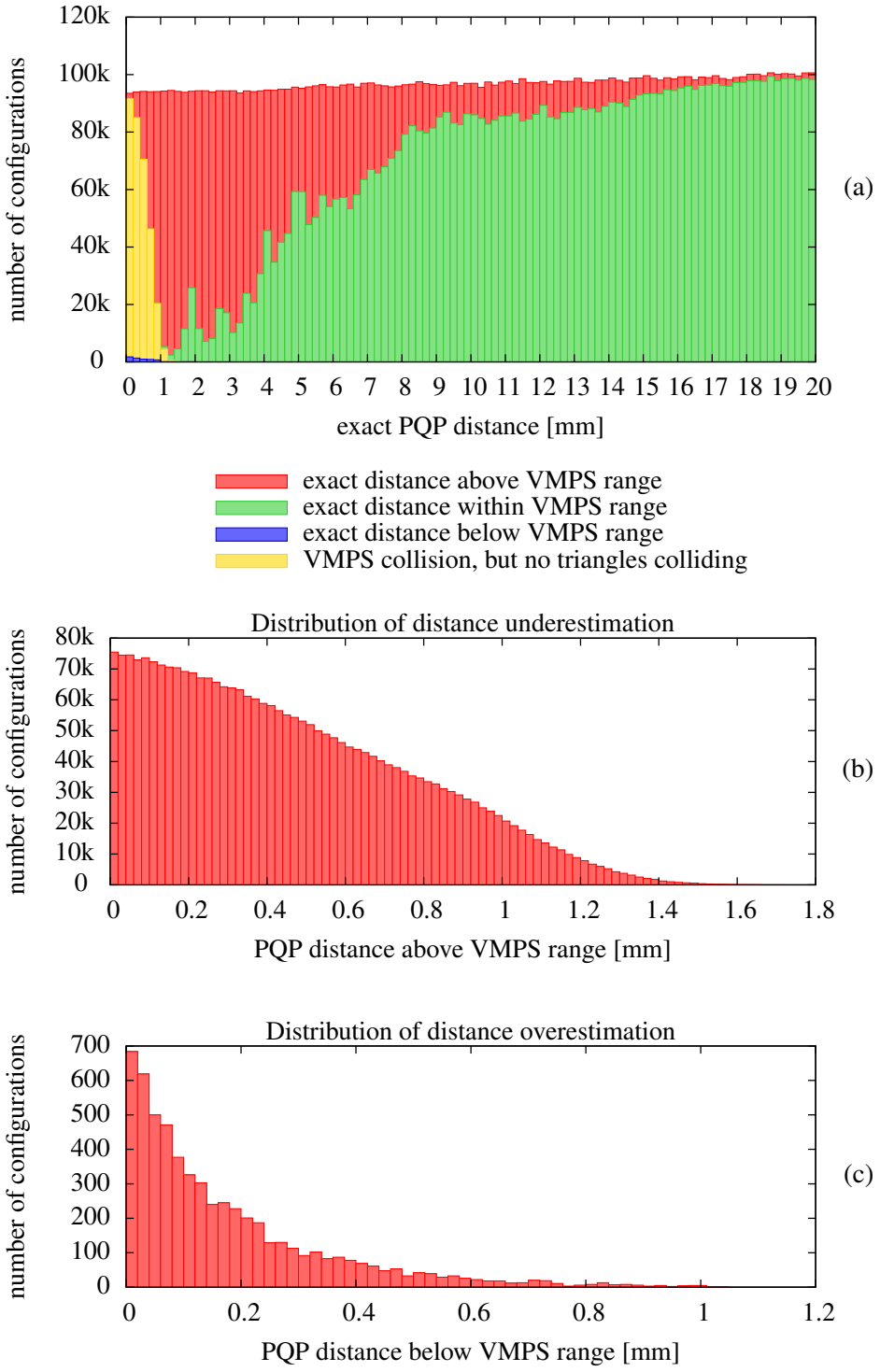


Figure 2.24: Because the original triangle geometry is replaced by an enclosing voxelization, it cannot be expected that the exact distance calculated by PQP always lies in the VMPS range. Figure (a) shows the distribution for the different cases depending on the exact distance and figure (b) and (c) the distribution how much the exact distance lies above and below the VMPS range in these cases.

2.9 Basics About Simulated Annealing and Final Packing Results

Having a fast function for estimating the penetration, simulated annealing can be applied as a standard probabilistic metaheuristic to generate packings as the solution of a combinatorial optimization problem. As already mentioned at the beginning of this chapter, the objective function to minimize is the sum over the penetrations of all pairs of packing objects as well as every packing object and the complement of the trunk resp. container:

$$f:\ \left(SO(3)\times\mathbb{R}^3\right)^n\ \rightarrow\ \mathbb{R}\quad,\quad(\mathbf{R}_1,\mathbf{t}_1,\ldots,\mathbf{R}_n,\mathbf{t}_n)\ \mapsto$$

$$\sum_{i=1}^{n} p(P_i(\mathbf{R}_i,\mathbf{t}_i),\mathbb{R}^3\setminus C)+\sum_{i=1}^{n}\sum_{j=i+1}^{n} p(P_i(\mathbf{R}_i,\mathbf{t}_i),P_j(\mathbf{R}_j,\mathbf{t}_j)) \tag{2.18}$$

$$\text{with}\quad P_i(\mathbf{R}_i,\mathbf{t}_i)=\left\{\mathbf{x}\in\mathbb{R}^3\,\middle|\,\mathbf{R}_i^T(\mathbf{x}-\mathbf{t}_i)\in P_i\right\}$$

The basic idea of simulated annealing is to make a random walk through the configuration space by making random moves, accept all moves which reduce the objective function f and reject resp. undo all moves which increase the function f with a certain probability. The probability P to accept a f increasing move is calculated by the so-called Metropolis criterion [38]:

$$P=\min\left\{1,\exp\left(-\frac{\Delta f}{T}\right)\right\}\quad\text{with}\quad\Delta f=f_{\text{after move}}-f_{\text{before move}}$$

The Metropolis criterion is physically motivated and T is the absolute temperature that starts at a certain value and is successively reduced towards 0 so that P goes to 0, too, and at the end of the simulation only f decreasing moves are accepted. In the physical model Δf is the energy difference between the state after and before the move divided by the Boltzmann constant. The basic idea is that at first, when the temperature is high, f increasing moves are accepted quite often so that the random walk is not caught up in a local minimum and a configuration is found which might still have a high f value but has already a good structure at a coarse level. Later when the temperature becomes lower the f value can be heavily reduced based on this good coarse structure by optimizing the configuration in its details. In this way simulated annealing can also be considered as some kind of divide and conquer approach. Further details can be found in [24].
The temperature is usually decreased during the simulation in an exponential manner and for a certain temperature a certain number of moves is done. Since the penetration sum is divided by the temperature, the initial temperature must be related in some way to penetrations typically occurring. Thus, as basic initial temperature half the radius of the biggest smallest enclosing spheres of all packing objects was used. On a CPU with several cores one simulation per core is started with an initial temperature of 100 %, 50 %, 200 %, 25 %, 200 % and so on of this basic value. The

temperature is decreased by a fixed factor of 0.9 when a certain number of moves have been applied at this temperature. The number of moves per temperature is continuously adapted to the remaining running time as an overall running time can be given by the user which is roughly tried to be met.
At the end the configuration with the smallest penetration sum of all configuration that occurred throughout all simulations is presented as the final packing result. Ideally this smallest penetration sum should be zero if the packing object could really be packed into the container. When a valid configuration is found with a zero penetration sum all simulations are stopped, of course.
An important ingredient of simulated annealing is the move set. That is the set of moves one of which is randomly chosen and applied to modify the current configuration on the random walk through configuration space. The move set used for packing a container with different packing objects was:

- a **big translation** of one randomly chosen packing object by a translation vector with components uniformly distributed in $(-0.5d, 0.5d)$ with d the length of the bounding box's diagonal, that encloses the container – for small random values this might be in fact a small translation,
- a **small translation** that is randomly chosen just like a big translation, but by a translation vector with components uniformly distributed in $(-0.1d$,$0.1d)$,
- a **full rotation** of one randomly chosen packing object by taking a rotation quaternion uniformly distributed over $S^3 = \{\mathbf{q} \in \mathfrak{Q}^* \,|\, |\mathbf{q}| = 1\}$ with $\mathfrak{Q}^*$ the set of unit quaternions and converting it to a rotation matrix.
- a **small rotation** of one randomly chosen packing object by an angle (nearly) uniformly distributed in $(0, 10°)$ around a random axis uniformly distributed over $S^2 = \{\mathbf{v} \in \mathbb{R}^3 \,|\, |\mathbf{v}| = 1\}$,
- a **swap** of two randomly chosen packing objects by swapping the centers of their enclosing spheres as well as their orientation in relation to their principal axes.

In every step of the random walk during simulated annealing one of these moves is chosen randomly uniformly distributed and the packing object or objects to which the move is applied are also chosen randomly uniformly distributed.
For generating a rotation by a random angle uniformly distributed in $(0, 10°)$ the cosine of half the angle would have to be calculated in order to generate a rotation matrix from a proper rotation quaternion. To avoid this expensive trigonometric function instead the function $1 - x^2$ with x uniformly distributed in $(0, \sqrt{1 - \cos(5°)})$ is used which is nearly the same because of $\cos x = 1 - {}^{x^2}/_2 + O(x^4)$ but does not result in a perfect equal distribution.
For the final packing a resolution of 1 mm for the voxelization was used and the surface point sampling was generated by an 8 mm voxelization with one sampling

	initial "temperature" range [mm]	final "temperature" range [mm]	number of temperature steps range
scenario A	97.4− 779.0	0.08−1.02	50−88
scenario B	80.4− 643.1	0.46−5.61	35−66
scenario C1	158.5−1268.4	0.23−1.85	49−80
scenario C2	158.5−1268.4	0.08−0.10	71−92
scenario D1	158.5−1268.4	0.82−6.54	40−65
scenario D2	158.5−1268.4	0.08−0.09	72−92

Table 2.2: Ranges for the initial and final temperatures and number of temperature steps for the simulated annealing runs of the different scenarios. Because the penetration sum as objective function is a length, the temperatures must be lengths, too.

	actual running time [h]	number of moves per temperature	per second	move acceptance rate [%]	range of final penetration sum [mm]
scenario A	7.58	403571	1088	35.2	0.0− 3.9
scenario B	5.05	468206	1226	41.7	0.0− 0.0
scenario C1	6.11	360070	1013	40.8	0.0− 0.0
scenario C2	8.00	223673	634	34.4	9.1−34.9
scenario D1	4.77	253609	760	40.9	0.0− 0.0
scenario D2	8.00	201588	574	27.5	8.9−89.1

Table 2.3: Average values for the actual running time, the number of moves per temperature and per second, the percentage of accepted moves and the range of the final penetration sum for all runs.

point per voxelization cell (see section 2.2) since a precision around 1 mm is usually sufficient for our industrial partner. For the container resp. the trunk no voxelization resp. adaptive distance field is calculated but only a surface point sampling because the container is usually too big for calculating a voxelization. That is no problem because usually the container geometry resp. the trunk does not have sharp features in its interior, and it has a soft coating so that for our industrial partner usually penetrations with the container geometry of even some millimeters are acceptable.
In figure 2.25 the packings for 6 scenarios generated on an Intel® Core™2 Extreme QX9650 CPU at 3.00 GHz with 4 cores by simulated annealing runs scheduled for 8 hours are shown. The runs were repeated 5 times. For scenario A, B, C1 and D1 valid packings could be generated. For scenario B, C1 and D1 all runs yielded valid packings while for scenario A that was only for one run the case, but the other runs finished with only small penetration sums up to 3.9 mm, which is usually tolerable by our industrial partner.
Scenario C2 is the same trunk as scenario C1 but with another set of suitcases and one golf bag. For this set no valid packing could be generated. The best run yielded

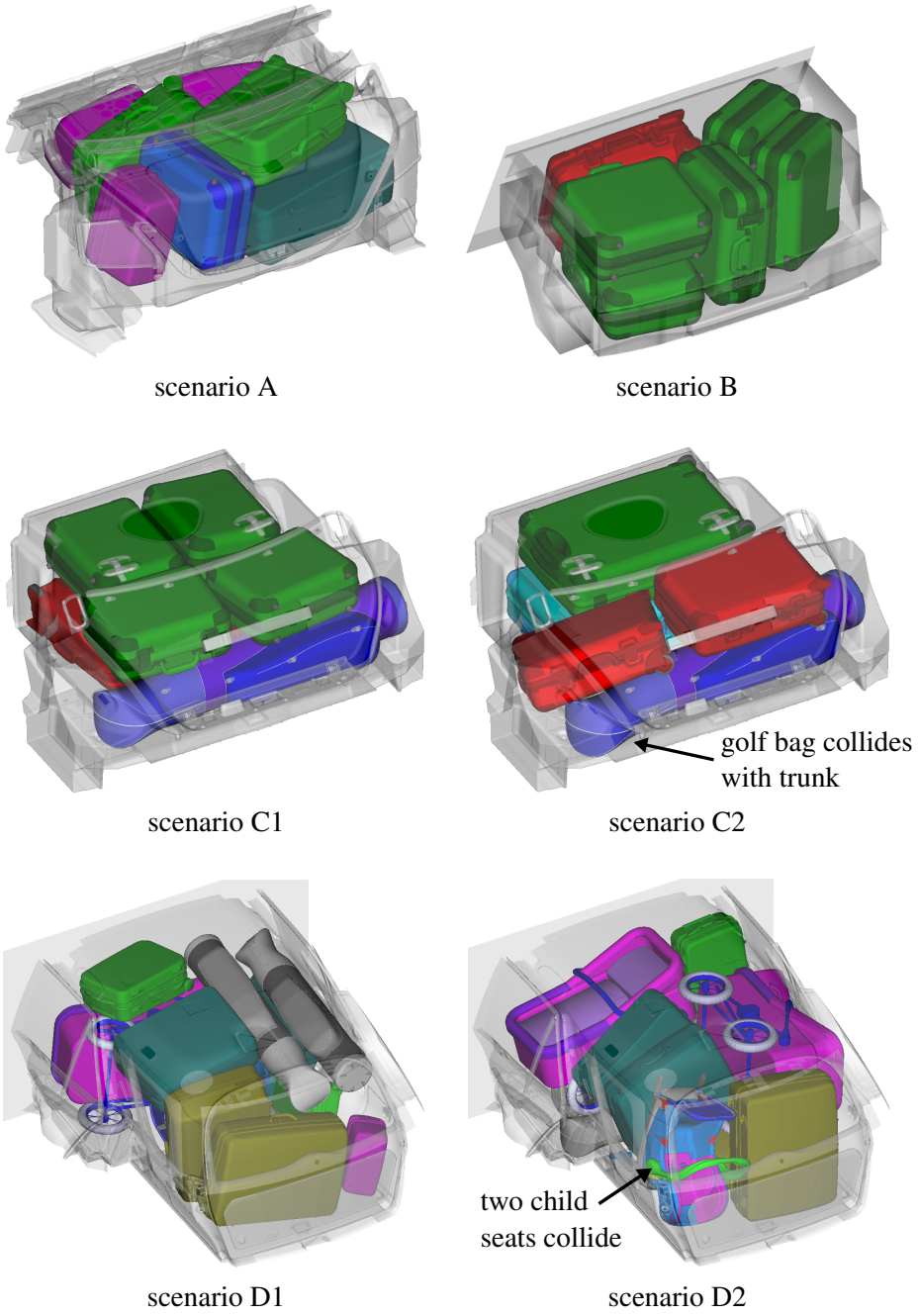
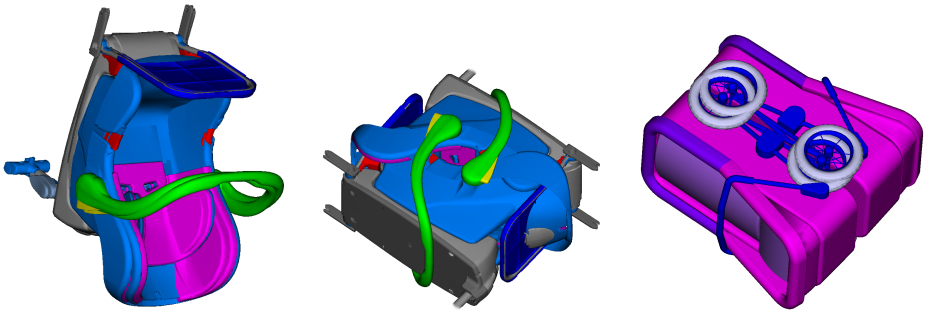


Figure 2.25: Packing results for different packing scenarios. Scenario C1 and C2 show the same trunk with a golf bag and different types of suitcases. With the larger suitcases in scenario C2 the golf bag seems to no longer fit into the trunk. The packing in scenario D1 is legal while that one in scenario D2 is not because it contains two child seats which are stacked on each other with penetration (see figure 2.26(a)).



(a) (b) (c)

Figure 2.26: Objects that are thin and for which no large enclosed box can be calculated by the algorithm of section 2.5 might end up in positions with small penetration where they do actually not fit. Thus, two child seats were stacked on each other (a) (look at the double contours) with small penetration or interweaved so that they collided with the carry handle of the other seat (b), and two baby carriages were packed with their wheels overlapping (c).

a final penetration sum of 9.1 mm. Thus it can be concluded that the set of suitcases in scenario C1 fits into the trunk while that one in scenario C2 does not.
The set of packing objects in scenario D1 includes, besides suitcases and a golf bag, a child seat and a baby carriage. Scenario D2 is again the same trunk, but the packing objects include two child seats and two baby carriages. For this scenario no valid packing could be generated. The final penetration sums were in the range from 8.9 mm to 89.1 mm. A main reason that no valid packing could be generated for this scenario seems to be that, if there are two packing objects of the same shape which is not similar to a box, like child seats or baby carriages, these objects are likely to get finally into a position which has only a small penetration sum but can not become legal by a small move. Such positions, which occurred after runs for scenario D2, are illustrated in figure 2.26.
Table 2.2 contains some statistics about the temperatures of the simulated annealing runs. The ranges result from the fact that on the four cores four simulated annealings are computed in parallel with different initial temperatures. Table 2.3 contains the average values for the actual running time, the number of moves per temperature and per second, the move acceptance rate and the range of the final penetration sum. For the scenarios for which valid packings could be generated the actual running time is smaller than 8 hours because a run is aborted as soon as a valid packing has been found. The number of moves per temperature are in the range of several hundred thousands and per second in the range from 500 up to 1000. To achieve such move numbers is crucial for simulated annealing to work and the consequence of the fast approximative penetration calculation, which is the basic workhorse doing most of the work under the hood of this packing algorithm.
In figure 2.27(a) the move acceptance rate for every temperature step is illustrated for one of the best runs for every scenario and in figure 2.27(b) the current pene-

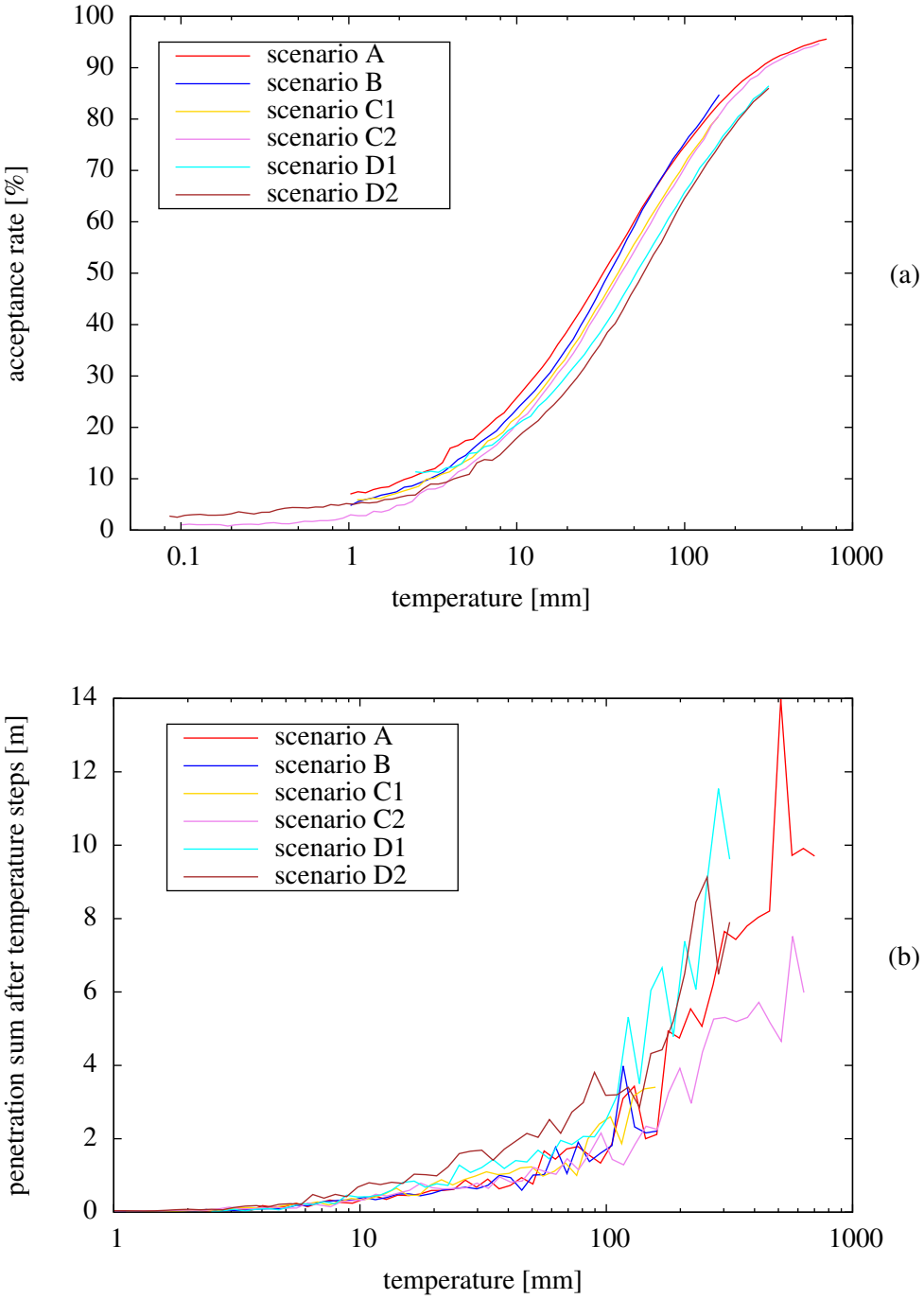


Figure 2.27: Move acceptance rate for every temperature step (a) and penetration sum after every temperature step (b) for one of the best runs for every scenario.

tration sum after every temperature step. Because the temperature is exponentially reduced by multiplying by 0.9, the temperature axis is scaled logarithmically. As it can be seen, the decrease of the acceptance rate is quite similar for all scenarios and the volatility of the penetration sum is large for high temperatures.
On the whole we can conclude that the presented packing algorithm works quite well for packing objects which are similar to boxes, like suitcases and golf bags, but probably needs further improvement for more complex packing objects because of the problems illustrated in figure 2.26. One approach could be to calculate not only one largest enclosed box of the voxelization, but iterate this process by calculating a largest enclosed box for the remaining voxelization and so on. On this way an approximation of the packing objects by boxes could be calculated, for which easily penetrations could be calculated as described in section 2.6.

Chapter 3

Rigid Transformation Hierarchies

3.1 Minimum Distance Calculation for Two Objects on a Rigid Transformation Track

Another field concerning computer aided design (CAD) and modelling (CAM) applications in the automotive industry, besides packing tests for the trunk of a car, are geometrical tests concerning the engine. Thus our industrial partner makes test drives with new car models and measures the movements of the engine during these test drives with different measurement systems. In figure 3.1 a measurement setup, as it is used by our industrial partner, can be seen. By the gained data he checks if the engine is mounted in its flexible bearings in such a way that it cannot touch parts of the casing of the engine compartment and keeps a certain safety distance even during extreme driving maneuvers.
The measurement outputs the exact position of the engine within the engine compartment in its rest position as well as a sequence of rigid transformations describing the movement of the engine during the test drive, which we want to denote as transformation track. Because the movement of the engine is measured with quite a high sampling rate of up to 200 Hz, the sequence of rigid transformations can become quite large and consist of several hundred thousand transformations. The check if a certain safety distance between engine and casing of the compartment is kept was done with this data by standard CAD software so far with an unsatisfactory performance.
Therefore a software has been developed for our industrial partner which calculates the minimum distance between a set of fixed parts belonging to the casing and a set of moving parts belonging to the engine, to which the rigid transformations are applied. By applying the standard technique of calculating bounding volume hierarchies (BVHs) [39, 40] for the fixed parts and the moving parts, both described by triangles, and then calculating the distance between both BVHs for all transforma-

Figure 3.1: Setup for measuring the motion of an engine during a test drive as it is used by our industrial partner.

tions an already sufficient performance could be reached. While the calculation of the solution using standard CAD software took several days on a common workstation for typical cases, this could be reduced to some hours for the same cases by the usage of BVHs.
An acceleration like this can be achieved by BVHs because during the distance calculation of two BVHs an upper bound for the distance is calculated and continuously updated and, if two bounding volumes have a distance larger than this upper bound, the triangles contained by these bounding volumes have not to be checked for their distance. Thus there are many early exits when traversing both hierarchies and only quite few triangles have finally to be checked for their distance. This is done only when the leaf level is reached in both hierarchies and then the upper bound is updated by the distance between all triangle pairs of these leaves. After traversing both hierarchies completely the upper bound becomes sharp and is just the distance between both BVHs resp. the minimum distance between both set of triangles the BVHs contain.
There are several different types of bounding volumes which can be used for BVHs. The most common ones are:

- spheres
- axis aligned bounding boxes (AABB)
- orientated bounding boxes (OBB)
- rectangles swept spheres (RSS), which are the Minkowski sum of a rectangle and a sphere

For AABBs and spheres a smallest enclosing bounding volume can be calculated exactly pretty fast. For AABBs this is trivial and for spheres this is done by the Welzl algorithm [29]. But for OBBs and RSS this cannot be done so easily. Especially it is not clear what smallest should mean for OBBs and RSSs although

Figure 3.2: A typical scenario with some parts belonging to the engine (tube shaped) and some parts of the engine compartment casing (sheets) for which it must be checked if they keep a safety distance when the engine moves during a test drive.

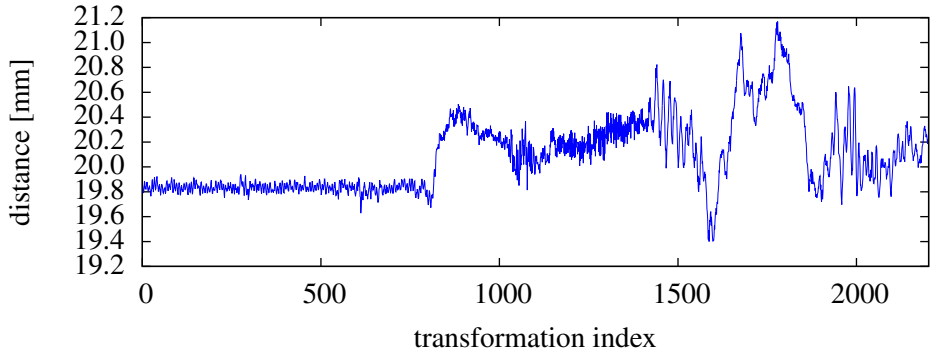


Figure 3.3: Distance profile for the above geometry for an about 11 seconds long drive-off maneuver consisting of 2202 transformations.

volume might be the most obvious scalar property to minimize. Thus for BVHs only "small" OBBs or RSSs for a set of points or triangles are usually calculated by some kind of heuristic, and the most common heuristic is to calculate the orientation of an OBB or RSS for a set of points or triangles by the principal axes of their covariance matrix in their centroid system.
In figure 3.2 a typical scenario with some parts belonging to the engine and some parts belonging to the casing of the engine compartment of a car can be seen for which it must be checked if they keep a certain safety distance. In figure 3.3 the distance profile of these parts can be seen for an about 11 seconds long drive-off maneuver during a test drive consisting of 2202 transformations. To generate the distance profile for every transformation the distance is calculated by using BVHs for the fixed and the moving geometry, which are calculated in a preprocessing step.
Our industrial partner is not only interested in the closest approach during the test drive, which would be sufficient to check a certain safety distance, but all constrictions are to be examined by an expert so that he can draw conclusions during the car's design process if the engine or the casing design has to be revised. Because of this it is also calculated which points on which triangles get closest to each other. And because so far no criteria have been specified when a constriction is relevant and when not, a complete exact distance profile is calculated as shown in figure 3.3 and presented by the software.
However, it is clear that in the future there will be additional requirements on the software which will make a speed-up of the distance calculation at least desirable if not inevitable. So far there seems to be no previous work about rigid transformation tracks how this could be done. So far only numerous publications about swept volumes exist which are a representation of the volume that a geometry sweeps if a track of rigid transformation is applied to it. A survey of publications about swept volumes can be found in [41].
Yet those approaches do not suite well for our application. Firstly, for just calculating the minimum distance between fixed and moving geometry during a track the effort for calculating a swept volume is much too high and it would be like cracking a nut with a sledgehammer. Secondly, our industrial partner is also interested in when the closest point is reached during a track, but just this information gets lost when calculating a swept volume.
An acceptable compromise in order to achieve an acceleration of the calculation could be dropping the requirement for calculating a complete exact distance profile. And an obvious question is whether the approach of calculating some kind of BVH, as it is done for the geometry, can be transferred to the transformations as well, especially because the engine makes only quite small moves during the test drive, which can be called a vibration movement.
If the engine was only shifted during the test drive and the sequence of transformations $T_1, \ldots, T_n$ consisted of only translations by translations vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$

$$T_i : \mathbb{R}^3 \to \mathbb{R}^3 , \mathbf{x} \mapsto \mathbf{x} + \mathbf{b}_i \quad , \quad 1 \leq i \leq n ,$$

the solution would be clear. A BVH of bounding spheres could be build up over the translation vectors. For example, if $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are contained by a sphere with center $\mathbf{b}$ and radius r, which can be calculated with minimum radius by the Welzl algorithm [29], instead of $T_1, \ldots, T_n$ the translation $T : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \mathbf{x} + \mathbf{b}$ can be applied to any $\mathbf{x} \in \mathbb{R}^3$ leading to an error of at most r concerning the Euclidean norm:

$$|T_i(\mathbf{x}) - T(\mathbf{x})| = |\mathbf{x} + \mathbf{b}_i - (\mathbf{x} + \mathbf{b})| = |\mathbf{b}_i - \mathbf{b}| \leq r \ , \ 1 \leq i \leq n \ . \tag{3.1}$$

However, the engine is not only shifted during the test drive but also makes tilt movements which can be described by rotations $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathrm{SO}(3)$ additionally to the translation vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$. Thus we have not only translations but a sequence of rigid transformations $T_1, \ldots, T_n$:

$$T_i \ : \ \mathbb{R}^3 \to \mathbb{R}^3 \ , \ \mathbf{x} \mapsto \mathbf{R}_i\mathbf{x} + \mathbf{b}_i \quad , \quad 1 \leq i \leq n$$

However, similar to the "mean" translation according to (3.1) by taking the center of the enclosing sphere of the translation vectors we can try to find some kind of mean rigid transformation $T : \mathbb{R}^3 \to \mathbb{R}^3$, $\mathbf{x} \mapsto \mathbf{R} + \mathbf{b}$ so that we can bound the error when applying T to any $\mathbf{x} \in \mathbb{R}^3$:

$$\begin{aligned} |T_i(\mathbf{x}) - T(\mathbf{x})| &= |\mathbf{R}_i\mathbf{x} + \mathbf{b}_i - (\mathbf{R}\mathbf{x} + \mathbf{b})| = |(\mathbf{R}_i - \mathbf{R})\mathbf{x} + (\mathbf{b}_i - \mathbf{b})| \\ &\leq \|\mathbf{R}_i - \mathbf{R}\|_2|\mathbf{x}| + |\mathbf{b}_i - \mathbf{b}| \end{aligned} \tag{3.2}$$

In the last step the triangle inequation was applied and the consistency of the spectral norm and the Euclidean norm was used. Again we can minimize the latter term in (3.2) over all $1 \leq i \leq n$ by calculating the smallest enclosing sphere of the translation vectors $\mathbf{b}_i$, $1 \leq i \leq n$, and take its center for $\mathbf{b}$. Minimizing the first term over all $1 \leq i \leq n$ leads to the minimization problem

$$\operatorname*{arg\,min}_{\mathbf{R} \in \mathrm{SO}(3)} \max_{i \in \{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{R}\|_2 \tag{3.3}$$

for the "mean" rotation $\mathbf{R}$. To find a way how this minimization problem can be solved, it is helpful to understand what the expression $\|\mathbf{R}_1 - \mathbf{R}_2\|$ for two rotation matrices $\mathbf{R}_1, \mathbf{R}_2 \in \mathrm{SO}(3)$ means geometrically. This is clarified by the following proposition:

Proposition 3.1 *Let $\mathbf{R}_1, \mathbf{R}_2 \in \mathrm{SO}(3)$ be two rotation matrices and φ the function that maps a rotation matrix to its rotation angle:*

$$\varphi : \mathrm{SO}(3) \to [0, \pi] \quad , \quad \mathbf{R} \mapsto \arccos\left(\frac{\mathrm{trace}(\mathbf{R}) - 1}{2}\right) \tag{3.4}$$

As the angle $\delta(\mathbf{R}_1, \mathbf{R}_2) \in [0, \pi]$ between $\mathbf{R}_1$ and $\mathbf{R}_2$ we define the rotation angle of the rotation $\mathbf{R}_1^T\mathbf{R}_2$:

$$\delta(\mathbf{R}_1, \mathbf{R}_2) := \varphi(\mathbf{R}_1^T\mathbf{R}_2) = \arccos\left(\frac{\mathrm{trace}(\mathbf{R}_1^T\mathbf{R}_2) - 1}{2}\right) \tag{3.5}$$

Then it is

$$\|\mathbf{R}_1 - \mathbf{R}_2\|_2 = \sqrt{2(1 - \cos(\delta(\mathbf{R}_1, \mathbf{R}_2))} = \sqrt{3 - \mathrm{trace}(\mathbf{R}_1^T \mathbf{R}_2)} \tag{3.6}$$

with $\|.\|_2$ *the spectral norm.*

Proof:
Because the trace is invariant under basis transitions and every rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ with a rotation angle $\varphi \in [0, \pi]$ can be transformed to a rotation around the z axis by a proper basis transition, it is

$$\mathrm{trace}(\mathbf{R}) = \mathrm{trace} \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} = 2\cos\varphi + 1 \,, \tag{3.7}$$

which explains (3.4). Since the Euclidean norm and thus the spectral norm are invariant under rotations, it is

$$\|\mathbf{R}_1 - \mathbf{R}_2\|_2 = \|\mathbb{1} - \mathbf{R}_1^T \mathbf{R}_2\|_2 \,.$$

By applying the basis transition trick in (3.7) to $\mathbf{R}_1^T \mathbf{R}_2 \in \mathrm{SO}(3)$ we get denoting $\delta(\mathbf{R}_1, \mathbf{R}_2)$ briefly by δ:

$$\|\mathbf{R}_1 - \mathbf{R}_2\|_2 = \Bigg\| \underbrace{\begin{pmatrix} 1 - \cos\delta & \sin\delta & 0 \\ -\sin\delta & 1 - \cos\delta & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{=:\mathbf{M}} \Bigg\|_2$$

The spectral norm of matrix $\mathbf{M}$ can be calculated as the square root of the largest eigenvalue of $\mathbf{M}^T\mathbf{M}$, and because of

$$\mathbf{M}^T\mathbf{M} = \begin{pmatrix} 2(1 - \cos\delta) & 0 & 0 \\ 0 & 2(1 - \cos\delta) & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

we have shown (3.6)

□

Because the function $f(x) = \sqrt{2(1 - \cos(x))}$ is strictly monotonically increasing in $[0, \pi]$, we get for (3.3):

$$\underset{\mathbf{R} \in \mathrm{SO}(3)}{\arg\min} \max_{i \in \{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{R}\|_2 = \underset{\mathbf{R} \in \mathrm{SO}(3)}{\arg\min} \max_{i \in \{1,\ldots,n\}} \delta(\mathbf{R}_i, \mathbf{R})$$

The next step to get a geometric view of the angle between two rotations is to switch to rotation quaternions. It is well-known that the $\mathrm{SO}(3)$ can be uniquely mapped

to the set of unit quaternions $\mathfrak{Q}^*$, which can be expressed by 4-dimensional unit vectors $\mathfrak{Q}^* = \{\mathfrak{q} \in \mathbb{R}^4 \,|\, |\mathfrak{q}| = 1\}$, if we identify antipodal quaternions $\mathfrak{q} \sim -\mathfrak{q}$, i.e. divide $\mathfrak{Q}^*$ by this equivalence relation. The inverse of this mapping has the form

$$\rho : \mathfrak{Q}^* \to \mathrm{SO}(3) \quad , \tag{3.8}$$

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \mapsto \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} .$$

The quaternion multiplication, which corresponds to the matrix multiplication on the $\mathrm{SO}(3)$ side, can be briefly expressed by the cross product in the following way if we split a quaternion $\mathfrak{q} = (q_0, q_1, q_2, q_3)^T$ into a scalar part $q_0 \in \mathbb{R}$ and a vector part $\mathbf{q} = (q_1, q_2, q_3)^T \in \mathbb{R}^3$:

$$\mathfrak{q} \cdot \mathfrak{w} = \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} q_0 \\ \mathbf{q} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} q_0 w_0 - \mathbf{q}^T \mathbf{w} \\ q_0 \mathbf{w} + w_0 \mathbf{q} + \mathbf{q} \times \mathbf{w} \end{pmatrix}$$

Furthermore there is a close relation between the axis angle representation $\boldsymbol{\varphi} = \varphi\hat{\boldsymbol{\varphi}}$ of a rotation around the axis $\hat{\boldsymbol{\varphi}} \in \mathbb{R}^3$, $|\hat{\boldsymbol{\varphi}}| = 1$ by the angle $\varphi \in [0, 2\pi)$ and the corresponding rotation quaternion $\mathfrak{q}(\boldsymbol{\varphi})$. The relation between the axis angle representation and the corresponding rotation matrix is given by Rodrigues' rotation formula [42]:

$$R(\boldsymbol{\varphi}) = \exp(\boldsymbol{\varphi}^\times) = \cos\varphi \, \mathbb{1} + (1 - \cos\varphi) \, \hat{\boldsymbol{\varphi}}\hat{\boldsymbol{\varphi}}^T + \sin\varphi \, \hat{\boldsymbol{\varphi}}^\times \tag{3.9}$$

$$\text{with } \mathbb{1} := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}^\times := \begin{pmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{pmatrix}$$

Since the epimorphism (3.8) from $\mathfrak{Q}^*$ to $\mathrm{SO}(3)$ gives the relation between a rotation quaternion and the corresponding rotation matrix, it can be easily verified that the rotation quaternion

$$\mathfrak{q}(\boldsymbol{\varphi}) = \begin{pmatrix} \cos\frac{\varphi}{2} \\ \hat{\boldsymbol{\varphi}} \sin\frac{\varphi}{2} \end{pmatrix} \tag{3.10}$$

corresponds to the rotation with the axis angle representation $\boldsymbol{\varphi} = \varphi\hat{\boldsymbol{\varphi}}$.
Transposing on the $\mathrm{SO}(3)$ side means negating the vector part of the corresponding rotation quaternion, which is called quaternion conjugation and usually indicated by a star just like for complex numbers. If we denote by $\mathfrak{q}(\mathbf{R})$ the rotation quaternion in $\mathfrak{Q}^*/\sim$ that corresponds to a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ and given two rotation matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ with $\mathfrak{q}(\mathbf{R}_1) = \mathfrak{q} = (q_0, \mathbf{q})^T$ and

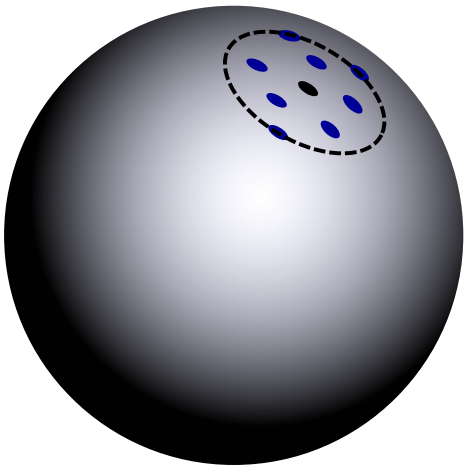


Figure 3.4: Solving the minimization problem

$$\arg\min_{\mathbf{R}\in\mathrm{SO}(3)} \max_{i\in\{1,\dots,n\}} \|\mathbf{R}_i - \mathbf{R}\|_2$$

for a set of rotations matrices $\mathbf{R}_1, \dots, \mathbf{R}_n \in \mathrm{SO}(3)$ means geometrically finding a smallest enclosing circle for properly chosen corresponding rotation quaternions on the quaternion sphere and taking its center. The quaternion sphere can be embedded in 4-dimensional space, but to get a geometric idea the projection to the sphere in 3-dimensional space as illustrated can be considered.

$\mathbf{q}(\mathbf{R}_2) = \mathfrak{w} = (w_0, \mathbf{w})^T$ and δ the angle between them according to proposition 3.1, the following holds because of (3.10):

$$\begin{pmatrix} \cos\frac{\delta}{2} \\ \dots \end{pmatrix} = \mathbf{q}(\mathbf{R}_1^T\mathbf{R}_2) = \mathbf{q}(\mathbf{R}_1)^*\mathbf{q}(\mathbf{R}_2) = \begin{pmatrix} q_0 \\ -\mathbf{q} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix} = \begin{pmatrix} q_0 w_0 + \mathbf{q}^T\mathbf{w} \\ \dots \end{pmatrix}$$

Because we are in $\mathfrak{Q}^*/\sim$, this equation only holds except a sign. However, the angle between two rotations is in the range $[0, \pi]$ per definition, $\cos(\delta/2)$ on the left hand side is positive, and it is clear that we can choose proper representatives for $\mathbf{q}$ and $\mathfrak{w}$ in $\mathfrak{Q}^*/\sim$ so that their scalar product as 4-dimensional vectors $q_0 w_0 + \mathbf{q}^T\mathbf{w}$ on the right hand side is positive as well. This means geometrically that the angle between properly chosen rotation quaternions is just half the angle between the corresponding rotation matrices.
By comparing to (3.5) and since it is $\cos(\alpha) = 2\cos^2(\alpha/2) - 1$, we can conclude

$$\left(\frac{\mathrm{trace}(\mathbf{R}(\mathfrak{w})\mathbf{R}(\mathbf{q})^T) - 1}{2}\right)^2 = 2(\mathbf{q}^T\mathfrak{w})^2 - 1\ .$$

This can be verified by (3.8) directly with a huge computational effort. Thereby the angle between two orientations can be calculated much easier by rotation quaternions than by rotation matrices:

$$\delta(\mathbf{q}, \mathfrak{w}) := \delta\left(\mathbf{R}(\mathbf{q}), \mathbf{R}(\mathfrak{w})\right) = 2\arccos\left(|\mathbf{q}^T\mathfrak{w}|\right) \tag{3.11}$$

Especially by the knowledge of this relation it can be shown quite easily that δ according to (3.5) is actually a metric on $\mathrm{SO}(3)$. While non-negativity, identity of indiscernibles and symmetry can be easily shown by the trace formula, the triangle inequality is much harder to prove. However, by formula (3.11) it can be reduced to the triangle inequality for triangles on the sphere in 4-dimensional space. A

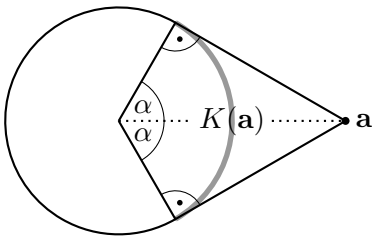


Figure 3.5: Definition of a circle $K(\mathbf{a})$ on the sphere by a point $\mathbf{a}$ outside of (or on) the sphere (here in its two-dimensional projection).

proof for arbitrary dimension can be found as lemma A.2 in the appendix.
Now we have shown that solving the minimization problem

$$\arg\min_{\mathbf{R}\in\mathrm{SO}(3)} \max_{i\in\{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{R}\|_2 = \arg\min_{\mathbf{R}\in\mathrm{SO}(3)} \max_{i\in\{1,\ldots,n\}} \delta(\mathbf{R}_i, \mathbf{R})$$

means geometrically finding for properly chosen corresponding rotation quaternions a smallest enclosing circle on the quaternion sphere as illustrated in figure 3.4.

3.2 The Welzl Algorithm on the Sphere

In the following section we want to show that the Welzl algorithm [29] for calculating the smallest enclosing sphere of a set of points in $\mathbb{R}^d$ can be transferred to the d-sphere, the surface of a d-dimensional ball with radius 1. At first we want to state how to define circles on the d-sphere mathematically:

Definition 3.2 *Let $S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d \,|\, |\mathbf{x}| = 1\}$, $d \geq 2$, be the d-dimensional sphere with radius 1 and $\mathbf{a} \in \mathbb{R}^d$, $|a| \geq 1$, a point outside of or on the sphere. As circle on the sphere for the point $\mathbf{a}$ we define the set:*

$$K(\mathbf{a}) := \left\{\mathbf{x} \in S^{d-1} \,|\, \mathbf{a}^T\mathbf{x} \geq 1\right\} .$$

As a measure for its size we define its half "opening angle" $\alpha \in [0, \pi/2)$ as

$$\alpha := \arccos\left(\frac{1}{|\mathbf{a}|}\right)$$

Because the arc cosine is strictly monotonically decreasing on $[0, \pi]$, the opening angle of a circle rises with increasing norm of $\mathbf{a}$, this means the circle becomes larger.

The geometric relation between $\mathbf{a}$, $K(\mathbf{a})$ and α is illustrated by figure 3.5 as 2-dimensional projection. Lines through $\mathbf{a}$ and boundary points of the circle $K(\mathbf{a})$

are just tangents of the sphere, what becomes clear by the cathetus theorem. The circle $K(\mathbf{a})$ spans a cone over the center of the sphere with half an opening angle of α.
Consider that for $|\mathbf{x}| \to \infty$ $K(\mathbf{x})$ approaches a hemisphere and the boundary of $K(\mathbf{x})$ approaches a great circle with opening angle π, but never reaches it. Thus two antipodal points can obviously not be contained by a circle according to definition 3.2, but need a hemisphere to be contained by. A hemisphere containing two antipodal points $\mathbf{x}$ and $-\mathbf{x} \in S^{d-1}$ is not unique because all hemispheres whose dividing plane contains the line through $\mathbf{x}$ and $-\mathbf{x}$ contain both points. Thus, a smallest circle containing two antipodal points cannot be defined uniquely anyway.
Actually the smallest enclosing circle of a set of points on the d-dimensional sphere could also be calculated as the intersection of this sphere and the smallest enclosing sphere of the points in $\mathbb{R}^d$ calculated according to the standard Welzl algorithm. However, transferring the Welzl algorithm to the sphere is more elegant and faster since some things can be better conceived mathematically and the dimension is smaller by one:

Lemma 3.3 *Let $\mathbf{p}_1, \ldots, \mathbf{p}_n$ n be linearly independent points on the d-dimensional sphere $S^{d-1} := \left\{\mathbf{x} \in \mathbb{R}^d \,|\, |\mathbf{x}| = 1\right\}$ with $n \leq d$. Then an unique smallest circle $K(\mathbf{x})$ on the d-dimensional sphere exists with the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ on its boundary, i.e. $\mathbf{x}^T\mathbf{p}_i = 1$ for all $1 \leq i \leq n$. We denote this circle by $K_b(\mathbf{p}_1, \ldots, \mathbf{p}_n)$. $\mathbf{x}$ is determined as the solution of the system of equations*

$$\underbrace{\begin{pmatrix} \mathbf{p}_1^T \\ \vdots \\ \mathbf{p}_n^T \end{pmatrix}}_{=:A^T} \mathbf{x} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \tag{3.12}$$

with minimum norm. If $\mathbf{A} = \mathbf{QR}$ is the QR decomposition of the matrix $\mathbf{A}$, then $\mathbf{x}$ can be calculated as $\mathbf{x} = \mathbf{Qz}$ with $\mathbf{z}$ the solution of

$$\mathbf{R}^T\mathbf{z} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

calculated by forward substitution.

Proof:
In principle clear. The matrix equation (3.12) just combines the equations $\mathbf{x}^T\mathbf{p}_i = 1, 1 \leq i \leq n$, which mean that the points $\mathbf{p}_i$ are on the circle boundary, and it might be under-determined because of $n \leq d$. According to definition 3.2 we can take the norm of $\mathbf{x}$ as measure for the size of $K(\mathbf{x})$, the norm of $\mathbf{x}$ must be minimum because we are searching for a smallest circle with the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ on its

boundary. But it is a well-known result that the matrix equation $\mathbf{A}^T\mathbf{x} = \mathbf{b}$ has an unique solution under the condition that $\mathbf{x}$ has minimum norm and $\mathbf{A}$ has full column rank. The latter is the case, because $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are linearly independent. Furthermore it is well known, that this unique solution can be calculated by the QR decomposition of $\mathbf{A}$, as described above and can be read in [30], for example.

□

At this point we want to briefly recall how for a set of points $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^d$, $n \leq d+1$, the center $\mathbf{c} \in \mathbb{R}^d$ for a sphere $S = \{\mathbf{x} \in \mathbb{R}^d \,|\, |\mathbf{x} - \mathbf{c}| \leq r\}$ with minimum radius r and the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ on its surface can be calculated, which is necessary for the "classical" Welzl algorithm. For that we can put $\mathbf{p}_1$ to the origin by subtracting it from all points $\tilde{\mathbf{p}}_i := \mathbf{p}_i - \mathbf{p}_1$, $2 \leq i, j \leq n$, so that the Euclidean norm of $\tilde{\mathbf{c}} := \mathbf{c} - \mathbf{p}_1$ is just the radius r, and use the fact that the foot of the perpendicular from $\tilde{\mathbf{c}}$ to any secant from $\mathbf{p}_1$ to $\mathbf{p}_i$, $2 \leq i \leq n$, is just its mid $\frac{1}{2}\tilde{\mathbf{p}}_i$:

$$\tilde{\mathbf{p}}_i^T(\tilde{\mathbf{c}} - \frac{1}{2}\tilde{\mathbf{p}}_i) = 0 \quad \Leftrightarrow \quad \tilde{\mathbf{p}}_i^T\tilde{\mathbf{c}} = \frac{1}{2}|\tilde{\mathbf{p}}_i|^2 \quad , \quad 2 \leq i \leq n$$

Thus $\tilde{\mathbf{c}}$ can be found as solution of the system of equations

$$\begin{pmatrix} \tilde{\mathbf{p}}_2^T \\ \vdots \\ \tilde{\mathbf{p}}_n^T \end{pmatrix} \tilde{\mathbf{c}} = \frac{1}{2} \begin{pmatrix} |\tilde{\mathbf{p}}_2|^2 \\ \vdots \\ |\tilde{\mathbf{p}}_n|^2 \end{pmatrix}$$

with minimum norm again by a QR decomposition. However, taking $\mathbf{p}_1$ as origin is arbitrary and of course every other point from $\mathbf{p}_2, \ldots, \mathbf{p}_n$ could be taken, too. Hence the corresponding problem from lemma 3.3 on the sphere is nicer from a mathematical point of view because it can be solved in a way which is completely symmetrical for all points.

A fundamental element for the proof of the correctness of the Welzl algorithm, as it can be found in [29], is that enclosing spheres with a non-empty intersection can be transformed continuously into each other so that every sphere in between contains the intersection of both original spheres. In order to transfer this proof to our case we have to prove something equal for circles on a sphere:

Lemma 3.4 *Let $\mathbf{a}_0, \mathbf{a}_1 \in \mathbb{R}^d$ be vectors with $|\mathbf{a}_0| \geq 1$ and $|\mathbf{a}_1| \geq 1$ and $K(\mathbf{a}_0) \cap K(\mathbf{a}_1) \neq \emptyset$. Then $K(\mathbf{a}_0)$ can be transformed continuously into $K(\mathbf{a}_1)$ by a convex combination of $\mathbf{a}_0$ and $\mathbf{a}_1$ and it is:*

$$K(\mathbf{a}_0) \cap K(\mathbf{a}_1) \subset K\left((1-t)\mathbf{a}_0 + t\mathbf{a}_1\right) =: K(t) \quad \textit{and} \tag{3.13}$$

$$\partial K(\mathbf{a}_0) \cap \partial K(\mathbf{a}_1) \subset \partial K(\underbrace{(1-t)\mathbf{a}_0 + t\mathbf{a}_1}_{=:\mathbf{a}(t)}) =: \partial K(t) \quad \forall\, t \in [0,1]\,, \tag{3.14}$$

with $\partial K(\mathbf{a}) = \{\mathbf{x} \in S^{d-1} | \, \mathbf{a}^T\mathbf{x} = 1\}$ *the boundary of* $K(\mathbf{a})$ *and*

$$\mathbf{p} \in K(t_0) \wedge \mathbf{p} \in K(t_1) \;\Rightarrow\; \mathbf{p} \in K(t) \; \forall \, t \in [t_0, t_1] \; , \; 0 \leq t_0 \leq t_1 \leq 1 \quad (3.15)$$

Besides, the opening angle of all "intermediate circles" $K(t)$, $t \in (0,1)$, *is smaller than the opening angle of the largest "edge circle"* $K(\mathbf{a}_0)$ *or* $K(\mathbf{a}_1)$ *provided that* $\mathbf{a}_0 \neq \mathbf{a}_1$.

Proof:
At first we have to show $|(1-t)\mathbf{a}_0 + t\mathbf{a}_1| \geq 1$ for $t \in [0,1]$ so that $K(t)$ is well-defined. Because of $K(\mathbf{a}_0) \cap K(\mathbf{a}_1) \neq \emptyset$ a $\mathbf{p} \in S^{d-1}$ exists with $\mathbf{p} \in K(\mathbf{a}_0)$ and $\mathbf{p} \in K(\mathbf{a}_1)$ and hence:

$$1 \leq \mathbf{a}_0^T\mathbf{p} \wedge 1 \leq \mathbf{a}_1^T\mathbf{p} \;\Rightarrow\; 1 \leq \left((1-t)\mathbf{a}_0 + t\mathbf{a}_1\right)^T \mathbf{p} \overset{\text{Cauchy-Schwarz}}{\underset{|\mathbf{p}|=1}{\leq}} |(1-t)\mathbf{a}_0 + t\mathbf{a}_1|$$

Analogous to the first conclusion we can deduce (3.13) resp. (3.14), if we consider the equality case, as well as (3.15), if we replace $\mathbf{a}_0$ by $(1-t_0)\mathbf{a}_0 + t_0\mathbf{a}_1$ and $\mathbf{a}_1$ by $(1-t_1)\mathbf{a}_0 + t_1\mathbf{a}_1$.
That the opening angle of the "intermediate circles" is smaller than the opening angle of the largest "edge circle" follows again by the Cauchy-Schwarz inequality:

$$|(1-t)\mathbf{a}_0 + t\mathbf{a}_1| \leq (1-t)|\mathbf{a}_0| + t|\mathbf{a}_1| \leq \max\{|\mathbf{a}_0|, |\mathbf{a}_1|\}$$

If $\mathbf{a}_0 \neq \mathbf{a}_1$, one of the two inequality relations must be strict. Because if it was $|\mathbf{a}_0| = |\mathbf{a}_1|$ as well as $\mathbf{a}_0$ and $\mathbf{a}_1$ linearly dependent, it would be $\mathbf{a}_1 = -\mathbf{a}_0$. But that is impossible because of $\mathbf{p}^T\mathbf{a}_0 \geq 1 \wedge \mathbf{p}^T\mathbf{a}_1 \geq 1 \Rightarrow \mathbf{p}^T(\mathbf{a}_0 + \mathbf{a}_1) \geq 2$.

□

Lemma 3.5 *Let* $P, R \subset S^{d-1}$ *be finite sets of points on the sphere,* P *non-empty and* $\mathbf{p} \in P$ *a point in* P.

(i) If a circle $K(\mathbf{a})$, $\mathbf{a} \in \mathbb{R}^d$, $|\mathbf{a}| \geq 1$, *exists with* $P \subset K(\mathbf{a})$ *and* $R \subset \partial K(\mathbf{a})$, *then there is also a unique mimimum circle* $K(\mathbf{a}_{min})$ *with* $P \subset K(\mathbf{a}_{min})$ *and* $R \subset \partial K(\mathbf{a}_{min})$. *We denote this circle as* $K_{mb}(P, R)$.

(ii) If $K_{mb}(P, R)$ *exists and* $\mathbf{p} \notin K_{mb}(P \backslash \{\mathbf{p}\}, R)$, *then* $\mathbf{p}$ *is on the boundary of* $K_{mb}(P, R)$, *i.e.* $K_{mb}(P, R) = K_{mb}\left(P \backslash \{\mathbf{p}\}, R \cup \{\mathbf{p}\}\right)$.

(iii) If $K_{mb}(P, R)$ *exists and the points in* R *are linearly independent, then it exists a set* S *of at most* $\max\{0, d - |R|\}$ *points from* P *so that* $K_{mb}(P, R) = K_{mb}(S, R)$ *and the points in* $R \cup S$ *are linearly independent, too. Particularly the points in* S *are on the boundary of* $K_{mb}(P, R)$, *i.e.* $K_{mb}(P, R) = K_b(R \cup S)$ *with* K_b *defined according to lemma 3.3.*

Proof:
(i): $\mathbf{a}_{min}$ is obviously the point with minimum norm fulfilling the condition $\mathbf{a}_{min}^T\mathbf{p}_i \geq 1 \; \forall \; \mathbf{p}_i \in P$ and $\mathbf{a}_{min}^T\mathbf{r}_i = 1 \; \forall \; \mathbf{r}_i \in R$. Therefore we have a quadratic program at hand. If we claim aditionally that the norm is limited by the norm of $\mathbf{a}$, the admissible region described by this condition is compact, convex and non-empty as it contains $\mathbf{a}$. Because of the continuity of the norm there must be a point with mimimum norm in the admissible region. If it were several points, there would be even points with smaller norm on the connecting line according to lemma 3.4 in contradiction to the minimality. Thus $\mathbf{a}_{min}$ and $K_{mb}(P,R)$ are determined uniquely.
(ii): If $K_{mb}(P,R)$ exists, then it is also unique. In the case $P = \{\mathbf{p}\}$ and $R = \emptyset$ it is obviously $K_{mb}(P,R) = K(\mathbf{p})$ and $\mathbf{p}$ is on the boundary of $K_{mb}(P,R)$. In all other cases $K_{mb}(P\backslash\{\mathbf{p}\},R)$ and $K_{mb}(P,R)$ contain a common point from $P \cup R$ and hence they can be transformed continuously into each other according to lemma 3.4 by a family of circles $K(t)$, $t \in [0,1]$. If $\mathbf{p}$ was not on the boundary of $K_{mb}(P,R)$, then there would be a $t_0 \in (0,1)$, so that $\mathbf{p}$ was on the boundary of $K(t_0)$. But since the opening angle of $K_{mb}(P\backslash\{\mathbf{p}\},R)$ is truely smaller than the opening angle of $K_{mb}(P,R)$ because of $\mathbf{p} \notin K_{mb}(P\backslash\{\mathbf{p}\},R)$, $K(t)$ would have a smaller opening angle as $K_{mb}(P,R)$ according to lemma 3.4 in contradiction to its minimality.
(iii): Let S be a maximal subset of the points in P on the boundary of $K_{mb}(P,R)$ so that $R \cup S$ is linearly independent. From linear algebra theory we know that such a set must exist and it can contain at most $\max\{0, d - |R|\}$ points. Let us suppose that it is $K_{mb}(P,R) \neq K_b(R \cup S)$. Because of the uniqueness and minimality of $K_b(R \cup S)$ the opening angle of $K_{mb}(P,R)$ must be larger than the opening angle of $K_b(R \cup S)$. According to lemma 3.4 there is now a continuous transformation $K(t)$, $t \in [0,1]$ from $K(0) = K_b(R \cup S)$ to $K(1) = K_{mb}(R,S)$. For every $\mathbf{p} \in P$ an unique $t(\mathbf{p}) \in [0,1]$ exists so that $\mathbf{p} \in K(t) \; \forall \; t \in [t(\mathbf{p}),1]$ and $\mathbf{p} \notin K(t) \; \forall \; [0,t(\mathbf{p}))$. If $t_{max} := \max\{t(\mathbf{p}) | \mathbf{p} \in P\}$ was smaller than 1, $K(t_{max})$ would be a circle with the points in R on its boundary which contains all points in P but has a smaller opening angle than $K_{mb}(P,R)$ – in contradiction to its minimality. Hence it is $t_{max} = 1$ and there must be a point $\mathbf{p}_{max} \in P$ so that $\mathbf{p}_{max} \in K(1) = K_{mb}(R,S)$ and $\mathbf{p}_{max} \notin K(t) \; \forall \; t \in [0,1)$. Thus because of the continuity of this transformation $\mathbf{p}_{max}$ must be on the boundary of $K_{mb}(P,R)$ but not in S. But that is only possible if $\mathbf{p}_{max}$ is linearly dependent from $R \cup S$. Thus let be $R \cup S = \{\mathbf{s}_1, \ldots, \mathbf{s}_m\}$ and $\mathbf{p}_{max} = \sum_{i=1}^m \lambda_i \mathbf{s}_i$. As $\mathbf{p}_{max}$ and $\mathbf{s}_1, \ldots, \mathbf{s}_m$ are on the boundary of $K_{mb}(P,R) =: K(\mathbf{a})$, it is:

$$1 = \mathbf{a}^T\mathbf{p}_{max} = \mathbf{a}^T \sum_{i=1}^m \lambda_i \mathbf{s}_i = \sum_{i=1}^m \lambda_i \mathbf{a}^T\mathbf{s}_i = \sum_{i=1}^m \lambda_i$$

Thereby it follows because $\mathbf{s}_1, \ldots, \mathbf{s}_m$ are on the boundary of $K_b(R \cup S) =: K(\mathbf{b})$:

$$\mathbf{b}^T\mathbf{p}_{max} = \mathbf{b}^T \sum_{i=1}^m \lambda_i \mathbf{s}_i = \sum_{i=1}^m \lambda_i \mathbf{b}^T\mathbf{s}_i = \sum_{i=1}^m \lambda_i = 1$$

Hence $\mathbf{p}_{max}$ is on the boundary of $K_b(R \cup S)$ in contradiction to $\mathbf{p}_{max} \notin K(t)\, \forall\, t \in [0,1)$. Thus it is $K_{mb}(P,R) = K_b(R \cup S) = K_{mb}(S,R)$. However, we have not yet considered the case $R \cup S = \emptyset$. That means it must be shown that there is at least one point in P which is on the boundary of $K_{mb}(P,\emptyset)$. But that can be done by using a continuous transformation from $K(\mathbf{p})$ to $K_{mb}(P,\emptyset)$ for any point $\mathbf{p} \in P$ in the same way as it has been shown above that $\mathbf{p}_{max}$ is on the boundary of $K_{mb}(P,R)$. If the last remaining point (it could also be several points at once) entered the transformed circle before the transformation had finished, the transformed circle at that moment would contain all points in P, but the circle's opening angle would be smaller than the opening angle of $K_{mb}(P,\emptyset)$ – in contradiction to its minimality. Hence there is a point in P which is not contained by any of the "intermediate circles" but only by $K_{mb}(P,\emptyset)$. Therefore this point must be on the boundary of $K_{mb}(P,\emptyset)$ and thus S cannot be empty.

□

Lemma 3.5 always assumes in its statements the existence of an enclosing circle. But so far we have not yet elaborated on a criterion when for a given set of points an enclosing circle actually exists.

Proposition 3.6 *Let $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\} \subset S^{d-1}$, $n \in \mathbb{N}$, be a set of points on the sphere $S^{d-1} = \left\{\mathbf{x} \in \mathbb{R}^d \,|\, |\mathbf{x}| = 1\right\}$, $d \geq 2$, lying within a hemisphere (without boundary plane), i.e. $\exists\, \mathbf{h} \in S^{d-1}$ with $\mathbf{h}^T\mathbf{p}_i > 0$ for all $1 \leq i \leq n$. Under this condition there is an unique smallest circle $K(\mathbf{a}_{min}) := \{\mathbf{x} \in S^{d-1} | \mathbf{a}_{min}^T\mathbf{x} \geq 1\}$ on the sphere with $\mathbf{h}^T\mathbf{a}_{min} > 0$ that contains all points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$. We denote this circle as $K_m(\mathbf{p}_1, \ldots, \mathbf{p}_n)$.*

Proof:
It is clear that

$$K\left(\frac{\mathbf{h}}{\min\{\mathbf{h}^T\mathbf{p}_i \mid 1 \leq i \leq n\}}\right)$$

is a well-defined circle containing all points $\mathbf{p}_1, \ldots, \mathbf{p}_n$. Thereby it follows by lemma 3.5(i) with $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ and $R = \emptyset$ that $K_m(\mathbf{p}_1, \ldots, \mathbf{p}_n) = K(\mathbf{a}_{min})$ exists and is unique. We have still to show that $\mathbf{h}^T\mathbf{a}_{min} > 0$. If $\mathbf{h}^T\mathbf{a}_{min} < 0$ also the circle $K(\mathbf{a}_{min} - \mathbf{h}\mathbf{h}^T\mathbf{a}_{min})$ would contain $\mathbf{p}_1, \ldots, \mathbf{p}_n$:

$$\mathbf{p}_i^T\left(\mathbf{a}_{min} - \mathbf{h}\mathbf{h}^T\mathbf{a}_{min}\right) = \underbrace{\mathbf{p}_i^T\mathbf{a}_{min}}_{\geq 1 \text{ because } \mathbf{p}_i \in K(\mathbf{a}_{min})} - \underbrace{\mathbf{p}_i^T\mathbf{h}}_{>0}\underbrace{\mathbf{h}^T\mathbf{a}_{min}}_{<0} \geq 1$$

But it would be smaller than $K(\mathbf{a}_{min})$ in contradiction to its minimality:

$$\begin{aligned}\left|\mathbf{a}_{min} - \mathbf{h}\mathbf{h}^T\mathbf{a}_{min}\right|^2 &= |\mathbf{a}_{min}|^2 - 2\,\mathbf{a}_{min}^T\mathbf{h}\mathbf{h}^T\mathbf{a}_{min} + \mathbf{a}_{min}^T\mathbf{h}\underbrace{\mathbf{h}^T\mathbf{h}}_{=1}\mathbf{h}^T\mathbf{a}_{min} \\ &= |\mathbf{a}_{min}|^2 - \left(\mathbf{h}^T\mathbf{a}_{min}\right)^2 < |\mathbf{a}_{min}|^2\end{aligned}$$

Suppose that $\mathbf{h}^T\mathbf{a}_{min} = 0$. Let $a := |\mathbf{a}_{min}|^2$ and m be a positive number smaller than $\min\{\mathbf{h}^T\mathbf{p}_i | 1 \leq i \leq n\}$ and $1/\sqrt{a}$. Then the circle $K(\mathbf{b})$ with

$$\mathbf{b} := \frac{1 - m^2 a}{1 + m^2 a}\,\mathbf{a}_{min} + \frac{2ma}{1 + m^2 a}\,\mathbf{h}$$

contains $\mathbf{p}_1, \ldots, \mathbf{p}_n$, too, because

$$\begin{aligned} \mathbf{b}^T\mathbf{p}_i &= \frac{1 - m^2 a}{1 + m^2 a}\,\underbrace{\mathbf{a}_{min}^T\mathbf{p}_i}_{\geq 1} + \frac{2ma}{1 + m^2 a}\,\underbrace{\mathbf{h}^T\mathbf{p}_i}_{> m} \\ &> \frac{1 - m^2 a}{1 + m^2 a} + \frac{2m^2 a}{1 + m^2 a} = 1\,, \end{aligned}$$

and $K(\mathbf{b})$ has the same opening angle as $K(\mathbf{a}_{min})$ in contradiction to its uniqueness:

$$|\mathbf{b}|^2 = \frac{(1 - m^2 a)^2}{(1 + m^2 a)^2}\,a + \frac{(2ma)^2}{(1 + m^2 a)^2} = \frac{1 + 2m^2 a + m^4 a^2}{(1 + m^2 a)^2}\,a = a = |\mathbf{a}_{min}|^2$$

Thus it must be $\mathbf{h}^T\mathbf{a}_{min} > 0$.

□

Theorem 3.7 *Let P and R be finite sets of points on the sphere $S^{d-1} := \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{x}| = 1\}$. If a smallest enclosing circle $K_{mb}(P, R)$ of the points in P with the points in R on its boundary exists, then this circle can be calculated by the following recursive function:*

Algorithm 3.1 Recursive calculation of $K_{mb}(P, R)$

```
1:  function K_mb(P, R)
2:      if P = ∅ or |R| = d then
3:          return K_b(R)
4:      else
5:          choose p ∈ P
6:          K ← K_mb(P \ {p}, R)
7:          if p ∉ K then
8:              return K_mb(P \ {p}, R ∪ {p})
9:          else
10:             return K
11:         end if
12:     end if
13: end function
```

The function $K_b(R)$ returns the smallest circle according to lemma 3.3 with the points in R on its boundary.

Especially by $K_{mb}(P, \emptyset)$ the smallest enclosing circle $K_m(P)$ of the points in P can be calculated, which exists and is unique according to theorem 3.6 if there is a $\mathbf{h} \in S^{d-1}$ with $\mathbf{h}^T\mathbf{p} > 0 \; \forall \, \mathbf{p} \in P$. The average running time for calculating $K_{mb}(P, \emptyset) = K_m(P)$ is in $O(dd!|P|)$ if a random point from P is chosen at line 5.

Proof:
Correctness of the algorithm / function: In the case $P = \emptyset$ all is clear. For $|R| = d$ it is clear from the proof of lemma 3.3 that there is exactly one circle with the points in R on its boundary if these points are linearly independent since the system of linear equations of this proof has an unique solution in this case. If the points in R are not linearly independent, we even get an over-determined system of linear equations in the subspace spanned by those points, which must have a solution because of the existence of $K_{mb}(P, R)$.
If we choose $\mathbf{p} \in P$, the existence of $K_{mb}(P \backslash \{\mathbf{p}\}, R)$ is ensured by the assumed existence of $K_{mb}(P, R)$ according to lemma 3.5(i). If it is furthermore $\mathbf{p} \notin K_{mb}(P \backslash \{\mathbf{p}\}, R)$, it can be concluded by lemma 3.5(ii), that $K_{mb}(P \backslash \{\mathbf{p}\}, R \cup \{\mathbf{p}\})$ is just $K_{mb}(P, R)$.
Running time for calculating $K_{mb}(P, \emptyset) = K_m(P)$: If we suppose that the points in R are linearly independent, we can conclude by applying lemma 3.5(iii) to $K_{mb}(\{\mathbf{p}\}, R)$ that the at line 5 from P chosen point $\mathbf{p}$ must be linearly independent from R if it is $\mathbf{p} \notin K$ at line 7. By induction it is clear that when calculating $K_m(P)$ the sets R occurring during recursion consist of linear independent points. By applying lemma 3.5(iii) once again we can conclude that among the points in R there are at most $d - |R|$ points which are necessary for determining the circle K by lying on its boundary. The case $\mathbf{p} \notin K$ at line 7 can obviously only occur if $\mathbf{p}$ is one of these $d - |R|$ points determining K and the probability of that is just $(d - |R|)/|P|$. The further calculation for the running time can be done in just the same way as for the Welzl algorithm which has according to [29] a running time in $O(\delta\delta!|P|)$ with $\delta := d + 1$ since $d + 1$ points determine a sphere in d-dimensional space. Because according to lemma 3.3 a circle on the sphere S^{d-1} in d-dimensional space is already determined by d points, we get a running time in $O(dd!|P|)$ for the Welzl algorithm transferred to the sphere.

□

An obvious application of the Welzl algorithm on the sphere is, when considering a set of directions in $\mathbb{R}^3$ as points on the S^2, to determine the "average" direction which deviates from the original directions at most by a certain angle by calculating the minimum enclosing circle.
However, as it has already been suggested at the end of the previous section we want to apply this algorithm to a set of rotation quaternions on the S^3. But as already mentioned, the unit quaternions $\mathfrak{Q}^*$ are only isomorph to the $\mathrm{SO}(3)$ if we divide them by the equivalence relation $\mathfrak{q} \sim \mathfrak{w} \; :\Leftrightarrow \; \mathfrak{q} = \mathfrak{w} \vee \mathfrak{q} = -\mathfrak{w}$. So when

transferring rotation matrices to rotation quaternions there is an ambiguity concerning the sign. But we can use this ambiguity so that the hemisphere condition of lemma 3.6 is always fulfilled for $\mathfrak{Q}^*/\sim$ because of the following lemma.

Lemma 3.8 *On the set* $S^{d-1} = \{\mathbf{x} \in \mathbb{R}^d \mid |\mathbf{x}| = 1\}$ *given the following equivalence relation:*

$$\mathbf{a} \sim \mathbf{b} \quad :\Leftrightarrow \quad \mathbf{a} = \mathbf{b} \ \vee \ \mathbf{a} = -\mathbf{b}$$

Then for every finite subset $P = \{\tilde{\mathbf{p}}_1, \ldots, \tilde{\mathbf{p}}_n\}$ *of* $S^{d-1}/\sim$ *there exists a set* $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset S^{d-1}$ *of representatives with* $\mathbf{p}_i \sim \tilde{\mathbf{p}}_i$, $1 \leq i \leq n$, *and a* $\mathbf{h} \in S^{d-1}$ *so that* $\mathbf{h}^T\mathbf{p} > 0$ *for all* $1 \leq i \leq n$.

Proof:
We prove by induction for d. For $d = 1$ the statement is trivial since it is $S^0 = \{-1, 1\}$ and $|S^0/\sim| = 1$. Thus we can take $\{1\}$ as set of representatives and as "vector" $\mathbf{h}$ we can take 1, too, unless P is empty. For the induction step we can choose an arbitrary $\mathbf{h}_d \in S^d$ and a set of representatives $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ by properly flipping the sign so that it is $\mathbf{h}_d^T\mathbf{p}_i \geq 0$. If there is no representative with $\mathbf{h}_d^T\mathbf{p}_i = 0$, that means $J := \{j \in \{1, \ldots, n\} \mid \mathbf{h}_d^T\mathbf{p}_j = 0\}$ is empty, we can choose $\mathbf{h} = \mathbf{h}_d$ and we are done. Otherwise since $S^d \cap \langle \mathbf{h}_d \rangle^\perp = \{\mathbf{x} \in S^d \mid \mathbf{h}_d^T\mathbf{x} = 0\}$ can be considered as S^{d-1}, because of the induction hypothesis there exist representatives $\mathbf{p}_j$ with $j \in J$ a $\mathbf{h}_{d-1} \in S^d \cap \langle \mathbf{h}_d \rangle^\perp$ so that it is $\mathbf{h}_{d-1}^T\mathbf{p}_j > 0$ for all $j \in J$. If $I := \{1, \ldots, n\} \backslash J$ is empty, we can choose $\mathbf{h} = \mathbf{h}_{d-1}$ and are done. Otherwise define $u := \max\{|\mathbf{h}_{d-1}^T\mathbf{p}_i| \mid i \in I\}$ and let v be a number between 0 and $\min\{\mathbf{h}_d^T\mathbf{p}_i \mid i \in I\}$. We define:

$$\mathbf{h} := \underbrace{\frac{u}{\sqrt{u^2 + v^2}}}_{=:\lambda} \mathbf{h}_d + \underbrace{\frac{v}{\sqrt{u^2 + v^2}}}_{=:\mu} \mathbf{h}_{d-1}$$

Because of $\lambda^2 + \mu^2 = 1$ and $\mathbf{h}_d^T\mathbf{h}_{d-1} = 0$ it is obviously $|\mathbf{h}| = 1$ and thus $\mathbf{h} \in S^d$. Besides, for $j \in J$ it is obviously $\mathbf{h}^T\mathbf{p}_j = \mu\mathbf{h}_{d-1}^T\mathbf{p}_j > 0$. For $i \in I$ we have:

$$\mathbf{h}^T\mathbf{p}_i = \frac{u}{\sqrt{u^2 + v^2}} \underbrace{\mathbf{h}_d^T\mathbf{p}_i}_{>v} + \frac{v}{\sqrt{u^2 + v^2}} \underbrace{\mathbf{h}_{d-1}^T\mathbf{p}_i}_{\geq -u} > 0$$

□

In order to solve for a finite set of rotation matrices $\{\mathbf{R}_1, \ldots, \mathbf{R}_n\} \subset \mathrm{SO}(3)$ the problem

$$\underset{\mathbf{R} \in \mathrm{SO}(3)}{\arg\min} \max_{i \in \{1, \ldots, n\}} \delta(\mathbf{R}, \mathbf{R}_i) \tag{3.16}$$

by the Welzl algorithm on the sphere in 4-dimensional space, we need to know which signs of the corresponding quaternions $\pm\mathbf{q}(\mathbf{R}_1), \ldots, \pm\mathbf{q}(\mathbf{R}_n)$ to choose. According to lemma 3.8 we can do this in a way so that the hemisphere condition in lemma 3.6 is fulfilled. Unfortunately this is not sufficient because only if the

angles between all pairs of quaternions are smaller than $\pi/2$, they correspond to half the angle between the corresponding rotation matrices due to the absolute value in (3.11).

Theorem 3.9 *Let* $\{\mathbf{R}_1, \ldots, \mathbf{R}_n\} \subset \mathrm{SO}(3)$ *be a finite set of rotation matrices. Furthermore corresponding rotation quaternions* $\mathfrak{q}_1, \ldots, \mathfrak{q}_n$ *exist so that* $\mathbf{R}_i = \mathbf{R}(\mathfrak{q}_i)$ *and* $\mathfrak{q}_1^T \mathfrak{q}_i > 0$*,* $2 \leq i \leq n$*. Let* $K(\mathfrak{q}) = K_b(\mathfrak{q}_1, \ldots, \mathfrak{q}_n)$ *have an opening angle smaller than* $\pi/2$*, that means* $|\mathfrak{q}|^2 < 1/\cos^2(\pi/4) = 2$*. In the following we want to talk of "*$\mathbf{R}_1, \ldots, \mathbf{R}_n$ *fulfilling the cone condition" if these conditions are met.*
Thus if $\mathbf{R}_1, \ldots, \mathbf{R}_n$ *fulfill the cone condition, then* $\mathbf{R}_m(\mathbf{R}_1, \ldots, \mathbf{R}_n) := \mathbf{R}(\mathfrak{q}/|\mathfrak{q}|)$ *is the unique solution of the minimization problem*

$$\underset{\mathbf{R} \in \mathrm{SO}(3)}{\arg\min} \max_{i \in \{1,\ldots,n\}} \delta(\mathbf{R}, \mathbf{R}_i)$$

and it is

$$\alpha_m(\mathbf{R}_1, \ldots, \mathbf{R}_n) := \min_{\mathbf{R} \in \mathrm{SO}(3)} \max_{i \in \{1,\ldots,n\}} \delta(\mathbf{R}, \mathbf{R}_i) = 2 \arccos\left(\frac{1}{|\mathfrak{q}|}\right) \in \left[0, \frac{\pi}{2}\right) .$$

Proof:
If the opening angle of $K(\mathfrak{q}) = K_m(\mathfrak{q}_1, \ldots, \mathfrak{q}_n)$ is smaller than $\pi/2$, the angles of all pairs of quaternions are smaller than $\pi/2$ because of the triangle inequation for triangles on the sphere in 4-dimensional space (see lemma A.2):

$$\forall\, 1 \leq i, j \leq n \,:\, \measuredangle(\mathfrak{q}_i, \mathfrak{q}_j) \leq \measuredangle(\mathfrak{q}_i, \mathfrak{q}) + \measuredangle(\mathfrak{q}, \mathfrak{q}_j) < \frac{\pi}{4} + \frac{\pi}{4} = \frac{\pi}{2} \;\Rightarrow\; \mathfrak{q}_i^T \mathfrak{q}_j > 0$$

If the circle $K(\mathfrak{q})$ has an opening angle smaller or equal to $\pi/2$, it is not possible that such a circle with an opening angle smaller than $\pi/2$ exists for any other choice of signs for $\mathfrak{q}_2, \ldots, \mathfrak{q}_n$. If that would be the case, there had to be a $k \in \{2, \ldots, n\}$ so that $\mathfrak{q}_1^T \mathfrak{q}_k < 0$ in contradiction to the above inequation.
If the opening angle of $K(\mathfrak{q})$ is smaller than $\pi/2$, the angle between two rotation quaternions is in fact equal to half the angle between the corresponding rotation matrices according to (3.11) and the original minimization problem is equivalent to the minimization problem

$$\underset{\mathfrak{q} \in \mathfrak{Q}^*}{\arg\min} \max_{i \in \{1,\ldots,n\}} \arccos(\mathfrak{q}^T \mathfrak{q}_i) .$$

This minimization problem is actually solved by the smallest enclosing circle of $\mathfrak{q}_1, \ldots, \mathfrak{q}_n$ on the sphere in 4-dimensional space. Because this smallest enclosing circle has an opening angle smaller than $\pi/2$, that means $\arccos(1/|\mathfrak{q}|) \in [0, \pi/4)$, it follows $\alpha_m(\mathbf{R}_1, \ldots, \mathbf{R}_n) \in [0, \pi/2)$.

□

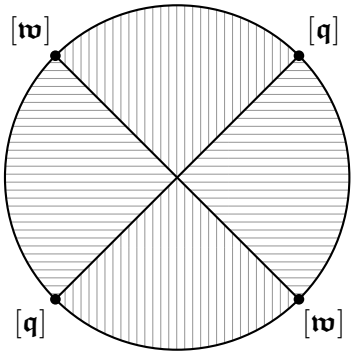


Figure 3.6: For two quaternions $\mathfrak{q}$ and $\mathfrak{w}$ resp. their cosets $[\mathfrak{q}]$ and $[\mathfrak{w}]$ which are perpendicular to each other there are two smallest enclosing double cones (contrary hatched) with an opening angle of $\pi/2$ on whose boundary they lie.

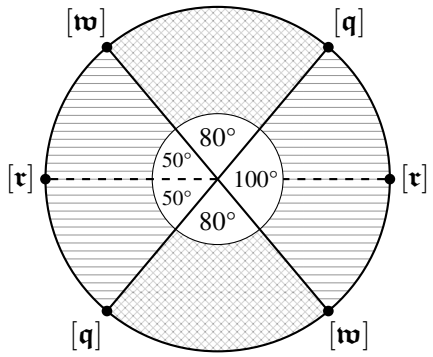


Figure 3.7: Although $[\mathfrak{r}]$ is not within the smallest enclosing double cone of $[\mathfrak{q}]$ and $[\mathfrak{w}]$ (cross hatched), it does not lie on the boundary of the smallest enclosing double cone of $[\mathfrak{q}]$, $[\mathfrak{w}]$ and $[\mathfrak{r}]$ (horizontally hatched).

The cone condition in theorem 3.9 is formulated constructively. To check it and, if it is fulfilled, to solve the minimization problem of theorem 3.9, appropriate rotation quaternions $\mathfrak{q}_1, \ldots, \mathfrak{q}_n$ are calculated at first so that $\mathbf{R}_i = \mathbf{R}(\mathfrak{q}_i)$ for $1 \leq i \leq n$ and then the signs of $\mathfrak{q}_2, \ldots, \mathfrak{q}_n$ are possibly flipped so that it is $\mathfrak{q}_1^T \mathfrak{q}_i > 0$ for $2 \leq i \leq n$. If a $j \in \{2, \ldots, n\}$ exists so that $\mathfrak{q}_1^T \mathfrak{q}_j = 0$, the cone condition is not fulfilled and we can abort. If that is not the case, we can calculate $K(\mathfrak{q}) = K_m(\mathfrak{q}_1, \ldots, \mathfrak{q}_n)$ by the Welzl algorithm on the sphere in 4-dimensional space. The opening angles of the "intermediate circles" $K_i := K_m(\mathfrak{q}_{\pi^{-1}(1)}, \ldots, \mathfrak{q}_{\pi^{-1}(i)})$, which are calculated during the Welzl algorithm (with π a random permutation applied at the beginning of the algorithm), increase strictly monotonically until finally the circle $K_n = K(\mathfrak{q})$ is reached. Hence we can check during the Welzl algorithm for each of these intermediate circles if its opening angle is greater or equal to $\pi/2$ and, if this is the case, we can abort immediately because the cone condition is not fulfilled.

The question rises if and how the optimization problem (3.16) can be solved if the rotation quaternions corresponding to the given rotation matrices can only be enclosed by a cone with an opening angle greater or equal to $\pi/2$.
In fact, in this case the solution is no longer unique as it is shown by figure 3.6 for a two-dimensional projection of the quaternion sphere: For two quaternions which are perpendicular to each other there are two double cones with an opening angle

of $\pi/2$ on whose boundary the two quaternions lie.
Furthermore, it is not possible in this case to calculate any optimum solution by the Welzl algorithm since a for the Welzl algorithm necessary property is no longer fulfilled: If the smallest enclosing circle for a subset of the points does not contain another point, this point must lie on the boundary of the smallest enclosing circle of the union of the subset and the other point. But this is not true in the case of an opening angle larger than $\pi/2$ as illustrated by figure 3.7 again for the two-dimensional projection of the quaternion sphere. Thus the Welzl algorithm cannot be applied any longer.

Now the question arises what is the best "approximation" for $\mathbf{R}_1, \ldots, \mathbf{R}_n$ in terms of the spectral norm if we accept as approximation not only rotations but also any other linear transformation. This we want to investigate in the following lemma:

Lemma 3.10 *Let $\mathbf{R} \in \mathrm{SO}(3)$ be a rotation by the angle $\alpha \in [0, \pi]$ and let $\mathrm{Sym}_{\mathrm{def}}(3)$ be the set of real symmetric semidefinite 3×3 matrices, i.e. the union of negative and positive definite real matrices. Then the minimization problem*

$$\underset{\mathbf{S}\in\mathrm{Sym}_{\mathrm{def}}(3)}{\arg\min} \|\mathbf{R}-\mathbf{S}\|_2$$

is solved by

$\cos\alpha\,\mathbb{1}$	*with* $\|\mathbf{R}-\cos\alpha\,\mathbb{1}\|_2 = \sin\alpha$	*for* $\alpha \in [0, \pi/2)$ *and*
the zero matrix $\mathbf{0}$	*with* $\|\mathbf{R}-\mathbf{0}\|_2 = \|\mathbf{R}\|_2 = 1$	*for* $\alpha \in [\pi/2, \pi]$.

Proof:
Let $\tilde{\mathbf{R}} \in \mathrm{SO}(3)$ be a rotation mapping the rotation axis of $\mathbf{R}$ to the z-axis. For arbitrary $\mathbf{S} \in \mathrm{Sym}_{\mathrm{def}}(3)$ it is:

$$\|\mathbf{R}-\mathbf{S}\|_2 = \left\|\tilde{\mathbf{R}}\,(\mathbf{R}-\mathbf{S})\,\tilde{\mathbf{R}}^T\right\|_2 = \left\|\begin{pmatrix}\cos\alpha & -\sin\alpha & 0\\ \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{pmatrix} - \underbrace{\tilde{\mathbf{R}}\mathbf{S}\tilde{\mathbf{R}}^T}_{=:\mathbf{S}'}\right\|_2$$

Obviously it is $\mathbf{S}' \in \mathrm{Sym}_{\mathrm{def}}(3)$. We divide $\mathbf{S}'$ into blocks:

$$\mathbf{S}' = \begin{pmatrix}\mathbf{S}'_{2\times2} & a'\\ a' \quad b' & c\end{pmatrix} \quad \text{with } \mathbf{S}'_{2\times2} \in \mathrm{Sym}_{\mathrm{def}}(2) \text{ and } a', b', c \in \mathbb{R}$$

According to the principal axis [43] theorem there is a $\mathbf{R}_{2\times2} \in \mathrm{SO}(2)$ and $s_1, s_2 \in \mathbb{R}$ so that

$$\mathbf{S}'_{2\times2} = \mathbf{R}^T_{2\times2}\begin{pmatrix}s_1 & 0\\ 0 & s_2\end{pmatrix}\mathbf{R}_{2\times2}$$

and thus (consider the commutativity of $\mathrm{SO}(2)$):

$$\begin{aligned}
&\|\mathbf{R}-\mathbf{S}\|_2 \\
= &\left\| \begin{pmatrix} \mathbf{R}_{2\times 2} & 0 \\ 0 \quad 0 & 1 \end{pmatrix} \left[\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} \mathbf{S}'_{2\times 2} & a' \\ a' \quad b' & c \end{pmatrix} \right] \begin{pmatrix} \mathbf{R}^T_{2\times 2} & 0 \\ 0 \quad 0 & 1 \end{pmatrix} \right\|_2 \\
= &\left\| \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} s_1 & 0 & a \\ 0 & s_2 & b \\ a & b & c \end{pmatrix} \right\|_2 \qquad (3.17) \\
= &\underbrace{\left\| \begin{pmatrix} \cos\alpha - s_1 & -\sin\alpha & -a \\ \sin\alpha & \cos\alpha - s_2 & -b \\ -a & -b & 1-c \end{pmatrix} \right\|_2}_{=:\mathbf{M}} \quad \text{with} \quad \begin{pmatrix} a \\ b \end{pmatrix} := \mathbf{R}_{2\times 2} \begin{pmatrix} a' \\ b' \end{pmatrix}
\end{aligned}$$

$\|\mathbf{M}\|_2$ can be bounded in the following way (consider $\sin\alpha \geq 0$ because $\alpha \in [0,\pi]$):

$$\begin{aligned}
\|\mathbf{M}\|_2 &\geq \left| \mathbf{M} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right|_2 = \sqrt{(\cos\alpha - s_1)^2 + \sin^2\alpha + a^2} \geq \sin\alpha \quad \text{resp.} \\
\|\mathbf{M}\|_2 &\geq \left| \mathbf{M} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right|_2 = \sqrt{(\cos\alpha - s_2)^2 + \sin^2\alpha + b^2} \geq \sin\alpha
\end{aligned}$$

The two last inequalities are sharp iff $s_1 = s_2 = \cos\alpha$ and $a = b = 0$. In this case it is:

$$\mathbf{M}^T\mathbf{M} = \begin{pmatrix} \sin^2\alpha & 0 & 0 \\ 0 & \sin^2\alpha & 0 \\ 0 & 0 & (1-c)^2 \end{pmatrix} \quad \Rightarrow \quad \|\mathbf{M}\|_2 = \max\{\sin\alpha, |1-c|\}$$

In fact $\|\mathbf{M}\|_2$ reaches the lower bound $\sin\alpha$ if $|1-c| \leq \sin\alpha \Leftrightarrow c \in [1-\sin\alpha, 1+\sin\alpha]$. Especially this is true for $c = \cos\alpha$ in the case of $\alpha \in [0, \pi/2]$ due to

$$\alpha \in [0,\pi/2] \;\Rightarrow\; 0 \leq \cos\alpha \leq 1 \;\wedge\; 0 \leq \sin\alpha \leq 1$$

$$\Rightarrow\; \cos\alpha + \sin\alpha \geq \cos^2\alpha + \sin^2\alpha = 1 \;\Rightarrow\; \cos\alpha \geq 1 - \sin\alpha\,.$$

In order to get the primary matrix $\mathbf{S}$ the orthogonal basis transitions applied to $\mathbf{R}-\mathbf{S}$ have to be applied to (3.17) in transposed form in reverse order. Because of $s_1 = s_2 = \cos\alpha$ and $a = b = 0$ the latter matrix is just $\cos\alpha\,\mathbb{1}$. Because it is a scaled unit matrix and hence commutes with any other matrix, it is not affected by basis transitions and thus we have proven the first statement of this lemma.
In the case of $\alpha \in [\pi/2, \pi]$ it is $\cos\alpha \leq 0$. Because $\mathbf{S}$ is semidefinite and this

property is not affected by orthogonal basis transitions, $s_1, s_2, c \geq 0$ or $s_1, s_2, c \leq 0$ must be valid. If $s_1, s_2, c \geq 0$ holds, we get:

$$\|\mathbf{M}\|_2 \geq \left| \mathbf{M} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right|_2 = \sqrt{(\cos\alpha - s_1)^2 + \sin^2\alpha + a^2} \geq \sqrt{\cos^2\alpha + \sin^2\alpha} = 1$$

If $s_1, s_2, c \leq 0$ holds, we get:

$$\|\mathbf{M}\|_2 \geq \left| \mathbf{M} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right|_2 = a^2 + b^2 + (1-c)^2 \geq 1$$

But $\|\mathbf{R} - \mathbf{S}\|_2$ obviously reaches the lower bound of 1 for $\mathbf{S} = \mathbf{0}$ and thus we have proven the second statement of this lemma.

□

Theorem 3.11 *Let* $\{\mathbf{R}_1, \ldots, \mathbf{R}_n\} \subset \mathrm{SO}(3)$ *be a set of rotation matrices and* $\mathrm{Mat}_{3\times3}$ *the set of* 3×3 *matrices. Then the minimization problem*

$$\arg\min_{\mathbf{M}\in\mathrm{Mat}_{3\times3}} \max_{i\in\{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{M}\|_2$$

is solved by

$$\mathbf{M}_{opt} := \cos\big(\alpha_m(\mathbf{R}_1, \ldots, \mathbf{R}_n)\big)\mathbf{R}_m(\mathbf{R}_1, \ldots, \mathbf{R}_n) \quad \textit{with}$$

$$\max_{i\in\{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{M}_{opt}\|_2 = \sin\big(\alpha_m(\mathbf{R}_1, \ldots, \mathbf{R}_n)\big)$$

with α_m *defined according to theorem 3.9 if* $\mathbf{R}_1, \ldots, \mathbf{R}_n$ *fulfill the cone condition. Otherwise it is solved by the zero matrix with*

$$\max_{i\in\{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{0}\|_2 = 1\ .$$

Proof:
Let $\mathbf{M} \in \mathrm{Mat}_{3\times3}$ be arbitrary, but fixed. It is a well-known result of linear algebra that $\mathbf{M}$ can be written by the so-called polar decomposition as $\mathbf{M} = \mathbf{RS}$ with an orthogonal matrix $\mathbf{R}$ and a real symmetric positive semidefinite matrix $\mathbf{S}$ [44]. By possibly transferring a factor (-1) from $\mathbf{R}$ to $\mathbf{S}$ we achieve that it is $\mathbf{R} \in \mathrm{SO}(3)$. Then $\mathbf{S}$ might be negative semidefinite. By the polar decomposition of $\mathbf{M}$ and by lemma 3.10 we can conclude for all $1 \leq i \leq n$:

$$\|\mathbf{R}_i - \mathbf{M}\|_2 = \|\mathbf{R}_i - \mathbf{RS}\|_2 = \left\|\mathbf{R}^T\mathbf{R}_i - \mathbf{S}\right\|_2 \geq f\big(\delta(\mathbf{R}, \mathbf{R}_i)\big)$$

$$\text{with} \quad f(\alpha) := \begin{cases} \sin\alpha & \text{if} \quad \alpha \in [0, \pi/2) \\ 1 & \text{if} \quad \alpha \in [\pi/2, \pi] \end{cases}$$

Because f is monotonically increasing it is:

$$\min_{\mathbf{M}\in\mathrm{Mat}_{3\times 3}} \max_{i\in\{1,\dots,n\}} \|\mathbf{R}_i - \mathbf{M}\|_2 \geq f\left(\min_{\mathbf{R}\in\mathrm{SO}(3)} \max_{i\in\{1,\dots,n\}} \delta(\mathbf{R},\mathbf{R}_i)\right) \tag{3.18}$$

If $\mathbf{R}_1,\dots,\mathbf{R}_n$ fulfill the cone condition, we get according to theorem 3.9:

$$\min_{\mathbf{M}\in\mathrm{Mat}_{3\times 3}} \max_{i\in\{1,\dots,n\}} \|\mathbf{R}_i - \mathbf{M}\|_2 \geq \sin\big(\alpha_m(\mathbf{R}_1,\dots,\mathbf{R}_n)\big) \tag{3.19}$$

Let us denote $\alpha_m(\mathbf{R}_1,\dots,\mathbf{R}_n)$ by only α_m hereafter. Because of the above inequation it is sufficient to show that the lower bound $\sin(\alpha_m)$ for $\mathbf{M}_{opt}$ is reached. For all $1 \leq i \leq n$ it is:

$$\begin{aligned} d_i &:= \|\mathbf{R}_i - \cos\alpha_m \mathbf{R}_m(\mathbf{R}_1,\dots,\mathbf{R}_n)\|_2 \\ &= \big\|\underbrace{\mathbf{R}_m(\mathbf{R}_1,\dots,\mathbf{R}_n)^T\mathbf{R}_i}_{=:\mathbf{R}'} - \cos\alpha_m \mathbb{1}\big\|_2 \end{aligned}$$

By transforming by an appropriate orthogonal basis transition the rotation axis of $\mathbf{R}'$ to the z-axis we get by $\delta_i := \delta(\mathbf{R}_m(\mathbf{R}_1,\dots,\mathbf{R}_n),\mathbf{R}_i)$:

$$\begin{aligned} d_i &= \left\| \begin{pmatrix} \cos\delta_i & -\sin\delta_i & 0 \\ \sin\delta_i & \cos\delta_i & 0 \\ 0 & 0 & 1 \end{pmatrix} - \cos\alpha_m \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right\|_2 \\ &= \left\| \begin{pmatrix} \cos\delta_i - \cos\alpha_m & -\sin\delta_i & 0 \\ \sin\delta_i & \cos\delta_i - \cos\alpha_m & 0 \\ 0 & 0 & 1-\cos\alpha_m \end{pmatrix} \right\|_2 \\ &= \max\left\{ \sqrt{(\cos\delta_i - \cos\alpha_m)^2 + \sin^2\delta_i}\,,\ 1-\cos\alpha_m \right\} \text{ with } 0 \leq \delta_i \leq \alpha_m < \frac{\pi}{2} \end{aligned}$$

The following inequations are valid:

$$\begin{aligned} (\cos\delta_i - \cos\alpha_m)^2 + \sin^2\delta_i &= 1 + \cos\alpha_m^2 - 2\cos\delta_i\cos\alpha_m \\ &\overset{\cos\delta_i \geq \cos\alpha_m \geq 0}{\leq} 1 + \cos^2\alpha_m - 2\cos\alpha_m\cos\alpha_m = 1 - \cos^2\alpha_m = \sin^2\alpha_m \\ \text{and}\quad (1-\cos\alpha_m)^2 &= 1 + \cos^2\alpha_m - 2\cos\alpha_m \\ &\overset{0 \leq \cos\alpha_m \leq 1}{\leq} 1 + \cos^2\alpha_m - 2\cos\alpha_m\cos\alpha_m = 1 - \cos^2\alpha_m = \sin^2\alpha_m \end{aligned}$$

Thus it is:

$$\max_{i\in\{1,\dots,n\}} \|\mathbf{R}_i - \mathbf{M}_{opt}\|_2 = \max_{i\in\{1,\dots,n\}} d_i \ \leq \sin\alpha_m$$

And thus because of (3.19)

$$\max_{i\in\{1,\dots,n\}} \|\mathbf{R}_i - \mathbf{M}_{opt}\|_2 = \sin\alpha_m$$

must be valid.
If $\mathbf{R}_1, \ldots, \mathbf{R}_n$ do not fulfill the cone condition, it is:

$$\forall\, \mathbf{R} \in \mathrm{SO}(3): \quad \max_{i \in \{1,\ldots,n\}} \delta(\mathbf{R}, \mathbf{R}_i) \geq \frac{\pi}{2}$$

Because otherwise rotation quaternions $\mathbf{q}$ and $\mathbf{q}_1, \ldots, \mathbf{q}_n$ corresponding to $\mathbf{R}$ and $\mathbf{R}_1, \ldots, \mathbf{R}_n$ with $\mathbf{q}^T \mathbf{q}_i \geq 0$, $1 \leq i \leq n$, would exist which lie within a circle with an opening angle less than $\pi/2$ because of (3.5). Thus in conjunction with (3.18) we get:

$$\min_{\mathbf{M} \in \mathrm{Mat}_{3\times 3}} \max_{i \in \{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{M}\|_2 \geq f\left(\frac{\pi}{2}\right) = 1$$

Obviously this lower bounding value is reached for $\mathbf{M} = \mathbf{0}$.

□

3.3 Rigid Transformation Hierarchies Buildup and Usage

Now as we have shown how for a finite set $\Theta = \{T_1, \ldots, T_n\}$ of rigid transformations $T_i : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \mathbf{R}_i\mathbf{c} + \mathbf{b}_i$, $\mathbf{R}_i \in \mathrm{SO}(3)$, $\mathbf{b}_i \in \mathbb{R}^3$, $1 \leq i \leq n$, we can find a transformation $T(\Theta) : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto c(\Theta)\mathbf{R}(\Theta)\mathbf{x} + \mathbf{b}(\Theta)$, $\mathbf{M} \in \mathrm{Mat}_{3\times 3}$, $\mathbf{b} \in \mathbb{R}^3$ so that we have an upper bound

$$|T_i(\mathbf{x}) - T(\mathbf{x})| \leq r_{\mathbf{R}}(\Theta)\,|\mathbf{x}| + r_{\mathbf{b}}(\Theta) \quad , \quad 1 \leq i \leq n\,, \tag{3.20}$$

we can build up some kind of bounding volume hierarchy (BVH) over these transformations that we want to call a rigid transformation hierarchy (RTH). Because the upper bound on the right side in (3.20) contains $|\mathbf{x}|$, the geometry these transformations are applied to has to be taken into account. In order to replace $|\mathbf{x}|$ by an as small as possible constant so that (3.20) is really a global bound, the smallest enclosing sphere of the whole geometry to which the transformations are applied has to be calculated. This can be done again by the Welzl algorithm, the center of this sphere has to be moved to the origin and then $|\mathbf{x}|$ can be bounded by the radius of this sphere. When moving the center $\mathbf{c} \in \mathbb{R}^3$ of the enclosing sphere to the origin by the coordinate transformation $V : \mathbb{R}^3 \to \mathbb{R}^3, \mathbf{x} \mapsto \tilde{\mathbf{x}} = \mathbf{x} - \mathbf{c}$ the transformations T_i have to be replaced by VT_iV^{-1}:

$$VT_iV^{-1}:\ \mathbb{R}^3 \to \mathbb{R}^3\ ,\ \mathbf{x} \mapsto [\mathbf{R}_i(\mathbf{x} + \mathbf{c})] + \mathbf{b}_i - \mathbf{c} = \mathbf{R}_i\mathbf{x} + \mathbf{b}_i + \mathbf{R}_i\mathbf{c} - \mathbf{c}$$

Thus the translation vectors $\mathbf{b}_i$ have to be replaced by $\mathbf{b}_i + \mathbf{R}_i\mathbf{c} - \mathbf{c}$ before the RTH is built up. Then $|\mathbf{x}|$ in (3.20) can be bounded by the radius r of the enclosing sphere of the geometry to which the transformations are applied, and we get the global bound

$$|T_i(\mathbf{x}) - T(\mathbf{x})| \leq r_{\mathbf{R}}(\Theta)\,r + r_{\mathbf{b}}(\Theta) \quad , \quad 1 \leq i \leq n\,. \tag{3.21}$$

Algorithm 3.2 Recursive build up of an RTH node for a set Θ of rigid transformations

1: **function** BUILDUP_NODE(Θ)
2: calculate and store $c(\Theta)$, $\mathbf{R}(\Theta)$ and $r_{\mathbf{R}}(\Theta)$ according to lemma 3.3 and theorems 3.7, 3.9 and 3.11
3: calculate and store $\mathbf{b}(\Theta)$ and $r_{\mathbf{b}}(\Theta)$ as center and radius of the smallest enclosing sphere of the translation vectors of the transformations in Θ by the standard Welzl algorithm
4: **if** abort criterion is not fulfilled **then**
5: divide Θ in two subsets Θ_1 and Θ_2
6: build up children recursively by calling BUILDUP_NODE(Θ_1) and BUILDUP_NODE(Θ_2)
7: **end if**
8: **end function**

The RTH can now be built up in just the same way like a conventional BVH by the recursive function for building up a node for a set Θ of rigid transformations shown in algorithm 3.2.
Typical choices for the abort criterion at line 4 for any kind of BVH are an upper bound for the recursion resp. tree depth or a lower bound for the number of elements – in this case transformations – per leaf. Another obvious abort criterion specifically for RTHs is to abort if the error $r_{\mathbf{R}}(\Theta)\, r + r_{\mathbf{b}}(\Theta)$ for the transformation approximation according to (3.21) drops below a certain threshold. And of course combinations of all these criteria can be used.
Another important point is how the transformations are divided into two subsets at line 5 in algorithm 3.2 in order to recursively build up the child nodes in the hierarchy. The transformations consist of rotations and translations. Because it seems not to be justified to assume that two transformations with similar rotations must have similar translations without making requirements for the input data, the transformations are divided according to either the rotations or the translations depending on if $r_{\mathbf{R}}(\Theta)\, r$ or $r_{\mathbf{b}}(\Theta)$ is bigger for the current node. Such an alternating division criterion is also known from k-d trees [45] in which points are organized and at every node the points in the left and right subtree are divided according to either their x, y or z coordinate.
An usual strategy to divide a set of points during the buildup of a BVH is to calculate their covariance matrix in their centroid system and divide them according to their projection to the principal axis which corresponds to the biggest eigenvalue of the covariance matrix. We proceed in the same way when dividing based on the translations by considering the translation vectors as points in $\mathbb{R}^3$ and when dividing based on the rotations by considering the rotation quaternions as 4-dimensional points. That is done, of course, after the quaternions have been possibly flipped yielding $\mathfrak{q}_1^T \mathfrak{q}_i \geq 0$, $2 \leq i \leq n$, for a set of rotation quaternions $\mathfrak{q}_1, \ldots, \mathfrak{q}_n \in \mathfrak{Q}^*$. If the cone condition is not fulfilled, this simple flipping criterion might not be ideal

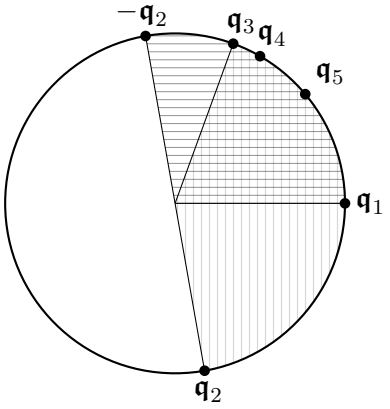


Figure 3.8: If the cone condition is not fulfilled so that a set a of unit quaternions cannot be embraced by a circle with an opening angle smaller than $\pi/2$, flipping the quaternions so that

$$\mathbf{q}_1^T \mathbf{q}_i \geq 0 \quad , \quad 2 \leq i \leq n$$

might not be ideal since the horizontally hatched cone is bigger than the vertically hatched cone.

as shown in figure 3.8. However, after some subdivisions of the transformations in the RTH, the cone condition should be fulfilled in any case for any input data for the deeper hierarchy levels. Because the data provided by our industrial partner describes only a vibration movement of an engine, the cone condition is fulfilled for that data even for all transformations on the whole.
Finally if $A \subset \mathbb{R}^3$ is the fixed geometry and $B \subset \mathbb{R}^3$ the moving geometry and by a function $\mathrm{dist}(A, B, T)$ their distance according to

$$\mathrm{dist}(A, B, T) := \min_{\mathbf{a} \in A, \mathbf{b} \in B} |\mathbf{a} - T(\mathbf{b})|$$

can be calculated, the minimum distance for A and B over a set Θ of rigid transformations

$$\min_{T \in \Theta} \mathrm{dist}(A, B, T) \tag{3.22}$$

can be calculated by traversing an RTH which has been built up over the transformations. For that an upper bound d for the distance according to (3.22) is initialized by infinity which will become exact after traversing the tree by the recursive function shown in algorithm 3.3.
Some things have to be considered when traversing an RTH: At first the function dist(A, B, T), which calculates the distance between A and transformed B by traversing a classical BVH, is usually designed only for rigid transformations, but at line 4 in algorithm 3.3 it is applied to T according to

$$T : \mathbb{R}^3 \to \mathbb{R}^3 \quad , \quad \mathbf{x} \mapsto c(\Theta_n)\mathbf{R}(\Theta_n) + \mathbf{b}(\Theta_n) \, . \tag{3.23}$$

This is not a rigid transformation because of the scaling factor $c(\Theta_n) \in [0, 1]$. But internally this function calculates its result as minimum of distances between triangles of A and transformed triangles of B and there are early exits based on calculated distances between bounding volumes of triangles of A and transformed bounding volumes of triangles of B. The transformation of triangles is done vertexwise, which remains correct for an affine transformation according to (3.23) because affine transformations map straight lines on straight lines. The transforma-

Algorithm 3.3 Minimum distance calculation by traversing an RTH
d is passed by reference and initialized by ∞

```
1:  function TRAVERSE_NODE(RTH node n, minimum distance d so far)
2:      Θ_n ← set of transformations contained by node n
3:      Retrieve c(Θ_n), R(Θ_n), b(Θ_n), r_R(Θ_n) and r_b(Θ_n) stored for node n
4:      d_n ← dist(A, B, T) with T : ℝ³ → ℝ³, x ↦ c(Θ_n)R(Θ_n)x + b(Θ_n)
5:      e ← r_R(Θ_n)r + r_b(Θ_n)
6:      if d_n − e ≥ d then
7:          return
8:      end if
9:      if n is leaf then
10:         for all T ∈ Θ_n do
11:             d = min{d, dist(A, B, T)}          ▷ update distance bound
12:         end for
13:     else
14:         TRAVERSE_NODE(left child of n, d)
15:         TRAVERSE_NODE(right child of n, d)
16:     end if
17: end function
```

tion of a bounding volume by a rigid transformation is usually done by transforming the in some way defined center and possibly rotating an orientation, e.g. for OBBs. When applying a transformation according to (3.23) the bounding volume has to be additionally scaled by $c(\Theta_n)$. However, since the bounding volumes are usually convex and $c(\Theta_n) \in [0, 1]$, the correctly transformed bounding volume is a subset of the transformed bounding volume with omitted scaling factor resp. scaling factor set to 1. Thus and because the distance of the bounding volumes is only used as bound to decide if an early exit is possible, the $\mathrm{dist}(A, B, T)$ function does not need necessarily to be adapted for non-rigid transformations according to (3.23), especially because the scaling factor for deeper levels in the RTH will usually be nearly 1.

Secondly, a BVH is usually not traversed by a simple DFS as it would be done by the recursive function above, but the traversal order is optimized by a heuristic so that the upper bound drops as fast as possible and there are more early exits at line 7 in algorithm 3.3. This is usually done by managing the nodes to traverse next in a priority queue, and in our case it is an obvious idea to sort this priority queue by the d_n values according to line 4 of algorithm 3.3 and assign nodes with smaller d_n values a higher priority.

Now we have shown how the calculation of the minimum distance between a fixed and moving geometry resp. set of triangles for a set of rigid transformations can be done with the help of an RTH. Of course, just as it can be discovered for which triangles the minimum distance is reached when traversing classical BVHs, it can be discovered when traversing the RTH for which rigid transformation the mini-

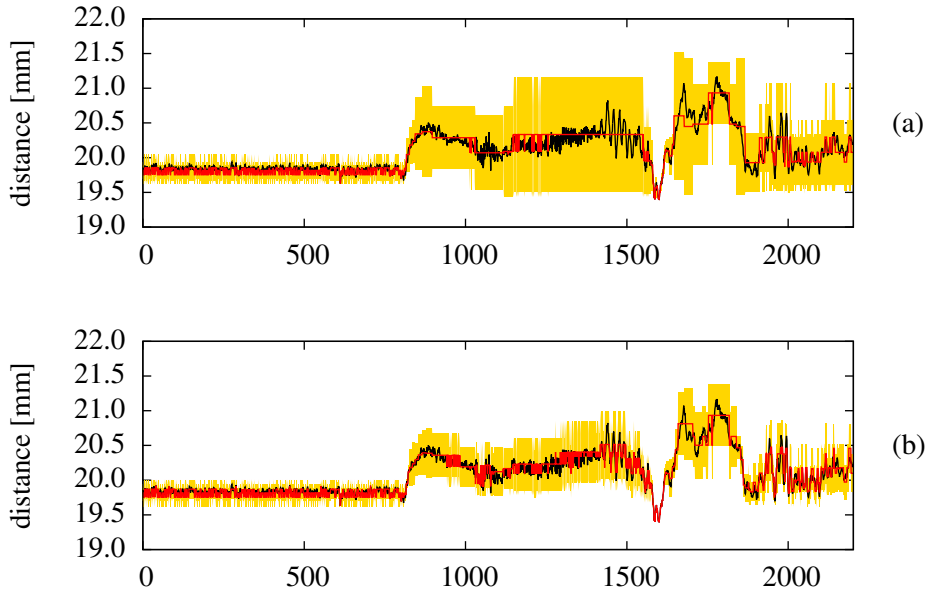


Figure 3.9: Exactly calculated distances (black) for a track consisting of 2202 transformations compared to distances approximately calculated by an RTH (red, error margin yellow) with (a) $d_n - e \geq d$ and (b) $d_n - 2e \geq d$ as early exit criterion.

mum distance is reached. This is done by extending line 11 in algorithm 3.3 so that when the upper bound d is reduced by a distance value for a certain transformation this transformation is stored and thus the time point on the track can be found for which the fixed and the moving geometry get closest.
However, as mentioned at the beginning of this chapter our industrial partner is not only interested in this single time point but also other time points when both geometries get quite close, too. But the RTH approach can be adapted in order to calculate the distance approximately. The error e calculated at line 5 in algorithm 3.3 means that the distance for the transformations contained by this node is in the range $[d_n - e, d_n + e]$. Thus, by adapting the criterion at line 6 we can make the early exits dependent on certain accuracy requirements for the approximation error of d_n. An obvious adaption is to tighten up this criterion by adding a factor f bigger than 1:

$$d_n - fe \geq d \quad , \quad f \geq 1 \tag{3.24}$$

In figure 3.9 a distance profile of an about 11 second long section of a test drive with 2202 transformations is shown. It is exactly calculated as well as approximately by an RTH with $f = 1$ for the upper and $f = 2$ for the lower graph according to (3.24) as early exit criterion at line 4 in algorithm 3.3. By the RTH the calculation could be sped up by a factor of about 5 for $f = 1$ and about 2.5 for $f = 2$. (Further and more detailed results will be presented in the next section.)
As it can be seen especially for the transformations 1000 to 1500, the approxima-

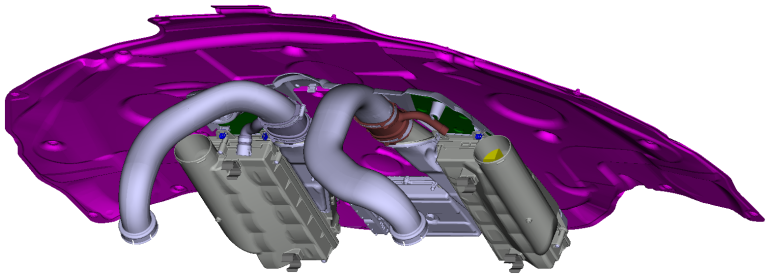


Figure 3.10: A test scenario for evaluating the RTH approach consisting of some upper parts of the engine and the hood.

tive calculation resembles the exact course of the distance for $f = 2$ much better than for $f = 1$. The error is strongly overestimated in both cases. For this overestimation there are two reasons. Firstly, the triangle inequation is used in (3.2) when combining the rotational and translational error while both errors might even cancel out each other to a certain degree in many cases. And secondly, the rotational error for the moving geometry is bounded by $r_{\mathbf{R}}(\Theta_n)r$ with r the radius of a sphere enclosing the moving geometry. But the point on the moving geometry which gets closest to the fixed geometry might be nearer to the center of this enclosing sphere than to its surface and then the "real" radius for this point is actually much smaller. However, finally this overestimation has to be accepted to keep conservativeness of the bound. But of course it is clear, that the radius r in the bound of the rotational error limits the application of the RTH approach to transformation tracks with many quite similar rotations resp. orientations. But usually that is the case for vibration movements. In the outlook some ideas will be sketched how the RTH approach can be extended in order to reduce the radius in the rotational error bound and thus to overcome this limitation.

3.4 Experimental Results

The RTH approach was evaluated for a track from a test drive consisting of over 180.000 rigid transformations delivered by our industrial partner. The test drive has been split into 28 driving maneuvers by our industrial partner. The RTH approach was applied to calculate an approximate course of the minimum distance between fixed parts of the engine compartment and moving parts of the engine during all these maneuvers as well as the track on the whole. This approximate calculation is done with a factor of $f = 1$ according to (3.24), as it is necessary for calculating the minimum distance throughout the track, as well as a factor of $f = 2$ in order to tighten up the approximation.
Usually our industrial partner does not test a complete engine and the complete engine compartment at once, but picks out some critical parts only. Thus he provided two scenarios for testing. The less complex scenario A consists of parts of

	moving parts		fixed parts	
	input	stored in BVH leaves	input	stored in BVH leaves
scenario A	109.094	180.826	89.187	188.986
scenario B	427.450	936.147	84.236	253.697

Table 3.1: Number of triangles for the moving parts belonging to the engine and for the fixed parts belonging to the engine compartment.

the exhaust system as parts of the engine and some shielding metal plates as parts of the compartment as they could already be seen in figure 3.2 at the beginning of this chapter. The other more complex scenario B consists of some upper parts of the engine and the hood and can be seen in figure 3.10.
RSSs are used as bounding volumes for the BVH managing the triangles of the geometry. The RSS BVH is built up in a top-down manner by iteratively splitting the set of triangles and calculating a left and a right bounding volume for the two subsets of triangles. That one of the three middle planes of every RSS is used for the splitting whose normal points into the direction of the biggest extension of the RSS. Then a tolerance region is defined by extending this plane by 2 mm to every side. Triangles spanning this tolerance region, i.e. there is a triangle vertex left and a triangle vertex right of this plane, are cut at the plane. If the third triangle vertex lies within the tolerance region, the triangle is split into two triangles, otherwise into three triangles. By the tolerance region it is avoided that very small triangles are generated which could lead to numerical stability issues.
The buildup recursion is aborted and a RSS bounding volume not further split if it has an extent smaller than 10 mm in all of its principal directions or if the contained triangle fragments come all from the same input triangle. Then in the BVH leaf node not the split triangles are stored but the input triangles because a leaf bounding volume can contain several fragments from the same input triangle. Thus after the BVH build up an input triangle is possibly stored in several BVH leaves whose bounding volumes, as a whole, cover the input triangle.
In table 3.1 the number of input triangles for the fixed and moving parts are listed as well as the number of triangles that are finally stored in the BVH leaf nodes. Scenario B is more complex than scenario A because the number of triangles for the moving parts is four times higher and because in scenario B the upper parts of the engine are quite "parallel" to the hood and so the region where fixed and moving parts can get closest is quite large.
Just like the BVH, the RTH data structure for the rigid transformations is also built up in a top down manner as described in the previous section and the transformations are divided until there is only one transformation on the leaf level, too. For comparison with the better, but slower $f = 2$ approximation the running time for calculating the exact minimum distances for all transformation was measured. The running time for the faster $f = 1$ approximation was compared with a naive

	number transfor-mations	RTH buildup time [ms]	exact calc. [sec]	RTH $f = 2$ [sec]	speed-up	naive min. dist. calc. [sec]	RTH $f = 1$ [sec]	speed-up
1	2202	8.4	3.33	0.41	8.2	1.105	0.207	5.3
2	2220	8.0	1.67	0.30	5.6	0.960	0.196	4.9
3	2531	10.2	5.98	1.00	6.0	1.403	0.471	3.0
4	2802	11.0	2.17	0.22	9.8	1.289	0.140	9.2
5	3824	15.0	2.97	0.19	15.5	1.511	0.136	11.1
6	3864	14.6	4.38	0.12	35.5	1.226	0.089	13.8
7	3894	14.0	30.15	6.89	4.4	17.179	3.315	5.2
8	4109	15.2	5.90	0.41	14.5	1.884	0.187	10.1
9	4208	17.0	4.68	0.26	18.0	1.907	0.149	12.8
10	4292	17.0	5.53	0.80	6.9	2.331	0.347	6.7
11	4605	17.0	6.53	0.94	7.0	2.202	0.458	4.8
12	4764	19.5	3.83	0.17	22.3	2.116	0.110	19.2
13	4899	20.0	8.40	0.96	8.7	2.626	0.433	6.1
14	5240	20.0	38.65	7.82	4.9	22.257	3.791	5.9
15	5898	21.8	22.25	0.86	26.0	2.936	0.437	6.7
16	6160	22.0	25.47	0.93	27.4	2.624	0.516	5.1
17	6227	22.9	27.42	1.11	24.7	2.731	0.500	5.5
18	6612	26.2	8.00	0.53	15.0	2.882	0.197	14.7
19	6686	25.9	9.40	0.25	37.6	2.839	0.214	13.3
20	7404	28.4	40.62	1.02	40.0	3.272	0.521	6.3
21	7659	28.0	33.17	1.06	31.4	3.522	0.488	7.2
22	7889	30.4	12.43	0.77	16.1	4.005	0.378	10.6
23	8400	31.6	44.48	1.32	33.7	3.710	0.570	6.5
24	11974	54.1	14.11	1.17	12.1	5.137	0.366	14.0
25	12978	60.5	12.41	0.98	12.7	12.930	0.370	34.9
26	13757	63.3	24.67	1.08	22.9	5.099	0.336	15.2
27	13976	63.0	27.07	1.84	14.7	7.837	0.705	11.1
28	14128	63.2	143.48	16.17	8.9	30.363	4.345	7.0
	183472	1012.6	586.24	2.88	203.9	99.788	0.797	125.2

Table 3.2: Running time results for a minimum distance computation by the RTH approach for scenario A with $f = 1$ and $f = 2$ according to (3.24) in comparison to an exact computation of all minimum distances for all transformations and a naive minimum distance calculation (see text).

	number transfor-mations	RTH buildup time [ms]	exact calc. [sec]	RTH $f = 2$ [sec]	speed-up	naive min. dist. calc. [sec]	RTH $f = 1$ [sec]	speed-up
1	2202	8.0	63.8	9.7	6.6	9.7	7.2	1.3
2	2220	8.0	70.1	8.6	8.2	31.4	6.1	5.1
3	2531	10.0	73.6	13.8	5.3	10.7	8.0	1.3
4	2802	10.7	72.2	5.3	13.6	23.6	4.2	5.6
5	3824	15.0	117.0	14.5	8.1	38.6	6.8	5.7
6	3864	14.0	120.9	1.9	63.9	23.4	1.9	12.6
7	3894	14.0	143.0	8.5	16.8	16.1	5.9	2.7
8	4109	15.0	158.4	7.0	22.6	42.3	4.8	8.8
9	4208	17.0	117.5	8.9	13.2	23.1	6.5	3.5
10	4292	17.0	127.1	9.1	14.0	20.9	5.5	3.8
11	4605	17.0	253.4	11.6	21.8	29.2	9.2	3.2
12	4764	20.0	135.0	3.5	38.2	44.7	2.6	17.2
13	4899	20.0	175.7	11.1	15.9	19.7	6.9	2.8
14	5240	20.0	191.2	6.2	30.8	22.1	4.7	4.7
15	5898	22.0	189.0	8.8	21.4	34.2	7.2	4.8
16	6160	22.9	204.3	9.9	20.5	80.2	4.4	18.1
17	6227	22.7	211.8	15.5	13.7	73.9	9.2	8.0
18	6612	25.7	200.6	7.3	27.4	63.9	5.2	12.3
19	6686	26.0	305.6	20.0	15.2	43.7	13.1	3.3
20	7404	28.0	216.4	10.7	20.2	100.2	4.9	20.3
21	7659	27.9	268.3	13.8	19.5	58.2	8.6	6.7
22	7889	30.0	267.0	5.9	44.9	57.8	3.7	15.6
23	8400	32.0	243.2	12.0	20.2	128.8	6.0	21.4
24	11974	52.6	380.9	21.1	18.0	148.9	7.3	20.5
25	12978	58.9	370.0	12.8	28.8	138.0	4.9	28.2
26	13757	64.5	607.9	18.8	32.4	205.2	9.8	20.8
27	13976	62.0	484.9	6.5	74.1	101.0	4.5	22.5
28	14128	62.3	324.8	4.1	79.3	102.0	3.3	30.5
	183472	1013.0	6784.6	13.0	522.4	1417.3	6.4	221.5

Table 3.3: Running time results for a minimum distance computation by the RTH approach for scenario B with $f = 1$ and $f = 2$ according to (3.24) in comparison to an exact computation of all minimum distances for all transformations and a naive minimum distance calculation (see text).

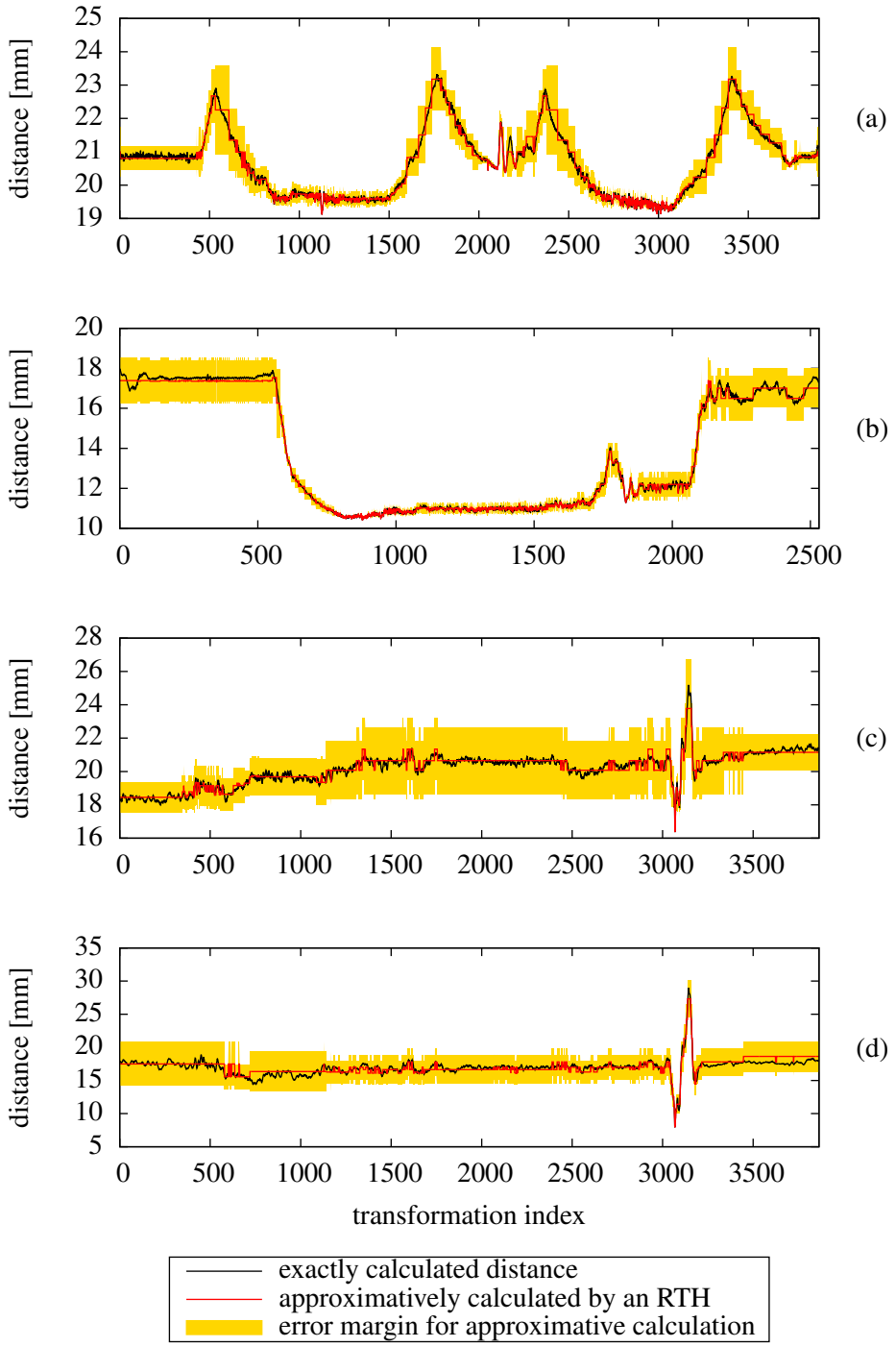


Figure 3.11: Exact and by RTH approach for $f = 2$ approximatively calculated distances for driving maneuver 7 for scenario A (a), maneuver 3 for scenario B (b) and maneuver 6 for scenario A (c) and scenario B (d). In the two former cases the approximative calculation is quite slow and in the two latter cases it is quite fast.

approach for calculating the minimum distance throughout a track, which was already much faster than calculating the minimum distances for all transformations. For this naive approach the BVHs for the fixed and moving geometry are traversed just like for a distance check for a single transformation. But a pair of bounding volumes is only skipped if the bounding volume containing triangles of the fixed geometry has a distance larger than the currently smallest distance to the bounding volume containing the triangles of the moving geometry transformed by *all* transformations of the track. On the leaf level the distance for every pair of triangles of both geometries is calculated for every transformation. By this naive approach a speed-up of up to 12.4 could be achieved compared to calculating the minimum distance for all transformations except one case in which it was equally fast.
The results for scenario A are shown in table 3.2 and for scenario B in table 3.3 with the driving maneuvers sorted in ascending order of the transformation count (2nd column) and with the total track at the bottom. The measurement was done on an Intel® Core™ i7-970 CPU at 3.20 GHz with 6 cores, parallelized by 6 threads and repeated 10 times. The tables contain the average of the measured values.
Additionally the tables contain the build up times for generating the RTH data structure (3rd column), which are quite small and thus listed in milliseconds although the buildup is done by only one thread. The build up of the RTH for all over 180.000 transformations takes only about a second.
Obviously it can be expected that the longer the track, the higher the achievable speed-up should be because the criterion for early exits when traversing the RTH depends on the minimum distance throughout the whole track found so far. And the more transformation there are, the more transformations are likely to lead to a larger minimum distance than the minimum distance throughout the whole track and and thus might be skipped by an early exit. This expectation is confirmed by the measurement at least in this point that the speed-ups for the whole track are an order of magnitude greater than for the single driving maneuvers. The speed-ups for the single driving maneuvers lie between 4.4 and 40.0 for scenario A and 5.3 and 79.3 for scenario B for the comparison between exact computation and approximative calculation with an RTH with $f = 2$. For the comparison between the naive calculation of the minimum distance throughout the whole track and the approximative calculation with an RTH with $f = 1$ they lie between 3.0 and 34.9 for scenario A and between 1.3 and 30.5 for scenario B.
However, there are some driving maneuvers for which the RTH based minimum distance calculation is quite slow compared to the others. For example, this is the case for maneuver 7 for scenario A, for which the exact and approximatively calculated distance is illustrated in figure 3.11(a), and it is the case for maneuver 3 for scenario B (figure 3.11(b)). Furthermore there are some maneuvers for which the RTH based minimum distance calculation is quite fast compared to the others. For example, for maneuver 6 that is the case for both scenarios (scenario A figure 3.11(c), scenario B figure 3.11(d)). As it can be seen in the figures for the two slow cases, there are many transformations for which nearly the total minimum distance is reached. Therefore these transformations cannot be skipped at high levels in the

RTH because the lower bound of the error margin would be smaller than the total minimum. Instead the RTH has to be traversed down to the leaf level, which makes the RTH approach not so efficient for these cases.
In contrast, for the two latter cases the total minimum value is reached in a sharp minimum. Thus for all other transformations outside of this minimum the distance is much larger than the total minimum and thus the RTH traversal can be skipped for these transformations at high levels making the RTH approach very efficient.

3.5 Future Work

So far the RTH approach has not yet been integrated into the software for our industrial partner. An integration would be possible in several ways. For example it would be conceivable to present the user quickly the result for the total minimum distance throughout the track by the RTH approach as well as an approximate course of the distance as shown in figure 3.11 with big error margins. But then the computation could continue in background and the nodes of the RTH which were skipped so far could be further processed by keeping them in a priority queue and making the error margins smaller and smaller and the course of the distance more and more precise. The user could proceed with his work at the car model and perhaps already take the decision that the minimum distance throughout the track is too small and the model has to be revised.
Another feature request by our industrial partner is that not only the two points on fixed and moving parts should be calculated which get closest in the course of a track, but he also wants to know if there are other points on the parts with a certain spatial distance to the totally closest points which reach a similar proximity.
A naive approach to realize this feature could be to divide the parts into cells by a cubic grid and to do a separate distance calculation for every cell. Then for every cell the total minimum distance could be quickly calculated by the RTH approach and based on these total minimum distances for every cell it could be decided due to certain criteria for which cells the calculation must be continued for more precise values throughout the track. For example, such a criterion could be that only distances are of interest which are at most twice as big as the total minimum distance.
Another advantage of dividing the parts into cells would be that for every cell an own RTH data structure could be built up with the cell's center as origin of the coordinate system. Then the radius of the moving geometry's enclosing sphere in the transformation bounding inequation (3.21), which is used in algorithm 3.3 at line 5, could be replaced by half the cell diameter and thus be reduced by using smaller cells.
A similar effect could be reached by building up several RTH data structures with the centers of the moving geometry's bounding volumes at a certain level of the BVH as origin. Then the radius of the enclosing spheres of these bounding volumes could be used as radius at line 5 in algorithm 3.3. The deeper the level at

Algorithm 3.4 Swept volume containment test by traversing an RTH

```
1: function TRAVERSE_NODE(rigid transformation hierarchy node n)
2:     Θ_n ← set of transformations contained by node n
3:     Retrieve c(Θ_n), R(Θ_n), b(Θ_n), r_R(Θ_n) and r_b(Θ_n) stored for node n
4:     if f(c(Θ_n)R(Θ_n)x + b(Θ_n)) > r_R(Θ_n)|x| + r_b(Θ_n) then
5:         return false
6:     end if
7:     if n is leaf then
8:         for all T ∈ Θ_n do
9:             if T(x) ∈ A then
10:                return true
11:            end if
12:        end for
13:        return false
14:    else
15:        return TRAVERSE_NODE(left child of n) or
16:                TRAVERSE_NODE(right child of n)
17:    end if
18: end function
```

which that was done, the smaller would be the radius and thus the lower the upper bound for the error, but the more RTHs would have been to be built up.
Besides there are also completely different applications of the RTH data structure which are of highly interest in the automotive industry in the context of computer aided design as well.
For example, when an RTH for a track of rigid transformations $T_1, \ldots, T_n$, $T_i : \mathbb{R}^3 \to \mathbb{R}^3$, $1 \leq i \leq n$, is combined with an adaptive distance field for an object $A \subset \mathbb{R}^3$ as described in section 2.1, the swept volume of that object along the track can be described implicitly. If we define a "discrete" swept volume S as the union of A transformed by all rigid transformations

$$S := \bigcup_{i=1}^{n} T_i(A) \quad \text{with} \quad T_i(A) := \{T(\mathbf{x}) \,|\, \mathbf{x} \in A\} \,, \tag{3.25}$$

a point $\mathbf{x}$ is contained by S iff

$$\mathbf{x} \in S \;\Leftrightarrow\; \exists\, i \in \{1, \ldots, n\} : \mathbf{x} \in T_i(A) \;\Leftrightarrow\; \exists\, i \in \{1, \ldots, n\} : T_i^{-1}(\mathbf{x}) \in A \,.$$

If we generate an adaptive distance field for A delivering a fast function $f : \mathbb{R}^3 \to \mathbb{R}_{\geq 0}$ that returns a lower bound for the distance of $\mathbf{x}$ to A for any point $\mathbf{x} \in \mathbb{R}^3$

$$f(\mathbf{x}) \leq d(\mathbf{x}, A) = \sup_{\mathbf{a} \in A} d(\mathbf{x}, \mathbf{a}) \,,$$

we can build up an RTH over the inverse transformations and then the recursive function traversing the RTH shown in algorithm 3.4 returns if a point $\mathbf{x} \in \mathbb{R}^3$ is

contained by the swept volume S.
In order to keep $|\mathbf{x}|$ small and to increase the likelihood of an early exit at line 5 in algorithm 3.4 it is important to choose the origin of the coordinate system properly. For example, this can be done by calculating the smallest enclosing sphere of A with center $\mathbf{c}_A$ and radius r_A and then the smallest enclosing sphere of $T_1(\mathbf{c}_A), \ldots, T_n(\mathbf{c}_A)$ with center $\mathbf{c}_T$ and radius r_T by the Welzl algorithm. The swept volume is then contained by the sphere with center $\mathbf{c}_T$ and radius $r_A + r_T$. Thus, if $\mathbf{c}_T$ is taken as origin and $T_1^{-1}, \ldots, T_n^{-1}$ transformed accordingly before building up the RTH, only points $\mathbf{x} \in \mathbb{R}^3$ with $|\mathbf{x}| \leq r_A + r_T$ in this coordinate system can be contained by the swept volume.
This approach for implicitly representing a "discrete" swept volume according to (3.25) can also be extended in order to represent some kind of continuous swept volume. This can be done by building up the RTH without reordering of the transformations and keeping an overlap of one transformation when splitting the set of transformations for an RTH node for the child nodes. Additionally the containment test for a point $\mathbf{x}$ at line 9 in algorithm 3.4 has to be adapted. For example, this can be done by testing all lines between $T_i(\mathbf{x})$ and $T_{i+1}(\mathbf{x})$ for any consecutive transformations T_i and T_{i+1} in Θ_n for intersection with the ADF by a sampling strategy.

Chapter 4

Comparing Two Independent Tracking Systems

In the last years several types of motion tracking systems have been developed independently. A big group among these system, which usually achieves the highest precision, works by placing so-called markers on the object to be tracked. Most of these systems works optically by using cameras and image processing to retrieve the marker positions, but also non-optical systems, for example using ultrasound, have been developed.
Many of these systems are offered at quite low costs on the market but cannot be easily compared because one has to rely completely on the manufacturer information about their accuracy. Thus it is an obvious approach to use two or even more system simultaneously to compare or combine their tracking data. In this context several mathematical problems arise, two of which will be dealt with in the following two sections.

4.1 Calculating Orientations Relative to "Average" Marker Coordinates

The available systems using markers can be divided into two groups. For the one group it is assumed that all tracked markers are fixed to a single object and thereby have a fixed relative position to each other. This is taken into account in the mathematical model for retrieving the marker coordinates from the raw data for noise reduction and thus these systems output only coordinates which have fixed relative positions to each other. The other group of systems do not make this assumption and track each marker independently from the others. If the markers are fixed to a single object, these systems will output marker coordinates which will not have a fixed relative position to each other because of noise.
Calculating the rigid transformation that transforms the marker coordinates at one

time point into the coordinates at the next or any other time point can be done for the first group by solving a simple system of linear equations. To do that for the second group is much harder because some kind of optimization has to be done. For example this optimization can be the minimization of the sum of the squared Euclidean distances of the transformed marker coordinates at one time point and the marker coordinates at the other time point. If we denote the coordinates at one time point by $\mathbf{p}_1, \ldots, \mathbf{p}_n$ and at the other by $\mathbf{p}'_1, \ldots \mathbf{p}'_n$ and look for the rigid transformation $(\mathbf{R}, \mathbf{b}) \in \mathrm{SO}(3) \times \mathbb{R}^3$, $(\mathbf{R}, \mathbf{b}) : \mathbb{R}^3 \to \mathbb{R}^3$, $\mathbf{x} \mapsto \mathbf{R}\mathbf{x} + \mathbf{b}$, this can be written as

$$\underset{(\mathbf{R},\mathbf{b})\in \mathrm{SO}(3)\times\mathbb{R}^3}{\arg\min} \sum_{i=1}^{n} \left|\mathbf{R}\mathbf{p}_i + \mathbf{b} - \mathbf{p}'_i\right|^2 . \tag{4.1}$$

We want to assume that the markers are properly fixed to the object to be tracked so that $\mathbf{P}$ and $\mathbf{P}'$ have full row rank and the solution of (4.1) is unique. The minimization problem (4.1) has already been solved in 1986 for the 3-dimensional case [46] by finding the optimal rotation's unit quaternion as the eigenvector of a 4×4 matrix for the largest eigenvalue. In 1987 another solution was presented that also applies to the n-dimensional case and finds the solution by the singular value decomposition (SVD) of an $n\times n$ matrix [47]. Later the problem has been extended to the case that the assignment $\mathbf{p}_i \leftrightarrow \mathbf{p}'_j$, which point has to be transformed into which other point, is unknown and solved by an iterative algorithm, which is known as the *ICP* (iterative closest point) algorithm [48].
However, in our case the assignment of points is known due to our ability to distinguish between the markers, and thus the original solution [46, 47] is sufficient. The first statement of these solutions is that the translation vector of the rigid transformation can be retrieved as the translation vector between both point sets' centroids and the rotational problem

$$\underset{\mathbf{R}\in \mathrm{SO}(3)}{\arg\min} \sum_{i=1}^{n} \left|\mathbf{R}\bar{\mathbf{p}}_i - \bar{\mathbf{p}}'_i\right|^2 \tag{4.2}$$

with $\bar{\mathbf{p}}_i := \mathbf{p}_i - \bar{\mathbf{p}}$, $\bar{\mathbf{p}} := {}^1\!/_n \sum_{i=1}^n \mathbf{p}'_i$, $\bar{\mathbf{p}}'_i := \mathbf{p}'_i - \bar{\mathbf{p}}'$, $\bar{\mathbf{p}}' := {}^1\!/_n \sum_{i=1}^n \mathbf{p}'_i$, denoting the points in their centroid system, has to be solved separately. By writing the points $\bar{\mathbf{p}}_i$ and $\bar{\mathbf{p}}'_i$ as columns of matrices $\mathbf{P}, \mathbf{P}'$ and using the Frobenius norm $\|.\|_F$ (4.2) can be written as

$$\underset{\mathbf{R}\in \mathrm{SO}(3)}{\arg\min} \left\|\mathbf{R}\mathbf{P} - \mathbf{P}'\right\|_F . \tag{4.3}$$

Because the solution of this problem can be extended easily to the $\arg\max$-case with a plus in (4.3) and its solution is the essential ingredient of the algorithms presented in this chapter, we want to recapitulate it including its proof briefly in the following lemma.

Lemma 4.1 *Let $\mathbf{P}, \mathbf{P}' \in \mathbb{R}^{d\times n}$ be two $d\times n$ matrices with $d \leq n$ and full row rank. Then it is*

$$\underset{\mathbf{R}\in \mathrm{SO}(d)}{\arg\min} \left\|\mathbf{R}\mathbf{P} - \mathbf{P}'\right\|_F = \underset{\mathbf{R}\in \mathrm{SO}(d)}{\arg\max} \left\|\mathbf{R}\mathbf{P} + \mathbf{P}'\right\|_F \tag{4.4}$$

$$= \mathbf{V} \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{UV}) \end{pmatrix} \mathbf{U}^T$$

$$\textit{with} \quad \mathbf{PP'}^T = \mathbf{U\Sigma V}^T = \mathbf{U} \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \end{pmatrix} \mathbf{V}^T$$

the SVD of $\mathbf{PP'}^T$ *so that* $\mathbf{U}$ *and* $\mathbf{V}$ *are orthogonal matrices and* $\sigma_1 \geq \ldots \geq \sigma_d \geq 0$ *its singular values in descending order.*

Proof:
Because it is $\|\mathbf{A}\|_F^2 = \mathrm{trace}(\mathbf{A}^T\mathbf{A})$ for any matrix $\mathbf{A}$, $\mathrm{trace}(\mathbf{AB}) = \mathrm{trace}(\mathbf{BA})$ for any matrices $\mathbf{A}, \mathbf{B}$ with matching dimensions and $\mathrm{trace}(\mathbf{A}+\mathbf{B}) = \mathrm{trace}(\mathbf{A}) + \mathrm{trace}(\mathbf{B})$, $\mathrm{trace}(\mathbf{A}) = \mathrm{trace}(\mathbf{A}^T)$ for any square matrices $\mathbf{A}$, $\mathbf{B}$, we can conclude:

$$\begin{aligned} \|\mathbf{RP} \pm \mathbf{P'}\|_F^2 &= \mathrm{trace}(\mathbf{P}^T\mathbf{R}^T\mathbf{RP}) + \mathrm{trace}(\mathbf{P'}^T\mathbf{P'}) \pm 2\,\mathrm{trace}(\mathbf{P'}^T\mathbf{RP}) \\ &= \|\mathbf{P}\|_F^2 + \|\mathbf{P'}\|_F^2 \pm 2\,\mathrm{trace}(\mathbf{RPP'}^T) \end{aligned}$$

Now the equivalence of the optimization problems in (4.4) is clear because in both cases $\mathrm{trace}(\mathbf{RPP'})$ has to be maximized. With $\mathbf{PP'}^T = \mathbf{U\Sigma V}^T$ the singular value decomposition of $\mathbf{PP'}^T$ and $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ its singular values sorted in descending order we get for that term:

$$\begin{aligned} \mathrm{trace}(\mathbf{RPP'}^T) &= \mathrm{trace}(\mathbf{RU\Sigma V}^T) = \mathrm{trace}(\mathbf{V}^T\mathbf{RU\Sigma}) \\ &= \mathrm{trace}(\mathbf{R'\Sigma}) = \sum_{i=1}^{d} r'_{ii}\sigma_i \leq \sum_{i=1}^{d} \sigma_i \end{aligned} \tag{4.5}$$

$$\text{with} \quad \mathbf{R'} := \mathbf{V}^T\mathbf{RU} =: \begin{pmatrix} r_{11} & \ldots & r_{1d} \\ \vdots & \ddots & \vdots \\ r_{d1} & \ldots & r_{dd} \end{pmatrix} \quad \text{an orthogonal matrix}$$

$$\text{and hence} \quad |r_{ij}| \leq 1 \quad \forall\, 1 \leq i, j \leq d$$

If $\det(\mathbf{UV}) = 1$, it is clear that the maximum for $\mathrm{trace}(\mathbf{RPP'})$ is reached for $r'_{ii} = 1$, $1 \leq i \leq d$, i.e. $\mathbf{R'} = \mathbb{1}$ and hence $\mathbf{R} = \mathbf{VU}^T$. In this case the inequation in (4.5) becomes an equation.
However, for $\det(\mathbf{UV}) = -1$ this is not possible because of $\det(\mathbf{R'}) = -1$. As orthogonal matrices are isometries, all their eigenvalues over $\mathbb{C}$ have an absolute value of 1. For a real orthogonal matrix like $\mathbf{R'}$ non-real eigenvalues can only appear together with their conjugate because the characteristic polynomial has real coefficients. So the trace, which is the sum of all eigenvalues, has the form

$$\lambda_1 + \bar{\lambda}_1 + \ldots + \lambda_k + \bar{\lambda}_k + 1 + \ldots + 1 + (-1) + \ldots + (-1)$$

$$= 2\,\mathrm{Re}(\lambda_1) + ... + 2\,\mathrm{Re}(\lambda_k) + 1 + \ldots + 1 + (-1) + \ldots + (-1)$$

with some complex eigenvalues $\lambda_1, \bar{\lambda}_1, \ldots, \lambda_k, \bar{\lambda}_k$. For a determinant of -1, which is the product of all eigenvalues, there must be at least one eigenvalue of -1 because the product of the complex eigenvalues with their conjugates is always +1. By that one can easily conclude that $d-2$ is an upper bound for the trace of a $d \times d$ orthogonal matrix with determinant -1.
Thereby we can consider the maximization of the term $\sum_{i=1}^{d} r'_{ii}\sigma_i$ in (4.5) as a linear optimization with the constraints $-1 \leq r'_{ii} \leq 1$ and $\sum_{i=1}^{d} r'_{ii} \leq d-2$. Because the boundary hyperplane defined by $\sum_{i=1}^{d} r'_{ii} \leq d-2$ intersects the hypercube $[-1,+1]^d$ defined by $-1 \leq r'_{ii} \leq 1$ in its corners $(-1,1,\ldots,1),\ldots,(1,\ldots,1,-1)$, the corners of the feasible region are just $\{-1,+1\}^d$ except $(1,\ldots,1)$. Because there must be corners in which the optimal value is reached we can simply compare the values achieved in the corners and it is easy to see that because of $\sigma_1 \geq \ldots \geq \sigma_d \geq 0$ the best value is achieved in any case at $(1,\ldots,1,-1)$. Thus the term $\sum_{i=1}^{d} r'_{ii}\sigma_i$ is maximized by $(r'_{11},\ldots,r'_{dd}) = (1,\ldots,1,-1)$, which can be obviously realized by $\mathbf{R}' = \mathrm{diag}(1,\ldots,1,-1)$ and thus $\mathbf{R} = \mathbf{V}\mathrm{diag}(1,\ldots,1,-1)\mathbf{U}^T$.

□

For a series of coordinates $\mathbf{p}_1^1,\ldots,\mathbf{p}_n^1$, ..., $\mathbf{p}_1^k,\ldots,\mathbf{p}_n^k$ of n markers for k time points one usually wants to translate these marker coordinates into poses of the tracked object consisting of translations $\in \mathbb{R}^3$ and orientations $\in \mathrm{SO}(3)$, which can be considered as rigid transformations that move the object from a reference pose in a reference coordinate system into its position in the reference coordinate system at any time point.
The most obvious procedure to get such a series of poses resp. rigid transformation is to take as translation the centroid of every set of n marker points, take as orientation for one reference point set, e.g. the first, $\mathbb{1} \in \mathrm{SO}(3)$ and calculate the orientations for the other point sets $\mathbf{p}_1^2,\ldots,\mathbf{p}_n^2$, ..., $\mathbf{p}_1^k,\ldots,\mathbf{p}_n^k$ according to the preceding lemma by the SVD of $\mathbf{P}_1\mathbf{P}_2^T$, ..., $\mathbf{P}_1\mathbf{P}_k^T$ with $\mathbf{P}_1,\ldots,\mathbf{P}_k \in \mathbb{R}^{d\times n}$ the matrices with the coordinates of the points of every set in their centroid system as columns.
However, this approach has one big disadvantage. If one takes a point set as reference which is strongly afflicted by noise or is even some kind of outlier, this will influence all calculated orientations negatively. Hence it is a natural idea to calculate the orientations relative to some kind of "average" marker coordinates in the centroid system. It is a natural idea as well to define these "average" coordinates, if considered as the columns of a matrix $\mathbf{P} \in \mathbb{R}^{d\times n}$, as a solution of the minimization problem

$$\underset{\mathbf{P}\in\mathbb{R}^{d\times n},(\mathbf{R}_1,\ldots,\mathbf{R}_k)\in\mathrm{SO}(d)^k}{\arg\min} \sum_{i=1}^{k} \|\mathbf{R}_i\mathbf{P} - \mathbf{P}_i\|_F^2 \ . \tag{4.6}$$

Then the matrices $\mathbf{R}_1,\ldots,\mathbf{R}_k$ are the orientations of the k poses. It is clear that the solution of (4.6) is not unique because if $\tilde{\mathbf{P}}$, $\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_k$ is a solution, $\mathbf{R}^T\tilde{\mathbf{P}}$,

$\tilde{\mathbf{R}}_1\mathbf{R}, \ldots, \tilde{\mathbf{R}}_k\mathbf{R}$ is a solution for any $\mathbf{R} \in \mathrm{SO}(d)$ as well. But that is only a question of how the reference coordinate system is orientated.
The minimization problem (4.6) can be simplified, which we want to show in the following lemma.

Lemma 4.2 *Given k matrices $\mathbf{P}_1, \ldots, \mathbf{P}_k \in \mathbb{R}^{d\times n}$. The minimization problem*

$$\underset{\mathbf{P}\in\mathbb{R}^{d\times n},(\mathbf{R}_1,\ldots,\mathbf{R}_k)\in\mathrm{SO}(d)^k}{\arg\min} \sum_{i=1}^{k} \|\mathbf{R}_i\mathbf{P} - \mathbf{P}_i\|_F^2 \tag{4.7}$$

is solved by the solution of the maximization problem

$$\underset{(\mathbf{R}_1,\ldots,\mathbf{R}_k)\in\mathrm{SO}(d)^k}{\arg\max} \left\| \sum_{i=1}^{k} \mathbf{R}_i^T\mathbf{P}_i \right\|_F \quad \textit{and} \quad \mathbf{P} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{R}_i^T\mathbf{P}_i \; .$$

Proof:
We start in quite a similar way as in the proof of lemma 4.1:

$$\begin{aligned} \sum_{i=1}^{k} \|\mathbf{R}_i\mathbf{P} - \mathbf{P}_i\|_F^2 &= \sum_{i=1}^{k} \left[\|\mathbf{P}_i\|_F^2 + \|\mathbf{P}\|_F^2 - 2\,\mathrm{trace}(\mathbf{P}^T\mathbf{R}_i^T\mathbf{P}_i)\right] \\ &= \sum_{i=1}^{k} \|\mathbf{P}_i\|_F^2 + k\left[\|\mathbf{P}\|_F^2 - 2\,\mathrm{trace}\left(\mathbf{P}^T\frac{1}{k}\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right)\right] \end{aligned} \tag{4.8}$$

We now would like to apply the so-called von Neumann's trace inequality

$$|\mathrm{trace}(\mathbf{AB})| \le \sum_{i=1}^{m} \alpha_i\beta_i \tag{4.9}$$

for $m\times m$ matrices $\mathbf{A}, \mathbf{B}$ with singular values $\alpha_1 \ge \ldots \ge \alpha_m$ and $\beta_1 \ge \ldots \ge \beta_m$ with

$$\mathbf{A} = \mathbf{P}^T \quad \text{and} \quad \mathbf{B} = \mathbf{P}' := \frac{1}{k}\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i$$

in our case. It was first published by John von Neumann in 1937 [49] and a compact proof can be found in [50].
However, the matrices $\mathbf{P}$ and $\mathbf{P}'$ in our case are not necessarily square matrices. But von Neumann's trace inequality can be easily generalized to non-square matrices with matching dimensions by simply filling up the matrices by zeros to square matrices. One easily verifies that this does not change the trace of the product and thus the left hand side of (4.9) is not affected. Furthermore a SVD of a non-square matrix can be extended to a SVD of the square matrix filled up by zeros by also filling up the diagonal matrix containing the singular values with zeros and extending

the smaller orthogonal matrix with a properly sized $\mathrm{diag}(1,\ldots,1)$ block. Hence filling up a non-square matrix with zeros simply yields additional 0 singular values and thus the right hand side in (4.9) is not affected either.
If $\sigma_1 \geq \ldots \geq \sigma_m$ are the singular values of $\mathbf{P}$ (which are also the singular values of $\mathbf{P}^T$) and $\sigma'_1 \geq \ldots \sigma'_m$ are the singular values of $\mathbf{P'}$ with $m := \min\{d, n\}$, von Neumann's trace inequality yields:

$$\mathrm{trace}\left(\mathbf{P}^T \frac{1}{k}\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right) \leq \left|\mathrm{trace}\left(\mathbf{P}^T \frac{1}{k}\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right)\right| \leq \sum_{i=1}^{m}\sigma_i\sigma'_i \qquad (4.10)$$

Because the Frobenius norm is unitary and orthogonally invariant, it is clear that

$$\|\mathbf{P}\|_F^2 = \sum_{i=1}^{m}\sigma_i^2 \,.$$

Thus we get by (4.10) as lower bound for the term in square brackets in (4.8):

$$\|\mathbf{P}\|_F^2 - 2\,\mathrm{trace}\left(\mathbf{P}^T \frac{1}{k}\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right) \geq \sum_{i=1}^{m}\sigma_i^2 - 2\sum_{i=1}^{m}\sigma_i\sigma'_i \qquad (4.11)$$

If we consider the right hand side of (4.11) as a function f of the σ_i with fixed σ'_i, it is clear because of

$$\frac{\partial f}{\partial \sigma_i} = 2(\sigma_i - \sigma'_i) \quad \text{and} \quad \frac{\partial^2 f}{\partial\sigma_i\partial\sigma_j} = 2\,\delta_{ij}$$

that this function has a global minimum at $\sigma_i = \sigma'_i$ of

$$-\sum_{i=1}^{k}\sigma_i'^2 = -\|\mathbf{P'}\|_F^2 = -\frac{1}{k^2}\left\|\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right\|_F^2 \,.$$

Obviously this is the case if we choose $\mathbf{P} = \mathbf{P'}$ and the inequation (4.10) becomes an equation. Hence this must be a global minimum for $\sum_{i=1}^{k}\|\mathbf{R}_i\mathbf{P} - \mathbf{P}_i\|_F^2$ in dependence of $\mathbf{P}$ for fixed $\mathbf{R}_1,\ldots,\mathbf{R}_i$ and $\mathbf{P'}$. In this global minimum it is

$$\sum_{i=1}^{k}\|\mathbf{R}_i\mathbf{P} - \mathbf{P}_i\|_F^2 = \sum_{i=1}^{k}\|\mathbf{P}_i\|_F^2 - \frac{1}{k}\left\|\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right\|_F^2 \,.$$

This is minimal if

$$\left\|\sum_{i=1}^{k}\mathbf{R}_i^T\mathbf{P}_i\right\|_F$$

is maximal.

□

But how should we calculate $\mathbf{R}_1, \ldots, \mathbf{R}_k$ so that

$$F := \left\| \sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i \right\|_F$$

becomes maximal? If we have reasonable starting values for $\mathbf{R}_1, \ldots, \mathbf{R}_k$, lemma 4.1 gives us an instruction how we can iteratively "improve" the $\mathbf{R}_i$ by increasing F. According to lemma 4.1 F increases if we set

$$\mathbf{R}_i^T \to \mathbf{R}_i^{imp} \mathbf{R}_i^T \quad \text{with} \quad \mathbf{R}_i^{imp} = \mathbf{V} \operatorname{diag}(1, \ldots, 1, \det(\mathbf{UV}))\, \mathbf{U}^T$$

$$\text{with } \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \text{ the SVD of } \quad \mathbf{R}_i^T \mathbf{P}_i \left(\sum_{\substack{j=1 \\ j \neq i}}^{k} \mathbf{R}_j^T \mathbf{P}_j \right)^T$$

We can do that successively for all $\mathbf{R}_i$ and iterate it several times. Because F increases in doing so, but is bound by

$$F = \left\| \sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i \right\|_F \leq \sum_{i=1}^{k} \left\| \mathbf{R}_i^T \mathbf{P}_i \right\|_F = \sum_{i=1}^{k} \left\| \mathbf{P}_i \right\|_F \ ,$$

this process must converge.
Unfortunately, it could not be shown that this convergence leads to a global maximum. But by choosing proper initial values for the $\mathbf{R}_i$ it should be clear that this convergence will lead in any case to reasonable values for the $\mathbf{R}_i$. In the case $n = d$ the $\mathbf{P}_i$ are square matrices and we can take

$$\mathbf{R}_i = \mathbf{U}_i \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{U}_i \mathbf{V}_i) \end{pmatrix} \mathbf{V}_i^T \text{ with } \mathbf{U}_i \Sigma_i \mathbf{V}_i^T \text{ the SVD of } \mathbf{P}_i. \quad (4.12)$$

This is a good choice because of two reasons.
At first if we have no noise and the columns of the $\mathbf{P}_i$ have fixed relative positions to each other, $\mathbf{R}_{ji} \in \mathrm{SO}(3)$ with $\mathbf{P}_j = \mathbf{R}_{ji} \mathbf{P}_i$ exist. But then, if $\mathbf{U}_i \boldsymbol{\Sigma}_i \mathbf{V}_i^T$ is the SVD of $\mathbf{P}_i$, $\mathbf{R}_{ji} \mathbf{U}_i \boldsymbol{\Sigma}_j \mathbf{V}_j^T$ is obviously a SVD of $\mathbf{P}_j$. Because of $\det(\mathbf{R}_{ji}) = 1$ and hence $\det(\mathbf{R}_{ji} \mathbf{U}_i \mathbf{V}_i) = \det(\mathbf{U}_i \mathbf{V}_i)$ according to (4.12) we get $\mathbf{R}_j = \mathbf{R}_{ji} \mathbf{R}_i$, which is obviously an ideal choice. The only thing we still have to prove is that for a $\mathbf{R}_i$ delivered by (4.12) the product $\mathbf{R}_i^T \mathbf{P}_i$ is the same for different SVDs of $\mathbf{P}_i$. The orthogonal matrices of a SVD are not unique concerning a factor of (-1) as well as orthogonal transformations in subspaces corresponding to a multiple singular value. We will show in lemma 4.3 that the products $\mathbf{R}_i^T \mathbf{P}_i$ are uniquely defined despite this ambiguity.

The second reason why (4.12) is a good choice is that then the summands of $\sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i$, whose Frobenius we have to maximize, are

$$\mathbf{V}_i \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{U}_i \mathbf{V}_i) \end{pmatrix} \mathbf{\Sigma}_i \mathbf{V}_i^T = \mathbf{V}_i \begin{pmatrix} \sigma_1^i & & & \\ & \ddots & & \\ & & \sigma_{d-1}^i & \\ & & & \sigma_d^i \det(\mathbf{U}_i \mathbf{V}_i) \end{pmatrix} \mathbf{V}_i^T$$

with $\sigma_1^i \geq \ldots \geq \sigma_k^i$ the singular values of $\mathbf{P}_i$. Because the columns of the $\mathbf{P}_i$ are the marker coordinates in their centroid system, it is $\mathbf{P}_i(1, \ldots, 1)^T = 0$ and thus the smallest singular values σ_d^i must be 0. Thereby the summands of $\sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i$ are positive semidefinite symmetric matrices, and hence that is also true for their sum. For positive semidefinite matrices the trace norm $\|.\|_{tr}$, which is defined as the sum of the singular values, can obviously be calculated as the trace actually, and thus we get:

$$\begin{aligned} \left\| \sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i \right\|_{tr} &= \operatorname{trace} \left(\sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i \right) = \sum_{i=1}^{k} \operatorname{trace} \left(\mathbf{R}_i^T \mathbf{P}_i \right) \\ &= \sum_{i=1}^{k} \operatorname{trace} \left(\mathbf{V}_i \begin{pmatrix} \sigma_1^i & & & \\ & \ddots & & \\ & & \sigma_{d-1}^i & \\ & & & 0 \end{pmatrix} \mathbf{V}_i^T \right) \\ &= \sum_{i=1}^{k} \sum_{j=1}^{d-1} \sigma_j^i = \sum_{i=1}^{k} \|\mathbf{P}_i\|_{tr} = \sum_{i=1}^{k} \|\mathbf{R}_i^T \mathbf{P}_i\|_{tr} \end{aligned}$$

Because of the triangle inequation for the trace norm that means that with the $\mathbf{R}_i$ according to (4.12) the trace norm of $\sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i$ is maximized. However, actually we want to maximize the sum's Frobenius norm, but it can be easily proven that the trace norm is strongly related to the Frobenius norm by the inequation

$$\|\mathbf{A}\|_F \leq \|\mathbf{A}\|_{tr} \leq \sqrt{\operatorname{rank}(A)}\, \|\mathbf{A}\|_F$$

for any matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. Because in our case it is only $\operatorname{rank}(A) = 2$, it should be a good idea to start searching for a maximum Frobenius norm at a point where the trace norm is maximum.

Lemma 4.3 *Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be an arbitrary square matrix and $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ a SVD of $\mathbf{A}$, i.e. $\mathbf{U}, \mathbf{V}$ orthogonal and $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_d)$ with $\sigma_1 \geq \ldots \geq \sigma_d \geq 0$ the singular values of $\mathbf{A}$. Then the matrix*

$$\mathbf{V} \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{U}_i \mathbf{V}_i) \end{pmatrix} \mathbf{U}^T \mathbf{A}$$

is independent of the choice of the SVD.

Proof:
Let $\mathbf{A} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T = \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T$ be two SVDs of $\mathbf{A}$. Because the singular values $\sigma_1 \geq \ldots \sigma_d \geq 0$ of $\mathbf{A}$ are uniquely determined and sorted on the diagonal of the central matrix in the SVD, it is $\mathbf{\Sigma}_1 = \mathbf{\Sigma}_2 = \mathrm{diag}(\sigma_1, \ldots, \sigma_d) =: \mathbf{\Sigma}$. If we consider $\mathbf{U}_1 = (\mathbf{u}_1^1 \ldots \mathbf{u}_d^1)^T$, $\mathbf{U}_2 = (\mathbf{u}_1^2 \ldots \mathbf{u}_d^2)^T$, $\mathbf{V}_1 = (\mathbf{v}_1^1 \ldots \mathbf{v}_d^1)^T$ and $\mathbf{V}_2 = (\mathbf{v}_1^2 \ldots \mathbf{v}_d^2)^T$ by their rows $\in \mathbb{R}^d$, we can write

$$\mathbf{U}_1\mathbf{\Sigma}\mathbf{V}_1^T = \mathbf{U}_2\mathbf{\Sigma}\mathbf{V}_2^T \tag{4.13}$$

componentwise as

$$\sigma_i\delta_{ij}\,{\mathbf{u}_i^1}^T\mathbf{v}_j^1 = \sigma_i\delta_{ij}\,{\mathbf{u}_i^2}^T\mathbf{v}_j^2\,. \tag{4.14}$$

Let be $\sigma_1 \geq \ldots \geq \sigma_t > 0$ and $\sigma_{t+1} = \ldots = \sigma_d = 0$ for a proper $t \in \{1, \ldots, d\}$. For $i, j \leq t$ we can divide equation (4.14) by σ_i and get

$$\delta_{ij}\,{\mathbf{u}_i^1}^T\mathbf{v}_j^1 = \delta_{ij}\,{\mathbf{u}_i^2}^T\mathbf{v}_j^2 \quad \text{for } i, j \leq t. \tag{4.15}$$

If $t = d$, it is $\det(\mathbf{\Sigma}) \neq 0$ and by applying $\det$ to (4.13) and dividing by $\det(\mathbf{\Sigma})$ we get $\det(\mathbf{U}_1\mathbf{V}_1) = \det(\mathbf{U}_2\mathbf{V}_2)$. Thus (4.15) yields

$$\det(\mathbf{U}_1\mathbf{V}_1)\,{\mathbf{u}_d^1}^T\mathbf{v}_d^1 = \det(\mathbf{U}_2\mathbf{V}_2)\,{\mathbf{u}_d^2}^T\mathbf{v}_d^2\,. \tag{4.16}$$

(4.15) and (4.16) deliver componentwise the matrix equation $\mathbf{U}_1\mathbf{D}\mathbf{V}_1^T = \mathbf{U}_2\mathbf{D}\mathbf{V}_2^T$ and transposed

$$\mathbf{V}_1\mathbf{D}\mathbf{U}_1^T = \mathbf{V}_2\mathbf{D}\mathbf{U}_2^T \tag{4.17}$$

with the diagonal matrix $\mathbf{D}$ defined as:

$$\mathbf{D} := \begin{cases} \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & \underbrace{1}_{t \text{ columns}} & & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix} & \text{for } t < d \\ \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{U}_1\mathbf{V}_1) \end{pmatrix} & \text{for } t = d \end{cases}$$

Obviously it is $\mathbf{D}\mathbf{\Sigma} = \mathbf{D}'\mathbf{\Sigma}$ with

$$\mathbf{D}' := \begin{pmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(\mathbf{U}_1\mathbf{V}_1) \end{pmatrix}$$

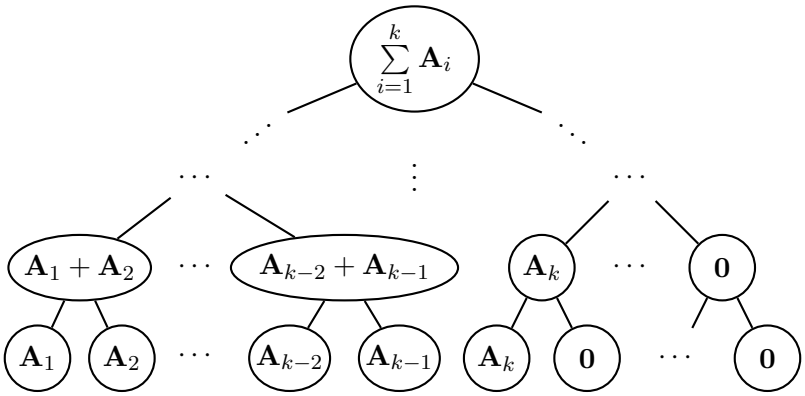


Figure 4.1: Partial sums over the $\mathbf{A}_i$ in algorithm 4.1 organized in a heap. The $\mathbf{A}_i$ in the leaves are filled up by $\mathbf{0}$s up to the next power of 2. A node is the sum of its both children.

for $t < d$ as well as $t = d$. Thus we get by multiplying (4.17) from right by $\mathbf{A}$ as $\mathbf{U}_1\mathbf{\Sigma}\mathbf{V}_1^T$ and $\mathbf{U}_2\mathbf{\Sigma}\mathbf{V}_2^T$:

$$\begin{aligned}
& \mathbf{V}_1\mathbf{D}\mathbf{U}_1^T\mathbf{U}_1\mathbf{\Sigma}\mathbf{V}_1^T &=& \mathbf{V}_2\mathbf{D}\mathbf{U}_2^T\mathbf{U}_2\mathbf{\Sigma}\mathbf{V}_2^T \\
\Rightarrow \quad & \mathbf{V}_1\mathbf{D}\mathbf{\Sigma}\mathbf{V}_1^T &=& \mathbf{V}_2\mathbf{D}\mathbf{\Sigma}\mathbf{V}_2^T \\
\Rightarrow \quad & \mathbf{V}_1\mathbf{D}'\mathbf{\Sigma}\mathbf{V}_1^T &=& \mathbf{V}_2\mathbf{D}'\mathbf{\Sigma}\mathbf{V}_2^T \\
\Rightarrow \quad & \mathbf{V}_1\mathbf{D}'\mathbf{U}_1^T\mathbf{U}_1\mathbf{\Sigma}\mathbf{V}_1^T &=& \mathbf{V}_2\mathbf{D}'\mathbf{U}_2^T\mathbf{U}_2\mathbf{\Sigma}\mathbf{V}_2^T \\
\Rightarrow \quad & \mathbf{V}_1\mathbf{D}'\mathbf{U}_1^T\mathbf{A} &=& \mathbf{V}_2\mathbf{D}'\mathbf{U}_2^T\mathbf{A}
\end{aligned}$$

□

Now we know how to choose the initial values for the $\mathbf{R}_i$ if $n = d$, but what about $n \neq d$. It is clear that $n < d$ does not make sense. For example an object in 3-dimensional space cannot be tracked by only 2 markers because then rotations around the axis defined by those two marker points could not be recognized. But if $n > d$ we can consider only the d first marker points, i.e. take only a square submatrix of $\mathbf{P}_i$, shift their columns into their centroid system and calculate the initial $\mathbf{R}_i$s according to the $d = n$ case.

The presented algorithm for retrieving k poses from k sets of n marker points in the 3-dimensional case is shown in pseudo code in algorithm 4.1. As abort criterion at line 13 it is used that the rotation angle of the k last correction rotations $\mathbf{R}_{cor}$ must be below a certain small threshold. This is equivalent to demanding that its cosine, which can be calculated easily by the trace, is above a certain threshold slightly below 1.

When analyzing the running time of algorithm 4.1, it is clear that the running time bottleneck is at line 9 if we assume that n can be considered as constant because it is usually $k \gg n$. The running time for calculating this sum is in $O(k)$ while being

Algorithm 4.1 Retrieving k poses $(\mathbf{R}_i, \mathbf{b}_i)$, $1 \leq i \leq k$, and "average marker coordinates" from k sets $(\mathbf{p}_1^i, \ldots, \mathbf{p}_n^i)$, $1 \leq i \leq k$, of n marker coordinates

[t]

1: **for** $i \in \{1, \ldots, k\}$ **do**
2: set as translation $\mathbf{b}_i$ of the i-th pose the centroid of $\mathbf{p}_1^i, \ldots, \mathbf{p}_n^i$
3: define a $3 \times n$ matrix by $\mathbf{P}_i \leftarrow (\mathbf{p}_1^i - \mathbf{b}_k \ \ldots \ \mathbf{p}_n^i - \mathbf{b}_n)$
4: calculate the centroid $\mathbf{c}_i$ of the first three marker points $\mathbf{p}_1^i, \mathbf{p}_2^i, \mathbf{p}_3^i$ and the SVD $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ of the 3×3 matrix $(\mathbf{p}_1^i - \mathbf{c}_i \ \mathbf{p}_2^i - \mathbf{c}_i \ \mathbf{p}_3^i - \mathbf{c}_i)$ (if $n = 3$, this matrix has already been calculated as $\mathbf{P}_i$) and set

$$\mathbf{R}_i \leftarrow \mathbf{U} \begin{pmatrix} 1 & & \\ & 1 & \\ & & \det(\mathbf{U}\mathbf{V}) \end{pmatrix} \mathbf{V}^T$$

as initial orientation of the i-th pose
5: $\mathbf{A}_i \leftarrow \mathbf{R}_i^T \mathbf{P}_i$
6: **end for**
7: **while true do**
8: **for** $i \in \{1, \ldots, k\}$ **do**
9: $\mathbf{B} \leftarrow \sum_{j=1, j \neq i}^{k} \mathbf{A}_j$
10: calculate the SVD $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ of the 3×3 matrix $\mathbf{A}_i \mathbf{B}^T$ and set

$$\mathbf{R}_{cor} \leftarrow \mathbf{V} \begin{pmatrix} 1 & & \\ & 1 & \\ & & \det(\mathbf{U}\mathbf{V}) \end{pmatrix} \mathbf{U}^T$$

11: $\mathbf{A}_i \leftarrow \mathbf{R}_{cor} \mathbf{A}_i$
12: $\mathbf{R}_i \leftarrow \mathbf{R}_i \mathbf{R}_{cor}^T$
13: **if** some abort criterion is fulfilled **then**
14: **break** loop at line 7
15: **end if**
16: **end for**
17: **end while**
18: calculate the "average" marker coordinates in their centroid system as the columns of $\frac{1}{k} \sum_{i=1}^{k} \mathbf{A}_i$

Algorithm 4.2 Sum calculation by a heap

1: $sum \leftarrow \mathbf{0}$
2: Node $n \leftarrow$ leaf node containing $\mathbf{A}_i$
3: **while** n has a parent **do**
4: $\quad sum \leftarrow sum +$ entry of n's sibling
5: $\quad n \leftarrow$ parent of n
6: **end while**

executed k times in one round of the loop at line 7. However, the asymptotic running time for calculating this sum can be reduced to $O(\log k)$ by not only storing the $\mathbf{A}_i$, but a whole heap of partial sums over the $\mathbf{A}_i$ according to figure 4.1. With such a heap the sum

$$\sum_{\substack{j=1 \\ j \neq i}}^{k} \mathbf{A}_j$$

can be calculated in obviously logarithmic running time as shown in algorithm 4.2. The running time for building up the heap, what is done at line 5 in algorithm 4.1, is obviously in $O(k)$ with a memory consumption in $O(k)$ as well. Instead of updating only $\mathbf{A}_i$ at line 11 the whole branch from $\mathbf{A}_i$ up to the root has to be updated consuming a running time in $O(\log k)$. Thus an iteration of the loop at line 8 is in $O(\log k)$ and hence an iteration of the main loop at line 7 is in $O(k \log k)$ while being in $O(k^2)$ without using a heap.
The method for calculating average marker coordinates presented in this section was tested for a dataset of 3823 positions of three markers tracked by an ultrasound system. For the necessary singular value decompositions the Eigen library was used [51]. The running time of algorithm 4.1 for this dataset is about 150 ms on a single core of a Intel® Core™ i7-720QM CPU at 1.60 GHz. Thus for such a small dataset the running time is negligible.
Figure 4.2 illustrates some statistics for algorithm 4.1 run for this test dataset. Figure 4.2(a) shows the angles of the correction rotations calculated in the inner loop at line 4. The rounds on the x-axis correspond to the inner loop at line 8. The range above 0.2° is scaled by a factor of $1/20$ to get the huger values – up to 3.3° at maximum – into the plot. As can be seen, just after the first round the correction angles become extremely small and after the second round they cannot be distinguished any longer from zero at the plot's angle resolution.
Figure 4.2(b) shows some kind of error value according to the formula

$$\sqrt{\frac{1}{3k} \left\| \sum_{i=1}^{k} \mathbf{R}_i \mathbf{P}_i - \mathbf{P} \right\|_F^2} \quad \text{with} \quad \mathbf{P} = \frac{1}{k} \sum_{i=1}^{k} \mathbf{R}_i^T \mathbf{P}_i \tag{4.18}$$

which can be considered as the mean marker deviation from the average marker coordinates when the algorithm is stopped at this point. As can be seen, this deviation drops when there are bigger correction angles in the plot above. Because the

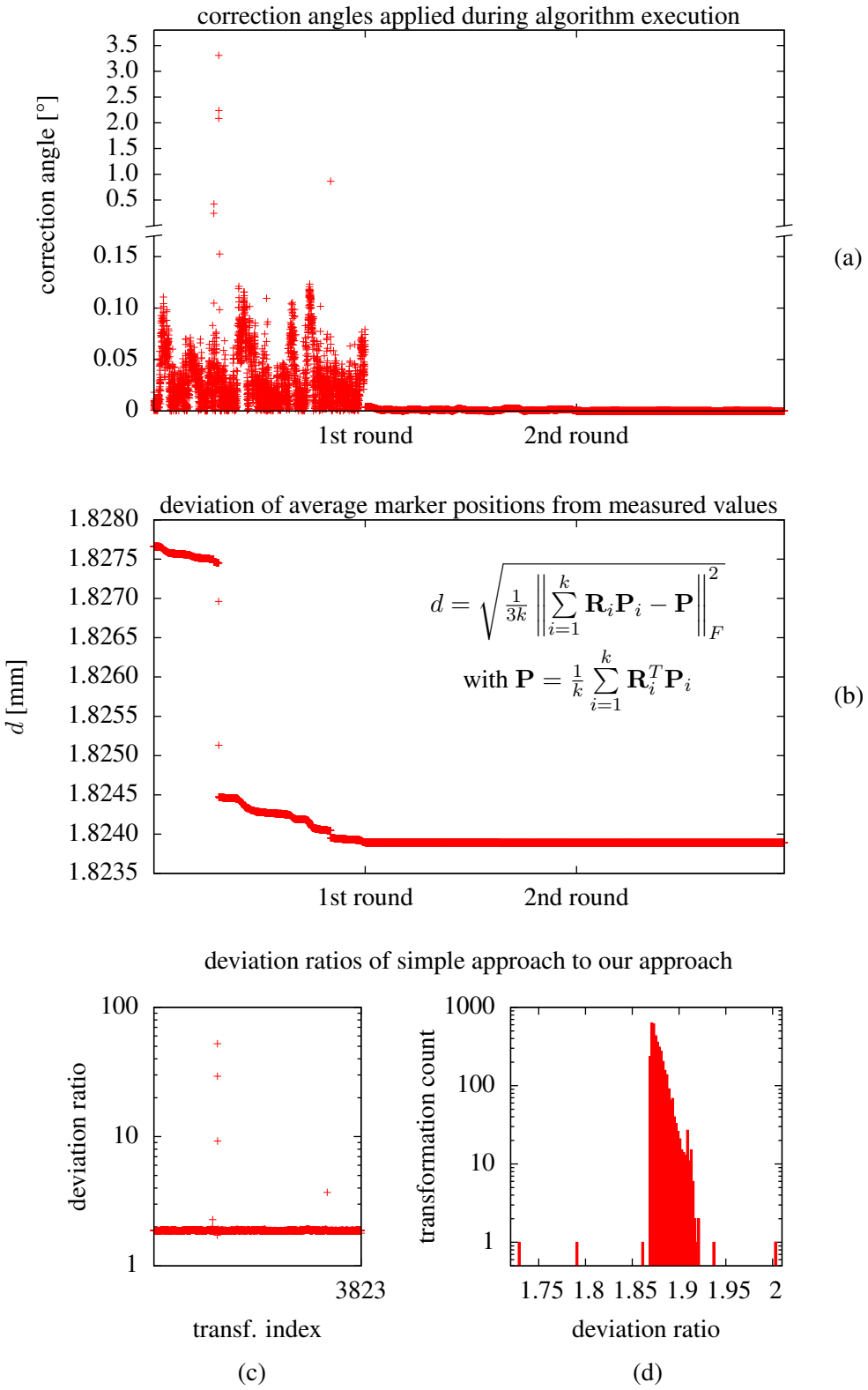


Figure 4.2: Some statistics of algorithm 4.1 run for a test dataset of 3823 noise afflicted positions of 3 markers tracked by an ultrasound system. For explanation see the text.

values for initial rotations seems to be very good for the test dataset this deviation is already quite low at the beginning of the algorithm and decreases only slightly during the execution of the algorithm.
As mentioned at the beginning of this section, a very simple approach to retrieve rotations from marker positions is to take one of the sets of marker coordinates in their centroid system as reference and calculate the orientation of the other marker coordinates in their centroid system by lemma 4.1. Figures 4.2(c) and (d) show the ratio of the deviation according to (4.18) for this simple approach compared to the average marker coordinates approach when using algorithm 4.1.
Figure 4.2(c) shows these rations for taking each of the 3823 coordinate sets as reference with its index on the x-axis. The most ratios are slightly below 2 except five outliers with a maximum ratio of 52.4 which again correspond to the drops in figure 4.2(b) above. The distribution of the ratios except these five outliers, which are out of the plot's range, is shown in the histogram in figure 4.2(d).
That the average marker coordinates approach yields an error that is at least about 1.7 and at most about 52 times smaller than when calculating the poses by that simple approach relative to one of the 3823 sets of marker coordinates shows that it is worth its additional computational effort for the sake of robustness.

4.2 Calculating the Position of Two Tracking Systems to Each Other

The poses of a tracked object, that a tracking system delivers and that might be calculated from marker coordinates as described in the previous section, can be considered as rigid transformations which transform coordinates from the markers' coordinate system to the sensor's coordinate system. When the poses are calculated as in the previous section, the origin of the markers' coordinate system is their centroid. Thus when two independent tracking systems are used, we have two pairs of coordinate systems and get pairs A_i, B_i of rigid transformations by the measurement of the tracking system when the tracked object moves. We make the assumption that the sensors as well as the markers, which are usually fixed to the tracked object, do not change their relative position to each other during a measurement. Then there are two rigid transformation S and T which transform coordinates between the sensors' and markers' coordinate systems as illustrated in figure 4.3 so that it is

$$SA_i = B_iT \; . \tag{4.19}$$

In order to compare the poses of the two tracking systems or to combine them for a lower error, we must know S and T. Because there is usually noise on the values measured for the A_i and B_i, equation 4.19 will not hold exactly and the question is how we can calculate S and T from a series A_i, B_i, $1 \leq i \leq n$, of n measurements in "a least mean square sense".
This problem already occurred in robotics. An overview over the existing pub-

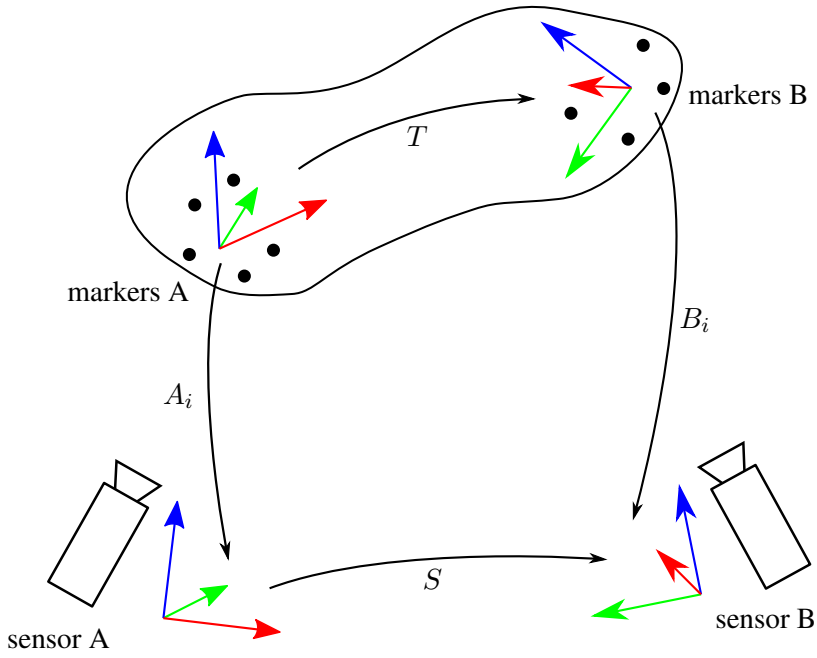


Figure 4.3: When using two independent tracking systems delivering poses that can be interpreted as rigid transformations A_i and B_i this transformations are related to each other by two other rigid transformations S and T.

lications can be found in [52]. We want to show a closed form solution using quaternions which is mathematically equivalent to the solution shown in [53, 54], but can be calculated with less computational effort and additionally gives a geometric view of the problem. Furthermore by this geometric view it becomes clear that there is a critical ambiguity concerning rotation quaternions which can be overcome by a RANSAC-like approach. This ambiguity seems to have been neglected in existing publications [53, 54, 55] so far.
At first we want to motivate the way how to solve this problem. For its solution it is natural to minimize an expression of the form

$$\sum_{i=1}^{n} d(SA_i, B_iT)^2 \tag{4.20}$$

with d a proper metric on the group of rigid transformation $\mathrm{SE}(3)$ (special Euclidean group). In [56] it is shown that there is unfortunately no metric on $\mathrm{SE}(3)$ which is bi-invariant. Bi-invariant means that it is left-invariant, i.e. $d(A, B) = d(TA, TB)$ for any $T \in \mathrm{SE}(3)$, as well as right-invariant, i.e. $d(A, B) = d(AT, BT)$ for any $T \in \mathrm{SE}(3)$. However, in [56] the following parametrized left-

invariant metric is suggested:

$$d_l(A,B) = l\ (\delta(\mathbf{R}_A, \mathbf{R}_B))^2 + |\mathbf{b}_A - \mathbf{b}_B|^2$$

with l a positive length scale factor, $\mathbf{R}_A, \mathbf{R}_B \in \mathrm{SO}(3)$ and $\mathbf{b}_A, \mathbf{b}_B \in \mathbb{R}^3$ so that is

$$A : \mathbb{R}^3 \to \mathbb{R}^3\ ,\ \mathbf{x} \mapsto \mathbf{R}_A\mathbf{x} + \mathbf{b}_A \quad \text{and} \quad B : \mathbb{R}^3 \to \mathbb{R}^3\ ,\ \mathbf{x} \mapsto \mathbf{R}_B\mathbf{x} + \mathbf{b}_B\ .$$

By $\delta(\mathbf{R}_A, \mathbf{R}_B)$ we want denote the angle "between" the rotations $\mathbf{A}$ and $\mathbf{B}$ which is defined as the angle of the rotation $\mathbf{A}^T\mathbf{B}$ and can be calculated by the trace of this matrix (see section 3.1).
Because it is

$$\begin{aligned} SA_i\mathbf{x} &= \mathbf{R}_S\left(\mathbf{R}_{A_i}\mathbf{x} + \mathbf{b}_{A_i}\right) + \mathbf{b}_S = \mathbf{R}_S\mathbf{R}_{A_i}\mathbf{x} + \mathbf{R}_S\mathbf{b}_{A_i} + \mathbf{b}_S \\ B_iT\mathbf{x} &= \mathbf{R}_{B_i}\left(\mathbf{R}_T\mathbf{x} + \mathbf{b}_T\right) + \mathbf{b}_{B_i} = \mathbf{R}_{B_i}\mathbf{R}_T\mathbf{x} + \mathbf{R}_{B_i}\mathbf{b}_T + \mathbf{b}_{B_i} \end{aligned} \quad (4.21)$$

for any $\mathbf{x} \in \mathbb{R}^3$, that means we have to minimize

$$l\sum_{i=1}^{n}\left(\delta\left(\mathbf{R}_s\mathbf{R}_{A_i}, \mathbf{R}_{B_i}\mathbf{R}_T\right)\right)^2 + \sum_{i=1}^{n}|\mathbf{R}_S\mathbf{b}_{A_i} + \mathbf{b}_S - \mathbf{R}_{B_i}\mathbf{b}_T - \mathbf{b}_{B_i}|^2$$

Now we want to make a simplification. We minimize the first sum representing the rotational problem separately and, when we have found optimal $\mathbf{R}_S$ and $\mathbf{R}_T$, we minimize the second sum. Because the second sum contains $\mathbf{R}_S$, which will be replaced by the optimal value $\mathbf{R}_{S,opt}$ from optimizing the first sum, this is not completely correct, but it should be a good approximation. Besides, the scale length factor l does obviously not matter when minimizing the first sum separately.
When we have found $\mathbf{R}_{S,opt}$, minimizing the second sum is easy because it can be written as

$$\left|\begin{pmatrix} \mathbb{1} & -\mathbf{R}_{B_1} \\ \mathbb{1} & -\mathbf{R}_{B_2} \\ \vdots & \vdots \\ \mathbb{1} & -\mathbf{R}_{B_n} \end{pmatrix}\begin{pmatrix}\mathbf{b}_S \\ \mathbf{b}_T\end{pmatrix} - \begin{pmatrix} \mathbf{b}_{B_1} - \mathbf{R}_{S,opt}\mathbf{b}_{A_1} \\ \mathbf{b}_{B_2} - \mathbf{R}_{S,opt}\mathbf{b}_{A_2} \\ \vdots \\ \mathbf{b}_{B_n} - \mathbf{R}_{S,opt}\mathbf{b}_{A_n} \end{pmatrix}\right|^2 .$$

Minimizing such an expression for $\mathbf{b}_S, \mathbf{b}_T \in \mathbb{R}^3$ resp. $(\mathbf{b}_S, \mathbf{b}_T)^T \in \mathbb{R}^6$ is a well-known problem from numerics and is solved by a QR decomposition of the matrix containing the $\mathbf{R}_{B_i}$ [30].
Thus we still have to solve the rotational problem, that means finding $\mathbf{R}_S$ and $\mathbf{R}_T$ so that it is

$$\mathbf{R}_S\mathbf{R}_{A_i} \approx \mathbf{R}_{B_i}\mathbf{R}_T \quad \Leftrightarrow \quad \mathbf{R}_{B_i} \approx \mathbf{R}_S\mathbf{R}_{A_i}\mathbf{R}_T^T$$

for $1 \le i \le n$. When translating the latter expression to unit quaternions instead of rotation matrices, we get an expression of the form

$$\mathfrak{b}_i \approx \mathfrak{s}\mathfrak{a}_i\mathfrak{t}^*\ .$$

The expression $\mathfrak{s}\mathfrak{a}_i\mathfrak{t}^*$ is known from representation theory because it describes how $\mathfrak{Q}^* \times \mathfrak{Q}^*$ with $\mathfrak{Q}^*$ denoting the group of unit quaternions, which is isomorph to $\mathrm{SU}(2)$, operates on $\mathrm{SO}(4)$. To understand that we calculate this expression componentwise and factor out the components of $\mathfrak{a}_i$ in the result:

$$\mathfrak{s}\mathfrak{a}_i\mathfrak{t}^* = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} \begin{pmatrix} a_0^i \\ a_1^i \\ a_2^i \\ a_3^i \end{pmatrix} \begin{pmatrix} t_0 \\ -t_1 \\ -t_2 \\ -t_3 \end{pmatrix} = \begin{pmatrix} s_0a_0^i - s_1a_1^i - s_2a_2^i - s_3a_3^i \\ s_1a_0^i + s_0a_1^i + s_2a_3^i - s_3a_2^i \\ s_2a_0^i + s_0a_2^i + s_3a_1^i - s_1a_3^i \\ s_3a_0^i + s_0a_3^i + s_1a_2^i - s_2a_1^i \end{pmatrix} \begin{pmatrix} t_0 \\ -t_1 \\ -t_2 \\ -t_3 \end{pmatrix}$$

$$= \begin{pmatrix} a_0^i(+s_0t_0 + s_1t_1 + s_2t_2 + s_3t_3) + a_1^i(+s_0t_1 - s_1t_0 - s_2t_3 + s_3t_2) \\ a_0^i(-s_0t_1 + s_1t_0 - s_2t_3 + s_3t_2) + a_1^i(+s_0t_0 + s_1t_1 - s_2t_2 - s_3t_3) \\ a_0^i(-s_0t_2 + s_1t_3 + s_2t_0 - s_3t_1) + a_1^i(+s_0t_3 + s_1t_2 + s_2t_1 + s_3t_0) \\ a_0^i(-s_0t_3 - s_1t_2 + s_2t_1 + s_3t_0) + a_1^i(-s_0t_2 + s_1t_3 - s_2t_0 + s_3t_1) \end{pmatrix}$$

$$\begin{pmatrix} +a_2^i(+s_0t_2 + s_1t_3 - s_2t_0 - s_3t_1) + a_3^i(+s_0t_3 - s_1t_2 + s_2t_1 - s_3t_0) \\ +a_2^i(-s_0t_3 + s_1t_2 + s_2t_1 - s_3t_0) + a_3^i(+s_0t_2 + s_1t_3 + s_2t_0 + s_3t_1) \\ +a_2^i(+s_0t_0 - s_1t_1 + s_2t_2 - s_3t_3) + a_3^i(-s_0t_1 - s_1t_0 + s_2t_3 + s_3t_2) \\ +a_2^i(+s_0t_1 + s_1t_0 + s_2t_3 + s_3t_2) + a_3^i(+s_0t_0 - s_1t_1 - s_2t_2 + s_3t_3 \end{pmatrix}$$

This quaternion treated as a 4-dimensional vector can be written as the result of the product of a 4×4 matrix $\mathbf{M}$, which depends on $\mathfrak{s}$ and $\mathfrak{t}$, and $\mathfrak{a}_i$ considered as 4-dimensional vector:

$$= \begin{pmatrix} +s_0t_0 + s_1t_1 + s_2t_2 + s_3t_3 & +s_0t_1 - s_1t_0 - s_2t_3 + s_3t_2 \\ -s_0t_1 + s_1t_0 - s_2t_3 + s_3t_2 & +s_0t_0 + s_1t_1 - s_2t_2 - s_3t_3 \\ -s_0t_2 + s_1t_3 + s_2t_0 - s_3t_1 & +s_0t_3 + s_1t_2 + s_2t_1 + s_3t_0 \\ -s_0t_3 - s_1t_2 + s_2t_1 + s_3t_0 & -s_0t_2 + s_1t_3 - s_2t_0 + s_3t_1 \end{pmatrix}$$

$$\begin{pmatrix} +s_0t_2 + s_1t_3 - s_2t_0 - s_3t_1 & +s_0t_3 - s_1t_2 + s_2t_1 - s_3t_0 \\ -s_0t_3 + s_1t_2 + s_2t_1 - s_3t_0 & +s_0t_2 + s_1t_3 + s_2t_0 + s_3t_1 \\ +s_0t_0 - s_1t_1 + s_2t_2 - s_3t_3 & -s_0t_1 - s_1t_0 + s_2t_3 + s_3t_2 \\ +s_0t_1 + s_1t_0 + s_2t_3 + s_3t_2 & +s_0t_0 - s_1t_1 - s_2t_2 + s_3t_3 \end{pmatrix} \begin{pmatrix} a_0^i \\ a_1^i \\ a_2^i \\ a_3^i \end{pmatrix}$$

$$= \mathbf{M}\mathfrak{a}_i$$

One easily verifies by a computer algebra program that it is $\mathbf{M}^T\mathbf{M} = \mathbb{1}$ and $\det(\mathbf{M}) = 1$ and thus $\mathbf{M} \in \mathrm{SO}(4)$. Hence we have a mapping

$$\zeta : \mathfrak{Q}^* \times \mathfrak{Q}^* \to \mathrm{SO}(4) ,$$

$$\left(\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}, \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} \right) \mapsto \begin{pmatrix} +s_0t_0 + s_1t_1 + s_2t_2 + s_3t_3 & \dots \\ -s_0t_1 + s_1t_0 - s_2t_3 + s_3t_2 & \dots \\ -s_0t_2 + s_1t_3 + s_2t_0 - s_3t_1 & \dots \\ -s_0t_3 - s_1t_2 + s_2t_1 + s_3t_0 & \dots \end{pmatrix} \quad \text{(see above)} .$$

Furthermore, one can easily verify by a computer algebra program that it is

$$\zeta(\mathfrak{s}\mathfrak{u}, \mathfrak{t}\mathfrak{v}) = \zeta(\mathfrak{s}, \mathfrak{t})\zeta(\mathfrak{u}, \mathfrak{v}) \quad \text{for } \mathfrak{s}, \mathfrak{t}, \mathfrak{u}, \mathfrak{v} \in \mathfrak{Q}^*.$$

Thus ζ is even a group homomorphism if we provide $\mathfrak{Q}^* \times \mathfrak{Q}^*$ with the componentwise multiplication.
But what does that mean for our original problem? Originally we wanted to find rotations $\mathbf{R}_S$ and $\mathbf{R}_T$ so that

$$\mathbf{R}_{B_i} \approx \mathbf{R}_S \mathbf{R}_{A_i} \mathbf{R}_{B_i}^T$$

holds. By switching to unit quaternions and applying ζ that means we are looking for a rotation $\mathbf{M} \in \mathrm{SO}(4)$ so that it is

$$\mathfrak{b}_i \approx \mathbf{M}\mathfrak{a}_i \ .$$

But for just this problem lemma 4.1 offers a solution if the $\mathfrak{a}_i$ and the $\mathfrak{b}_i$ are written as the columns of two $4 \times n$ matrices. Of course not exactly in the way as we wanted to solve this problem. Originally we wanted to minimize the sum over the squared angles $\delta_i := \delta(\mathbf{R}_{B_i}, \mathbf{R}_S \mathbf{R}_{B_i} \mathbf{R}_T^T)$ between the rotations $\mathbf{R}_{B_i}$ and $\mathbf{R}_S \mathbf{R}_{B_i} \mathbf{R}_T^T$ while when applying lemma 4.1 we minimize the sum over the squared distances of the corresponding unit quaternions $\mathfrak{b}_i$ and $\mathfrak{s}\mathfrak{a}_i\mathfrak{t}^*$ considered as 4-dimensional vectors. However, as already shown in section 3.1, the angle between unit quaternions is just half the angle between the corresponding rotations, and the angle between vectors is related to the length of their distance vector by the cosine rule. Thus we have the relation

$$|\mathfrak{b}_i - \mathfrak{s}\mathfrak{a}_i\mathfrak{t}^*| = \sqrt{2\left(1 - \cos\left(\frac{\delta_i}{2}\right)\right)} = \frac{\delta_i}{2} + O(\delta_i^3) \quad \text{for} \quad \delta_i \in [0, \pi] \ .$$

The factor of $1/2$ does not affect the minimization and because the next term in the Taylor series has order 3 there should be no big difference as long as the δ_i are small for the found solution.
But when we now have found a matrix $\mathbf{M} \in \mathrm{SO}(4)$ by lemma 4.1, how do we get back to the unit quaternions $\mathfrak{s}$ and $\mathfrak{t}$ and thus the corresponding rotations $\mathbf{R}_S$ and $\mathbf{R}_T$? Or mathematically spoken, how can we invert the homomorphism ζ. The solution can be seen when considering

$$\mathbf{M} - \mathbf{M}^T = \begin{pmatrix} 0 & +2s_0t_1 - 2s_1t_0 & +2s_0t_2 - 2s_2t_0 & +2s_0t_3 - 2s_3t_0 \\ -2s_0t_1 + 2s_1t_0 & 0 & -2s_0t_3 - 2s_3t_0 & +2s_0t_2 + 2s_2t_0 \\ -2s_0t_2 + 2s_2t_0 & +2s_0t_3 + 2s_3t_0 & 0 & -2s_0t_1 - 2s_1t_0 \\ -2s_0t_3 + 2s_3t_0 & -2s_0t_2 - 2s_2t_0 & +2s_0t_1 + 2s_1t_0 & 0 \end{pmatrix}$$

and $\mathrm{trace}(\mathbf{M}) = 4\, s_0 t_0$. Thus it is

$$\begin{pmatrix} M_{00}+M_{11}+M_{22}+M_{33} \\ (M_{32}-M_{23})-(M_{01}-M_{10}) \\ (M_{13}-M_{31})-(M_{02}-M_{20}) \\ (M_{21}-M_{12})-(M_{03}-M_{30}) \end{pmatrix} = \begin{pmatrix} 4s_0t_0 \\ 4s_1t_0 \\ 4s_2t_0 \\ 4s_3t_0 \end{pmatrix} = 4t_0\mathfrak{s} \tag{4.22}$$

$$\begin{pmatrix} M_{00}+M_{11}+M_{22}+M_{33} \\ (M_{32}-M_{23})+(M_{01}-M_{10}) \\ (M_{13}-M_{31})+(M_{02}-M_{20}) \\ (M_{21}-M_{12})+(M_{03}-M_{30}) \end{pmatrix} = \begin{pmatrix} 4s_0t_0 \\ 4s_0t_1 \\ 4s_0t_2 \\ 4s_0t_3 \end{pmatrix} = 4s_0\mathfrak{t} \tag{4.23}$$

if we denote the entries of $\mathbf{M}$ by $M_{00}, \ldots, M_{33}$. By normalizing the 4-dimensional vectors in (4.22) and (4.23) we can obviously retrieve $\mathfrak{s}$ and $\mathfrak{t}$. However, this only works for $s_0 \neq 0$ and $t_0 \neq 0$ and thus $\mathrm{trace}(\mathbf{M}) \neq 0$.
To avoid this case we can take advantage of ζ being a group homomorphism. Obviously every component of a quaternion can be swapped into the scalar component (i.e. the first component, see section 3.1) – plus possibly two sign flips – by multiplying by $\mathfrak{e}_0 = (1,0,0,0)^T, \ldots, \mathfrak{e}_3 = (0,0,0,1)^T$. Hence we can find $k, l \in \{0,1,2,3\}$ so that $\mathfrak{e}_k\mathfrak{s}$ and $\mathfrak{e}_l\mathfrak{t}$ have a scalar component $\neq 0$ and thus it is

$$\mathrm{trace}(\zeta(\mathfrak{e}_k\mathfrak{s}, \mathfrak{e}_l\mathfrak{t})) = \mathrm{trace}(\zeta(\mathfrak{e}_k, \mathfrak{e}_l)\zeta(\mathfrak{s}, \mathfrak{t})) \neq 0\ .$$

Hence we have 16 matrices $\zeta(\mathfrak{e}_k, \mathfrak{e}_l) \in \mathrm{SO}(4)$, $k, l \in \{0,1,2,3\}$, and there must be one matrix $\mathbf{P} = \zeta(\mathfrak{e}_{k_0}, \mathfrak{e}_{l_0})$ among them so that $\mathrm{trace}(\mathbf{PM}) \neq 0$. When we have found this matrix, we can retrieve $\mathfrak{e}_{k_0}\mathfrak{s}$ and $\mathfrak{e}_{l_0}\mathfrak{t}$ by applying (4.22) and (4.23) and thus by multiplying by $\mathfrak{e}_{k_0}^*$ resp $\mathfrak{e}_{l_0}^*$ we get $\mathfrak{s}$ and $\mathfrak{t}$.
Because the matrices $\zeta(\mathfrak{e}_k, \mathfrak{e}_l)$ can be divided into 4 groups of 4 matrices of the form

$$\begin{pmatrix} \pm1 & & & \\ & \pm1 & & \\ & & \pm1 & \\ & & & \pm1 \end{pmatrix}, \begin{pmatrix} & \pm1 & & \\ \pm1 & & & \\ & & & \pm1 \\ & & \pm1 & \end{pmatrix}, \begin{pmatrix} & & \pm1 & \\ & & & \pm1 \\ \pm1 & & & \\ & \pm1 & & \end{pmatrix}, \begin{pmatrix} & & & \pm1 \\ & & \pm1 & \\ & \pm1 & & \\ \pm1 & & & \end{pmatrix},$$

multiplying from left by them means only swapping some rows and/or flipping some signs. Thus the computational costs for finding $\mathbf{P}$ are lower as they might appear at first glance. For maximum numerical stability when normalizing the vectors in (4.22) and (4.23) we can simply apply all 16 matrices and take the one with the largest absolute value of $\mathrm{trace}(\mathbf{PM})$. For doing this only $15 \cdot 3 = 45$ extra additions are necessary for calculating the 15 additional traces.
Because of $\zeta(\mathfrak{s}, \mathfrak{t}) = \zeta(-\mathfrak{s}, -\mathfrak{t})$ ζ is obviously no isomorphism, but when reflecting on (4.22) and (4.23) it becomes clear that this is the only ambiguity. This ambiguity is no problem because, when translating the unit quaternions back into $\mathrm{SO}(3)$ rotations, a sign flip does not matter anyway.

The algorithm we have sketched so far is mathematically equivalent to the algorithm presented in [54] because both algorithms minimize the expression

$$\sum_{i=1}^{n} |\mathfrak{b}_i - \mathfrak{s}\mathfrak{a}_i\mathfrak{t}^*|^2 \ .$$

However, while our algorithm finds the optimal solution by a SVD of the 4×4 matrix

$$\sum_{i=1}^{n} \mathfrak{a}_i\mathfrak{b}_i^T \ , \tag{4.24}$$

the algorithm presented in [54] finds the solution as an eigenvector of the 4×4 matrix

$$\mathbf{C}^T\mathbf{C} \quad \text{with} \quad \mathbf{C} = \sum_{i=1}^{n} \begin{pmatrix} a_0^i & a_1^i & a_2^i & a_3^i \\ -a_1^i & a_0^i & a_3^i & -a_2^i \\ -a_2^i & -a_3^i & a_0^i & a_1^i \\ -a_3^i & a_2^i & -a_1^i & a_0^i \end{pmatrix} \begin{pmatrix} b_0^i & -b_1^i & -b_2^i & -b_3^i \\ b_1^i & b_0^i & b_3^i & -b_2^i \\ b_2^i & -b_3^i & b_0^i & b_1^i \\ b_3^i & b_2^i & -b_1^i & b_0^i \end{pmatrix} . \tag{4.25}$$

Thus it can also be found as a right singular vector of $\mathbf{C}$ which is numerically better since $\mathbf{C}^T\mathbf{C}$ has a squared condition compared to $\mathbf{C}$.
Obviously for calculating a summand in (4.24) only $4^2 = 16$ multiplications are necessary while calculating a summand in (4.25) requires $4^3 = 64$ multiplications. Hence our approach is much faster as long as the input data set is large so that the effort for calculating (4.24) resp. (4.25) exceeds the effort for calculating the SVD of (4.24) resp. the eigenvectors of (4.25), which is independent of n.
However, there is still a big problem concerning the ambiguity that an unit quaternion and its negated union quaternion represent the same rotation when transforming the rotations to quaternions at the beginning of the algorithm. As we showed, the solution of the problem means geometrically that a set of unit quaternions considered as 4-dimensional vectors has to be rotated onto another set of unit quaternions. Hence it becomes clear at once that it makes a big difference if some of the quaternions are arbitrarily negated or , to use a more geometric term, flipped.
In [54] an ambiguity is only mentioned in the case that a quaternion's scalar component is zero because rotation quaternions are apparently assumed to have a non-negative scalar component. However, this assumption is arbitrary and not backed mathematically.
E.g. consider the two unit quaternion pairs

$$\mathfrak{u}_1 = [\cos(-30°), \sin(-30°), 0, 0]^T = \left[\frac{\sqrt{3}}{2}, -\frac{1}{2}, 0, 0\right]^T ,$$

$$\mathfrak{v}_1 = [\cos(\ \ 30°), \sin(\ \ 30°), 0, 0]^T = \left[\frac{\sqrt{3}}{2}, \ \ \frac{1}{2}, 0, 0\right]^T$$

and

$$\mathfrak{u}_2 = [\cos(\ 60°), \sin(\ 60°), 0, 0]^T = \left[\ \frac{1}{2}, \frac{\sqrt{3}}{2}, 0, 0\right]^T,$$
$$\mathfrak{v}_2 = [\cos(120°), \sin(120°), 0, 0]^T = \left[-\frac{1}{2}, \frac{\sqrt{3}}{2}, 0, 0\right]^T.$$

Obviously

$$\underset{\mathbf{R}\in \mathrm{SO}(4)}{\arg\min} \left(|\mathbf{R}\mathfrak{u}_1 - \mathfrak{v}_1|^2 + |\mathbf{R}\mathfrak{u}_2 - \mathfrak{v}_2|^2\right)$$

is perfectly solved by the rotation matrix

$$\begin{pmatrix} \cos(90°) & -\sin(90°) & 0 & 0 \\ \sin(90°) & \cos(90°) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

with a minimum of 0. However, if we assume that rotation quaternions have non-negative scalar components, we have to flip $\mathfrak{v}_2$. But as solution for

$$\underset{\mathbf{R}\in \mathrm{SO}(4)}{\arg\min} \left(|\mathbf{R}\mathfrak{u}_1 - \mathfrak{v}_1|^2 + |\mathbf{R}\mathfrak{u}_2 + \mathfrak{v}_2|^2\right)$$

lemma 4.1 gives the $\mathrm{SO}(4)$ matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

with a minimum of $4(1 - \cos(30°)) \approx 0.5359$, which is no longer a perfect solution.

Thus when rotating a set of rotations quaternions $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ onto another set of rotation quaternions $\mathfrak{b}_1, \ldots, \mathfrak{b}_n$ all possible "flipping configurations" have to be considered. However, flipping $\mathfrak{a}_i$ *and* $\mathfrak{b}_i$ for a certain i does not make a difference as well as flipping all $\mathfrak{a}_i$ or $\mathfrak{b}_i$ since this can be compensated by the rotation because of $(-\mathbb{1}) \in \mathrm{SO}(4)$. Thus, for n pairs $(\mathfrak{a}_i, \mathfrak{b}_i)$ there are 2^{n-1} relevant flipping configurations.

Because of this exponential relation it is not possible to check every flipping configuration. But the problem can be solved by a RANSAC-like approach [57]. We randomly choose four quaternion pairs, calculate the matrix $\mathbf{M}$ according to lemma 4.1 for all $2^3 = 8$ flipping configurations for these four pairs and check if for a certain threshold $c \in (0, 1)$

$$|\mathfrak{b}_i^T \mathbf{M} \mathfrak{a}_i| > c \quad \text{for all} \quad 1 \leq i \leq n \tag{4.26}$$

is fulfilled for a flipping configuration. If yes, we take this flipping configuration resp. the corresponding $\mathbf{M}$ with the smallest sum

$$\sum_{i=1}^{n} |\mathfrak{b}_i - \mathbf{M}\mathfrak{a}_i|^2$$

if there are several configurations fulfilling (4.26). Then for all pairs with $\mathfrak{b}_i^T \mathbf{M} \mathfrak{a}_i < -c$ we flip $\mathfrak{a}_i$ and calculate $\mathbf{M}$ again for all pairs in this flipping configuration. If no flipping configuration fulfills (4.26), we iterate this process for four new random pairs.
Criterion (4.26) means that after a possible flip the angle between $\mathfrak{b}_i$ and $\mathbf{M}\mathfrak{a}_i$ is below the threshold $\arccos(c)$. For the dataset the algorithm was tested with this procedure terminated quite quickly for $c = 0.5 \Rightarrow \arccos(c) = 60°$. However, for noisier data this might not be the case and one has to abort the iteration after a certain time and take the largest subset of all $\mathfrak{a}_i$ and $\mathfrak{b}_i$ fulfilling (4.26). In the RANSAC terminology this is called taking the largest consensus set.
The dataset the algorithm was tested with consisted of 3823 poses retrieved by an infrared tracking system and 3823 corresponding poses simultaneously retrieved by an ultrasound tracking system. The raw data consisted of 4 marker positions for the infrared system and 3 marker positions for the ultrasound system, which were transformed to poses by the algorithm described in the previous section.
But in fact this algorithm was only necessary for the ultrasound marker positions because the ultrasound markers were arbitrarily placed on the tracked object and tracked independently. Thus the relative positions of the delivered marker positions to each other were not completely consistent because of noise.
The infrared tracking system used a cross-shaped carrier with the markers in well-defined positions on it, which was fixed to the tracked object. Because this is taken into account in the mathematical model of the infrared tracking system, it delivered completely consistent marker positions.
To evaluate the presented approach, that we want to call quaternion rotation approach because quaternions considered as 4-dimensional vectors are rotated, it has been compared to another approach presented in [55] that uses the Kronecker product defined by

$$\otimes : \mathbb{R}^{k\times l} \times \mathbb{R}^{m\times n} \to \mathbb{R}^{km\times ln}, \begin{pmatrix} a_{11} & \dots & a_{1l} \\ \vdots & \ddots & \vdots \\ a_{k1} & \dots & a_{kl} \end{pmatrix} \otimes \mathbf{B} \mapsto \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1l}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{k1}\mathbf{B} & \dots & a_{kl}\mathbf{B} \end{pmatrix}.$$

By that the equations

$$\begin{aligned} \mathbf{R}_S \mathbf{R}_{A_i} &\approx \mathbf{R}_{B_i} \mathbf{R}_T \\ \mathbf{R}_S \mathbf{b}_{A_i} + \mathbf{b}_S &\approx \mathbf{R}_{B_i} \mathbf{b}_T + \mathbf{b}_{B_i}, \end{aligned}$$

which follows from (4.21), can be written as

$$\begin{pmatrix} \mathbf{R}_{A_i}^T \otimes \mathbb{1}_{3\times 3} & -\mathbb{1}_{3\times 3} \otimes \mathbf{R}_{B_i} & \mathbf{0} & \mathbf{0} \\ \mathbf{b}_{A_i}^T \otimes \mathbb{1}_{3\times 3} & \mathbf{0} & \mathbb{1}_{3\times 3} & -\mathbf{R}_{B_i} \end{pmatrix} \begin{pmatrix} \mathbf{v}_S \\ \mathbf{v}_T \\ \mathbf{b}_S \\ \mathbf{b}_T \end{pmatrix} \approx \begin{pmatrix} \mathbf{0} \\ \mathbf{b}_{B_i} \end{pmatrix}$$

with $\mathbf{v}_S$ and $\mathbf{v}_T$ defined as 9-dimensional vectors stacking the columns of $\mathbf{R}_S$ and $\mathbf{R}_T$. Thus it is obvious to find $\mathbf{v}_S$, $\mathbf{v}_T$, $\mathbf{b}_S$ and $\mathbf{b}_T$ as the solution of the

minimization problem

$$\arg\min \left| \begin{pmatrix} \mathbf{R}_{A_1}^T \otimes \mathbb{1}_{3\times3} & -\mathbb{1}_{3\times3} \otimes \mathbf{R}_{B_1} & \mathbf{0} & \mathbf{0} \\ \mathbf{b}_{A_1}^T \otimes \mathbb{1}_{3\times3} & \mathbf{0} & \mathbb{1}_{3\times3} & -\mathbf{R}_{B_1} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{R}_{A_n}^T \otimes \mathbb{1}_{3\times3} & -\mathbb{1}_{3\times3} \otimes \mathbf{R}_{B_n} & \mathbf{0} & \mathbf{0} \\ \mathbf{b}_{A_n}^T \otimes \mathbb{1}_{3\times3} & \mathbf{0} & \mathbb{1}_{3\times3} & -\mathbf{R}_{B_n} \end{pmatrix} \begin{pmatrix} \mathbf{v}_S \\ \mathbf{v}_T \\ \mathbf{b}_S \\ \mathbf{b}_T \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ \mathbf{b}_{B_1} \\ \vdots \\ \mathbf{0} \\ \mathbf{b}_{B_n} \end{pmatrix} \right|,$$

which can be found by a QR decomposition of the big $12n\times24$ matrix.
However, the resulting $\mathbf{v}_S$ and $\mathbf{v}_T$ resp. $\mathbf{R}'_S$ and $\mathbf{R}'_T$ when written as matrix are not necessarily orthogonal. Thus the final orthogonal $\mathbf{R}_S$ and $\mathbf{R}_T$ are found according to lemma 4.1 as the solution of

$$\underset{\mathbf{R}_S\in\mathrm{SO}(3)}{\arg\min} \|\mathbf{R}_S - \mathbf{R}'_S\|_F \quad \text{and} \quad \underset{\mathbf{R}_T\in\mathrm{SO}(3)}{\arg\min} \|\mathbf{R}_T - \mathbf{R}'_T\|_F$$

by a singular value decomposition of $\mathbf{R}'_S$ and $\mathbf{R}'_T$.
The comparison between both approaches yielded the following results for the test dataset:

	average rotation error [°]	average translation error [mm]
quat. rot. approach	3.06425	6.46702
Kronecker approach	3.06571	9.25119

with

$$\text{average rotation error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(\delta(\mathbf{R}_S\mathbf{R}_A, \mathbf{R}_B\mathbf{B})\right)^2} \tag{4.27}$$

$$\text{average translation error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} |\mathbf{R}_S\mathbf{b}_{A_i} + \mathbf{b}_S - (\mathbf{R}_{B_i}\mathbf{b}_T + \mathbf{b}_{\mathbf{B}_i})|^2} \tag{4.28}$$

The SVD and QR decompositions were calculated again with the help of the Eigen library [51]. Obviously the quaternion rotation approach yielded a slightly better result.
In [55] the Kronecker approach has been compared to the original quaternion approach according to [53] for synthetic random data and outperformed it clearly. This could be explained by the assumption that the quaternion flipping problem is not treated properly in [53] because apart from that the quaternion rotation approach is mathematically equivalent to our approach.
To confirm this thesis the quaternion rotation and the Kronecker approach were additionally compared for synthetic random data. For that a number of rigid random transformations A_i as well as S and T with uniformly distributed rotations and translations uniformly distributed in the range of [-1000m,1000mm] for every

number of transformations	mean value of average rotation error of quat. rot. approach [°]	mean value of average rotation error of Kronecker approach [°]	mean value of average translation error of quat. rot. approach [mm]	mean value of average translation error of Kronecker approach [mm]
4	5.481 ± 0.054	8.421 ± 0.206	71.8 ± 1.0	113.1 ± 4.3
5	6.133 ± 0.049	23.389 ± 0.788	74.6 ± 0.9	450.8 ± 25.0
6	6.498 ± 0.042	9.336 ± 0.119	76.6 ± 0.8	104.6 ± 2.4
7	6.748 ± 0.038	8.410 ± 0.068	77.1 ± 0.8	80.8 ± 1.3
8	6.971 ± 0.039	8.230 ± 0.055	78.3 ± 0.8	75.1 ± 1.0
10	7.233 ± 0.034	8.110 ± 0.042	79.6 ± 0.8	72.5 ± 0.8
12	7.406 ± 0.031	8.055 ± 0.035	79.7 ± 0.7	72.1 ± 0.8
14	7.481 ± 0.029	8.043 ± 0.031	80.5 ± 0.7	73.4 ± 0.7
16	7.560 ± 0.026	8.023 ± 0.028	79.1 ± 0.7	72.5 ± 0.7
32	7.887 ± 0.020	8.089 ± 0.020	79.4 ± 0.7	75.5 ± 0.7
64	8.005 ± 0.014	8.104 ± 0.014	81.3 ± 0.7	79.3 ± 0.7
128	8.095 ± 0.010	8.144 ± 0.010	80.3 ± 0.7	79.3 ± 0.7
256	8.130 ± 0.007	8.154 ± 0.007	79.3 ± 0.7	78.8 ± 0.7
512	8.140 ± 0.005	8.152 ± 0.005	81.7 ± 0.7	81.4 ± 0.7
1024	8.153 ± 0.003	8.159 ± 0.003	79.8 ± 0.7	79.6 ± 0.7
2048	8.161 ± 0.002	8.164 ± 0.002	79.3 ± 0.7	79.2 ± 0.7
4096	8.161 ± 0.002	8.163 ± 0.002	80.6 ± 0.7	80.6 ± 0.7

Table 4.1: Results for the mean values for 1000 runs with random transformations for the average rotation and translation error according to (4.27) and (4.28).

component were generated. Uniformly distributed rotations are generated by taking a rotation quaternion uniformly distributed over $S^3 = \{\mathfrak{q} \in \mathbb{R}^4 \mid |\mathfrak{q}| = 1\}$ and converting it to a rotation matrix. After that proper transformations B_i as $B_i = SA_iT^{-1}$ were calculated and the A_i and B_i were disturbed by applying random rotations to the transformations' rotational part and random shifts to the transformations' translational part. The disturbance rotations had uniformly distributed rotation axes and uniformly distributed rotation angles in [0,10°] and the random shifts were uniformly distributed in [-10mm,10mm] in every component. Table 4.1 shows the mean values for 1000 runs with a different number of transformations and figure 4.4 the corresponding plots. The errors $\Delta\mu$ of the mean values $\mu = 1/n \sum_{i=1}^{n} \alpha_i$ of n values $\alpha_1, \ldots, \alpha_n$ were calculated according to the formula

$$\Delta\mu = \sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (\alpha_i - \mu)^2} \; .$$

As can be seen, the Kronecker approach yields only poor results for below 7 transformation with a catastrophic outlier for 5 transformations. This is due to the fact that for such few transformations the calculated $\mathbf{R}'_S$ and $\mathbf{R}'_T$ are far away from

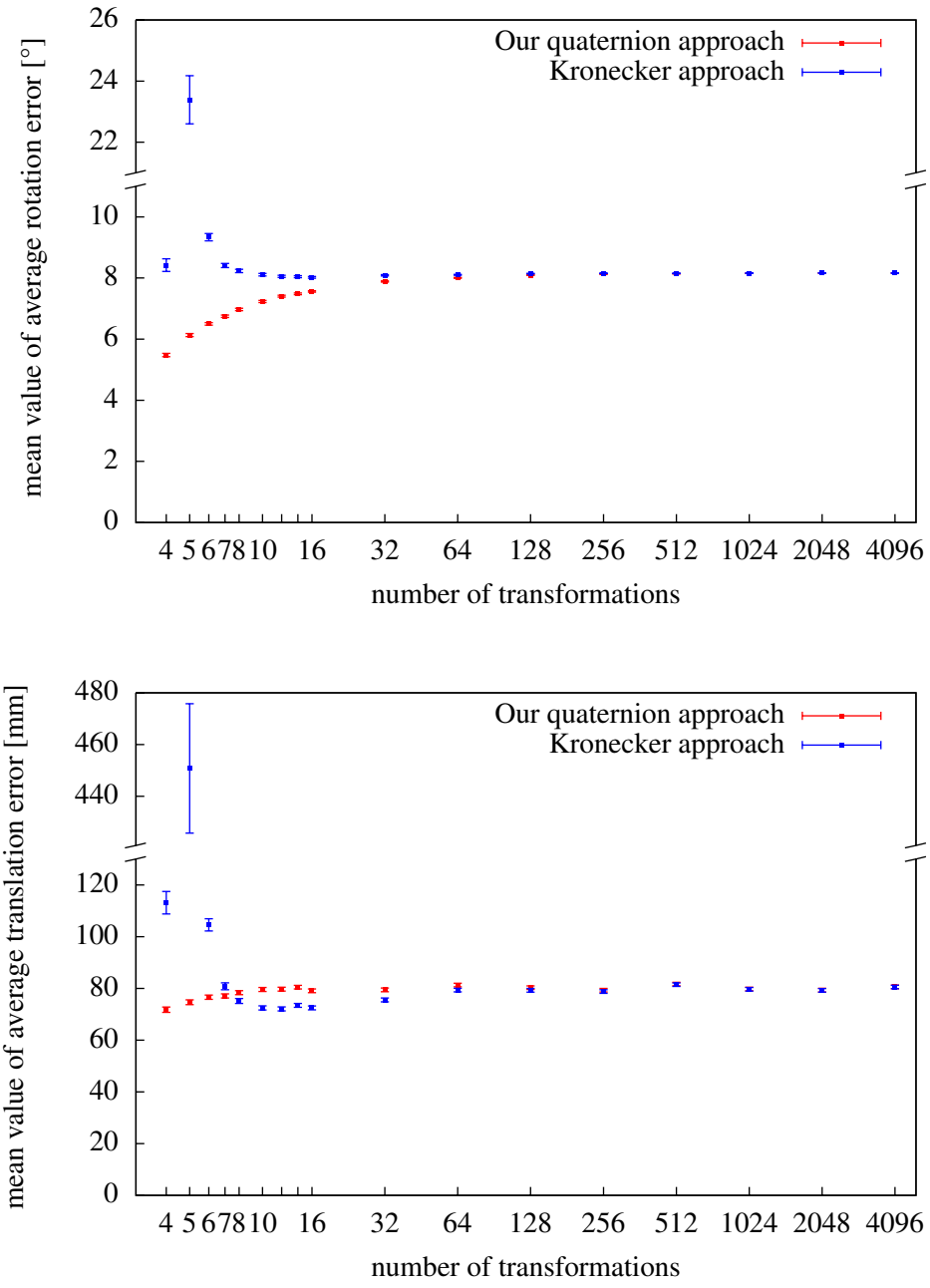


Figure 4.4: Plots of the mean values for 1000 runs with random transformations for the average rotation (a) and translation (b) error according to (4.27) and (4.28). The y-axes are broken so that the plots cover the outlier for 5 transformations for the Kronecker approach.

being orthogonal, which is revealed by singular values far away from 1 when orthogonalizing $\mathbf{R}'_S$ and $\mathbf{R}'_T$ by their SVD.
For a sufficient number of transformations both approaches yield comparable results with the tendency that the quaternion approach yields a slightly better rotational result while the Kronecker approach yields a slightly better translational result. However, this is to be expected because the quaternion approach solves at first only the rotational problem and calculates then the translations, which are hence afflicted by error propagation. The Kronecker approach calculates rotations as well as translations simultaneously, but does not account for the rotation matrices to be orthogonal and requires an orthogonalization as post processing.
For sure, it can be said that the quaternion rotation approach is not outperformed by the Kronecker approach as in [55] confirming the thesis that in [54, 55] the quaternion flipping problem was not treated properly.
Besides, it was verified for the synthetically generated and disturbed rotations that the RANSAC approach always found the correct flipping configuration which were determined by the undisturbed rotations. However, since the angle of the disturbance rotation was limited to 10° at maximum, this was probably to be expected.

Conclusion and Outlook

In this work several algorithms have been shown to solve fundamental problems in the scope of CAD and DMU. In chapter 1 we developed a heuristic to retrieve a voxelization of inner space for surface geometries consisting of triangles “soups”, which is robust even against large holes. By this heuristic even millimeter resolution and below can be reached within a reasonable running time.
Besides our usage many fields of application are conceivable for this heuristic. For example, by such a fine voxelization of the interior and the outside also the triangles of the input geometry could be determined which actually enclose the interior or form the outer surface of the object. This could be done by checking which triangles intersect the first voxel layers not belonging to the interior or the outside. Many other algorithms based on the triangle geometry, like packing algorithms using contact simulation, only need these triangles and their running time scales with the number of input triangles. Thus, such algorithms could be significantly accelerated by thinning out the set of input triangles by the presented heuristic.
In chapter 2 data structures and algorithms for calculating approximate distances and penetrations very fast were presented and applied to pack arbitrarily shaped objects into a trunk. It was shown that this approach works quite well for packing objects which have a shape similar to boxes like suitcases and golf bags though there might be room for improvement for packing more complexly shaped objects. In the scope of this packing algorithm a very fast and easy to implement algorithm to calculate a largest box in a 3D voxelization has been presented. An algorithm to solve such a basic geometric problem should be a very interesting result on its own and it could be a very promising approach to further improve the presented packing algorithm. Instead of calculating only one largest enclosed box this algorithm could be iterated in order to calculate an approximation of arbitrarily shaped objects by boxes for a better penetration calculation.
Distance and penetration calculation, as it is used for the presented packing algorithm, is an absolutely fundamental workhorse for high level algorithms in the scope of CAD and DMU. For most of these high level algorithms running time is crucial. Therefore they could largely benefit from the presented distance and penetration algorithm. Although it might not be applicable in all cases because distances and penetrations are only calculated approximately, this plays a minor role for many applications since in CAD and DMU there are always certain toler-

ance bounds.
Furthermore, this restriction might also be overcome algorithmically. For example, algorithms with a very large search space, as in motion planning, could use our fast approximative algorithm to find a solution and then validate or optimize this solution by exact algorithms in a post-processing step.
In chapter 3 an approach was presented how the concept of bounding volumes can be extended to rigid transformations, and it has been applied for an accelerated motion analysis of a car's engine. Rigid transformations play an important role in many fields of CAD and DMU, for instance whenever trajectories come into play. By our approach we have built a bridge between the algorithmic concepts for triangle based geometry, like meshes, on the one hand and trajectories on the other hand, which were quite separate so far.
An obvious application of the presented rigid transformation hierarchies is the implicit representation of swept volumes, as already sketched at the end of chapter 3. Besides digital mock-up, swept volumes also play an important role in numerically controlled machining and robot analysis. However, calculating explicit mesh representations of swept volumes might be too time-consuming for many application cases. Thus, the proposed implicit representation by rigid transformation hierarchies might be a very promising approach in such cases and should be the subject of further research.

Appendix A

Used Mathematical Theorems

Lemma A.1 *Let* $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$*,* $d \geq 2$*, be vectors with* $|\mathbf{a}| = 1$ *and* $|\mathbf{b}| = 1$*. Then the spectral norm of the matrix* $\mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T$ *can be calculated by:*

$$\left\|\mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T\right\|_2 = \sqrt{1 - (\mathbf{a}^T\mathbf{b})^2}$$

Proof:
Let be $\mathbf{A} := \mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T$. It is well-known that that the spectral norm of the matrix $\mathbf{A}$ can be calculated as the square root of the biggest eigenvalue of $\mathbf{A}^T\mathbf{A}$ [30]. Because the orthogonal complement of $\langle \mathbf{a}, \mathbf{b} \rangle$ is obviously contained by the kernel of $\mathbf{A}$ and $\mathbf{A}^T\mathbf{A}$, we have only to consider how $\langle \mathbf{a}, \mathbf{b} \rangle$ is transformed by $\mathbf{A}^T\mathbf{A}$. Considering the skew-symmetry of $\mathbf{A}$ as well as $\mathbf{a}^T\mathbf{b} = \mathbf{b}^T\mathbf{a}$ we get:

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\mathbf{a} &= -\left(\mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T\right)^2 \mathbf{a} = -\left(\mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T\right)\left(\mathbf{a}\mathbf{b}^T\mathbf{a} - \mathbf{b}\right) \\ &= -\mathbf{a}\left(\mathbf{b}^T\mathbf{a}\right)^2 + \mathbf{a} + \mathbf{b}\mathbf{b}^T\mathbf{a} - \mathbf{b}\mathbf{a}^T\mathbf{b} = \left(1 - \left(\mathbf{a}^T\mathbf{b}\right)^2\right)\mathbf{a} \end{aligned}$$

Because of the symmetry of $\mathbf{A}^T\mathbf{A}$ in $\mathbf{a}$ and $\mathbf{b}$ it follows in the same way:

$$\mathbf{A}^T\mathbf{A}\mathbf{b} = \left(1 - \left(\mathbf{a}^T\mathbf{b}\right)^2\right)\mathbf{b}$$

Hence we can conclude for $\mathbf{A}^T\mathbf{A}$ restricted to the subspace $\langle \mathbf{a}, \mathbf{b} \rangle$:

$$\mathbf{A}^T\mathbf{A}\big|_{\langle \mathbf{a},\mathbf{b} \rangle} = \left(1 - \left(\mathbf{a}^T\mathbf{b}\right)^2\right)\mathbb{1}$$

Thus $\mathbf{A}^T\mathbf{A}$ has besides 0 the (double) eigenvalue $1 - \left(\mathbf{a}^T\mathbf{b}\right)^2$ and we are done.

□

Proposition A.2 (Triangle inequation for spherical triangles) *Given three distinct points on the surface* S^{d-1} *of the* d*-dimensional sphere:* $\mathbf{a}, \mathbf{b}, \mathbf{c} \in S^{d-1} :=$

$\left\{\mathbf{x} \in \mathbb{R}^d | \, |\mathbf{x}| = 1\right\}$, $d \geq 3$. *By connecting these points by shortest paths on great circles on the sphere's surface we get a spherical triangle. For this triangle the triangle inequation is valid:*

$$\measuredangle(\mathbf{a}, \mathbf{b}) \leq \measuredangle(\mathbf{a}, \mathbf{c}) + \measuredangle(\mathbf{c}, \mathbf{b}) \tag{A.1}$$

The arc length resp. the angle between two points on the surface S^{d-1} of d-dimensional sphere can be calculated as:

$$\measuredangle(\mathbf{xy}) = \arccos\left(\mathbf{x}^T\mathbf{y}\right) \in [0, \pi] \quad , \quad \mathbf{x}, \mathbf{y} \in S^{d-1}$$

The equality case can only occur if $\mathbf{a}$, $\mathbf{b}$ *and* $\mathbf{c}$ *are linearly dependent and the spherical triangle is degenerated.*

Proof:
According to lemma A.1 it is:

$$\left|\mathbf{a}\left(\mathbf{b}^T\mathbf{c}\right) - \mathbf{b}\left(\mathbf{a}^T\mathbf{c}\right)\right|^2 = \left|\left(\mathbf{a}\mathbf{b}^T - \mathbf{b}\mathbf{a}^T\right)\mathbf{c}\right|^2 \leq 1 - \left(\mathbf{a}^T\mathbf{b}\right)^2$$

From the proof of lemma A.1 it is clear that the equality case only occurs iff $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are linearly dependent. Expanding yields:

$$\Leftrightarrow \quad \left(\mathbf{b}^T\mathbf{c}\right)^2 + \left(\mathbf{a}^T\mathbf{c}\right)^2 - 2\left(\mathbf{a}^T\mathbf{b}\right)\left(\mathbf{b}^T\mathbf{c}\right)\left(\mathbf{a}^T\mathbf{c}\right) \leq 1 - \left(\mathbf{a}^T\mathbf{b}\right)^2$$

Rearranging the terms, adding $\left(\mathbf{b}^T\mathbf{c}\right)^2\left(\mathbf{a}^T\mathbf{c}\right)^2$ on both sides and factorizing yields:

$$\begin{aligned} \left(\left(\mathbf{b}^T\mathbf{c}\right)\left(\mathbf{a}^T\mathbf{c}\right) - \mathbf{a}^T\mathbf{b}\right)^2 &\leq \left(1 - \left(\mathbf{a}^T\mathbf{c}\right)^2\right)\left(1 - \left(\mathbf{b}^T\mathbf{c}\right)^2\right) \\ \Rightarrow \qquad \left(\mathbf{b}^T\mathbf{c}\right)\left(\mathbf{a}^T\mathbf{c}\right) - \mathbf{a}^T\mathbf{b} &\leq \left|\left(\mathbf{b}^T\mathbf{c}\right)\left(\mathbf{a}^T\mathbf{c}\right) - \mathbf{a}^T\mathbf{b}\right| \\ &\leq \sqrt{\left|\left(1 - (\mathbf{a}^T\mathbf{c})^2\right)\left(1 - (\mathbf{b}^T\mathbf{c})^2\right)\right|} \end{aligned}$$

Obviously the bracketed terms below the square root are both nonnegative and thus the absolute value can be left away and the square root can be split. However, bounding the first term in the latter inequation by its absolute value restricts equality to the case that $\mathbf{c}$ is on the shorter of the two arcs that form the great circle containing $\mathbf{a}$ and $\mathbf{b}$.
With $\cos(\measuredangle(\mathbf{x}, \mathbf{y})) = \mathbf{x}^T\mathbf{y}$ and $\sin(\measuredangle(\mathbf{x}, \mathbf{y})) = \sqrt{1 - (\mathbf{x}^T\mathbf{y})^2}$ for $\mathbf{x}, \mathbf{y} \in S^{d-1}$ we get:

$$\begin{aligned} &\cos\left(\measuredangle\left(\mathbf{a}, \mathbf{c}\right)\right)\cos\left(\measuredangle\left(\mathbf{c}, \mathbf{b}\right)\right) - \sin\left(\measuredangle\left(\mathbf{a}, \mathbf{c}\right)\right)\sin\left(\measuredangle\left(\mathbf{c}, \mathbf{b}\right)\right) \\ &\quad = \cos\left(\measuredangle\left(\mathbf{a}, \mathbf{c}\right) + \measuredangle\left(\mathbf{c}, \mathbf{b}\right)\right) \leq \cos\left(\measuredangle\left(\mathbf{a}, \mathbf{b}\right)\right) \end{aligned}$$

By applying the arc cosine and flipping the inequality sign because the arc cosine is strictly monotonically decreasing on $[-1, 1]$ we get the triangle equation for spherical triangles.

□

Appendix B

Another Minimization Problem on the SO(3)

In chapter 3 the minimization problem

$$\underset{\mathbf{R}\in\mathrm{SO}(3)}{\arg\min} \max_{i\in\{1,\ldots,n\}} \|\mathbf{R}_i - \mathbf{R}\|_2$$

for a set of rotation matrices $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathrm{SO}(3)$ and in chapter 4 a minimization problem similar to

$$\underset{\mathbf{R}_S,\mathbf{R}_T\in\mathrm{SO}(3)}{\arg\min} \sum_{i=1}^{n} \left(\delta\left(\mathbf{R}_S\mathbf{R}_{A_i}, \mathbf{R}_{B_i}\mathbf{R}_T\right)\right)^2$$

for two sets $\mathbf{R}_{A_1}, \ldots, \mathbf{R}_{A_n} \in \mathrm{SO}(3)$ and $\mathbf{R}_{B_1}, \ldots, \mathbf{R}_{B_n} \in \mathrm{SO}(3)$ of corresponding rotation matrices has been solved (with δ the angle between two rotations, definition see section 3.1). In both cases this was done by translating the rotation matrices to rotation quaternions and in both cases much effort had to be taken to solve the problem that the corresponding rotation quaternions are ambiguous concerning a sign. Here we want to solve another minimization problem on the $\mathrm{SO}(3)$ again with the help of quaternions, but in this case the ambiguity concerning the sign does not matter. This minimization problem has been solved in the course of this thesis but was finally not applied in its scope.

Theorem B.1 *Given a set of rotations $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathrm{SO}(3)$ and the minimization problem*

$$\underset{R\in\mathrm{SO}(3)}{\arg\min} \max_{\mathbf{p}\in S^2} \sum_{i=1}^{n} |\mathbf{R}_i\mathbf{p} - \mathbf{R}\mathbf{p}|^2 \ . \tag{B.1}$$

with $S^{(n-1)} := \{\mathbf{v} \in \mathbb{R}^n | \, |\mathbf{v}| = 1\}$ and $|.|$ the Euclidean norm.
Furthermore let $\mathfrak{Q}^*$ be the group of unit quaternions represented by 4-dimensional vectors with ρ

$$\rho : \mathfrak{Q}^* \to \mathrm{SO}(3) \quad ,$$

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \mapsto \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3-2q_0q_1 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} \tag{B.2}$$

the mapping of unit quaternions to rotation matrices as already presented at the end of section 3.1. Then the minimization Problem (B.1) is solved by $\mathbf{R} = \rho(\mathfrak{q})$ *with the union quaternion* $\mathfrak{q}$ *which is - considered as a 4-dimensional vector - an eigenvector for the largest eigenvalue of the symmetric matrix*

$$\mathbf{\Sigma} := \sum_{i=1}^{n} \mathfrak{q}_i \mathfrak{q}_i^T \tag{B.3}$$

with $\mathbf{R}_i = \rho(\mathfrak{q}_i)$, $1 \leq i \leq n$, *and* $\mathfrak{q}_i$ *considered as 4-dimensional vector in (B.3). The solution is unique iff the eigenspace for the largest eigenvalue of* $\mathbf{\Sigma}$ *has dimension 1 and it is*

$$\min_{R \in \mathrm{SO}(3)} \max_{\mathbf{p} \in S^2} \sum_{i=1}^{n} |\mathbf{R}_i\mathbf{p} - \mathbf{R}\mathbf{p}|^2 = 4(n - \sigma_0 - \sigma_3)$$

with σ_0 *and* σ_3 *the smallest and largest eigenvalue of* $\mathbf{\Sigma}$.

Proof:

ρ is only an epimorphism because of $\rho(\mathfrak{q}) = \rho(-\mathfrak{q})$. However, obviously this ambiguity concerning a sign does not matter in the definition of $\mathbf{\Sigma}$.

For the expression in (B.1) to minimize we can conclude because rotations are isometries:

$$S_i \;:=\; |\mathbf{R}_i\mathbf{p} - \mathbf{R}\mathbf{p}|^2 = \left|\mathbf{p} - \mathbf{R}_i^T\mathbf{R}\mathbf{p}\right|^2$$

Let $\hat{\boldsymbol{\varphi}}_i \in S^2$ be the axis and $\varphi_i \in [0, \pi]$ the angle of the rotation $\mathbf{R}_i^T\mathbf{R}$ for $\mathbf{R} \in \mathrm{SO}(3)$ arbitrary, but fixed.Furthermore let $\alpha_i = \measuredangle(\hat{\boldsymbol{\varphi}}_i, \mathbf{p})$ be the angle between $\hat{\boldsymbol{\varphi}}_i$ and $\mathbf{p}$ and $\hat{\boldsymbol{\varphi}}_{i,\perp}$ the proper unit vector from the orthogonal complement of $\hat{\boldsymbol{\varphi}}_i$ so that $\mathbf{p}$ can be written as $\cos\alpha_i\hat{\boldsymbol{\varphi}}_i + \sin\alpha_i\hat{\boldsymbol{\varphi}}_{i,\perp}$. Since the to $\hat{\boldsymbol{\varphi}}_i$ parallel component of $\mathbf{p}$ is not affected by the rotation $\mathbf{R}_i\mathbf{R}$, we get for S_i

$$\begin{aligned} S_i &= \sin^2\alpha_i \left|\hat{\boldsymbol{\varphi}}_\perp - \mathbf{R}_i^T\mathbf{R}\hat{\boldsymbol{\varphi}}_\perp\right|^2 \\ &= \sin^2\alpha_i \left|\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} \cos\varphi_i & -\sin\varphi_i \\ \sin\varphi_i & \cos\varphi_i \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right|^2 \\ &= \sin^2\alpha_i\ (2 - 2\cos\varphi_i) = 4\ \sin^2\alpha_i\ \sin^2\frac{\varphi_i}{2} \\ &= 4\left(\sin^2\frac{\varphi_i}{2} - \cos^2\alpha_i\ \sin^2\frac{\varphi_i}{2}\right) \\ &= 4\left(1 - \cos^2\frac{\varphi_i}{2} - \cos^2\alpha_i\ \sin^2\frac{\varphi_i}{2}\right) \end{aligned} \tag{B.4}$$

At the end of section 3.1 the relation between the rotation angle and rotation axis of a rotation and the corresponding rotation quaternion has already been treated thoroughly. Furthermore it has been shown how the quaternion multiplication can be written by splitting a quaternion $\mathbf{q} = (q_0, \mathbf{q})^T$ in a "scalar part" $q_0 \in \mathbb{R}$ and a "vector part" $\mathbf{q} \in \mathbb{R}^3$.
Due to this knowledge it is clear that the rotation quaternion corresponding to the rotation matrix $\mathbf{R}_i^T\mathbf{R}$ has the form $(\pm\cos(\varphi_i/2), \pm\sin(\varphi_i/2)\hat{\boldsymbol{\varphi}})^T$. As it can be checked by (B.2), transposing on the SO(3) side means flipping the vector part on the quaternion side, which is called quaternion conjugation and usually denoted by a star, just like the conjugation of complex numbers. Thus, if $\mathbf{q_R} \in \mathfrak{Q}^*$ is a proper rotation quaternion with $\rho(\mathbf{q}_R) = \mathbf{R}$, $(\pm\cos(\varphi_i/2), \pm\sin(\varphi_i/2)\hat{\boldsymbol{\varphi}})^T$ can also be written as $\mathbf{q}_i^*\mathbf{q_R}$. The middle term in (B.4) is just the squared scalar component of this quaternion product and the right term the squared scalar component of

$$\mathbf{q}_i^*\mathbf{q_R}\begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \pm\cos\frac{\varphi_i}{2} \\ \pm\sin\frac{\varphi_i}{2}\hat{\boldsymbol{\varphi}}_i \end{pmatrix}\begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix}$$

$$= \begin{pmatrix} \mp\sin\frac{\varphi_i}{2}\,\hat{\boldsymbol{\varphi}}_i^T\mathbf{p} \\ \pm\cos\frac{\varphi_i}{2}\mathbf{p} \pm \sin\frac{\varphi_i}{2}\hat{\boldsymbol{\varphi}}_i \times \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mp\sin\frac{\varphi_i}{2}\cos\alpha_i \\ \pm\cos\frac{\varphi_i}{2}\mathbf{p} \pm \sin\frac{\varphi_i}{2}\hat{\boldsymbol{\varphi}}_i \times \mathbf{p} \end{pmatrix}.$$

It can be easily verified that unit quaternions can be embedded into 4-dimensional rotation matrices by the monomorphism

$$\tau : \mathfrak{Q}^* \to \mathrm{SO}(4) \quad , \quad \begin{pmatrix} q_0 \\ \mathbf{q} \end{pmatrix} \mapsto \begin{pmatrix} q_0 & -\mathbf{q}^T \\ \mathbf{q} & q_0\mathbb{1} + \mathbf{q}^\times \end{pmatrix}$$

$$\text{with} \quad \mathbf{q}^\times = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}^\times = \begin{pmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{pmatrix}.$$

The scalar quaternion component just appears as upper left matrix entry and conjugating on the quaternion side means transposing on the $\mathrm{SO}(4)$ side. With $\mathbf{e}_0 := (1,0,0,0)^T$, $\mathbf{Q}_i = \tau(\mathbf{q}_i)$, $\mathbf{Q_R} = \tau(\mathbf{q_R})$, $\mathbf{P} = \tau(0,\mathbf{p})$ we can write S_i by using the associativity of matrix multiplication as

$$\begin{aligned} S_i &= 4\left[1 - \left(\mathbf{e}_0^T\mathbf{Q}_i^T\mathbf{Q_R}\mathbf{e}_0\right)^2 - \left(\mathbf{e}_0^T\mathbf{Q}_i^T\mathbf{Q_R}\mathbf{P}\mathbf{e}_0\right)^2\right] \\ &= 4\left[1 - \left(\mathbf{q}_i^T\mathbf{q_R}\right)^2 - \left(\mathbf{q}_i^T\mathbf{Q_R}\begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix}\right)^2\right] \\ &= 4\left[1 - \mathbf{q_R}^T\mathbf{q}_i\mathbf{q}_i^T\mathbf{q_R} - \begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix}^T\mathbf{Q_R}^T\mathbf{q}_i\mathbf{q}_i^T\mathbf{Q_R}\begin{pmatrix} 0 \\ \mathbf{p} \end{pmatrix}\right] \end{aligned}$$

with the quaternions considered now as 4-dimensional vectors.
Thus we get for the original minimization problem:

$$\underset{R\in\mathrm{SO}(3)}{\arg\min}\max_{\mathbf{p}\in S^2}\sum_{i=1}^{n}\left|\mathbf{R}_i\mathbf{p} - \mathbf{R}\mathbf{p}\right|^2$$

$$
\begin{aligned}
&= \underset{R\in\mathrm{SO}(3)}{\arg\min}\max_{\mathbf{p}\in S^2}\sum_{i=1}^{n} 4\left[1-\mathbf{q}_\mathbf{R}^T\mathbf{q}_i\mathbf{q}_i^T\mathbf{q}_\mathbf{R}\right.\\
&\qquad\qquad\left.-\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}^T\mathbf{Q}_\mathbf{R}^T\mathbf{q}_i\mathbf{q}_i^T\mathbf{Q}_\mathbf{R}\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}\right]\\
&= \underset{R\in\mathrm{SO}(3)}{\arg\min}\max_{\mathbf{p}\in S^2} 4\left[n-\mathbf{q}_\mathbf{R}^T\mathbf{\Sigma}\mathbf{q}_\mathbf{R}-\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}^T\mathbf{Q}_\mathbf{R}^T\mathbf{\Sigma}\mathbf{Q}_\mathbf{R}\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}\right]\\
&= 4n-4\underset{R\in\mathrm{SO}(3)}{\arg\max}\left[\underbrace{\mathbf{q}_\mathbf{R}^T\mathbf{\Sigma}\mathbf{q}_\mathbf{R}}_{=:A(\mathbf{R})}+\min_{\mathbf{p}\in S^2}\underbrace{\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}^T\mathbf{Q}_\mathbf{R}^T\mathbf{\Sigma}\mathbf{Q}_\mathbf{R}\begin{pmatrix}0\\ \mathbf{p}\end{pmatrix}}_{=:B(\mathbf{R},\mathbf{p})}\right]
\end{aligned}
\tag{B.5}
$$

Because $\mathbf{\Sigma}$ is symmetric and positive semi-definite, it can be diagonalized with four eigenvalues $0 \le \sigma_0 \le \sigma_1 \le \sigma_2 \le \sigma_3$ and four corresponding orthonormal eigenvectors $\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$. A and B in (B.5) are of the form $\mathbf{v}^T\mathbf{\Sigma}\mathbf{v}$, $\mathbf{v} \in S^3$, which can be bounded by $\sigma_0 \le \mathbf{v}^T\mathbf{\Sigma}\mathbf{v} \le \sigma_3$.
One can easily verify that $\mathbf{q}_\mathbf{R}$, $\mathbf{Q}_\mathbf{R}(0,1,0,0)^T$, $\mathbf{Q}_\mathbf{R}(0,0,1,0)^T$, $\mathbf{Q}_\mathbf{R}(0,0,0,1)^T$ is an orthonormal basis of $\mathbb{R}^4$. Thus by properly choosing $\mathbf{p} \in \mathbb{R}^3$ $\mathbf{Q}_\mathbf{R}(0,\mathbf{p})^T$ can be any unit vector in the orthogonal complement of $\mathbf{q}_\mathbf{R}$. Obviously we can write $\mathbf{q}_\mathbf{R}$ as a linear combination $\sin\beta\,\mathbf{u}_0 + \cos\beta\,\mathbf{u}_{0,\perp}$, $\beta \in [0, 2\pi)$, of $\mathbf{u}_0$ and a proper unit vector $\mathbf{u}_{0,\perp}$ from the orthogonal complement of $\mathbf{u}_0$. By choosing $\mathbf{p}$ so that $\mathbf{Q}(0,\mathbf{p})^T = \cos\beta\,\mathbf{u}_0 - \sin\beta\,\mathbf{u}_{0,\perp}$, which is obviously orthogonal to $\mathbf{q}_\mathbf{R}$, we get the following bound:

$$
\begin{aligned}
A(\mathbf{R})+\min_{\mathbf{p}\in S^2}B(\mathbf{R},\mathbf{p}) \;\le\;& \left(\sin\beta\,\mathbf{u}_0+\cos\beta\,\mathbf{u}_{0,\perp}\right)^T\mathbf{\Sigma}\left(\sin\beta\,\mathbf{u}_0+\cos\beta\,\mathbf{u}_{0,\perp}\right)\\
&+\left(\cos\beta\,\mathbf{u}_0-\sin\beta\,\mathbf{u}_{0,\perp}\right)^T\mathbf{\Sigma}\left(\cos\beta\,\mathbf{u}_0-\sin\beta\,\mathbf{u}_{0,\perp}\right)\\
=\;& \sigma_0+\mathbf{u}_{0,\perp}^T\mathbf{\Sigma}\mathbf{u}_{0,\perp}\le\sigma_0+\sigma_3
\end{aligned}
\tag{B.6}
$$

However, this bound is reached for $\mathbf{q}_\mathbf{R} = \mathbf{u}_3$, which means $\beta = 0$ and $\mathbf{u}_{0,\perp} = \mathbf{u}_3$ in (B.6). Then $\min_{\mathbf{p}\in S^2} B(\mathbf{R} = \rho(\mathbf{u}_3), \mathbf{p})$ is obviously minimized by $\mathbf{p}$ so that $\mathbf{Q}(0,\mathbf{p})^T$ is $\mathbf{u}_0$. If the dimension of the eigenspace for σ_3 is 1, i.e. $\sigma_2 < \sigma_3$, the unit eigenvector $\mathbf{u}_3$ for σ_3 is unique except a sign and therefore $\mathbf{R} = \rho(\mathbf{u}_3)$ is unique. If $\mathbf{q}_\mathbf{R}$ was not $\mathbf{u}_3$ in this case, it would be $\mathbf{u}_{0,\perp} \neq \mathbf{u}_3$ and thus the inequation in (B.6) would be strict so that the optimal bound would not be reached.

□

Bibliography

[1] D. Will, "Normgerechte Volumenbestimmung von Kofferräumen im deutschen Automobilbau mit Hilfe von Kontaktsimulation," *Diploma thesis, Johannes Gutenberg-Universität, Mainz*, 2009.

[2] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[3] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[4] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[5] E. A. Dinic, "An algorithm for the solution of the max-flow problem with the polynomial estimation," *Doklady Akademii Nauk SSSR*, vol. 194, no. 4, pp. 1277–1280, 1970.

[6] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *Journal of the ACM (JACM)*, vol. 35, no. 4, pp. 921–940, 1988.

[7] D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," *Journal of computer and system sciences*, vol. 26, no. 3, pp. 362–391, 1983.

[8] S. Even and R. E. Tarjan, "Network flow and testing graph connectivity," *SIAM journal on computing*, vol. 4, no. 4, pp. 507–518, 1975.

[9] Y. Shiloach and U. Vishkin, "An $O(n^2 \log n)$ parallel max-flow algorithm," *Journal of Algorithms*, vol. 3, no. 2, pp. 128–146, 1982.

[10] A. V. Goldberg and R. E. Tarjan, "A parallel algorithm for finding a blocking flow in an acyclic network," *Information Processing Letters*, vol. 31, no. 5, pp. 265–271, 1989.

[11] U. Vishkin, "A parallel blocking flow algorithm for acyclic networks," *Journal of Algorithms*, vol. 13, no. 3, pp. 489–501, 1992.

[12] A. V. Karzanov, "Determining the maximal flow in a network by the method of preflows," *Doklady Akademii Nauk SSSR*, vol. 215, no. 1, pp. 49–52, 1974.

[13] "DIN 70020, Teil 1, Straßenfahrzeuge; Kraftfahrzeugbau; Begriffe von Abmessungen," tech. rep., Deutsches Institut für Normung e.V. DIN 70020, February 1993.

[14] "SAE J1100, Motor Vehicle Dimensions," tech. rep., Society of Automotive Engineers, February 2001.

[15] F. Eisenbrand, S. Funke, J. Reichel, and E. Schömer, "Packing a trunk," in *Algorithms-ESA 2003*, pp. 618–629, Springer, 2003.

[16] J. Reichel, *Combinatorial approaches for the Trunk packing problem*. PhD thesis, Universität des Saarlandes, Saarbrücken, 2007.

[17] E. Shellshear, J. S. Carlson, and R. Bohlin, "A Combinatorial Packing Algorithm and Standard Trunk Geometry for ISO Luggage Packing," in *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. 307–313, American Society of Mechanical Engineers, 2012.

[18] F. Eisenbrand, S. Funke, A. Karrenbauer, J. Reichel, and E. Schömer, "Packing a trunk: now with a twist!," in *Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pp. 197–206, ACM, 2005.

[19] A. Karrenbauer, "Packing boxes with arbitrary rotations," *Master's thesis, Universität des Saarlandes, Saarbrücken*, 2004.

[20] S. Tiwari, G. Fadel, and P. Fenyes, "A fast and efficient compact packing algorithm for SAE and ISO luggage packing problems," *Journal of Computing and Information Science in Engineering*, vol. 10, no. 2, p. 021010, 2010.

[21] U. Neumann, "Optimierungsverfahren zur normgerechten Volumenbestimmung von Kofferräumen im europäischen Automobilbau," *Master's thesis, Technische Universität Braunschweig*, 2006.

[22] T. Baumann, *Volumenpackungen nach SAE J1100*. PhD thesis, Johannes Gutenberg-Universität, Mainz, 2008.

[23] K. Werth, *Packing a trunk : a physically based approach with full motional freedom*. PhD thesis, Johannes Gutenberg-Universität, Mainz, 2013.

[24] S. Kirkpatrick, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[25] C. Tom and J. Judy, "The use of the voxmap Pointshell Method of Collision Detection in virtual assembly methods planning, 2001 ASME Design

Engineering Technical Conferenes and Computers and lnformation in Engineering Conferenee Vol. 2," in *27th Design Automation Conference*, vol. 2, pp. 1169–1172, 2000.

[26] S. A. Cameron and R. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3, pp. 591–596, IEEE, 1986.

[27] S. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra," in *ICRA*, vol. 97, pp. 20–25, 1997.

[28] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *Game developers conference*, vol. 170, 2001.

[29] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New results and new trends in computer science*, pp. 359–370, Springer, 1991.

[30] M. Hanke-Bourgeois, *Grundlagen der numerischen Mathematik und des wissenschaftlichen Rechnens*. Springer, 2009.

[31] S. Gottschalk, "Separating axis theorem," tech. rep., Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[32] C. Spielberger and M. Held, "Inner Approximation of Polygons and Polyhedra by Unions of Boxes," in *Twenty-second European Workshop on Computational Geometry Delphi, Greece March 27–29, 2006*, p. 189, 2006.

[33] M. MacKenna, J. O'Rourke, and S. Suri, *Finding the largest rectangle in an orthogonal polygon*. Johns Hopkins University. Electrical Engineering and Computer Science Department, 1985.

[34] A. Aggarwal and S. Suri, "Fast algorithms for computing the largest empty rectangle," in *Proceedings of the third annual symposium on Computational geometry*, pp. 278–290, ACM, 1987.

[35] "Programming languages – C++," ISO INCITS/ISO/IEC 14882-2012, American National Standards Institute, 2012.

[36] J. D. Valois, "Lock-free linked lists using compare-and-swap," in *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, pp. 214–222, ACM, 1995.

[37] "PQP – A Proximity Query Package." http://gamma.cs.unc.edu/SSV/, 1999.

[38] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[39] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, ACM, 1996.

[40] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," tech. rep., Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

[41] K. Abdel-Malek, J. Yang, D. Blackmore, and K. JOY, "Swept volumes: fundation, perspectives, and applications," *International Journal of Shape Modeling*, vol. 12, no. 01, pp. 87–127, 2006.

[42] O. Faugeras, *Three-dimensional computer vision: a geometric viewpoint*. MIT press, 1993.

[43] G. Strang, *Introduction to linear algebra*. Wellesley-Cambridge Press, 2011.

[44] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU Press, 2012.

[45] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[46] B. K. Horn, "Closed-form solution of absolute orientation using unit quaternions," *JOSA A*, vol. 4, no. 4, pp. 629–642, 1987.

[47] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 5, pp. 698–700, 1987.

[48] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Robotics-DL tentative*, pp. 586–606, International Society for Optics and Photonics, 1992.

[49] J. v. Neumann, "Some matrix-inequalities and metrization of matrix-space," *Tomsk Univ. Rev. 1*, pp. 286–300, 1937.

[50] L. Mirsky, "A trace inequality of John von Neumann," *Monatshefte für Mathematik*, vol. 79, no. 4, pp. 303–306, 1975.

[51] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[52] M. Shah, R. D. Eastman, and T. Hong, "An overview of robot-sensor calibration methods for evaluation of perception systems," in *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, pp. 15–20, ACM, 2012.

[53] H. Zhuang, Z. S. Roth, and R. Sudhakar, "Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form AX= YB," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 4, pp. 549–554, 1994.

[54] F. Dornaika and R. Horaud, "Simultaneous robot-world and hand-eye calibration," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 4, pp. 617–622, 1998.

[55] A. Li, L. Wang, and D. Wu, "Simultaneous robot-world and hand-eye calibration using dual-quaternions and Kronecker product," *Inter. J. Phys. Sci*, vol. 5, no. 10, pp. 1530–1536, 2010.

[56] F. C. Park, "Distance metrics on the rigid-body motions with applications to mechanism design," *Journal of Mechanical Design*, vol. 117, no. 1, pp. 48–54, 1995.

[57] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.