Packing a trunk: a physically based approach with full motional freedom

Dissertation
zur Erlangung des Grades
"Doktor der Naturwissenschaften"

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

vorgelegt von
Kai Werth
geboren in Illingen/Saar

Mainz, February 4, 2013

Für meinen Vater

Abstract

Geometric packing problems may be formulated mathematically as constrained optimization problems. But finding a good solution is a challenging task. The more complicated the geometry of the container or the objects to be packed, the more complex the non-penetration constraints become.

In this work we propose the use of a physics engine that simulates a system of colliding rigid bodies. It is a tool to resolve interpenetration conflicts and to optimize configurations locally. We develop an efficient and easy-to-implement physics engine that is specialized for collision detection and contact handling.

In succession of the development of this engine a number of novel algorithms for distance calculation and intersection volume were designed and implemented, which are presented in this work. They are highly specialized to provide fast responses for cuboids and triangles as input geometry whereas the concepts they are based on can easily be extended to other convex shapes. Especially noteworthy in this context is our ε-distance algorithm - a novel application that is not only very robust and fast but also compact in its implementation. Several state-of-the-art third party implementations are being presented and we show that our implementations beat them in runtime and robustness.

The packing algorithm that lies on top of the physics engine is a Monte Carlo based approach implemented for packing cuboids into a container described by a triangle soup. We give an implementation for the SAE J1100 variant of the trunk packing problem. We compare this implementation to several established approaches and we show that it gives better results in faster time than these existing implementations.

Zusammenfassung

Geometrische Packprobleme lassen sich mathematisch formulieren als Optimierungsprobleme mit Nebenbedingungen. Jedoch das Finden einer guten Lösung ist eine große Herausforderung. Je komplizierter die Geometrie des Containers oder der zu packenden Objekte ist, umso komplexer werden die Zwangsbedingungen für Überschneidungsfreiheit.

In dieser Arbeit schlagen wir die Verwendung einer Physics-Engine vor, die ein System von kollidierenden Starrkörpern simuliert. Dieses Werkzeug löst Überschneidungen auf und optimiert eine Konfiguration lokal. Wir entwickeln eine effiziente und leicht nachzubauende Physics-Engine welche spezialisiert ist auf Kollisionserkennung und Kollisionsbehandlung.

Infolge der Entwicklung dieser Engine wurden eine Reihe neuartiger Algorithmen zur Berechnung von euklidischem Abstand und Schnittvolumen entworfen und implementiert, welche in dieser Arbeit präsentiert werden. Sie sind hochspezialisiert um schnelle Antwortzeiten für Quader und Dreiecke zu liefern, wobei sich die verwendeten Konzepts ohne Weiteres auch für andere konvexe Objekte erweitern lassen. Erwähnenswert in diesem Kontext ist unser ε-Abstandsalgorithmus, ein neuartiges Werkzeug, welches sowohl sehr robust arbeitet als auch kompakt zu implementieren ist. Wir präsentieren einige anerkannte Drittimplementierungen und zeigen auf, daß unsere Implementierungen diese schlagen in Laufzeit und Robustheit.

Der Packalgorithmus welcher auf der Physics-Engine aufsetzt, ist ein Monte Carlo-basierter Ansatz für Quader in einem Container, der durch Dreiecke beschrieben ist. Wir beschreiben eine Implementierung für die SAE-J1100 Variante des Kofferraumpackproblems. Wir vergleichen diese Implementierung mit einer Reihe bereits bestehender Ansätze und zeigen auf, daß sie bessere Resultate in kürzerer Zeit liefert als diese existierenden Ansätze.

Danksagung

Ich danke meinem Betreuer für das grenzenlose Vertrauen, das in mich gesetzt und die Unterstützung die mir die letzten Jahre zuteil wurde. Einen besseren Chef kann man sich nicht wünschen.

Ebenso danke ich meinem Zweitgutachter für die Zweitbegutachtung meiner Arbeit, aber auch für die zahlreichen Fahrten zwischen dem Saarland und Mainz bei kompetenter Diskussion über Informatik und Jazz.

Meinen Kollegen danke ich besonders für ihre großartige Freundschaft, die weit über den wissenschaftlichen Konsens hinausgeht.

Meinen langjährigen Mitstreitern am Projekt danke ich für die Hilfe und die Zusammenarbeit.

Vor allem danke ich meiner Familie und meiner Freundin, die immer für mich da sind und mir das Leben lebenswert machen.

Contents

1 Introduction

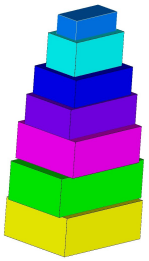
Geometric packing problems and contact simulation of rigid bodies are two fields of interest in Computer Science that are not usually considered to belong together. Nevertheless, allowing the items and the container to behave in a mechanical manner seems natural and would enable us to explore the solution space continuously, which means the items could translate and rotate freely within the container. At the same time, the rigid body behavior makes sure that the non-penetration constraints are fulfilled. In this work, we will introduce a *streamlined* physics engine which resolves contacts but omits unneeded properties like friction or resilience.

This approach is an example of a new class of algorithms that combine stochastic optimization methods with physical simulation strategies, which are being developed at the University of Mainz. Other approaches for different problem instances are described in Will (2009) and Erbes et al. (2013).

In collaboration with a German car manufacturer we will instantiate our method on a packing problem, which is known as the SAE variant of the trunk packing problem. For the DIN variant, we refer to previous works ((Eisenbrand et al., 2007, 2003, 2005)).

Problem definition The trunk packing problem, as suggested in Cagan and Ding (2003) is defined as follows:

Given a specific automobile compartment and the SAE standard luggage set, the task is to pick a subset of the luggage pieces and determine their optimal locations inside the trunk. The goal is to maximize the total volume of the subset while minimize the intersections and protrusions of these components inside the trunk.



The SAE standard luggage set defines 8 item types A to H that correspond to three dimensional cuboids with different dimensions (compare table 1.1).

box type	max. occurrences	inch (″) l	w	h	mm l	w	h	Volume l	ft^3
A	4	24	19	9	610	483	229	67.47	2.375
B	4	18	13	6.5	457	330	165	24.88	0.880
C	2	26	16	9	660	406	229	61.36	2.167
D	2	21	18	8.5	533	457	216	52.61	1.859
E	2	15	9	8	381	229	203	17.71	0.625
F	2	21	14	7	533	356	178	33.78	1.191
H	20	12.8	6	4.5	325	152	114	5.63	0.200

Table 1.1: Allowed box types for the SAE J1100 trunk packing problem.

The only exception is type G, a stylized golf bag, which is usually left out. The shoe box sized H-boxes can be seen as optional filling material. In section 9.2 of SAE (2001) we read: *Place in random order as many as one Standard Luggage Set of Luggage into the Luggage Compartment, excluding H boxes. When the best load is obtained using the Standard Luggage Set, H-boxes may be added to arrive at the final load. Pieces from subsequent Standard Luggage Sets may be used when the previous set is placed in the Luggage Compartment.*

The trunk geometry is usually described by a triangle mesh, which is the tessellated output of a set of NURBS surfaces. The common standard in computer aided design (CAD) are NURBS surfaces and they have their origin in the automotive design. Note that the tessellated triangles are an approximation on the real geometry. The majority of industrial CAD systems produce tessellations that are not conservative and give no information about the approximation error that occurs. This puts the need of accurate penetration constraints into question.

1.1 Organization of this work

The work is structured as follows: This chapter contains the introduction and the related work. In chapter 2, we describe our physical modeling techniques used to fulfill the non-penetration constraints.

In chapter 3 we discuss the geometrical algorithms that have been treated as black boxes so far. This is a major part of this work. These algorithms include descriptions and necessary extensions to known intersection tests and penetration depth calculations as well as novel techniques to compute the Euclidean distance and minimal translational distance of primitive convex objects. In

addition to the geometric algorithms that were used for our physical resolver, we also describe novel algorithms, which we designed for other packing strategies like a very robust and efficient ε-distance computation for cuboids and triangles as well as a fast and accurate intersection volume calculation for cuboids. We give runtime results of our implementations on these techniques and compare them to a number of available state-of-the-art implementations.

Chapter 4 describes the stochastic optimization strategies that are layered on top of the contact resolver. It explains the Monte Carlo based scheme and the set of moves we use to produce high quality packings. In order to fulfill the SAE standard we use a two level method. In the first phase, we leave out the H boxes, and in the second phase we add them using free space detection.

Chapter 5 shows the experimental results of our work. It also contains a theoretic examination of the coverage of a feasible package compared to the continuous volume of a trunk. It finishes with several comparisons to other existing packing strategies to prove the advantage of our techniques compared to other works.

Chapter 6 gives the conclusion and further work.

1.2 Related work

Physics engines and motion simulation of rigid bodies are popular fields of research particularly for computer games. An impulse based approach - similar to the one presented here - is found in Mirtich and Canny (1995). Loads of software solutions for physics engines exist like Geometrictools (2006); ODE (2006); BULLET (2010) as well as hardware supported engines like PhysX NVIDIA (2009). In the last few years GPU programming and CUDA have been heavily used for physics simulation and most of the mentioned contributors have parallelized their algorithms. In contrast to our approach common engines provide much more true-to-life features like friction modeling or link structures. Intrinsically tied to rigid body simulation are methods for fast collision detection. The concepts have been existing for years (see Eberly (2001) and Agarwal et al. (2001)). They are mostly based on broad-and-narrow-phase concepts from Mirtich (1997), which heavily profit from hierarchically Bounding Volume Structures like OBB-Trees from Gottschalk et al. (1996), where the *Separating Axis Theorem* was mentioned first. Kinetic data structures have been introduced in Basch et al. (1997); Abam and de Berg (2005) for the same issue. According to van den Bergen (2001), little is published

on penetration depth methods. Plenty distance solutions are given in Eberly (2001). A definition of the *Minimum Translational Distance (MTD)* can be found in Cameron and Culley (1986) including an approach for convex polyhedra based on the *Minkowski Sum (MS)* computation. Another MS-based approach for fast penetration depth calculation is described in Kim et al. (2002), which specialized for physical based animation using graphics hardware.

Recent publications that deal with discrete approaches for the SAE variant are Cagan and Ding (2003); Althaus et al. (2007); Tiwari et al. (2010). As far as the author knows these are the only existing publications on this subject, apart from ours. What all three have in common is that they limit the rotations to multiples of 90 degrees. Cagan and Ding (2003) even state that rotations are less important than translations for this problem. Their work also states that the trunk packing problem is NP-hard because it is an *example of the general* 3*d layout problem*. They solve the problem with a variant of their *extended pattern search based algorithm* described in Cagan et al. (2002). The overall implementation is a MC based approach very close to simulated annealing (SA) with a carefully chosen start configuration, an objective function and a set of moves (called *extensions*). The annealing is realized by a decrease of the absolutes of the discrete translation moves over time, up to a given limit. The packable region is represented by an octree.

A similar approach is given in Tiwari et al. (2010) by an evolutionary algorithm that uses a set of mutation and offspring operations (called *genetic variation operators*) on a given configuration (called *chromosome*). Although a sequence coding scheme *With Full Rotational Freedom* is given, the experimental results only show results with *Orthogonal Orientations*. The packing algorithm repeatedly performs two steps that are strictly separated from each other; the *optimization step* performs the genetic variation operators on the permutation and orientation of the items. The *layout step* uses a greedy strategy to find a placement for each item in the packable region represented by a fine grid. The quality of the packing is then rated by an objective function and further variations are performed according to this rating. The separation between orientation and translation makes it possible to produce offspring from configuration pairs, which is a significant advantage of this approach compared to others.

As simple as the strategy seems, it gives good results in acceptable time. Unfortunately the mentioned work only gives one example of a medium sized trunk of approximately 13ft^3. To compare their results with others they take into account the coverage of the packing volume against the continuous vol-

ume. Other models must have also been tested, because the authors point out that for bigger trunk volumes the coverage increases - an observation that we can confirm with our results. With medial coverage of 69.74% for the $13ft^3$ model their results are within the bounds of ours, which lie between 70% and 75% for this class of models. In section 5.1.1 we determine that Tiwari et al.'s definition of *coverage* differs from ours, which leads to a more pessimistic interpretation than ours. A direct comparison was possible, because we had the chance to interchange geometric data with Tiwari et al., which will be presented in 5.1.1. To be able to perform this test, the data had to be converted into a *waterproof* triangle mesh. This requirement makes Tiwari et al.'s approach less applicable for practical use. The conversion was done by an in-house-tool from our group as a joint venture from a recent work (von Dziegielewski et al. (2012)) to produce adaptive high quality meshes based on a closed voxel representation, which is described in Will (2009). A second disadvantage of the approach is that it allows an overall overlap of $5mm$ coming from a grid representation of $5mm$ spacing. Our usual approach does not allow box/box overlaps and tries to minimize the penetration of box and boundary.

The algorithm described in Althaus et al. (2007), which was also developed in our group, was the state-of-art implementation for the SAE variant at the time it was published. As a novelty, it can handle continuous translations and its resolution bound is restricted only by the precision of floating point arithmetic. It works - similar to our approach - on a fine point cloud representation of the trunk geometry. To describe the possible placements of a box (also known as *free space detection*), a *discrete Minkowski Sum* of the point cloud and all box types in every orthogonal orientation is calculated. A possible placement for a new box is one of the boundary points of the Minkowski Sum, and every time a box is added, the Minkowski Sum has to be updated. Like this, boxes are added sequentially and all packings can be enumerated brute-force. The authors also show strategies to eliminate redundant placements, which decreases the complexity tremendously. Also different orderings for the enumeration are discussed that lead to faster responses of acceptable results. As this brute-force strategy is obviously exponential in runtime and memory consumption for a given number of points, it terminates only for small to medium trunks up to about $9ft^3$. For trunks up to approximately $15ft^3$, it terminates if the smallest box type H is left out. For trunks larger than $15ft^3$ it exceeds the memory and time resources of common computer systems. Advantages of the algorithm are it produces dense packings due to the continuous point of view and it is able to produce results for most models in acceptable time.

For the German DIN 70020 variant of the trunk packing problem, several approaches exist for both discrete[1] and continuous[2] representations of the configuration space (Eisenbrand et al. (2003, 2005, 2007); Reichel (2006a); Karrenbauer (2004); Neumann (2006); Rieskamp (2005)). In Tiwari et al. (2010), a strategy to pack trunks according to the DIN standard is given using the same evolutionary algorithm as for SAE but with slightly different genetic variation operators. A very recent article (Shellshear et al. (2012)) describes a simple and easy-to-implement greedy strategy to find a packing solution. It works on a voxel representation of the interior of the trunk, but - unlike all other known grid based approaches - it is not restricted to orthogonal orientation. Another work that is inspired by physical motion similar to our approach is Will (2009), where the DIN variant is solved by packing slices of the trunk differently (actually in parallel) by means of a two-dimensional contact solver.

To date, physics is of little relevance in optimization strategies, even though we find work on this subject. *Spring embedder algorithms* e.g. try to find optimal placements of edges and vertices in a graph by a force based model of repelling vertices where adjacent vertices are connected by springs. This defines a dynamic system and a minimum energy configuration corresponds to an optimal placement, which can be obtained by standard optimization strategies. The easiest strategy to solve this uses *gradient descent* (Kamada and Kawai (1989)). We find other approaches that use *randomized adaptive* strategy (Bruss and Frick (1996)) or *simulated annealing (SA)* (Fruchterman and Reingold (1991) - where spring embedded algorithms are mentioned first in the context of $3D$ graph drawing).

[1] The discrete approaches work on a cubical grid that approximates the usable inner space of the trunk and its result is a set of axis aligned boxes.

[2] The continuous approach exploits all degrees of freedom.

2 A physics engine for packing issues

This chapter gives an overview of the contact resolver and the physical models used. The main issue of the contact resolver or physics engine is to maintain the non-penetration constraints during the packing simulation. We are interested in having a *legal* packing in every iteration. Legal in this context means that no box penetrates any other box or the boundary of the container more than a desired threshold ε.

As we do not know in advance if a given packing is legalizable, a resolving scheme can only be heuristic. The scheme we herewith present works as follows: eliminate mutual conflicts between the boxes and the container in a pairwise fashion. If a box still experiences an overlap, repeat this step. As this step might be repeated very often, a fast pairwise collision handling is desirable. After detecting a collision it is resolved by means of the well known concepts of conservation of momentum and impulse interchange.

The geometric functions used to detect the intersection and to calculate the penetration depth will be presented in chapter 3. In this chapter we show how to use this geometric information to define and solve differential equations numerically. To avoid cyclic conditions during the simulation we also present an abort criterion, which is based on a statistic evaluation of the penetration depth during the resolving. We further illustrate the broad-and-narrow-phase concepts used to reduce the number of intersection and penetration computations.

2.1 Rigid body motion

In order to describe the configuration of a rigid body, we use the following notation. The position of the center of mass is denoted by a vector $\mathbf{c} \in \mathbb{R}^3$ and the orientation of the body is specified by a quaternion $\mathfrak{q} \in \mathbb{R}^4$. $\mathbf{v}, \omega \in \mathbb{R}^3$ stand for the linear and the angular velocities. The equations of rigid body

motion are given by

$$\frac{d\mathbf{c}}{dt} = \dot{\mathbf{c}} = \mathbf{v}, \qquad \frac{d\mathfrak{q}}{dt} = \dot{\mathfrak{q}} = \frac{1}{2}\omega \cdot \mathfrak{q}$$

Here the product $\omega \cdot \mathfrak{q}$ has to be read as a quaternion product. Using the explicit Euler scheme we can discretize the differential equations above.

$$\mathbf{c}(t + \Delta t) = \mathbf{c}(t) + \Delta t\, \mathbf{v}, \qquad \mathfrak{q}(t + \Delta t) = \mathfrak{q}(t) + \frac{1}{2}\Delta t\, \omega \cdot \mathfrak{q}(t) \tag{2.1}$$

The change of position during a time step of length Δt is denoted by $\Delta\mathbf{c} = \Delta t\mathbf{v}$ and the change of orientation by $\Delta\varphi = \Delta t\omega$. The mass of the rigid body is given by $m \in \mathbb{R}$ and its inertia is characterized by a matrix $\mathbf{I} \in \mathbb{R}^{3\times3}$ that depends on the orientation $\mathfrak{q}$ of the body. Let $\mathbf{R}(\mathfrak{q}) \in \mathbb{R}^{3\times3}$ be the rotation matrix corresponding to the unit quaternion $\mathfrak{q}$ with scalar component $q_0 \in \mathbb{R}$ and vector component $\mathbf{q} \in \mathbb{R}^3$.

$$\mathbf{I}(\mathfrak{q}) = \mathbf{R}(\mathfrak{q}) \cdot \mathbf{I}_0 \cdot \mathbf{R}(\mathfrak{q})^\intercal \quad \text{with} \quad \mathbf{R}(\mathfrak{q}) = (q_0^2 - \mathbf{q}^\intercal\mathbf{q})\mathbf{E} + 2\mathbf{q}\mathbf{q}^\intercal + 2q_0\mathbf{q}^\times$$

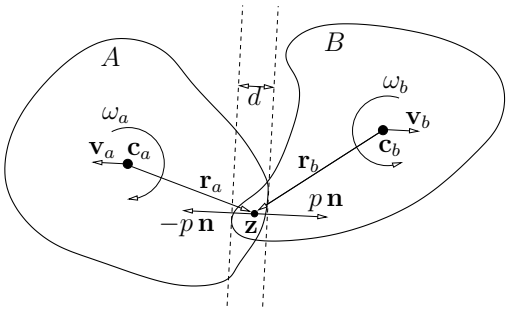
Here $\mathbf{I}_0$ is the inertia matrix of the body expressed in its local frame centered at $\mathbf{c}$. $\mathbf{E}$ denotes the identity matrix, $\mathbf{q}\mathbf{q}^\intercal$ the dyadic product and $\mathbf{q}^\times$ the skew symmetric matrix of vector $\mathbf{q}$.

2.2 Contact impulses

Consider two intersecting objects A and B (compare figure 2.1). Suppose that at time t the bodies have penetrated each other in direction $\mathbf{n}$ by distance d. we call the translation vector $\mathbf{n}$ from A to B the contact normal. The contact region itself is represented as a single point $\mathbf{z}$ within the geometric intersection $A \cap B$. The scalar value p is the impulse that is interchanged in point z between body A and B, and its value is chosen to separate A from B at time $t+\Delta t$ such that they touch tangentially. After this time the motion stops immediately. For this purpose, we examine the linear and angular velocities of the bodies caused by the impulse p.

Let us assume that the bodies are at rest at time t. The directed impulses $-p\,\mathbf{n}$ acting on A and $p\,\mathbf{n}$ acting on B instantaneously change the linear and angular velocities of the bodies from 0 to $\mathbf{v}_a, \mathbf{v}_b, \omega_a$, and ω_b.

If we linearly approximate the relative velocity of the bodies at the contact point $\mathbf{z}$ in the direction of the contact normal, we find the following relation



$$\begin{aligned}
\mathbf{v}_a &= -\frac{p}{m_a}\mathbf{n} \\
\mathbf{v}_b &= \frac{p}{m_b}\mathbf{n} \\
\omega_a &= -p\,\mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n}) \\
\omega_b &= p\,\mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})
\end{aligned}$$

Figure 2.1: Two penetrating objects interchange impulses

for the penetration depth, which is decreasing to zero during the time interval Δt.

$$\begin{aligned}
d &= \Delta t\,\mathbf{n}^\intercal(\mathbf{v}_b + \omega_b \times \mathbf{r}_b - \mathbf{v}_a - \omega_a \times \mathbf{r}_a) = \Delta t p M^{-1} \\
&\quad \text{with} \\
M^{-1} &= m_a^{-1} + m_b^{-1} + (\mathbf{r}_a \times \mathbf{n})^\intercal \mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n}) \\
&\quad + (\mathbf{r}_b \times \mathbf{n})^\intercal \mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})
\end{aligned} \tag{2.2}$$

Thus, the change of position $\Delta\mathbf{c} = \Delta t\,\mathbf{v}$ and the change of orientation $\Delta\varphi = \Delta t\omega$ necessary to resolve the penetration conflict of A and B can be computed in the following way

$$\begin{aligned}
\Delta\mathbf{c}_a &= -dMm_a^{-1}\mathbf{n} & \Delta\mathbf{c}_b &= dMm_b^{-1}\mathbf{n} \\
\Delta\varphi_a &= -dM\,\mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n}) & \Delta\varphi_b &= dM\,\mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})
\end{aligned} \tag{2.3}$$

As stated before we demand the movement of both bodies to stop immediately when they are separated - unlike a real physical motion. This avoids bouncing effects.

2.3 Contact resolving of boxes in a trunk

Consider a set of n boxes $B_1, \ldots, B_n$ in a polyhedral container like the trunk of a car. The boundary of the trunk may be described by a set of triangles or as a cloud of sufficiently dense points. Furthermore, consider that some of the boxes experience an overlap with other boxes or the container. The naive way to obtain a legal configuration is to detect all conflicts of a box B_i and resolve them one by one. This may not resolve all conflicts at once. However, it is very likely that the general overlap of all boxes is reduced after such a

resolving step, and that a finite number of resolving steps is sufficient to obtain a legal situation. Therefore, we distinguish between *box/box* collisions of two moving objects and *box/container* conflicts where only the boxes experience a motion.

A box B is described by its center of mass $\mathbf{c}$, its orientation quaternion $\mathfrak{q}$ and its three dimensional length vector $\mathbf{l} = (l_1, l_2, l_3)^\intercal$. The columns of the orientation matrix $\mathbf{R}(\mathfrak{q})$ correspond to the edge directions of B. For each B we also hold a translation vector $\Delta\mathbf{c} \in \mathbb{R}^3$ for the total translation at the end of one resolving step. The vector sum of all rotations is stored in a vector $\Delta\varphi \in \mathbb{R}^3$. We iteratively perform the resolving step until either the general overlap is less than a given threshold ε, or the computation is canceled because the configuration is not legalizable in a certain time interval. For the latter packing strategy, this can be an objective criterion to reject a move.

Geometric operations that are used in the sequel will be explained in chapter 3; at this point we assume they exist.

2.3.1 Box/box conflicts

We consider box/box conflicts first. Because the number of boxes for the SAE variant of the trunk packing problem rarely exceeds 20, a quadratic effort is justifiable. For the DIN case we can use a priority queue concept combined with a broad phase collision detection of bounding spheres that yields an amortized linear number of collisions per resolving step (comp. Mirtich (1997)).

For those boxes B_a, B_b that are in conflict we compute the translation $\Delta\mathbf{c}_a(\Delta\mathbf{c}_b)$ and rotation $\Delta\varphi_a(\Delta\varphi_b)$ from equations (2.3). As a central operator for resolving conflicts of boxes we use a function *penetration_ bb* that computes the penetration depth d, the contact normal $\mathbf{n}$ and a contact center $\mathbf{z} \in B_a \cap B_b$ for two intersecting boxes B_a and B_b. It will be described later in section 3.2.

Algorithm 1. RESOLVE BOXES
forall pairs (B_a, B_b) **with** $B_a \cap B_b \neq \emptyset$ **do**
$(d, \mathbf{n}, \mathbf{z}) =$ **penetration_bb** (B_a, B_b)
$\mathbf{r}_a = \mathbf{z} - \mathbf{c}_a,\ \mathbf{r}_b = \mathbf{z} - \mathbf{c}_b$
compute M **according to eqn** (2.2)
$\Delta\mathbf{c}_a \mathrel{-}= d\, M\, m_a^{-1}\mathbf{n}$
$\Delta\mathbf{c}_b \mathrel{+}= d\, M\, m_b^{-1}\mathbf{n}$
$\Delta\varphi_a \mathrel{-}= d\, M\, \mathbf{I}_a^{-1}(\mathbf{r}_a \times \mathbf{n})$
$\Delta\varphi_b \mathrel{+}= d\, M\, \mathbf{I}_b^{-1}(\mathbf{r}_b \times \mathbf{n})$

Figure 2.2 demonstrates a legalization involving seven boxes, which needed 27 resolving steps in our implementation.

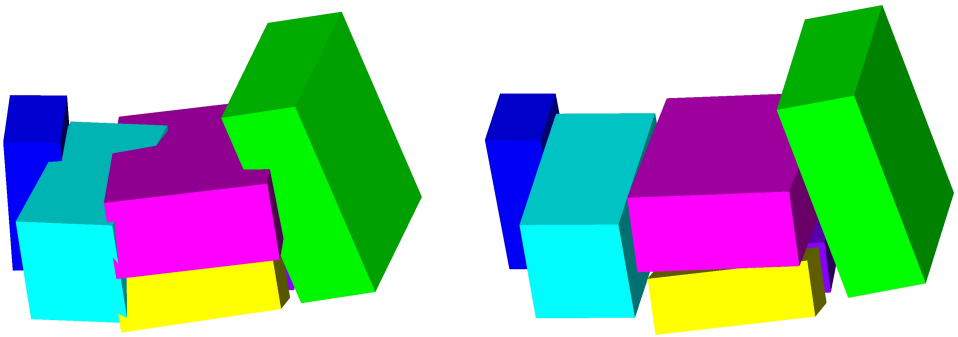


Figure 2.2: Seven boxes get resolved

2.3.2 Box/trunk conflicts

Our container typically consists of several thousand triangles. If we use a point cloud representation we also face some thousand points. In both cases, we use a *broad and narrow phase* concept from Mirtich (1997). In the point cloud case we use a Kd-Tree for the broad phase (Bentley (1975, 1990)). Otherwise, we insert the triangles into a space partition of spacing $s \times s \times s$, and we maintain a *triangle queue* $\mathcal{P}_i$ per box B_i where all triangles that overlap the same partition cell as B_i are ordered according to the distances to B_i. We hold a *mileage* β_i per box and every time $\beta_i > s$ we update $\mathcal{P}_i$. The eligible collision partners for B_i are taken from $\mathcal{P}_i$. This concept is similar to so called *kinetic data structures* introduced in Basch et al. (1997). For the space partition we refer to Reichel (2006a).

The function *penetration_tb* returns a triple $(d, \mathbf{n}, \mathbf{z})$ for a given box B_i and a triangle T_j. If the returned value d is negative then $-d$ is the distance of B_i and T_j. This fact is very useful to reinsert triangle T_j into $\mathcal{P}_i$ if $B_i \cap T_j = \emptyset$.

The computation for $\Delta\mathbf{c}_i$ and $\Delta\varphi_i$ is like the box/box case. The inverse value M for this contact involves only one box, hence we assume ${m_b}^{-1} = 0$.

2.3.3 One resolving step

For the resolving step itself we move a box B_a with center $\mathbf{c}_a$ and rotation quaternion $\mathfrak{q}_a$ as follows

forall (B_a) **do**
$\mathbf{c}_a \mathrel{+}= \Delta\mathbf{c}_a$
$\mathfrak{q}_a \mathrel{+}= \frac{1}{2}\Delta\varphi_a \cdot \mathfrak{q}_a$

Here we simply applied the discretized equations (2.1) of rigid body motion.

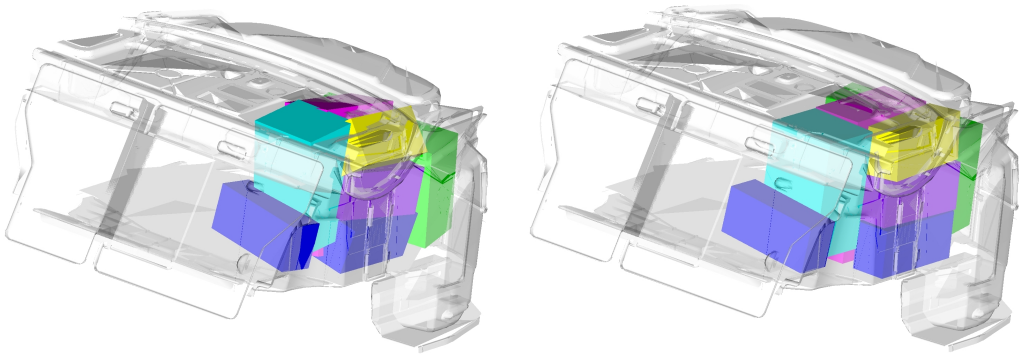


Figure 2.3: Seven boxes before and after legalizing

Figure 2.3 shows our box example from figure 2.2 this time inside a trunk described by 45456 triangles. Our implementation resolved this example in less than one second and performed more than 1000 single resolving steps.

2.4 Abort criterion

During the contact simulation, the items experience an expansion away from the center of all boxes. At the same time, the collisions with the triangles on the boundary of the container perform forces that push the boxes into the interior of the container - along the triangles' normals. This is a desired effect to obtain dense packings, but in some cases it may lead to cyclic states that are very unlikely to be resolved. The pairwise resolution that we use for performance reason is responsible for those cyclic states. But still, if the items do not fit, no strategy will be able to resolve the contacts. Therefore, we need a reliable abort criterion that tells us when to interrupt the legalization.

Especially during the packing simulation, when a box is added or altered in a way such that the current configuration experiences a high illegality, a fast

criterion for the legalizability is desired. Naive strategies would restrict the number of resolving steps or the amount of time allowed to perform these steps. The disadvantages of these strategies are obvious:

1. If cyclic states occur at an early point of the simulation, a high number of unneeded and expensive resolving steps are performed that are rejected in the end.

2. With a fixed maximum number of allowed resolving steps we might break up to early. It is imaginable - especially at a late point in the simulation - that a few more resolving steps would fit a box into a narrow area, which we would have discarded otherwise.

We could also forbid high forces such that situations that start with a high illegality are rejected. But this might always be the case when a box is added, even if this high illegality is resolved after a few steps. Hence a more general criterion would examine how the degree of illegality changes during time, and whether we experience a downhill or not. In the case that the overlap decreases, we would continue with the resolving continue. However if it increases we would stop and return `false`. But also a pure downhill strategy might break up too early because short term increases might lead to legal situations in longer term.

For the $2d$-packing strategy in Will (2009) a statistic evaluation of the *variation* of the sum of all pairwise penetration depths turned out to give a very good abort criterion. Here S_i denotes the sum of all penetration depths after the i's legalization step, and the variation V until the n's step is given as:

$$V_n \quad = \quad \sum_{i=1}^{n} |S_i - S_{i-1}|$$

The variation increases quickly if the penetration depth fluctuates a lot. Hence if we restrict it by a constant value, we avoid highly oscillating situations. Will also proposes to take the overall sum of the penetrations into account and to use the product of the sum over all penetrations and its variation. He further suggests to allow some time for a *setting phase* where high oscillations are tolerated in the first k steps. We use the sum $\sum_{i=1}^{k} S_i$ of the first k legalization trials to *normalize* the variation. For a given threshold $\mu \in \mathbb{R}$ the legalization

is aborted as soon as

$$\frac{\left(\frac{1}{k}\sum_{i=n-k}^{n} S_i\right) \cdot \sum_{i=k}^{n} |S_i - S_{i-1}|}{\left(\frac{1}{k}\sum_{i=1}^{k} S_i\right)^2} > \mu$$

The choice of the right μ is an empirical search. As for the $2d$ simulation, $\mu = 1.5$ was chosen. The $3d$ simulation presented in this work achieves the best result with $\mu = 0.3$. We work with $k = 8$ as in Will (2009), but we also interrupt the simulation immediately if the penetration sum in the setting phase grows over a given threshold $S_{\max}$, because this indicates a very bad start configuration that is very unlikely to be legalized. As a "safety net", we also restrict the maximum number of legalization steps by a value $k_{\max}$. This we might need in case of stable configurations where the penetration sum stays almost constant, but the packing cannot be legalized. This happens very rarely in practice. In our experiments, values $S_{\max} = 60.0$ and $k_{\max} = 1000$ turned out to be a good choice.

2.5 Broad and narrow phase collision detection

A box B_i can collide with either a subset of other boxes or a subset of triangles from the trunk (points from the trunk, respectively). Finding all pairs of colliding boxes (B_i, B_j) is a quadratic problem in the number of items (boxes) n. Finding all container collisions of one box is linear in the problem size of the container, respectively the number of triangles $|T|$ or points $|P|$ that describe the boundary and lies in $O(n \cdot |T|)$ or $O(n \cdot |P|)$ for all boxes.

Computational geometry knows a number of helping methods to detect at least subsets of all possible collision partners that are sufficiently close to B_i and that can be computed in nearly constant time. Finding these closest features is what we call the *broad phase*. In the *narrow phase* we take a closer look on the hopefully small subsets and use exact distance and penetration calculations as described later in chapter 3.

The terminology of broad and narrow phase has been introduced in Hubbard (1993) and tries to overcome the three so called *weaknesses* in naive rigid body motion:

fixed-timestep weakness Rigid body motion systems usually solve differential equation by discretization with a fixed timestamp Δt. An appropriate choice of Δt is crucial; if it is too low too many unnecessary discrete steps are performed and we waste runtime. Choosing Δt too high can cause the worst case scenario of *missing contacts*.

all-pairs weakness This specifies the mentioned problem of having a quadratic problem to detect all collision pairs.

pair-processing weakness The more complex the structure of the solids is, the harder it is to detect their collision.

In our case the pair-processing weakness for box pairs is negligible as we have efficient algorithms for this described in chapter 3. For box/container we could say that the all-pair weakness is of minor importance compared to the pair-processing weakness because the container consists of some thousand triangles (points). But if we change the point of view and regard each triangle of the container as a single object, then it is the pair-processing weakness (box/triangle) that we can neglect and the all-pair weakness dominates.

In Hubbard (1993) the broad phase uses *space time bounds* structures to overcome the first two weaknesses. For the narrow phase, the use of sphere trees is suggested to avoid the pair-processing weakness because of their invariance towards rotation.

For box/container collision tests we leave out the broad phase and only regard the narrow phase: Because the container is hardly moved or rotated we can use less sophisticated and therefore faster methods like a *space partition* to organize the triangles and a *Kd-Trees* for points. For box/box collisions the narrow phase consists of a *collision heap* that maintains a priority queue for box pairs sorted by a lower bound estimation on the first time the boxes will collide, based on the velocities of the boxes. This concept is very similar to the *space time bounds* structures from Hubbard (1993). The narrow phase for box/box conflicts exists in form of a simple comparison of the bounding spheres of the boxes.

In the following we describe the used structures like space partition, Kd-Tree, AABB and the collision heap.

Space partition A space partitioning is the subdivision of an Euclidean space into two or more disjoint subsets. Then for any point p it can be clearly decided in which region it lies.

A space partitioning is the maybe most straightforward concept to prune the majority of the $O(n^2)$ pairs in an n-body system and works for any representation of the objects. As easy it is as effective it proves and it is used nowadays heavily in time critical applications like games for ray shooting or collision detection.

Our space partition consists of a cubic axis aligned grid $\mathcal{G}$ that partitions the space into uni-sized cubic cells of length $s \times s \times s$ - the choice of s is an empirical value. Each cell c maintains a list of objects - triangles in our case - that have a non-empty intersection with c.

A query for a box B takes as argument B's AABB and returns a list of triangles that lie in the grid cells overlapped by B's AABB. Its runtime therefore is output sensitive but fast enough for our application. For explicit analysis on runtime and implementation for query, preprocessing and update mechanism we refer to Reichel (2006b).

Kd-Tree A Kd-tree is another space partitioning data structure very suitable for points whereas the partition cells are not restricted to a fixed size. It is a recursive structure organized hierarchically by a binary tree. Each leaf contains one or more points, and the union of all leafs corresponds to all points of the partitioning. Every node corresponds to a splitting of its leaves by a plane such that all leaves of the left subtree lie on the left side and those of the right tree lie on the right side. The orientation of the plane is chosen from one of the k dimension axes (in $\mathbb{R}^3$ its x, y and z-axis). Kd-tree stands for the k-dimensional tree and implies that this structure works for arbitrary dimensional points. Among the hierarchical search structures known Kd-trees are part of the *range tree* family.

The origin point of the splitting plane in a node often is a point from the set - not necessarily the median of its leaves - and each node usually contains exactly one point. In this case building up the $k = 1$ instance of a the Kd-tree would correspond to quicksort. The orientation usually alternates cyclicly from one level of the tree to the next. In $\mathbb{R}^3$ this means, the root splits at a x-aligned plane, its children both at a y-aligned plane, and so on (see figure (2.4)). By a x-aligned plane we denote a plane that is orthogonal to the x-axis, and the sideness test for that plane can be decided by simply checking the x-component of a given point. Under the condition that each plane origins in the median of its leaves, the tree is balanced and each leaf has about the same distance $\lceil \log n \rceil$ from the root. Building it up hence costs $O(n \log n)$ time. The leaves can be stored in an array in order of appearance

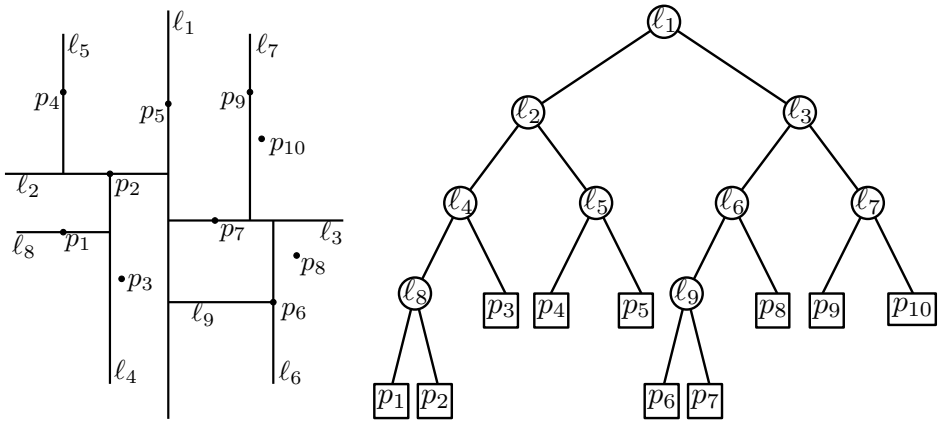


Figure 2.4: A set of points in $2d$ and its corresponding Kd-tree (Source: "Range searching and Kd-trees", Marc van Krefeld, Lecture notes 2011)

in the leaves such that the space consumption is $O(n)$.

For a given AABB B we can prune the points very easily that do not lie in B; We just climb down the tree and leave out the subtrees that are not cut by B, which only requires an inequality test between two `double` values. To collect points that lie in B efficiently, we can report whole subtrees that must lie in B. To achieve this we need to know what is the region ($region(\nu)$) of a node ν. On the path from the root to ν we visit a number of nodes, each one defining a splitting plane. The intersection of all planes from the root to ν is a (not necessarily finite) AABB, and this we call $region(\nu)$. If the region on a node ν is fully contained in B, we can report all leaves of ν (comp. figure (2.5)). In de Berg et al. (2000) the runtime for a range query is shown to be in $O(n^{1-\frac{1}{d}} + k)$ with dimension d and number of output points $k \leq n$.

For our implementation we changed the described structure a bit; first we do not cycle through the dimensions in alternation. Instead we split in the direction of longest dimension of the leaf points of a node ν. This gives more numerical stability for (quasi-)coplanar points. At the same time we can expect an acceptable range query runtime of $O(log^{(d-1)}n)$ as shown in Dickerson et al. (2000). Secondly, instead of having one point per leave, we organize a set of points per leaf and the region of a node is disregarded. We accept this in favor of a less deep tree and faster query responds.

The concept of Kd-trees can be extended to other primitive objects like trian-

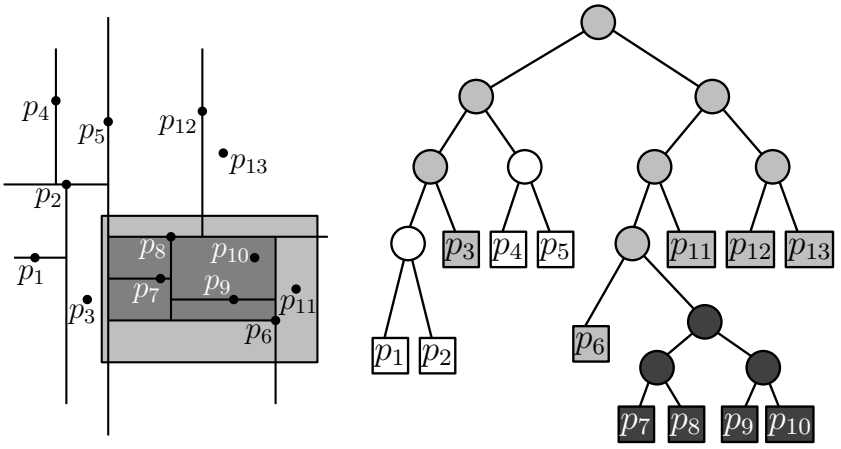


Figure 2.5: During the range search all subtrees that are fully contained are reported (black nodes and leaves). The regions of the grey nodes intersect the range, the grey leaves are fully contained. (Source: "Range searching and Kd-trees", Marc van Krefeld, Lecture notes 2011)

gles. The problem then is that a triangle may be intersecting more than one region. This violates the uniqueness of the representation. One possibility to overcome this is to split the triangle along the splitting plane into two triangles and insert them into either child's triangle list. But this technique holds numerical instabilities. The usual way would be to insert the triangle - or a representative like a pointer or an index to it - into the triangle lists of both children of the node that overestimates the split region. For big triangles it puts the use of a Kd-tree into question but for a large set of small triangles it may prove right.

AABB A very common, very easy-to-use and easy-to-implement concept - and at the same time very efficient - is the use of *axis aligned bounding boxes* to define a bounding volume for any given object. We only need to find the object's maximum values in every dimension, which is just a $\min - \max$ search for polyhedral bodies. The test for intersection of two AABBs consists of only three interval tests. AABBs can be organized in hierarchic structures like *AABB-trees* (van den Bergen (1998)). Apart from collision detection queries they are heavily used for ray shooting algorithms.

For rotating objects the use of AABBs might be a bottleneck of a simulation as it requires permanent recomputing of the AABB. This proves even worse

if AABB trees are used. In this case spheres or OBB are the better choice.

We mainly use AABBs to define a range to search in our Kd-tree or space partition. As broad phase to detect box/box collisions it turned out to be too inefficient because of the mentioned reasons.

Collision heap In Mirtich (1996b) the concept of collision heaps and conservative advancement is introduced for the IMPACT system, an impulse based rigid body simulation system similar to ours, which is better known as BULLET[1]. If we can define a maximum velocity $v_{\max}$ that is an upper bound on the velocities of all objects, and if we know the pairwise distances $d_{i,j}$ of all object pairs - or at least a lower bound like the distance of their bounding spheres - at the start point $t_0 = 0$, we can give a lower bound estimation on the first time they can collide by $\frac{d_{i,j}}{v_{\max}}$. In literature, the exact time of collision often is called *time of impact (TOI)*.

We sort the pairs into the collision heap according to the time estimation of TOI in ascending order and the key - also called time stamp - $t_{i,j} \in \mathbb{R}$ of the head element tells us that we can ignore all box/box collisions until we reach time step $\lfloor t_{i,j} \rfloor$. A time step is a discrete value in $\mathbb{N}$, and if we pop out the top element of the heap we perform the following:

Compute the distance $\tilde{d}_{i,j}$ of the *bounding spheres* of B_i and B_j.

if $\tilde{d}_{i,j} \leq 0$ remark the object pair for a narrow phase test in the following time step.

else sort the pair back into the collision heap according to this new and better TOI approximation $\frac{\tilde{d}_{i,j}}{v_{\max}}$.

Collision heaps prune a lot if the number of items is high enough and the maximal velocity $v_{\max}$ is well chosen. In the case that the number of items if small the costs of administration for the heap outbalances the benefit. If $v_{\max}$ is too high we test too many pairs per time step. On the other hand if $v_{\max}$ is lower than the real maximum we might miss collisions.

[1] BULLET is a state-of-the-art physics simulation system that nowadays is used by the majority of animation motion pictures.

3 Geometric operations and algorithms

This chapter presents a number of geometric algorithms that have been designed and implemented during the development of the packing algorithm. These algorithms include descriptions and necessary extensions to known intersection tests and penetration depth calculations as well as novel techniques to compute the Euclidean distance and minimal translational distance of primitive convex objects. In addition to geometric algorithms that were used for our physical resolver we also describe novel algorithms that we designed for other packing strategies like a very robust and efficient ε-distance computation for cuboids and triangles as well as a fast and accurate intersection volume calculation for cuboids.

The intersection volume algorithm was originally implemented to improve a simulated annealing approach described in Eisenbrand et al. (2005). With this implementation the evaluation of the objective function could be accelerated by a factor of 2. As this evaluation claims the biggest share on the runtime, this new implementation improved the whole simulation tremendously.

The development of an ε-distance algorithm is inspired from a second interpretation of box dimensions in the DIN 70100 standard; the edges are allowed to have a chamfer of radius $0.5mm$ and are no longer cuboids. The question if two chamfered boxes intersect corresponds to the detection if two cuboids are closer then $1mm$. An algorithm for the Euclidean distance is appropriate but if we only want to know if two cuboids are closer than a given ε we can include certain early-outs that accelerate the implementation a lot.

Structure of the chapter In 3.1 we introduce the basis of all our distance and penetration methods, the *separating axis theorem (SAT)*. In section 3.2 we describe the penetration depth computation for two intersecting boxes. In section 3.3 we follow by the pairwise distance algorithm for boxes. Section 3.4 presents the *minimal translational distance (MDT)* algorithm, which is a conclusion from section 3.2 and section 3.3. Section 3.5 will give a constructive

proof for the correctness of the MDT algorithm. In section 3.6 we present our novel ε-distance algorithm as the conclusion of the prior results. In section 3.7 we summarize our results and compare our ε-distance approach to existing implementations.

The last section 3.8 introduces a new fast approach to calculate the intersection volume of two boxes by only calculating the edges of the intersection. It also includes a complexity analysis and runtime comparison with existing state-of-the-art implementations.

3.1 Separating axis theorem (SAT)

Theorem 3.1.1 (Separating Axis Theorem (SAT)): *Given two convex objects A and B*

$$A \cap B = \emptyset \quad \Longleftrightarrow \quad \exists \text{ a plane that separates } A \text{ from } B$$

A proof for SAT can be found in (Gottschalk et al. (1996)). There we also learn, that for two convex polyhedra P_1 and P_2 it suffices to test a finite number of planes, namely those that are either

- parallel to one face f with $f \in \mathsf{P}_1$ or $f \in \mathsf{P}_2$ or
- parallel to two edges e_1, e_2 with $e_1 \in \mathsf{P}_1$ and $e_2 \in \mathsf{P}_2$.

If no such separating plane can be found, P_1 and P_2 have a non-empty intersection. The normal of a separating plane, if it exists, is called the *separating axis*. The *separating axis test* itself is done as follows: Project P_1 and P_2 on each eligible separating axis $\mathbf{n}$. If any two projections correspond to two disjoint line segments then $\mathbf{n}$ is a separating axis and P_1 and P_2 are disjoint. The separating axis theorem tells us that we only have to test a finite number of eligible axes namely the normals of all faces of P_1 and P_2 and the pairwise cross products of the edge directions (c.f. fig. (3.1)).

3.2 Penetration depth

Definition 3.2.1 (Penetration depth of two objects): *The* penetration depth *of two (non-disjoint) objects P_1 and P_2 is the minimal distance that one object has to be moved in order to nullify the intersection.*

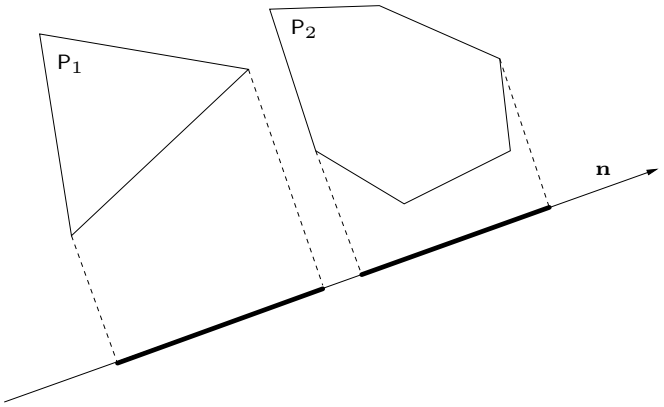


Figure 3.1: Two polygons in $\mathbb{R}^2$ and a separating axis.

For convex polyhedra the separating axis theorem can be used as follows: If P_1 and P_2 intersect we will not find any separating axis as all projections correspond to pairwise overlapping line segments. Then the minimum of all overlaps is the penetration (c.f. fig (3.2)).

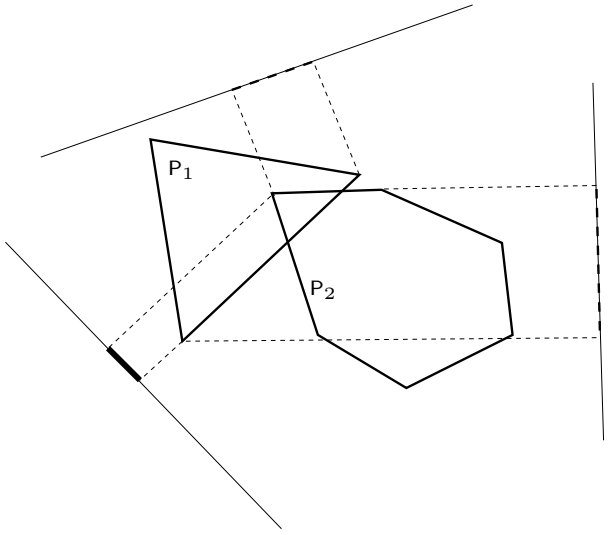


Figure 3.2: Two penetrating polygons in $\mathbb{R}^2$. We see 3 of the 9 possible projections including the one that gives us the penetration depth.

To compute the overlap we simply compare interval pairs and $\delta(I_1, I_2)$ gives us the penetration of two intervals I_1 and I_2:

Definition 3.2.2: *Let $I_1 = [i_0, i_1]$ and $I_2 = [j_0, j_1]$ be two (closed) intervals*

in $\mathbb{R}$. *The* penetration of I_1 and I_2 *is given as*

$$\delta(I_1, I_2) \quad = \quad \frac{i_1 - i_0}{2} + \frac{j_1 - j_0}{2} - \left| \frac{i_1 + i_0}{2} - \frac{j_1 + j_0}{2} \right|$$

The penetration of two intervals indicates the minimal shift to glue two intervals together. If $\delta(I_1, I_2) < 0$ the intervals are disjoint and $-\delta(I_1, I_2)$ equals the distance of I_1 and I_2.

For cuboids, the number of axes to be tested is small, and hence they are suitable for this method. Only 15 axes are possible ($2 \cdot 3$ for the faces of the boxes $+$ $3 \cdot 3$ pairwise cross products). We give a pseudo code for the penetration computation of two boxes here. The vector $\mathbf{a}_i$ (respectively $\mathbf{b}_j$) with $1 \leq i, j \leq 3$ denote the i'th edge axis of box A (B):

$$\begin{aligned}
&(1) \quad N = \{\mathbf{a}_i, \mathbf{b}_j, \frac{\mathbf{a}_i \times \mathbf{b}_j}{|\mathbf{a}_i \times \mathbf{b}_j|} \,|\, 1 \leq i, j \leq 3\} \\
&(2) \quad d = \min_{\mathbf{v} \in N} \{\delta([i_0, i_1], [j_0, j_1]) \,| \\
&\qquad\qquad i_0 = \min\{\mathbf{v}^\intercal \mathbf{x} \,|\, \mathbf{x} \in A\}, i_1 = \max\{\mathbf{v}^\intercal \mathbf{x} \,|\, \mathbf{x} \in A\}, \\
&\qquad\qquad j_0 = \min\{\mathbf{v}^\intercal \mathbf{y} \,|\, \mathbf{y} \in B\}, j_1 = \max\{\mathbf{v}^\intercal \mathbf{y} \,|\, \mathbf{y} \in B\}\} \\
&(3) \quad \text{if } (d < 0)\ d = 0
\end{aligned}$$

In (1) we number all possible axis directions. Every box gets projected on every axis in (2) and the minimal overlap of two projection intervals defines the penetration. Step (3) is only needed if A and B do not intersect. In this case d would be negative. We may stop the computation and return 0 as soon as two intervals with negative penetration are found.

The pseudo code description above would be very inefficient if implemented this way. The "shadow" that a box A casts on a given axis $\mathbf{n}$ can be regarded as an one-dimensional circle with radius r_A. We can calculate r_A very easily:

$$r_A \quad = \quad \sum_{0 \leq i < 3} |\mathbf{a}_i^\intercal \mathbf{n}| \cdot l_{A,i}$$

whereas $\mathbf{a}_i$ denotes the i'th axis of box A and $l_{A,i}$ the length of the i'th dimension (compare figure (3.3)). Then the overlap of two boxes A and B on axis $\mathbf{n}$ is given by $|\mathbf{c}_A - \mathbf{c}_B| - (r_A + r_B)$ with box centers $\mathbf{c}_A$ and $\mathbf{c}_B$.

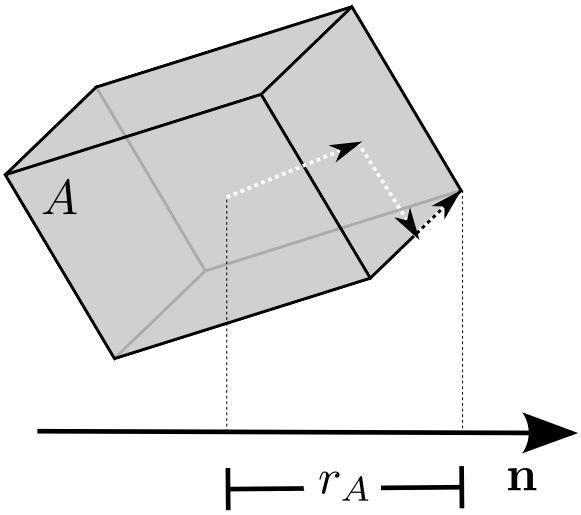
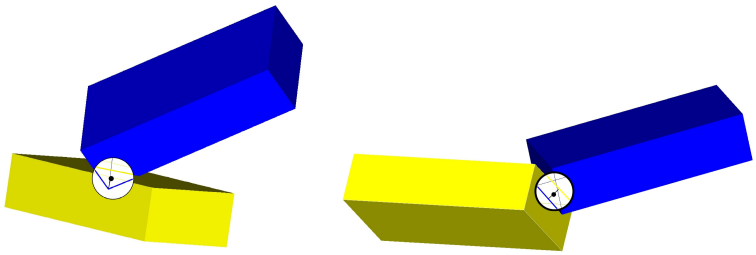


Figure 3.3: The calculation of the radius r_A of a box A.

3.2.1 Contact normal and collision center

To resolve a conflict of two objects we additionally need the contact normal **n** and a collision center **z**. The contact normal is the direction of the smallest overlap. It is parallel to **n** but might have a different sign. Without loss of generality we assume that **n** points from A to B (a simple comparison of the projection values of the centers of A and B upon **n** gives evidence whether the sign of **n** has to be changed).



(a) vertex/face collision (b) edge/edge collision

Figure 3.4: Possible collision scheme of two boxes

To find an appropriate collision center **z** we have to look at the way two boxes can intersect. In the case that **n** is a normal to one of the faces (let us say from B) a vertex **p** of A is penetrating the face of B with normal **n**.

The point $\mathbf{p}$ must be the extremal point of A that is maximal in direction $\mathbf{n}$. (compare figure 3.4(a)).

(3.1) **if** $(\mathbf{n} \in \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\})$
(3.2) $\quad \mathbf{p} = \text{argmax}_{\mathbf{x}\in A}\{\mathbf{n}^\intercal\mathbf{x} \,|\, \mathbf{x} \in A\}$
(3.3) $\quad \mathbf{z} = \mathbf{p} - \frac{d}{2}\mathbf{n}$
(3.4) **else if** $(\mathbf{n} \in \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\})$ ...

If $\mathbf{n}$ is the cross product of two directions we have an edge/edge penetration (see fig 3.4(b)). We locate $e_1 \in A$ and $e_2 \in B$ as the extremal edges in direction $\mathbf{n}$. The collision center $\mathbf{z}$ equals the midpoint $m(e_1, e_2)$ on the shortest connection between e_1 and e_2.

(3.5) **else** // $\quad \mathbf{n} = \frac{\mathbf{a}_i \times \mathbf{b}_j}{|\mathbf{a}_i \times \mathbf{b}_j|}$
(3.6) $\quad e_1 = \text{argmax}_{\mathbf{x}\in A}\{\mathbf{n}^\intercal\mathbf{x}\}$
(3.7) $\quad e_2 = \text{argmin}_{\mathbf{y}\in B}\{\mathbf{n}^\intercal\mathbf{y}\}$
(3.8) $\quad \mathbf{z} = m(e_1, e_2)$

3.3 Pairwise distance

A modified version of the penetration depth algorithm can be used to find the minimum distance of two boxes A and B. Compared to other approaches like Eberly (2006) this algorithm is easy to implement and even easier to describe. The running time of this calculation increases slightly compared to the penetration method as we need to identify all separating axes to get the one with the *maximal separation*.

Given two disjoint boxes A and B with axis directions $\mathbf{a}_i$ and $\mathbf{b}_j$ for $1 \le i, j \le 3$. In order to compute the minimal distance $d(A, B) = \min\{|\mathbf{y} - \mathbf{x}| \mid \mathbf{x} \in A, \mathbf{y} \in B\}$ we apply the following steps:

(1) $\quad N = \{\mathbf{a}_i, \mathbf{b}_j, \dfrac{\mathbf{a}_i \times \mathbf{b}_j}{|\mathbf{a}_i \times \mathbf{b}_j|} \,|\, 1 \le i, j \le 3\}$
(2) $\quad \mathbf{n} = \underset{\mathbf{v}\in N}{\text{argmax}}\ \min\{|\mathbf{v}^\intercal(\mathbf{y} - \mathbf{x})| \,|\, \mathbf{x} \in A,\, \mathbf{y} \in B\}$
(3) $\quad$ w.l.o.g. let $\mathbf{n}^\intercal\mathbf{x} \le \mathbf{n}^\intercal\mathbf{y}$ for $\mathbf{x} \in A,\, \mathbf{y} \in B$
(4) $\quad A' = \underset{\mathbf{x}\in A}{\text{argmax}}\ \mathbf{n}^\intercal\mathbf{x}$
(5) $\quad B' = \underset{\mathbf{y}\in B}{\text{argmin}}\ \mathbf{n}^\intercal\mathbf{y}$
(6) $\quad d(A, B) = d(A', B')$

The algorithm again uses SAT. As the two boxes are disjoint there must be at least one separating axis. As there may be more than one we calculate the direction of maximal separation $\mathbf{n}$. The separation itself usually is not the distance; but it leads us to the two sets A' and B' that are the closest extremal features of A and B in direction $\mathbf{n}$. Their distance defines the minimal distance of A and B.

An notion why this method works correctly on boxes is given in the following figures: If $\mathbf{n}$ is the direction of one axis (e.g. A in fig. (3.5)) the extremal

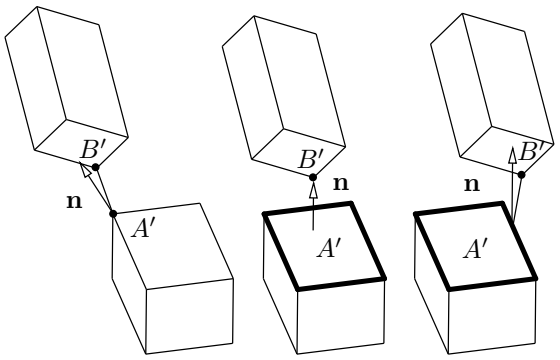


Figure 3.5: vertex-vertex, vertex-face and vertex-edge distance

feature B' always is a vertex of B whereas A' is a point of the face with normal $\mathbf{n}$. Depending on the perpendicular dropped on the supporting plane of this face, A' either lies on the boundary of this face (vertex/edge or vertex/vertex) or in the inner part of the face (vertex/face). If $\mathbf{n}$ is the cross product of two

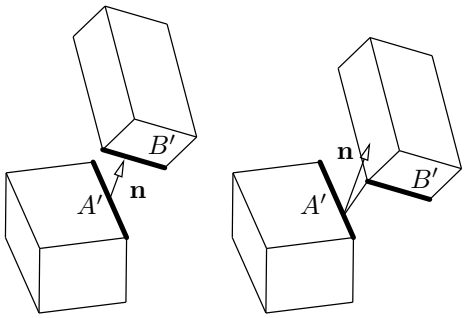


Figure 3.6: edge-edge and vertex-edge distance

edge directions (figure (3.6)) we are in the edge/edge case and the extremal features A' and B' are points on these edges. This case degenerates to a edge/vertex case if the perpendicular of the two carrier lines lies outside of one edge.

3.4 Minimal translational distance (MTD

The close relationship between the penetration calculation and the distance computation is obvious; if the boxes are disjoint, compute the axis of maximal separation. If they intersect, compute the axis of minimal overlap, which is the maximal separation, this time with a negative sign.

Having the maximal separation axis $\mathbf{n}$ on hand, we compute the maximal separation as

$$d \;=\; \min\{\mathbf{n}^\intercal(\mathbf{y}-\mathbf{x}) \mid \mathbf{x}\in A,\, \mathbf{y}\in B\}.$$

For negative d we are in the intersection case, and the penetration is $-d$. Otherwise we have to compute the minimal features of A and B along $\mathbf{n}$ as described in the last section to obtain the distance. If we are looking for the MTD, which we defined as a direction vector, then it is simply given by $d\cdot\mathbf{n}$.

3.5 Correctness

We gave an notion for the correctness of the computation scheme for boxes. SAT includes the correctness of the penetration computation for arbitrarily convex polygons. But is it correct that, if the objects are disjoint, the two extremal features of A and B in direction of the maximal separation $\mathbf{n}$ define the Euclidean distance? In the following we give a proof for this. We give the proof for two boxes but it should not be too hard to generalize it to arbitrary convex polyhedra.

Lemma 3.5.1: *Given two boxes A and B. Assume that the minimal translational distance $MTD(A,B)$ is known by the distance of two points $\mathbf{a}\in A$ and $\mathbf{b}\in B$. Further let the axis of the maximal separation be $\mathbf{n}_m$. Then the maximal separation $s_m = \min\{\mathbf{n}_m^T(\mathbf{y}-\mathbf{x}) \mid x\in A, y\in B\}$ equals the projection of $(\mathbf{b}-\mathbf{a})$ onto $\mathbf{n}_m$.*

To prove this lemma, we show the coincidence between A and B by their Minkowski Difference. It is motivated from the SAT correctness proof from Gottschalk (1998). We prove the lemma for the disjoint case as it has already been shown that for $A\cap B\neq\emptyset$ we get the penetration depth by the smallest overlap from all eligible separating axes. Regarding this as a negative distance, it is the same as the *maximal separation* for the non-disjoint case.

Definition 3.5.2: *Minkowski Sum*

The Minkowski-Sum $\oplus$ of two Sets $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$\mathcal{A} \oplus \mathcal{B} = \{a + b \,|\, a \in \mathcal{A},\, b \in \mathcal{B}\}$$

In the same way we can define the Minkowski Difference $\mathcal{A} \ominus \mathcal{B} = A \oplus (-\mathcal{B})$. If $\mathcal{A}$ and $\mathcal{B}$ are convex polyhedra, then $\mathcal{A} \ominus \mathcal{B}$ again is a convex polyhedron. The distance between $\mathcal{A}$ and $\mathcal{B}$ corresponds to the distance of $\mathcal{A} \ominus \mathcal{B}$ from the origin. If the origin lies within $\mathcal{A} \ominus \mathcal{B}$, their intersection is non-empty.

Geometrically the boundary of this polyhedron can be constructed by sweeping $-\mathcal{B}$ (the inversion of $\mathcal{B}$ in the origin) over the boundary of $\mathcal{A}$ or vice versa. The faces of this polyhedron then evolve either from sweeping a vertex over an original face or from sweeping an edge over another edge. The occurring normal vectors hence are either normals from one of the original polyhedra $\mathcal{A}$ or $\mathcal{B}$ or are cross products from two edge vectors. In other words the normals of the faces of $\mathcal{A} \ominus \mathcal{B}$ correspond to the eligible separating axes of $\mathcal{A}$ and $\mathcal{B}$.[1]

As shown in Gottschalk (1998) the Minkowski Difference of two boxes A and B is the intersection of 15 slabs (another word for the gap spanned by two parallel planes), and the normals of these slabs correspond to the eligible separating axes. The distance of a slab from the origin corresponds to the separation that the corresponding axis generates (comp. figure (3.7)). Every facet that is visible from the origin corresponds to a separating axis. If no facet is visible from the origin then the origin lies inside of the Minkowski Sum and A and B have a non empty intersection.

Definition 3.5.3: *We say a face f with normal vector* **n** generates *a separation s if f is visible from the origin. The separation s then is positive and equals the perpendicular dropped on the half-plane that contains f with normal vector n (also known as the affine hull of f).*

Now consider the statement from lemma (3.5.1) above and assume the distance $MTD(A, B)$ is known by the distance of two points $a \in A$ and $b \in B$. In the Minkowski Difference we find the point $p = a - b$ that must lie on the boundary of $A \ominus B$ and the distance of p from the origin equals $MTD(A, B)$. Any separating axis n is the normal vector of a visible facet f. The projection

[1] It is easy to see that this holds true if $\mathcal{B}$ is a point symmetric set like a box. In general the faces and edges of $\mathcal{B}$ and $(-\mathcal{B})$ differ in the sign of their normal vectors. The consequence is that the separating axes might change sign as well, but the sign of a separating axis is irrelevant to determine the separation.

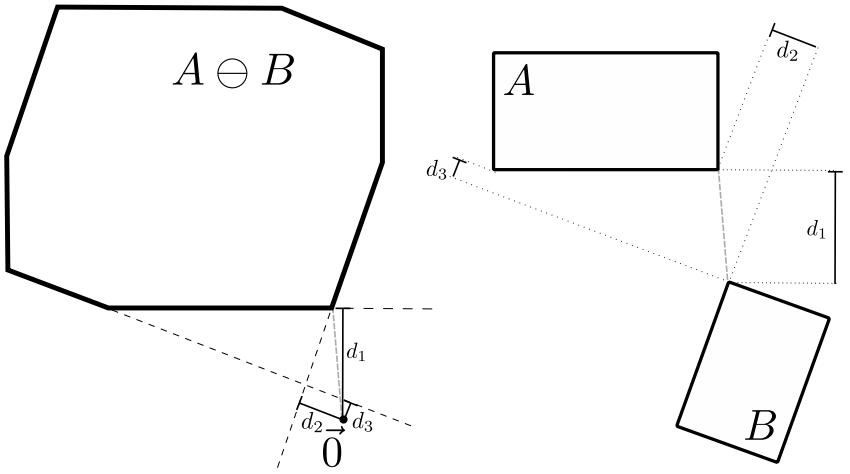


Figure 3.7: Two boxes A and B and their Minkowski Difference $A \ominus B$ in $2d$. We see the correspondence between the separating axes of A and B and the faces of $A \ominus B$ that are visible from the origin; three faces are visible and the origin drops the perpendiculars d_1, d_2 and d_3 on the planes they lie in. In the original geometry we find exactly those values as separations d_1, d_2 and d_3 of the three existing separating axes, whereas d_1 defines the maximal separation.

of $(p - \vec{0})$ onto that normal n is maximal if p lies on f (or at least if p lies on the affine hull of f). In order to proof the lemma (3.5.1) it suffices to show that p lies on the facet f_m whose normal vector $\mathbf{n}_m$ is the axis of maximal separation s_m.

The Minkowski Difference $A \ominus B$ - as being a convex polyhedron - is a waterproof 2-manifold and its boundary consists of faces, edges and vertices. The local neighborhood of p now defines what the closest features of A and B are.

1. p lies on a face f of $A \ominus B$. In this case the distance equals the separation that is generated from the axis normal to f. As the distance is an upper bound for all separations, this separation must be the maximal separation. It is also obvious that the statement of lemma (3.5.1) is fulfilled then.

2. p lies on an edge e of $A \ominus B$. Let f_1 and f_2 be the two faces adjacent to e in $A \ominus B$ as shown in figure (3.8). In this case we have at least two separations, namely the two generated by f_1 and f_2. Obviously one of them must correspond to the maximal separation, because any other

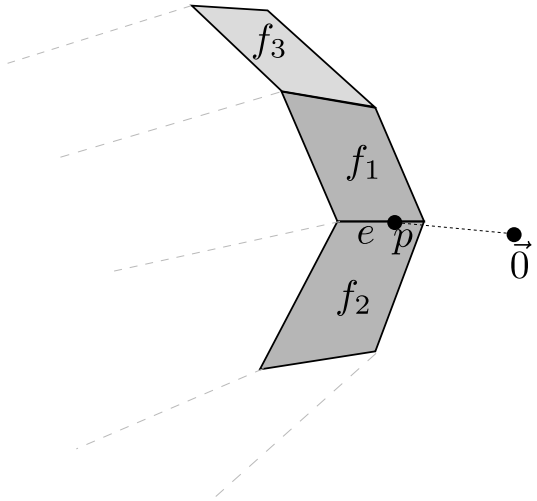


Figure 3.8: The closest feature of the Minkowski Sum to the origin is an edge e that is adjacent to the faces f_1 and f_2. These faces are visible from the origin, and so might be any other face f_3 that is not neighbored to e.

face that might be visible from the origin (e.G f_3) generates a separation that must be smaller than those from f_1 or f_2. We can conclude this from the fact that the Minkowski Sum is convex and that any other face apart from f_1 or f_2 must have a bigger incline from the view of the origin (comp. figure (3.8)). As p lies on both faces f_1 and f_2 the lemma holds for this case.

3. p is a vertex of $A \ominus B$. The case is similar to the last case, but now we have three or more faces that are adjacent to p. All those faces must generate separations as they all are visible from the origin. Any separation generated by non-adjacent faces cannot be greater than those due to the fact that $A \ominus B$ is convex. This means the maximal separation s_m is being generated by one of the faces adjacent to p; this proves the lemma.

3.6 ε-distance

In some cases we might need to know if two objects are closer than a given distance $\varepsilon \in \mathbb{R}$. This is the case if we need the penetration depth of two *offset-boxes*, in other words the `Minkowski`-sum of a box and a sphere. It also

works if only one object has an offset. In other scenarios it might be used as an exclusion criteria for brought/narrow phase tests of bounding boxes.

The ε-distance corresponds to the precise distance if it is less than ε, otherwise it is ε. The advantage of calculating the ε-distance compared to the real Euclidean distance is the possibility of a number of early-outs. This is based on our conclusions on SAT and permits the implementation of a fast and very robust algorithm. Let us take a look at the following facts:

1. The Euclidean distance d is an upper bound for any separation.
2. If any separation is greater than ε the Euclidean distance d is greater than ε

With the 15 separating axes we defined so far, we might not find the one that realizes the distance. But if two boxes are disjoint, we can extend the set of separating axes. Among those axes we will find the one realizing the distance. The trick is to stop as soon as we either find the greatest separation, means the distance, or find any separation greater than ε.

The additional axes for two boxes are the pairwise vertex distances and the pairwise perpendiculars of edge/vertex. For two boxes this is $8 \cdot 8 = 64$ plus $8 \cdot 12 = 96$ (so 160 axes to test), which is a tremendously high number. But with the previous knowledge of the maximum separation axis $\mathbf{n}_m$ from section 3.3, the closest features are given by the vertices that are extreme in $\mathbf{n}_m$ and their adjacent edges. Faces as extremal features have already been tested, so we only have to test the extreme vertices and their adjacent edges. In the worst case, we test the distance of two wireframe rectangles, each one having 4 vertices and 4 edges. This makes $3 \cdot 4^2 = 48$ additional tests in the worst case.

Every axis holds the chance for an early-out, if it separates more than ε. Further if we find two extremal features whose distance d is the maximal separation d_m, it is clear that d_m is the Euclidean distance and we can stop.

All in all it is very unlikely we have to test all 48 axes and our runtime experiments show that using the early-outs we outperform any other conventional distance algorithm. The ε-distance algorithm of course can be used to compute the real Euclidean distance, if we set $\varepsilon = \infty$. Even then our runtimes compete with existing implementations as we will show later.

Optimizations The resulting algorithm gives fast responses; for disjoint boxes the early-out-strategy is up to 4 times faster than computing the ex-

act distance. Novel to earlier implementation is that we abandon expensive square roots for SAT tests. In literature unit vectors are usually used as separating axes, if the penetration depth is needed, because the separation scales with the axis' length, and we need a common metric if we need to compare the separations of different axes. But we also can "square normalize" the axes and compare squared separations. Only in the end, when the distance has to be returned, we need one square root calculation. The only thing we have to care about is that we do not lose the sign of the separation, because it shows us if the boxes are distinct or not. If we take the actual calculation scheme for the separation of two objects, let's say boxes A and B, along a given unnormalized direction $\mathbf{n}$ we get the separation $s_\mathbf{n}$ as:

$$s_\mathbf{n} = \sum_{i=0}^{3} |\mathbf{a}_i^\intercal \frac{\mathbf{n}}{|\mathbf{n}|}| + \sum_{i=0}^{3} |\mathbf{b}_i^\intercal \frac{\mathbf{n}}{|\mathbf{n}|}| - |(\mathbf{c}_B - \mathbf{c}_A)^\intercal \frac{\mathbf{n}}{|\mathbf{n}|}|$$

and if $s_\mathbf{n} \geq \varepsilon$ we know that the Euclidean distance d of A and B is greater than ε as d is an upper bound of all separations. In the equation we see that $\frac{1}{|\mathbf{n}|}$ is a factor in every sum, and as it must be positive we can factorize the whole expression to $s_\mathbf{n} = \frac{1}{|\mathbf{n}|} \cdot \tilde{s_\mathbf{n}}$ whereas $\tilde{s_\mathbf{n}}$ is given by:

$$\tilde{s_\mathbf{n}} = \sum_{i=0}^{3} |\mathbf{a}_i^\intercal \mathbf{n}| + \sum_{i=0}^{3} |\mathbf{b}_i^\intercal \mathbf{n}| - |(\mathbf{c}_B - \mathbf{c}_A)^\intercal \mathbf{n}|$$

Instead of dividing by $|\mathbf{n}|$, which would cause expensive square roots, we can divide by $\mathbf{n}^2$. This is a much faster calculation. We obtain this by squaring the whole inequation:

$$s_\mathbf{n} = \frac{1}{|\mathbf{n}|} \cdot \tilde{s_\mathbf{n}} \geq \varepsilon \quad \Longrightarrow \quad \frac{1}{\mathbf{n}^2} \cdot \tilde{s_\mathbf{n}}^2 \geq \varepsilon^2 \tag{3.1}$$

Please note that this implication only is valid if $A \cap B = \emptyset$, or $\tilde{s_\mathbf{n}} \geq 0$. If not the relation sign gets inverted. In a nutshell we better use the following implication:

$$s_\mathbf{n} = \frac{1}{|\mathbf{n}|} \cdot \tilde{s_\mathbf{n}} \geq \varepsilon \quad \Longrightarrow \quad \frac{1}{\mathbf{n}^2} \cdot \tilde{s_\mathbf{n}} * |\tilde{s_\mathbf{n}}| \geq \varepsilon^2 \tag{3.2}$$

It is valid because ε is greater or equal zero. In the case that A and B intersect, there is no early-out as the inequality does not hold.

The algorithm is as easy-to-implement it is robust; earlier approaches suffered from degenerate cases, when edge directions are nearly parallel, or edges are

almost parallel to faces. It requires an explicit special case handling then and causes expensive branching. The new approach just computes more axes to be tested. This method is also suited to be parallelized; the detection of all eligible axes would be a pre-computation, and the later SAT-test would be a SIMD operation, which is the same work for every thread only on different axes. Of course we would lose the advantage of early-out then.

3.7 Experimental results

No external MTD methods specialized for cuboids exist that we could compete against. Known distance algorithms for rectangles in $\mathbb{R}^3$ (c.f. Eberly (2006)) already need 81 distinct tests and hence are hard to implement. Other implementations for convex polyhedra like Kim et al. (2002) usually use costly Minkowski sums computation inspired by one of the major works on MTD for convex polyhedra (Cameron and Culley (1986)).

Hence we compared our ε-distance algorithm against two existing implementations: the first is an open source physics library (SOLID) and the second is a very efficient in-house-implementation from our group specialized for boxes. We tested on a MacbookPro X68 machine under `G++ 4.2.1` with highest optimization level.

The first is an efficient penetration depth and distance algorithm for polytopes and simple quadric objects like spheres, cones and cylinders using a variant of the Gilbert-Johnson-Keerthi (GJK) algorithm described in van den Bergen (2001). This is a state-of-the-art implementation from the SOLID library (van Bergen (2006)). It provides an own specialized data structure for boxes that is very fast because it is also used for bounding box distance calculation. For every convex shape there exist two functions, `DT_GetClosestPair` to detect the distance and `DT_GetPenDepth` to compute the penetration depth. As our methods compute both, distance and penetration depth in one step, we used this information to compare with the right `SOLID` method. Surprisingly to us the penetration depth calculation in `SOLID` seems to be about two times slower than the distance calculation.

The second approach was developed in our group by Will. It uses a very optimized SAT implementation in case the boxes overlap. If the boxes are distinct, a $3d$ region code implementation similar to the $2d$ *Cohen-Sutherland-Algorithm for line clipping* (for further information see Foley et al. (1990)) is used to detect closest features. This is done twice in the coordinate system

of each box. The actual distance then is calculated by brute force testing all 144 edge pairs, where the pre-computations with region code checks help to prune most of the pairs that cannot realize the distance. The implementation was designed with regard to speed and to handle special cases that former approaches would not be able to cover. In the following we will show the experimental results and we will also go into detail about those special cases.

The reliability and robustness of a geometric algorithm shows best in degenerate cases. The non-degenerate case or so called general situation is only one scenario we test. Vast experiments in the past showed us that mainly two other scenarios need a special treatment, and each one occurs for distinct and intersecting items:

Axis parallel In this case the orientation of box one is axis parallel to the second box. Edges are pairwise parallel or orthogonal and so are the face normals. The problem then reduces to a lower dimensional problem in $2d$. If special care is not take, an algorithm might run into undefined conditions like computing the cross product of two parallel (edge/face)-vectors. The figure (3.9) shows this special case.

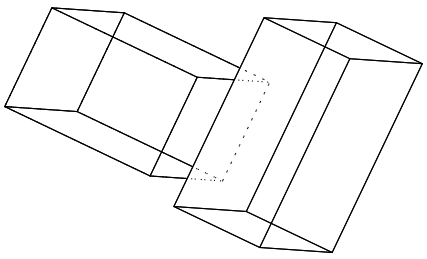
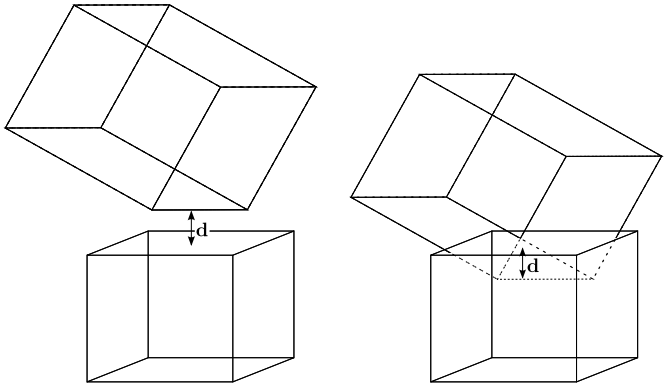


Figure 3.9: Two axis parallel boxes

Edge parallel to face This case is even more severe; an edge is orthogonal to a face normal. Even if detected this case is costly to solve because it is not trivial to compute the feature points that realize the distance. This case caused big problems in former approaches and was the reason why we were in need of finding a more satisfying solution.

The major advantage of our ε-distance computation is its robustness against degenerate cases. Indeed the implementation does not need to handle special cases at all as they are covered native by our method. This is the main reason why our implementation is fast compared to other algorithms as we can see in table (3.1).



(a) Disjoint boxes (b) Intersecting boxes

Figure 3.10: The two degenerate scenarios for the edge/face case

The following table (3.1) shows the runtime comparisons for the three scenarios, every scenario for disjoint and intersecting case. For every case we constructed 1000000 DIN box pairs of dimension $50mm \times 100mm \times 200mm$ by random and run the algorithms on a MacBook Pro with a 2.53 `GHz` Intel Core 2 Duo processor and 4`GB` RAM. We see that the ε-distance algorithm is always the fastest and the bitfield algorithm keeps up for intersecting boxes. The advantage of the early-out strategy of our implementation shows for the distinct case where the ε-distance algorithm is up to 4 times faster than the bitfield algorithm. The `SOLID` implementation is the slowest, but still very fast. We experienced the most incorrect outputs froms the GJK-implementation, which we show in the last column along with their relative occurrence. We compared the results of all three algorithms within a tolerance of 10^{-2}mm, and the ε-distance and bitfield algorithms are 100% conform.

	non-deg		orthogonal		edge || face	
	intersect	disjoint	intersect	disjoint	intersect	disjoint
ε-distance	$0.71s$	$0.35s$	$0.63s$	$0.45s$	$0.70s$	$0.88s$
bitfield algo	$0.81s$	$1.31s$	$0.86s$	$1.11s$	$0.82s$	$1.47s$
`SOLID`	$5.44s$	$2.83s$	$4.07s$	$2.49s$	$5.60s$	$3.20s$
`SOLID` % incorrect	0.021	0.004	0.0013	0.0001	0.027	0.029
$\varnothing$ failure $[mm]$	24.53	0.036	14.15	0.057	23.30	0.036

Table 3.1: Runtime tests on 1000000 DIN-box pairs of $200mm \times 100mm \times 50mm$ with $\varepsilon = 20mm$ in different scenarios and three tested algorithms.

Also notable is the fact that the distance calculation in SOLID seems to be implemented more carefully than their penetration depth computation that is not only slower but also gives less accurate results. Although the percentage of wrong results is less than 0.1%, the average failure of more than $10mm$ at box dimension of $50mm \times 100mm \times 200mm$ is unacceptably high. We can not assume that van den Bergen's interpretation of the penetration depth differs from ours as we adopted their definition of MDT from their paper (van den Bergen (2001)). Therefore we can only guess that the penetration depth algorithm doesn't cover special cases carefully enough.

	non-deg	orthogonal	edge || face
ε-distance	$1.16s$	$2.75s$	$1.38s$
bitfield algo	$1.26s$	$1.06s$	$1.42s$
SOLID	$2.77s$	$2.44s$	$3.23s$

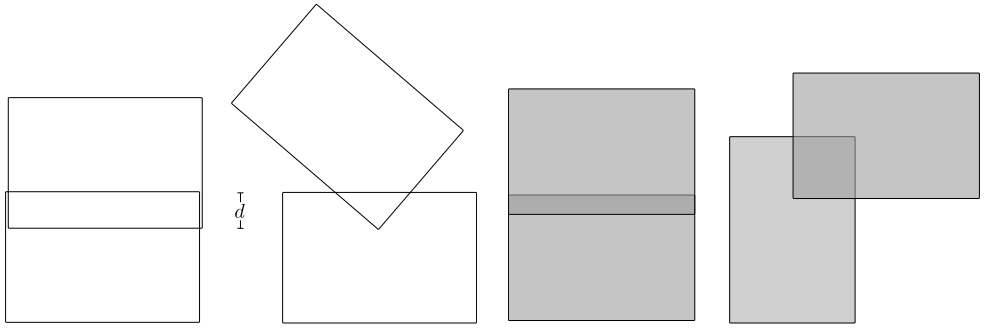
Table 3.2: Runtime tests on 1000000 disjoint DIN-box pairs of $200mm \times 100mm \times 50mm$ with $\varepsilon = \infty$ and three tested algorithms.

The early-out strategy for disjoint boxes works faster, the smaller ε is. When ε is greater than the real distance, the algorithm returns the distance and the early-out benefit is smaller. As table (3.2) shows for $\varepsilon = \infty$, the implementation is much slower then. For orthogonal edges we get the worst runtime, because the most separating axes are tested. But it is remarkable that our implementation still competes with both approaches in all other cases.

3.8 Intersection volume of two boxes

In Eisenbrand et al. (2005) and Eisenbrand et al. (2007) - earlier works from our group dealing with the trunk packing problem for the DIN standard - a simulated annealing approach (SA) is presented whose objective function heavily makes use of two characteristic values that give evidence on the amount of overlap of a current configuration during the simulation: (1) the penetration depth and (2) intersection volume.

The reason why the easier-to-calculate penetration depth alone is not sufficient to value a given configuration is shown in figure (3.11). Although it is much faster to compute, the penetration depth of all boxes of a configuration is not an adequate measure to rate a configuration. At the same time, as explained in Eisenbrand et al. (2005), the pure intersection volume might also be a disadvantage in some degenerate cases. Therefore the authors chose



(a) The two boxes overlap by the same penetration depth. But the volume (respectively the area) of overlap is much different.

(b) The overlapping area is equal in size. But to legalize the left pair only a slight move is needed opposite to the right one.

Figure 3.11: Intersection volume and penetration depth

a linear combination of penetration depth *and* intersection volume to rate a configuration after a move.

The penetration depth function used is the one described in section 3.2 and it is about 30 times faster than the intersection volume computation that was implemented. It works as follows (For a detailed description see Eisenbrand et al. (2005) and Eisenbrand et al. (2007)):

1. calculate the intersection of two boxes explicitly,
2. decompose the boundary of the intersection into triangles,
3. determine an inner point p of the intersection,
4. for every triangle calculate the volume of the tetrahedron spanned by p and the triangle,
5. sums all tetrahedral volumes up.

As the volume and penetration computations are used several billion times during the simulation, a faster and also more reliable implementation was desired. This section describes the actual implementation that has replaced the primarily used intersection volume computation. As it works about two times faster, it improves the SA implementation tremendously.

We first give a short description of how to obtain the volume of a convex polyhedron by reducing it to a sum of line integrals as suggested in Mirtich

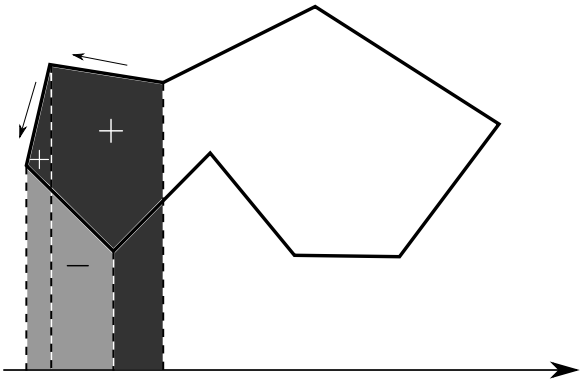


Figure 3.12: By integrating over the edges of a polygon we obtain the area of the interior of that polygon.

(1996a). Then we explain how we get the edges of the intersection without having to calculate the whole representation explicitly. A third part will give an notion of how complex the geometry of the intersection of two boxes might become. The last part will deal with runtime experiences of our implementation in comparison to the former triangle composition implementation and a state-of-the-art implementation for arbitrary polyhedra from the GNU Triangle Library.

3.8.1 Volume of a polyhedron

In Mirtich (1996a) the author gives a computation scheme to reduce the volume calculation of a (not necessarily convex) polyhedron P to a collection of line integrals in the plane. These integrals correspond to the edges of the convex faces $\mathsf{F} = \partial\mathsf{P}$ that define the boundary of P. Each edge $e = (\mathbf{a}, \mathbf{b})$ that bounds a face $f \in \mathsf{F}$ has a share in the complete volume of P, which is

$$\text{lineIntegral}(\mathbf{a}, \mathbf{b}, \mathbf{n}_f) \quad = \quad \frac{1}{6}\frac{n_x}{n_z}(b_y - a_y)(a_x^2 + a_x b_x + b_x^2) \tag{3.3}$$

and $\mathbf{n}_f = (n_x, n_y, n_z)^\intercal$ is the normal of f, which points to the outside of P. How this reduction works can be seen in figure (3.12) for a polygon in $\mathbb{R}^2$; given an arbitrary axis (usually one of the coordinate axes) we can calculate the signed area that every edge generates and sum them all up. The sign of an area is given by the direction of its edge. We only have to consider edges of the boundary as every inner edge exists twice with different direction such that their areas nullify each other.

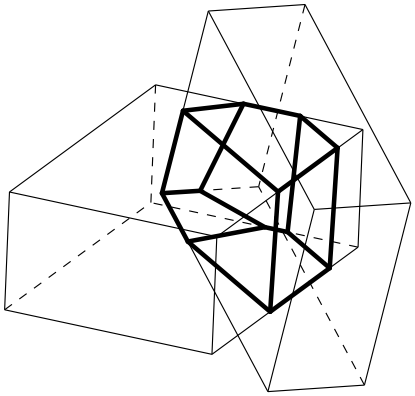


Figure 3.13: Intersection volume of two boxes

The closer n_z is to 0 in formula (3.3) the smaller the projection shadow of the face onto the (x, y)-plane is. This might cause numerical instabilities, and for $n_z = 0$ formula (3.3) is even undefined. Mirtich therefore proposes to permute the x-y-z axes right-handed such that n_z becomes maximal.

3.8.2 Intersection of two boxes

The intersection of two boxes is a convex polyhedron with maximal 12 faces. An edge of the intersection body is either a fraction of an original edge that is cut by one or two faces or is generated by the intersection of two rectangular faces (in $\mathbb{R}^3$ (see figure (3.13)).

To detect the new edges that arise from the intersection of two rectangles in $\mathbb{R}^3$ we use the idea from a fast rectangle intersection algorithm from Eberly (2001). We detect the relevant parts of the original edges by intersecting an edge $e \in P_i$ with box P_{3-i} for $i \in 1, 2$. This can be done by clipping line segments against a box and costs a few arithmetic operations. The normals are found among the original normals of the box faces.

$V = 0$
forall $(f_1, f_2) \in F_1 \times F_2$ with $f_1 \cap f_2 \neq \emptyset$ do
  $(\mathbf{a}, \mathbf{b}) = f_1 \cap f_2$ with $\det(\mathbf{b} - \mathbf{a}, \mathbf{n}_{f_1}, \mathbf{n}_{f_2}) > 0$
  $V +=$ lineIntegral$(\mathbf{a}, \mathbf{b}, \mathbf{n}_{f_1})$
  $V +=$ lineIntegral$(\mathbf{b}, \mathbf{a}, \mathbf{n}_{f_2})$
od
for $(i = 1, 2)$ do
  forall $(f \in F_i)$ do
    forall $(e \in f)$ with $e \cap P_{3-i} \neq \emptyset$ do
      $(\mathbf{a}, \mathbf{b}) = e \cap P_{3-i}$ with $\mathbf{n}_f^\intercal(\mathbf{b} - \mathbf{a}) > 0$
      $V +=$ lineIntegral$(\mathbf{a}, \mathbf{b}, \mathbf{n}_f)$
    od
  od
od

Vector $\mathbf{n}_{f_i}$ denotes the normal perpendicular to face f_i. It is self explanatory that this scheme can be generalized for non convex polyhedra. The efficiency of the computation mainly depends on the detection of intersecting face pairs and how fast we can determine, which parts of a line segment lie in the interior of a polyhedron.

3.8.3 Complexity of the intersection

The number of faces, edges and vertices of the intersection polyhedron of two boxes differs a lot depending on the dimension, position and orientation of the input boxes. It can hence become quite complex compared to the rather simple structure of a box with 8 vertices, 12 edges and 6 faces. In the following we try to give an notion of how many edges the intersection volume computation has to deal with in the worst case. To get a better view of the structure of the intersection body, we examine the dual problem of the convex hull of a number of points.

Duality

A box is the intersection of 6 pairwise parallel and mutual orthogonal half-spaces. The intersection of two boxes therefore is the intersection of 12 half-spaces.

In Werth (2004) the author shows the theory and the calculus of how to obtain the intersecting of a set halfspaces in $\mathbb{R}^d$ by means of computing the convex hull of a set of points, whereas a half-space $H = \{\mathbf{x} \in \mathbb{R}^d | \mathbf{n}^\intercal \mathbf{x} \leq 1\}$ is dualized to a point $\mathbf{n} \in \mathbb{R}^d$. We also learn about several dependencies between dual and primal subspaces. To simplify matters, we give a résumée of these observations as far as it is concerned to our problem in $\mathbb{R}^3$:

dual The dual of an arbitrary half-space $H_i = \{\mathbf{x} \in \mathbb{R}^3 | \mathbf{n}_i^\intercal \mathbf{x} \leq 1\}$ in $\mathbb{R}^3$ that contains the origin is a point $\mathbf{n_i} \in \mathbb{R}^3$.

intersection The dual of the intersection of a set of l half-spaces $\{H_1, \ldots, H_l\}$ is the convex hull $\mathcal{C}$ of its dual and the origin:
$\mathcal{C} = CH(\{\{\mathbf{n}_1, \ldots, \mathbf{n}_l\} \cup \vec{0}\})$.

dual faces The primal of a dual face on the boundary of $\mathcal{C}$ with normal vector $\mathbf{v}$ is the point $\mathbf{v}$.

dual edges The primal of an edge in $\mathcal{C}$ that is adjacent to two faces f_1 and f_2 is the primal edge that is adjacent to the primal of f_1 and f_2.

The dual of the intersection of two boxes consequentially is the convex hull of 12 points. In the sequel we assume that the intersection is a 2-manifold and does not consist of a single face, edge or point. Doing so we avoid degeneracies in the dual like collinear or coplanar points. Then the simplest intersection solid is a 3-simplex or tetrahedron with 4 vertices, 6 edges and 4 faces. Hence, its primal would again be a 3-simplex. This means the intersection of two boxes generally consists of minimum 4 vertices, 6 edges and 4 (triangular) faces.

The complexity of the structure of the intersection polygon obviously depends on the number of vertices on the boundary of the convex hull. The most complex solid we expect if each of the 12 points is a vertex of the boundary. Because of the duality between *convex hull* and *half space intersection* the faces of the dual polyhedron consists of triangles (in the general case) because a primal point is the intersection of 3 primal faces. Under all those conditions, namely that 12 points lie on the boundary of the convex hull and that it consists of triangles, we expect the dual convex hull to be something that is homomorphic to an icosahedron in worst case. This means the convex hull

at most has 12 vertices, 20 faces and 30 edges. Hence the primal intersection polyhedron has at most 20 vertices, 30 edges and 12 faces.

We can show the correctness of this estimation by an examination of the Euler equation for regular polyhedra

$$|V| - |E| + |F| \quad = \quad 2$$

It can be proved by recursion over $|V|$. In the case of a finite, simple and connected graph that is a 2-manifold of at minimum $|V| = 3$ vertices we can assume that every facet is bounded by at least three edges and each edge is adjacent to at most two faces. This leads to the in-equation $3 \cdot |F| \leq 2 \cdot |E|$, which we insert into Euler's equation to obtain (for a triangulated convex hull we obtain equality):

$$|E| \quad \leq \quad 3 \cdot |V| - 6$$

In our case we have 12 vertices and at most 30 edges. With Euler's equation $|V| + |F| - |E| = 2$ we can conclude that the convex hull consists of maximal 20 faces. The primal intersection polyhedron therefore has at most 12 faces, 30 edges and 20 vertices. All in all the number of edges we have to deal with lies between 6 and 30.

3.8.4 Runtime experiments

The described algorithm is implemented and used in a trunk packing software suite and evaluates the quality of a configuration during a Monte Carlo simulation (compare Eisenbrand et al. (2005)). Our implementation is in C++ and is platform independent. We tested it on Debian Linux (with g++ 3.3.5), Windows XP (Visual C++ 2005) and lately on a Macbook Pro (with g++ 4.2.1) where the benchmarks in table (3.3) where determined.

The GNU Triangulated Surface Library is an open source free software library intended to provide a set of useful functions to deal with 3D surfaces meshed with interconnected triangles (GTS (2006)), which is written in an object oriented C style. It provides operations on polyhedra, such as intersection, union, difference and much more. It organizes the geometrical objects in a hierarchical structure of vertices, edges, triangles, faces and surfaces and uses

bounding box trees to avoid dispensable intersection operations. It is possible to compute the volume of a closed surface where it determines, similar to our approach, the fraction that every triangle of the surface has on the complete volume.

To compare our implementation with the operations provided by GTS, we tried to work as efficiently as possible; we did not take pre-computation time into account, such as building up the bounding boxes tree. We also minimized the memory requirement as much as possible.

The tests were done by 1000000 box pairs of equal size $200 \times 100 \times 50$ with different intersection probabilities. Every box was converted to a *GtsSurface* with 12 triangles, 18 edges and 8 vertices. In table 3.3 we see that the new implementation is about two times faster than the prior triangle decomposition algorithm. The factor is smaller the smaller the fraction of overlapping pairs on all pairs is. In fact both algorithms use the same intersection test in advance such that the actual volume computation is only performed if the boxes have a non empty overlap. We also see in table 3.3 that our implementation is more than a factor 50 faster than the much more general implementation of GTS, which also allows intersection operations on non-convex objects. The benefit of the bounding box tree of GTS becomes apparent by the smallest intersection probability compared to the highest. In all tests the maximal deviation of the returned volume values between our implementation and GTS was less than 10^{-6}.

		12%	53%	95%
1	**line integral**	$0.933s$	$3.309s$	$6.327s$
2	**decomp. algo**	$1.512s$	$6.112s$	$12.532s$
3	**GTS**	$49.223s$	$185.609s$	$398.532s$
	factor 3/1	52	56	62

Table 3.3: Runtime tests on 1000000 box pairs with different intersection probabilities 12%, 53% and 95%

3.8.5 Vertex Neighborhoods: An alternative approach

In Franklin and Kankanhalli (1992) a very interesting approach is described for the intersection volume of arbitrary polyhedra that is very similar to our approach as the authors also calculate the volume without having to compute the intersection polyhedron explicitly.

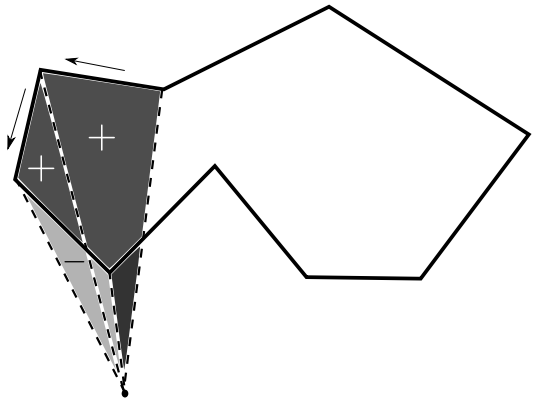


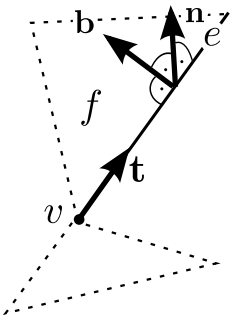
Figure 3.14: Same polyhedron from figure 3.12. Every edge spans a triangle with the origin and the sum of their signed areas is the whole volume.

The approach acts on the assumption of a given tetrahedral decomposition of the input polyhedra and computes the volumes of those tetrahedra overlaps. Instead of line integrals, the author uses the local neighborhood of the vertices to compute the volume as described in the authors prior work (Franklin (1987)).

In this way, the volume is calculated by only obtaining the vertices of the intersection. A vertex of the intersection is either an original vertex from one of the input polyhedra or an intersection of an edge and a face.

The mathematical foundation to obtain the volume from the edges differs from our line integral concept; instead Franklin and Kankanhalli sum up the signed volumes of the tetrahedra that every triangle on the boundary of the intersection spans with the origin (compare figure (3.14)). The interesting fact of this work is that the triangles do not have to be computed, as the local neighborhood of every vertex suffices to compute the volume.

Definition 3.8.1 (Vertex neighborhood): *The local neighborhood of a vertex $v \in \mathbb{R}^3$ that is adjacent to an edge e with normalized direction vector $\mathbf{t}$ on the boundary of a face f with unit normal vector $\mathbf{n}$ is given by a tuple $(v, \mathbf{t}, \mathbf{n}, \mathbf{b})$. $\mathbf{b}$ is a vector in the face f orthogonal to e (and $\mathbf{t}$) and it points into the interior of the face f. At the same time $\mathbf{n}$ must point into the interior of the polyhedron.*



There is a tuple for each case of adjacent face-edge-vertex combination and

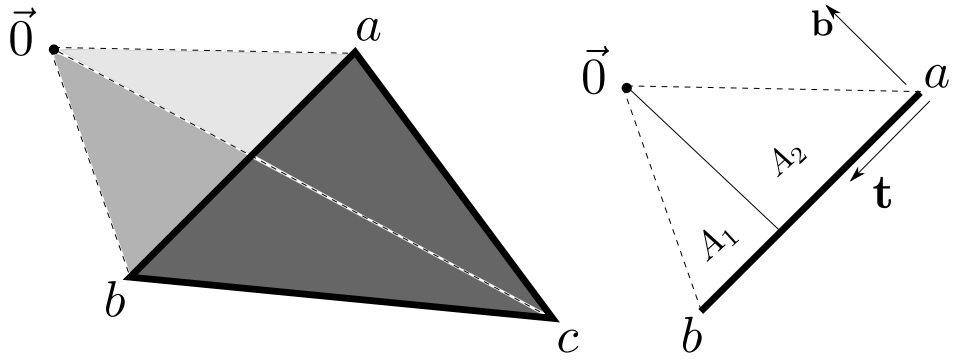


Figure 3.15: The area of triangle (a, b, c) can be decomposed into three triangles. Each one can again be decomposed into two right angled triangles with areas A_1 and A_2.

the total volume then is given as

$$V = \sum_{\forall(v,\mathbf{t},\mathbf{b},\mathbf{n})} \frac{1}{6}(v^\intercal \mathbf{t}) \cdot (v^\intercal \mathbf{b}) \cdot (v^\intercal \mathbf{n}) \tag{3.4}$$

The notion for the correctness of this formula is as follows; the polygon can be decomposed into triangles. Every triangle spans a tetrahedron with the origin and the sum of the volumes of all those tetrahedra is the volume of the polygon.

The volume of a single tetrahedron evolves from the following: if we consider a triangle (a, b, c) in $2d$, and we are interested in the area of this triangle, we can construct three triangles $(\vec{0}, a, b)$,$(\vec{0}, b, c)$,$(\vec{0}, c, a)$ and the sum of their (directed) areas then is the area of the triangle (a, b, c) (comp. figure 3.15). Now consider line segment (a, b) with normalized direction vector $\mathbf{t}$ and orthogonal normalized vector $\mathbf{b}$. The area of triangle $(\vec{0}, a, b)$ is

$$\begin{aligned} A &= \frac{1}{2} \cdot (b-a)^\intercal \mathbf{t} \cdot b^\intercal \mathbf{b} \\ &= \frac{1}{2} \left(b^\intercal \mathbf{t} \cdot b^\intercal \mathbf{b} + a^\intercal(-\mathbf{t}) \cdot a^\intercal \mathbf{b}\right) \\ &= \underbrace{\frac{1}{2} \left((b^\intercal \mathbf{t}) \cdot (b^\intercal \mathbf{b})\right)}_{A_1} + \underbrace{\frac{1}{2} \left((a^\intercal(-\mathbf{t})) \cdot (a^\intercal \mathbf{b})\right)}_{A_2} \end{aligned}$$

We could say that the vertex neighborhood of vertex b in $\mathbb{R}^2$ is the tuple $(b, \mathbf{t}, \mathbf{b})$ and the vertex neighborhood of a is $(a, -\mathbf{t}, \mathbf{b})$. In this sense we would be able to express the whole area of triangle (a, b, c) by the following sum

$$A_{(a,b,c)} = \sum_{\forall(v,\mathbf{t},\mathbf{b})} \frac{1}{2}(v^\intercal \mathbf{t}) \cdot (v^\intercal \mathbf{b})$$

Note that in figure 3.15, the areas A_1 and A_2 have negative signs as they are substracted from all other areas in order to obtain $A_{(a,b,c)}$.

As a next step, we change the point of view and imagine that the origin does not lie in the same plane as the triangle (a, b, c). We further say that the distance of the origin from that plane is a non zero value $h \in \mathbb{R}$. Now the origin spans a pyramid with the triangle (a, b, c), and its volume is given by

$$V = \frac{1}{3} \cdot A_{a,b,c} \cdot h$$

The height h of the pyramid is given by the projection of any of the three vertices a,b or c onto the normalized normal vector $\mathbf{n}$ that is orthogonal to face (a, b, c), or in other words $h = a^\intercal \mathbf{n} = b^\intercal \mathbf{n} = c^\intercal \mathbf{n}$. A simple conversion of the equation above yields equation (3.4) for the volume of the tetrahedron (or pyramid) $(\vec{0}, a, b, c)$.

A formal proof that also shows how to evaluate the vertex neighborhood values is the following:

Proof. (of equation (3.4)) We will again show that the equation holds for a single triangle. For every polygon there exists a triangulation of its boundary. The tetrahedra that those triangles span with the origin decompose the volume of the polyhedron (compare figure (3.14)). Furthermore redundant edges that are adjacent to parallel faces generate local neighborhoods that eliminate each other in the equation such that it also holds for non triangulated closed polyhedra.

Given a triangle T with vertices $a, b, c \in \mathbb{R}^3$. Together with the origin they form a tetrahedron with the volume

$$V = \frac{1}{6} a^\intercal (b \times c)$$

In the following we show that we get this known formula as a conclusion from equation (3.4): As we have 3 vertices, 2 edges per vertex and one face coincident to all, there exist 6 vertex neighborhoods. Exemplarily we show how to compute the neighborhoods $(a, \mathbf{t}_a, \mathbf{n}, \mathbf{n}_a)$ and $(b, \mathbf{t}_b, \mathbf{n}, \mathbf{n}_b)$ of a and b with respect to their coincident edge. Note that $\mathbf{n}$ points into the interior of the polygon or into the backside of the triangle:

$$\begin{aligned}
\mathbf{n} &= (c-a) \times (b-a) \\
&= -(a \times b + b \times c + c \times a) \\
\mathbf{t}_a &= (b-a) \\
&= -\mathbf{t}_b \\
\mathbf{n}_a &= -\mathbf{n} \times (b-a) \\
&= \mathbf{n}_b
\end{aligned}$$

As we see, every local neighborhood has the same face normal and each pair of coincident vertex neighborhoods have identical edge normal vectors. By evaluating the formula from (3.4) we first calculate

$$\begin{aligned}
a^\intercal \mathbf{n}_a &= -a^\intercal \left(\mathbf{n} \times (b-a)\right) \\
&= -\mathbf{n}^\intercal \left((b-a) \times a\right) \\
&= -\mathbf{n}^\intercal (b \times a) \\
&= \mathbf{n}^\intercal (a \times b) \\
&= \mathbf{n}^\intercal ((a-b) \times b) \\
&= -b^\intercal \mathbf{n}_b
\end{aligned}$$

We also need all direction vectors to be normalized, so we divide them by their lengths. The vector $\mathbf{n}_a$ is perpendicular to $\mathbf{n}$ and $(b-a)$ (and so is $\mathbf{n}_b$), the length of $\mathbf{n}_a$ (respectively $\mathbf{n}_b$) is given as

$$\begin{aligned}
|\mathbf{n}_a| &= |n| \cdot |b-a| \\
&= |\mathbf{n}_b|
\end{aligned}$$

such that the volume equation (3.4) can be formed up to

$$\begin{aligned}
6 \cdot V = & \left(\frac{a^\intercal(b-a)}{|b-a|} \cdot \frac{a^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(a \times b)}{|\mathbf{n}| \cdot |b-a|} + \frac{b^\intercal(a-b)}{|b-a|} \cdot \frac{b^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(a \times b)}{|\mathbf{n}| \cdot |b-a|} \right) \\
+ & \left(\frac{b^\intercal(c-b)}{|c-b|} \cdot \frac{b^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(b \times c)}{|\mathbf{n}| \cdot |c-b|} + \frac{c^\intercal(c-b)}{|c-b|} \cdot \frac{c^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(b \times c)}{|\mathbf{n}| \cdot |c-b|} \right) \\
+ & \left(\frac{c^\intercal(a-c)}{|a-c|} \cdot \frac{c^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(c \times a)}{|\mathbf{n}| \cdot |a-c|} + \frac{a^\intercal(a-c)}{|a-c|} \cdot \frac{a^\intercal \mathbf{n}}{|\mathbf{n}|} \cdot \frac{\mathbf{n}^\intercal(c \times a)}{|\mathbf{n}| \cdot |a-c|} \right)
\end{aligned}$$

Because of the symmetric structure of $\mathbf{n} = -(a \times b + b \times c + c \times a)$ we get

$$a^\intercal \mathbf{n} = b^\intercal \mathbf{n} = c^\intercal \mathbf{n} = -a^\intercal(b \times c)$$

which reminds us of the formula we want to have in the end. We can cancel this out from the whole equation. Doing so and taking a closer look on the first two summands we can convert this subtotal to

$$\begin{aligned}
& -\frac{a^\intercal(b \times c)}{|n|^2} \cdot \mathbf{n}^\intercal(a \times b) \left(\frac{a^\intercal(b-a) + b^\intercal(b-a)}{|b-a|^2} \right) \\
= & -\frac{a^\intercal(b \times c)}{|n|^2} \cdot \mathbf{n}^\intercal(a \times b) \left(\frac{b^2 - 2 \cdot a^\intercal b + a^2}{|b-a|^2} \right) \\
= & -\frac{a^\intercal(b \times c)}{|n|^2} \cdot \mathbf{n}^\intercal(a \times b) \left(\frac{(b-a)^2}{|b-a|^2} \right) \\
= & -\frac{a^\intercal(b \times c)}{|n|^2} \cdot \mathbf{n}^\intercal(a \times b)
\end{aligned}$$

and doing so for all three pairs of the complete sum, we get

$$6 \cdot V = -\frac{a^\intercal(b \times c)}{|n|^2} \cdot (\mathbf{n}^\intercal(a \times b) + \mathbf{n}^\intercal(b \times c) + \mathbf{n}^\intercal(c \times d))$$

The term in big brackets can be formed to $\mathbf{n}^\intercal(-\mathbf{n}) = -|\mathbf{n}|^2$, which prooves the equation.

□

The correctness of this interesting approach has been shown, the efficiency still is in question. Own experiments using this technique turned out to be about half as slow as the line integral method. Most likely this is caused by the high costs of the additional arithmetic operations such as square roots that are needed to calculate the helper vectors and to normalize them.

An important advantage of this algorithm - as Franklin and Kankanhalli state - is that it is easily parallelized. It can be used for GPU computing, which is a matter of particular interest nowadays. The fact that the given input polyhedra have to be decomposed into tetrahedra makes it uneasy to implement in a straight forward way. Nevertheless this approach might turn out to be of relevance when we are to develop a packing scheme on arbitrary triangle meshes for both, pure MC or the generalization of the packing algorithm described of our work.

4 Packing scheme

We are faced with a classic problem of *combinatorial optimization*. We have an objective function - the volume to be packed - and a configuration space - the valid placements and orientations of the objects (boxes). At first glance, discrete optimization approaches might not seem applicable, as the positions and orientations of the boxes might be continuous. But the volume of a packing is given by the number and sizes of the used boxes, which is a kind of discrete metric. Further, why should we consider a set of only slightly different configurations, if they more or less describe the same packing?

By using the simplified physics engine described in chapter 2, we can produce *valid* configurations where no box penetrates another or the bounding geometry. Starting with a valid configuration, we can perform moves on it such as adding a box, rotating a box, exchanging two equally sized boxes and much more. After each move, we let the resolver try to legalize all penetrations. If this legalization was successful we continue with the next move.

By doing so we generate a so called Markov chain of valid configurations and the objective function gives evidence on the quality of each configuration. Using concepts of *stochastic optimization*, we can combine all this to an intelligent control system that decides if the current configuration is to be accepted or rejected. In the latter case, we perform another move to produce a new configuration that is legalized and rated. Hopefully the produced configurations experience a quality increase such that, at the end of this simulation, we obtain an optimal solution.

In our case, we decided to use a simplified variant of *the method of simulated annealing* that we will present in section 4. We will give an overview on the general concept and explain where our approach differs from it. Then we will introduce the set of moves we implemented to generate new configurations. The objective function to rate a configuration will be presented in section 4.2.3.

4.1 Simulated annealing (SA)

The concept of annealing has its origin in material sciences. Metals or other fluid materials are annealed in order to obtain a desired structure. The annealing has to take place slow enough such that the molecules have time and energy enough to arrange in an optimal manner.

This concept works so well in nature that it has found its way into randomized optimization in the late 70's. Simulated annealing models the cooling down process of a physical system by using the fact that, if the cooling is slow enough, the system will reach a desired optimum.

The system has a number of molecules that are in a certain configuration. The agility of the system is simulated by a set of moves that are performed to the molecules. Each move leads to a new configuration and this changes the energy of the system. It is the goal of these moves to reach a state of minimal energy.

The current energy of the simulation is rated by the objective function and as we try to find an optimal state, we have to decide if a move has to be accepted or rejected. In the concept of simulated annealing this is done by the *Metropolis criterion*, which will be clarified later.

The system also has a certain temperature τ that is being annealed over time. τ is a variable of the Metropolis criterion and regulates the degree of agility of the system. In the beginning, when the temperature is high, moves are allowed that might lead a higher energy level - what is of disadvantage for the current state but might lead to better energy situation in the sequel. The idea behind this is to overcome local optima and to reach the global optimum in the energy landscape. At the end, when the temperature is low, only slight moves are accepted to fulfill certain quality constraints like overlap free packings.

In classic SA implementations the choice of the start temperature and the degree of annealing are crucial for the quality of the result of the optimization. In our scenario where configurations always are legal it turned out that the annealing part of the SA is of minor importance for our application. What really improved our results was the Metropolis criterion, which gave us a suitable stochastic measure to decide if a move is to be accepted or rejected.

Also very important is the choice of good moves, which we will present in the next section.

4.2 Moves

The actual approach differs between 9 kind of moves and can be separated into three different types:

1. Volume increasing move
2. Volume decreasing move
3. Perturbation move

At every iteration of the algorithm, a certain move is chosen by random whereas every move has its own probability to get selected. It should be obvious that the volume increasing moves have highest probability and the decreasing moves are the last to be chosen. But deteriorations during especially the early state of the simulation are important to move freely within the energy landscape to get into a better local optimum. Therefore the weightings of the decreasing moves should not be set too low. Further the objective function will rate them lower, which results into a lower probability for this move to be accepted. But a deterioration of a legal packing will very likely be valid again and this fact again increases its probability. These last three sentences explain, why the adjustment of these probabilities is both important and challenging and can often be only done by empirical studies.

Finding the right set of moves that can be chosen from is essential for obtaining good solutions in every kind of simulation. On the one hand, the possibility to get all kinds of configurations increases the freedom to find global optima. On the other hand, producing moves that will be rejected with high probability because they decrease the quality of the configuration too much are a waste of runtime.

The moves we present here are the attempt of a trade-off between a high degree of freedom and a certain determination to find global optima in the energy landscape fast and reliable. In a later stage we fine tune to crawl up (or rather crawl down) the landscape to reach the local extremum. Then we use slightly different moves, which we will present later.

4.2.1 Volume increasing moves

The way to increase the volume of a SAE packing can be done in two ways:

Add a box The most obvious box type to add is the smallest one, the H box, because it most likely fits the best. But the technical justification to add this type comes from the SAE norm description itself: whereas for all other box types we are never allowed to use more than 2 or 4, a valid packing can contain up to 20 H boxes. Every move - as it conforms to our strategy - leads from one valid configuration to another. Adding an H box gives higher assurance to be accepted, but also adding bigger box types would give good results. The placement of a new box can be completely randomly or can use some intelligence ranging from *gluing the new box to an existing one* up to *free space detection*. The more intelligence we use the more complexity we cause. During the early state of the simulation we need fast moves, and we work with the following:

1. Pick a box and glue an H box such that the resulting two boxes have smallest surface.
2. Pick a box and glue a clone on its biggest side.
3. Pick a box, rotate it randomly by $\frac{\Pi}{2}$ around any edge and glue a clone on its biggest side.

For the later state, the so called *fine tuning phase* we make use of the free space detection mentioned before It is based on a discrete Minkowski sum and works as follows:

Discrete Minkowski sum In section 3.5 we already introduced the Minkowski Sum. We now compute a discrete Minkowski sum of an axis aligned box b and the interior of the trunk represented by points. We then get an approximation on possible placements for b in the interior of the trunk with respect to an existing box set B by the following scheme (remark that we assume that all boxes in B are axis aligned as well):

1. As a preprocess we construct a fine voxelization of the interior of the trunk. For each voxel center $\mathbf{p} \in \mathbb{R}^3$ we place an instance of b. If this instance intersects the trunk geometry we discard $\mathbf{p}$, otherwise we store it in a placement set S.
2. To find a placement for b with respect to B we check for all points $p \in S$ if $p \in B \oplus b$, reject those that are contained and obtain possible placements. The set $B \oplus b$ is the element-wise Minkowski sum of all boxes $a \in B$ with b, and - as we talk about axis aligned boxes - we easily get it by extending all boxes in B by the half side lengths of b.

Blow a box up The SAE standard knows 7 different box types, and we can sort them according to their volume from type H, which covers the smallest volume, to the biggest type, the A box. By exchanging a chosen box with its successor we increase the quantity of the packing and hope at the same time that the box still fits after the legalization move.

Sometimes we are not allowed to blow a box up to its direct successor. For example we already have two boxes of type C and by blowing up a F box, which is the predecessor of a C box, we would exceed the maximum number of allowed C boxes, which is 2. In this case we could reject the move directly, because all configurations have to be valid at every time of the simulation. But this would be a waste. Instead we blow up as long as we have an invalid situation or until the biggest box type is reached. If this still is an invalid situation we reject the move.

The idea of a *blowing up* move comes from a former work (Baumann et al. (2007)), where we used a $2d$-variant of the physics engine to pack different sized circles such that the circumcircle is of minimum radius. The boxes do not have the same dimensions, e.g. the A box covers the biggest volume, but the C box is 2 inches longer. So it is questionable, if another order might work better in special situations, for example if the container is shaped very unusual. But the goal is to increase the volume and hence this natural order of boxes is both applicable and successful.

4.2.2 Volume decreasing moves

As mentioned before, these moves give more flexibility to move through the energy landscape during the simulation. But this weapon has to be used very carefully; we won't reach peaks in the landscape if we give up too early. As for the increasing case, we again differ between two types of decreasing moves:

Delete a box This simply chooses a random box and removes it from the actual packing. The volume of the deleted box affects the objective function and a removal of a big box will be rejected with more probably than of a smaller box. But the deletion of a big box like an A type still can happen and this would cause a tremendous change of the actual packing. We therefore restrict the deletion move only on the smallest occurring types. This method also makes sense concerning the fact that the creation of a new box also starts with the smallest type.

Shrink a box This move just is the inversion of the blowing up move. Choose a box and shrink it to the next smaller type. If the packing would become inconsistent according to the number of allowed boxes per type we go on shrinking until an allowed box type is found. In very rare cases this might convert a big box like type A into a quite small one (F or H). The objective function of cause will reject this move most likely and we do not restrict this possibility any further.

4.2.3 Perturbation move

For classical MC these moves are essential to lead into optimal situations because overlaps flow into the objective function as penalties. In this scenario perturbation moves alter the objective function and illegal situation would become more and more legal just by applying and accepting millions of those moves.

In our case, perturbation moves play another role, because the resolving of illegal situations is done by the physics layer. This also performs millions of small alterations to legalize the penetrating boxes but with a certain determination. This step is separated from the move and not visible to the objective function. There is no penalty and no decreasing of the quality of the packing after the move. Therefor a perturbation move does not alter the volume. The only reason for rejecting this move is that the packing is illegal, meaning that the physics engine failed to legalize the situation. In this case the move would have to be rejected then.

The moves we use make significant alteration in the structure of the current packing. A picked box gets rotated by multiples of $\frac{\Pi}{2}$ around an edge, or two random boxes are swapped. This is more or less what a human packer would also do; exchange items or arrange them differently in the trunk such that the lid can be closed.

Rotation move Our items are boxes, and the most space saving way to arrange two boxes is to stack them side by side. Therefor we restrict our rotations to multiples of $\frac{\Pi}{2}$, and the rotation axis is chosen from one of the three edge directions of the box by random. The shorter the chosen edge is, the more likely the rotated box will interfere with other boxes during the legalization step and the move might be rejected easier. So far we do not take this into account.

Swap two boxes In an early approach we admitted the swap of two arbitrarily boxes. We found out that, if the two boxes differ too much in volume and dimensions, the swap move is very likely to be rejected because the physics engine would not be able to resolve the situation. Again it is clever not to alter the configuration too much and thus swap two box types that are similar in volume and dimensions. We again use the natural order of the boxes' volume to find a suitable swap partner for a chosen box.

Finding the suitable box partner for a swap takes linear time. For large number n of boxes we might want random access to a box of a given type $t \in T$. A hash table with keys T then might be suitable. For big containers like trunks of vans or pick-up trucks, this really improves runtime. For common instances, like trunks of passenger cars, the disadvantage of managing insertions and deletions of the hash map doesn't justify the efford.

4.3 Objective function

We reject a move if one of the following criteria fail:

1. Consistence according to the SAE J1100 standard
2. Legalizability by the physics simulation
3. Metropolis criterion

SAE-consistence The first is the easiest test: The allowed numbers of box types have to correspond to the description in the SAE-J1100 standard (see SAE (2001)), otherwise the packing would not be valid.

Legalizability The second has been discussed in the last section: The physics simulation needs a certain amount of time to legalize the packing by performing thousands of small translations and rotations. There should be a decline of the pairwise penetrations, otherwise we might run into oscillating situations where no remarkable improvements take place. We decide this with a statistical evaluation by the variation of the overall movements (see section 2.4).

We also restrict the maximal number of moves if the prior test should fail. Further, in some situations, penetrations - thus the occurring forces - are that high that items might *tunnel* through the boundary of the container. In

this case we might obtain legality according to the physics but a part of the packing lies outside of the container. To discover this quickly we describe the outside of the container by a discrete grid and test the centers of all items if they lie in an outside grid cell.

In the early state of the simulation, we might want to allow illegal configurations up to a certain ε that we decrease during the simulation - comparable to the classical MC at high temperature τ. This can be achieved by setting the upper bound of allowed pairwise penetration depth ε high at the beginning and lower it over time. This improves both, the flexibility of the system and the runtime of the physics simulation to obtain an ε-legal situation.

Also, for early state scenarios or for pre-packing purposes, we can restrict the movements to only translational alterations. Then the whole simulation is much faster for boxes as they are axis aligned and penetration depth computation is only a comparison of three intervals.

Metropolis criterion The metropolis criterion tells us the probability to accept a move even if it leads to a smaller volume. It is a function ζ depending on the actual temperature τ, the difference of volume ΔV and some constant factor k, often denoted as *Boltzmann constant*, whereas in many practical applications it is chosen as $k = 1$.

$$\zeta = e^{\frac{-\Delta V}{k\tau}} \tag{4.1}$$

If the new configuration leads to a bigger or equal volume, ΔV is smaller or equal 0 and ζ is greater than 1, so the move will be accepted. Otherwise ζ is smaller than 1 depending on the deterioration of the packing. This means slightly smaller differences are accepted more easily than bigger ones.

The metropolis criterion give a very good measure to decide if a configuration has to be accepted or not. It has its practical justification because it describes the optimization process in real physical scenarios. Further, it has been shown in Metropolis et al. (1953) that the global optimum is reached if we anneal the temperature infinitely long, which of course is only a theoretical justification for this method.

Now the packing strategy is described as follows: Perform a random move, start the legalization and decide whether to accept or reject this move. Throughout the simulation we save the best solution seen. If we can define an upper bound U of the trunk volume (e.g. its continuous volume) we are able to perform a *fine tuning* phase when the volume of the packing is sufficiently

close to U. In this phase we disable the decrease of those boxes that define more than half of the covered volume. This is comparable to a simulated annealing approach and helps to lead the configuration into a local maximum. In our experiments the fine tuning starts when the best packing covers a volume more than $0.8U$. This bound has been found empirically; the size of the packing grows quite fast in the beginning of the algorithm but only few configurations reach a quality above $0.8U$. Those configurations are capable for a fine tuning.

To obtain faster simulations between two moves we use the point cloud representation of the trunk during the packing. In our experiments we could increase the moves-per-second-coefficient by about 10 compared to the triangulated representation, as the move rejection criterion responds faster. At the end of the packing we legalize triangle based to achieve a higher precision.

4.4 Use of more than one luggage set for bigger trunks

Due to the number of items allowed in the *Standard Luggage Set* the so far described strategy works fine on conventional trunks as we find them in most passenger cars. A `SAE`-set contains - if it is fully exploited - 36 boxes and occupies a volume of 28.7049ft^3 (approximately 813 liters), which is sufficient to approximate the trunk volume of most passenger cars. Nevertheless we want to use our strategy on bigger trunks to provide an all-embracing quality comparison to handmade packings and to the continuous volume. According to SAE (2001) we are allowed to use further standard sets:

> *Pieces from subsequent Standard Luggage Sets may be used when the previous set is placed in the Luggage Compartment.*

This means that we are only allowed to start a new standard set when all boxes from the previous set are exhausted. The smallest boxes, namely the H-boxes are excluded from this, as they usually are packed at the very end as a kind of *stuffing*. The actual SAE description leaves space for interpretation here and the fraction of the shoe-box-sized H-boxes is small anyway.

Previous implementations for the SAE variant as Althaus et al. (2007), Cagan and Ding (2003) or Tiwari et al. (2010) fail to solve this problem according the SAE standard because of different reasons. The first cannot cope with bigger trunks due to complexity reasons. The two others are Monte Carlo

and/or evolutionary algorithms and allow illegal configurations during the computation that are completely resolved only at the end. But to be allowed to start a second luggage set they would have to know in advance how many boxes can be packed. This would only work in several packing steps and would need more optimization to obtain dense packings.

The advantage of our strategy is that our configurations are always legal - no overlap and SAE conform. If we exhaust a standard set, we know that we can start with the next one. The only difference to the already existing implementation is that we have to make sure that the previous set(s) are still *placed in the Luggage Compartment*. To ensure this we mark all used boxes at the time we reach the exhaust (H boxes excluded!) and we prohibit those moves that would alter their type, such as *blow*, *shrink* or *delete*. All other moves still are valid, and for all new added boxes from the subsequent set, there is no restriction on moves.

The so computed packings fulfill the same quality requirements than the packings for smaller trunks, and this strategy is outstanding as this approach is the first that is able to also fulfill the standard requirements for bigger trunks.

5 Experimental results

The following chapter will summarize the output of the described algorithm. We will compare our results to existing approaches like Althaus et al. (2007), Cagan and Ding (2003) or Tiwari et al. (2010). The last is the most recent comparison as the author of this work had the opportunity to interchange data with Tiwari et al.. We also show the experimental dependencies between runtime, packing quality and input size. Latest at this point we may ask how packing quality can be defined. In Tiwari et al. (2010) the authors use the coverage of the packed volume in contrast to the continuous volume of the inner part of the trunk. As this seems to be a good measure, we further may ask what are reasonable coverage values in a real world application.

5.1 A word on coverage

As *coverage* of a packing we denote the volume that all boxes in this packing occupy divided by the continuous volume of the container. The container volume can only be approximated by computing a fine voxel representation of the inner part. In our work we used voxels of $0.75in$ (about $2cm$).

The main reason why the coverage is bound by a value less than 1 is the fact that we try to pack a rectangular item into a not-rectangular container. The edges of the trunks are often chamfered and sides - like the back seat side - are aslope. Especially arcuate faces lead to unsatisfying packing results. Here we realize that the standard box set of the SAE (2001)) norm seems not be not the optimal choice to reflect the packable region of a trunk. Luggage pieces are very often chamfered as well and give a better fit. To reflect this behavior the DIN 70020 norm has a further interpretation with equally sized boxes with a chamfer of $10mm$, which increases the coverage a lot for the DIN variant of the trunk packing problem.

It is obvious that packings with coverages of 100% are not possible for trunk as they are designed in nowadays passenger vehicle companies as their design has to satisfy many other aspects than packability. The few manually packed

models we were able to consult give maximal coverage of 76%, the median is less than 71%. Other publications like Tiwari et al. (2010) or Althaus et al. (2007) give similar bounds. But can we show theoretical upper bound on the coverage depending on the structure of the boundary? The following section tries to answer this question.

5.1.1 The standard container

Let us denote a standard container given by a cube of dimension $(d \times d \times d)$. The biggest box that would fit into this standard container would be the cube itself with a coverage of 1. If we now alter the shape of that cube how does this influence the possible coverage? We try to define two possible scenarios as they exist in real trunks and measure the change of coverage depending on the alteration; in the first scenario we add a chamfer of certain radius r to the edges and corners of the standard container. In a second scenario we bend a side of the cube by a certain angle α to reflect the back seat of most trunks.

Chamfered edges

Just imagine a cube of size d^3, whose edges are chamfered by a radius $r < \frac{d}{2}$. Like this it becomes the Minkowski sum of a smaller cube of size $(d - 2 \cdot r)^3$ and the sphere with radius r.

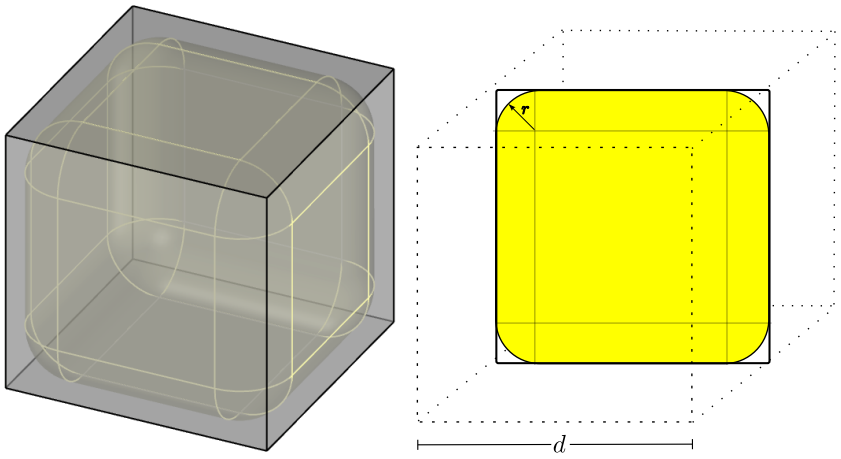


Figure 5.1: A cube of length d with chamfered edges of radius r. The yellow region denotes a slice of the volume. For $r = 0$ we have the regular cube, for $r = \frac{d}{2}$ it degenerates to a sphere.

Assuming that we try to fit in a box with dimensions greater than r, we might have two possibilities to define the packable region of this *volume swept cube*. One would be to fit the biggest cube into it, which is the one whose corners touch the spherical caps of the container (figure 5.2, left). A better approximation of the packable region might be the following: A human packer would pack a box as close to the border as possible. A second box would be stacked on top and could be pushed closer to the border then. The loss would be the rounded regions (as in figure 5.2, right).

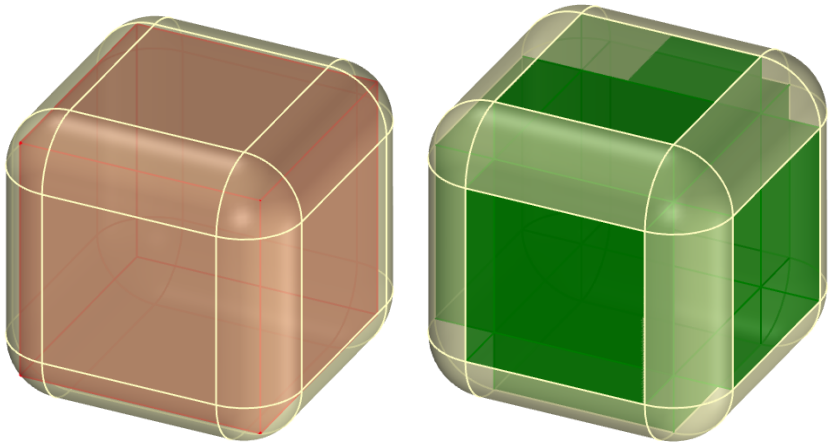


Figure 5.2: Two possible definitions of the packable region of the chamfered cube. The right one is a more optimistic approximation for smaller r as it degenerates to two lines for $r = \frac{d}{2}$ whereas the left one is a cube for any r.

The fitting cube in the interior on the left side of figure 5.2 has a side length of

$$l(r) = 2/3 * r * \sqrt{3} + d - 2 * r$$

For the volume of the second region we just sum the volume of the cube with side length $(d-2\cdot r)$ and of 6 boxes with dimension $(d-2\cdot r)^2 \times r$. The volume of the chamfered container is the sum of the inner second region's volume with the sphere with radius r and three cylinders of radius r and length $(d-2\cdot r)$. As mentioned the second approximation is more optimistic for $r < 0.25 \cdot d$ as shown in figure (5.3).

The next questing we ask is, how the coverage behaves depending on r. We assume small r and therefore choose the more optimistic approximation of the packable region in green. Figure 5.3 shows the coverage depending on the chamfer radius r. If we choose $d = 1m$ we get container that varies between

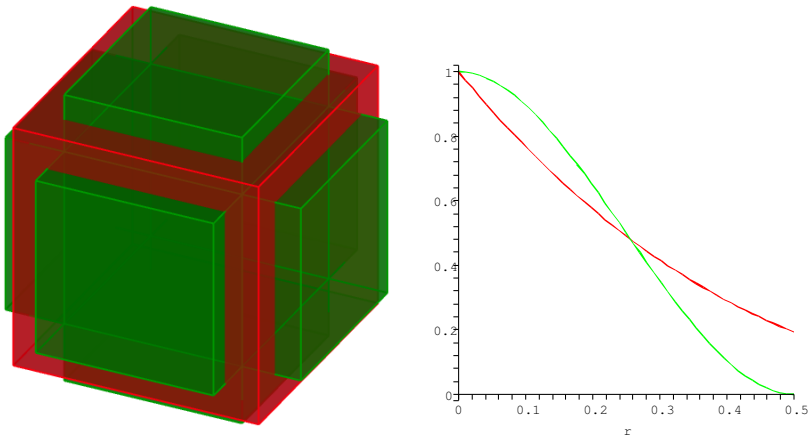


Figure 5.3: The comparison between two interpretations of the two packable region of the standard container depending on chamfer radius r. At $r \approx \frac{1}{4} \cdot d$ the red approximation outperforms the green one that is more optimistic for smaller r. If we choose the pessimistic red one we see at $r \approx 0.15 \cdot d$ that we reach a coverage of about 80%

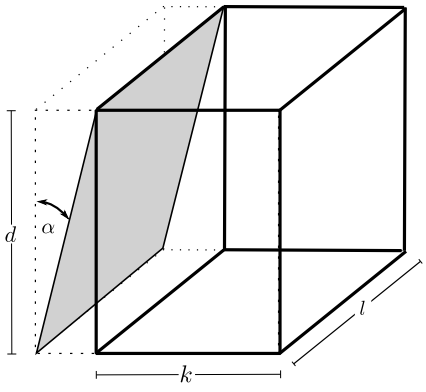
$1000 \approx 35\text{ft}^3$ and $523 \approx 18\text{ft}^3$. This is comparable to a bigger trunk. In this case the coverage of 80% can be reached at a radius of about $15cm$. The radius r scales with d. For a middle sized trunk of $343 \approx 13\text{ft}^3$ we would choose $d = 0.7m$ and would obtain a coverage of 80% at a radius of $10\frac{1}{2}cm$.

Standard chamfer radii in automotive design lie between $5cm$ and $15cm$. For middle sized trunks the ratio $\frac{r}{d}$ then lies between about 0.07 and 0.21 and gives coverages between 96% and 76%. Good real life packings for middle sized trunks reach 76% and more. This is not too far away from the upper bound we would expect from our model. In literature (Tiwari et al. (2010),Cagan and Ding (2003)) we read that bigger trunks lead to better coverage. This observation can be confirmed by our standard container model; for increasing d the ratio $\frac{r}{d}$ decreases and hence better coverages can be obtained. The mentioned theoretic value of 80% coverage for big trunks at a chamfer radius of $15cm$ has not been reached in practice.

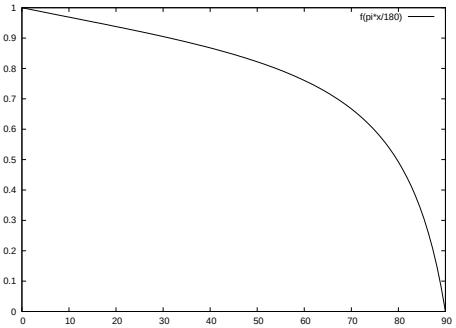
Aslope sides

The sides of a trunk usually are not at right angle, the boot lid for example may be crooked and most of all the backseat has an incline that usually reaches into the interior of the trunk. The loss of coverage in this case is

mainly depending on the proportion of the box dimension we want to pack. In figure 5.4(a) we have a proportion of $p = \frac{d}{k}$ between the hight of the box and the length.



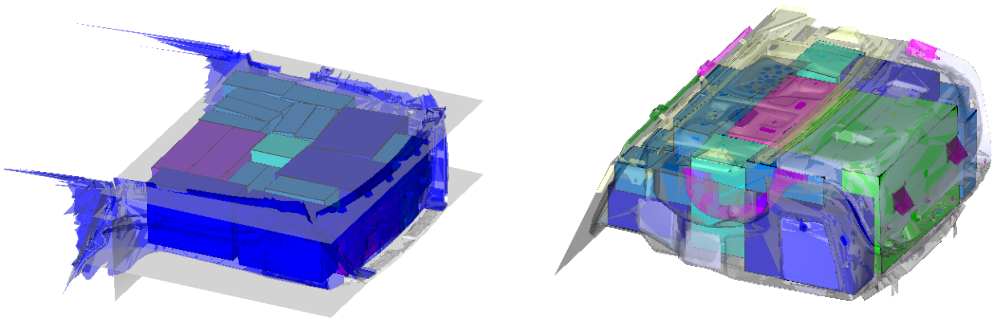
(a) The container is aslope on one side. As the rotation axis of the slope plane is parallel to the box depth, the coverage is only depending on the height d of the box and length k.



(b) The graph shows the dependency between the coverage and the angle of the aslope side. We have chosen $k = 2.75 \cdot d$. The usual backseat angle in cars is at about 26°. This corresponds to a maximal coverage of about 90%.

Figure 5.4: The change of the container shape compared to the coverage.

Note that only the height d and length k play a role for the coverage here, the depth l cancels out. The smaller p the better the coverage is. To find realistic values for d and k we compare the proportions of the boxes. These proportions differ between $1 \times 1.125 \times 1.875$ (type `E` box) and $1 \times 2 \times 3$ (type `F` box). If we weight each box with the maximal number allowed according to the `SAE J1100` standard we get a medial proportion of $1 \times 1.58 \times 2.75$. To be as pessimistic as possible we choose the extremal proportion $\frac{d}{k} = \frac{1}{2.75}$. In figure (5.4(b)) we see the graph that shows the dependency between coverage and angle for the mentioned proportion. The usual angle for backseats in cars (27°) corresponds to a coverage of about 90%. We do not reach this value in practice due the complexity of a trunk in real life. It is also perhaps too pessimistic to choose the proportion between the shortest side and the longest side, as an experienced packer would always try to pack the box such that the longest side is parallel to the slope edge. In this case, $k = 1.58$ would be a more optimistic approximation, which gives a coverage of 86%. This is closer to our best packings of about 78% coverage.



(a) Model with 18.09ft^3 continuos volume obtained the best coverage with 81% and 76% in average.

(b) Model with 17.655ft^3 continuos volume reached only 69% average coverage (best 73%).

Figure 5.5: The left trunk geometry (fig. 5.5(a)) is almost like a cube with three planes that cut the top, bottom and backseat side. It has almost no curves and no aslope faces. The right geometry (fig. 5.5(b)) has many curved patches at the back, the sides and even on the bottom and also the backseat side is aslope. As we see by means of the best packings, loads of space is left unused on the right trunk.

In opposition to Tiwari et al. (2010) we do not find a straight dependency between the continuous volume of a trunk and the coverage of the packing; in deed we can confirm that the smallest trunks give the worst coverage. But for big trunks of 25ft^3 and more we get varying coverages between 60% and 72%. What we can confirm - and what is also concluded by our simple model - is that we obtain best coverages of 78% for trunks whose shapes are more rectangular than others. The worst coverage values we obtained for trunks with arcuated lids; but this again can be considered as a conclusion from our standard model as we can interpret these lid bows as container edges with huge chamfer radii.

5.2 Scenario 1: Straight comparison of two different approaches with the same input data

We were in the lucky position to interchange data with Tiwari et al.. This gave us the possibility to compare our results directly. The approach described in Tiwari et al. (2010) is an evolutionary algorithm (EA) and gives very impressive results.

5.2.1 The evolutionary algorithm

The EA algorithm works with a number of configurations (the population of chromosomes) that are altered by moves (genetic variation operators). Moves that alter a single chromosome are called *mutations* whereas the fusion of two chromosomes into one new chromosome is called *crossover*. The later, namely the possibility to produce an offspring out of two configurations is the main difference to our technique and can be seen as a major advantage of this heuristic strategy. To be able to perform these crossovers, the authors separate the packing strategy into two independent phases, the *optimization* step and the *layout* step.

The chromosomes only hold the information about the orientation and permutation of the items. After a genetic variation in the *optimization* step, the items are placed on an fine grid of $5mm$ spacing that represents the packable region. This is called the *layout* process and the placement of the items is done straightforward in the order they appear in the chromosome. They start from the outermost bottom-left-back corner and propagate from here to the next possible placements, hence it is called the *bottom-left-back-fill (BLBF)*. This fill is performed until either no valid placement can be found for the nex item in the order or until all items fit inside the voxelization. After the layout process the actual chromosome is rated according to the packing efficiency and depending on this further classical genetic variations are performed to produce new configurations.

For the SAE-variant of their packing algorithm, items are boxes and orientations are orthogonal such that the *orientation gene* in the chromosome is an integer between 1 and 6. A typical chromosome of four items looks like this:

$$\begin{pmatrix} 1 & 3 & 4 & 2 & | & 2 & 5 & 3 & 3 \end{pmatrix}$$

The first numbers are a permutation of the order of the indices, in which the boxes are placed in the layout phase. The second half of the numbers define their orientation and only 6 different orthogonal orientations exist for a box. A mutation of this chromosome swaps randomly within the first half and alters randomly in the second half. A crossover between two chromosomes is done by *order-based crossover* for the first half of index permutations and *one-point crossover* for the orientation part. These are standard techniques in evolutionary algorithms and for further information we refer to Tiwari et al. (2010).

In a nutshell, the method values orthogonal box placements on an fine grid of $5mm$ spacing that represents the packable region. This grid is computed in advance and the algorithm's complexity is independent from the number of triangles of the input data. Instead it is is depending on the resolution of the voxel grid and as the author states out almost 90% of the overall runtime is spent for placement and overlap computations within the grid. Due to our observation their voxelization is not conservative, as they regard surface voxels to be inner voxels. They therefore obtain a slightly bigger trunk, and each box in the result may penetrate the boundary by the mentioned spacing of $5mm$. Further, placements are allowed to overlap by one voxel. This results in packings with pairwise box/box overlap of again $5mm$. On inquiry the author told us that this behavior is desired as it reflects the real world of flexible trunk sides and luggages.

To compete with this approach a tuning of our algorithm would be needed; as we normally allow maximal penetration depths of $\varepsilon = 10^{-3}mm$ in a physical resolving step, we choose $\varepsilon = 5mm$ here. This of course reduces the number of resolving steps during a move and leads to more moves per second.

The data we worked on is a re-meshed model produced by an in-house application described in von Dziegielewski et al. (2012). It is a Delaunay-triangulation of $25K$ triangles, it is waterproof and it is a $2-$manifold. We had to transform the input data in that manner because the EA only accepts waterproof input meshes.

The table 5.1 shows the comparison between both results. EA denotes the evolutionary algorithm from Tiwari et al. (2010), our approach is denoted as RI (random improvement):

We made 6 attempts of 5 hours each, they were done in parallel on a 6 core CPU. We assume that the EA is a parallelized implementation but we do not have information on that.

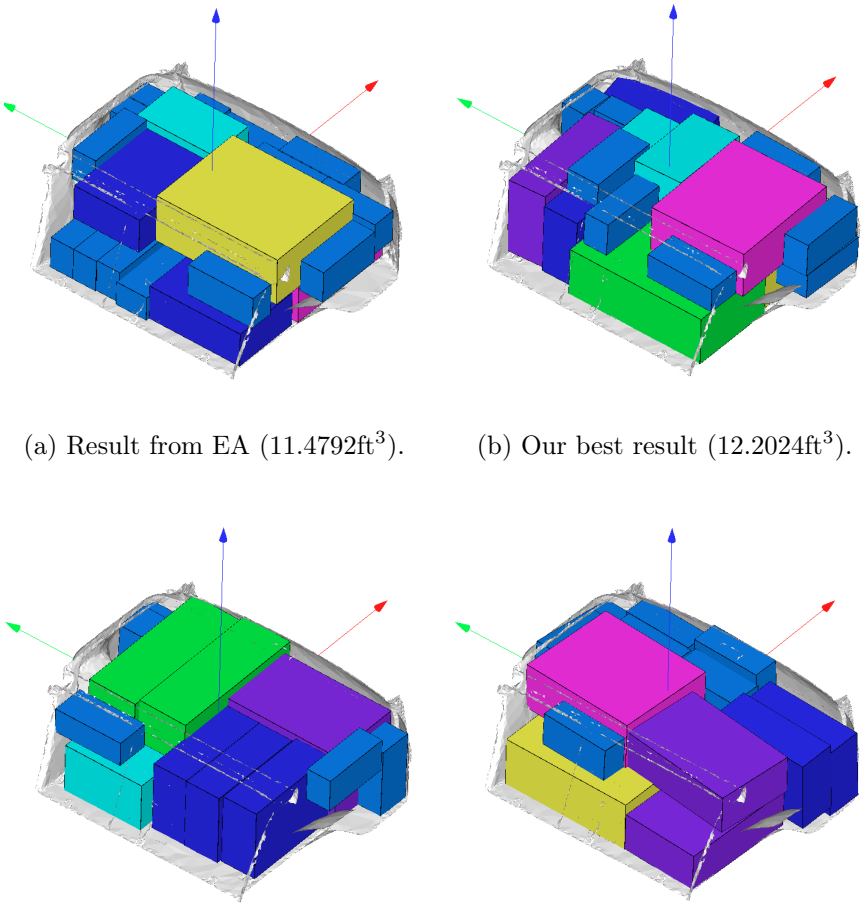
The row *packing efficiency* shows the differing calculation of coverage; as the

Algorithm:	EA	RI
Runtime:	$1 \times 24h$	$6 \times 5h$
Volume:	11.479ft^3	$10.9809\text{ft}^3 - 12.2024\text{ft}^3$ ($\emptyset 11.5053\text{ft}^3$)
Worst-case overlap:	$5mm$	$5mm$
Number of items packed:	21	$12 - 16$
Packing efficiency (Tiwari)	69.27%	n.n.
Packing coverage (our calculus):	78.42%	$75.02\% - 83.36\%$ ($\emptyset 78.59\%$)
Orientation:	Orthogonal	Prepack: Orthogonal Post-opt: Free rotations

Table 5.1: Results of two different packing schemes on the same input data. The evolutionary algorithm (EA) is described in Tiwari et al. (2010), the random improvement (RI) is the implementation described in this work in chapter 4.

authors denote their coverage as 69%, we get 78% for the same packing. As we did not really get an explanation from the authors, we can only guess the following: Our packable volume is computed by a *conservative* voxelization of $75in \approx 20mm$ spacing against a non-conservative voxelization of $5mm$ spacing. The detected free space in our model is underestimated, whereas in Tiwari et al.'s model, it is overestimated. Further we believe that the author's coverage is the fraction of covered voxels compared to all voxels. As boxes may overlap, one voxel may be covered by more than one box, but it is counted only once. We sum up the volumes of each box and divide it by the packable volume. Altogether this leads to a more pessimistic approximation of the coverage for the EA, or, if you like, a more optimistic one for us.

Comparing the placements of the different packings (see figure 5.6), we determine that the EA produces visually dense packings with a high number of small H-boxes (15), which results in a high number of items in the trunk (21). This might be seen as a violation of the SAE J1100 standard, which allows H boxes to be placed only if no other box fits into the compartment store. In our implementation, the addition of H-boxes is done at the end as part of the post optimization process. We hence obtain smaller numbers of boxes of type H (max 8) and also smaller numbers of packing items (max 16). If we compare our results with each other, we remark that the ability of free rotation is a crucial advantage to obtain good packings.



(a) Result from EA (11.4792ft^3). (b) Our best result (12.2024ft^3).

(c) Our worst result (10.9809ft^3). (d) Another result (11.9434ft^3).

Figure 5.6: Comparison of the result from the EA (5.6(a)) and three from our approach. The best packing (5.6(b)) exploits the free space at the round trunk lid - an advantage of the full freedom of rotation. As we see the worst packing (5.6(c)) does take only little advantage from free orientations compared to a much better one (5.6(d)).

Conclusion: Our approach is faster and more flexible and obtains better results. It prefers placements of bigger boxes, which - from our point of view - reflects the behavior of a real world packer in a better way. It can exploit more free space due to placements with free orientation. On the other hand the runtime of our approach depends much more on the input size. The success of evolutionary algorithms in all kind of optimization processes is justified and

we really appreciate the authors work and thank them for the collaboration on this little competition.

5.3 Scenario 2: Real world performance tests on 50 models and comparison to former approach

In the previous section we have already seen that the algorithm fulfills quality requirements that can compete with existing approaches on the market. Another approach we referred to (Althaus et al. (2007)) is an in-house application, called the MKS algorithm as is based on a continuous Minkowski sum computation and it was state-of-the-art at the time of its publication. From our industrial partner we got a pool of input data models that we could run our approaches. It was meant to help us developing an usable real world application but it also gives us the possibility to perform all-embracing performance tests and comparisons with the MKS approach.

We had the chance to run our algorithm on a linux cluster with 162 nodes with 2 AMD Opteron 2384 quad core CPUs @ 2.7 GHz each. Like this we were able to run our algorithm on every model 50 times. The runtime for each experiment was $4h$, which empirically proved to be a good trade-off to obtain high quality packings in acceptable time.

Although most automotive companies design their models with free form surfaces, better known as NURBS-surfaces, the models we got are described as triangle soups. These models are very different in shape, (continuous) volume and description size (number of triangles). The later is the consequence on different tessellation thresholds and differs between about $5K$ and $500K$.

The models can be distinguished into three classes according to their continuous volumes; the small ones usually designed for cabriolets or coupés have volumes less than 10ft^3. The large models we find in transporters, vans or station wagons have trunks of size bigger than 20ft^3. The class with the most representatives, which we usually find in personal cars, have volumes between 10ft^3 and 20ft^3. The standard luggage set defined by the SAE-J1100 standard suffices for those models. The larger models often need 2 or more luggage sets.

In figure (5.7) we see the variance of the coverage over all models in green. Colored in blue we see the middle sized models that show a more or less Gaussian distribution around 72% coverage. In yellow we see the large trunks,

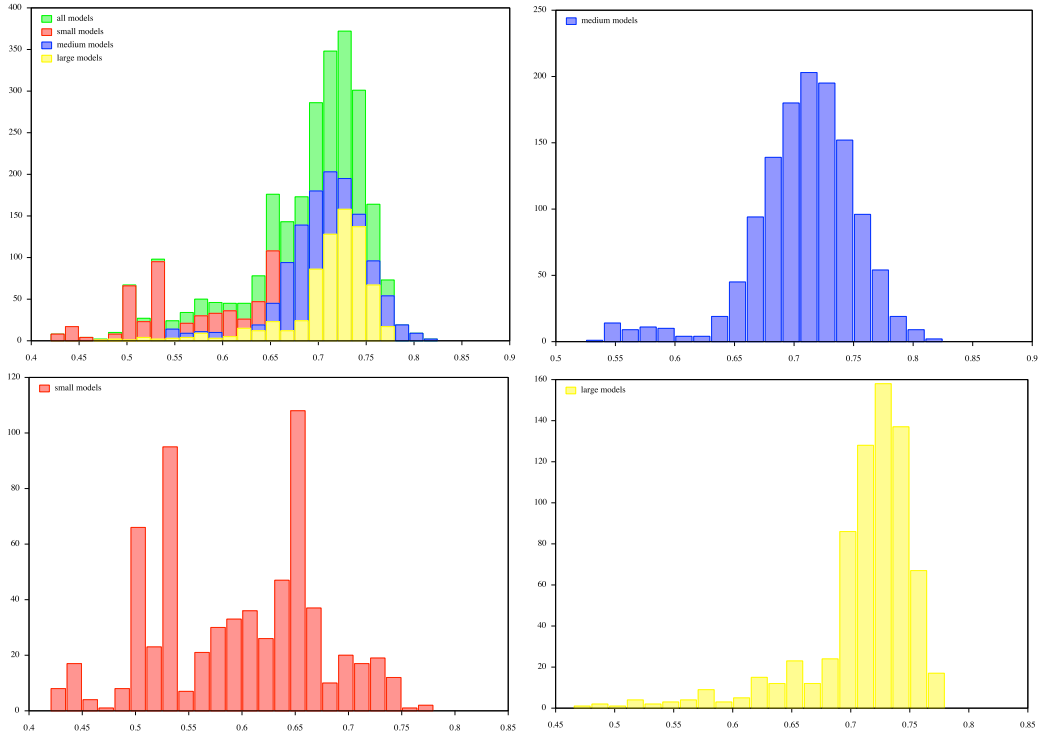


Figure 5.7: Histogram of the coverages of the random improvement algorithm on about 2500 attempts. The different colors represent the different trunk classes.

which are distributed at around 74%, a bit higher than the middle sized trunks. For the small trunks (red) we see three tremendous peaks at 50%, 57% and 65%. We would have expected a distribution with a median smaller than for larger trunks, so the value 65% is not surprising. The other two peaks are caused by exactly two models (D) and (F). They are very small, and hence only a few smaller boxes fit into them. Hence only few permutations of boxes are possible. Model (D) usually contains two boxes, one of type `E` and one of type `H`, model (F) usually has two more `H`-boxes. For the exact results we refer to the following table 5.8.

In table 5.8 we see our quality results. We also see the number of triangles per model (#TRI) and their continuous volume (VOL_C). The medial coverage (MED) and the extrema (MIN/MAX) are given and for statistical reason we calculated the weighted σ-value on all experiments per model. The results from the former MKS approach is given, as well as hand made packings (HAND.) If we compare our approach to the MKS algorithm we see that on average our results are more than 5% better. Still we obtain only about 92%

	#TRI	VOL_C (ft^3)	MED (%)	MIN (%)	MAX (%)	$SIGMA/VOL_C$	MKS (%)	HAND (%)
A	50000	11,19	70,18	67,00	74,65	0,017	66,13	67,19
AA	55402	10,89	73,41	69,70	76,68	0,017	71,33	66,89
AA2	128343	2,41	54,47	44,78	57,66	0,038	no data	58,41
AB1	157120	68,57	73,78	71,73	75,54	0,010	no data	72,91
AB2	111613	34,52	72,16	69,39	77,77	0,023	no data	90,52
AB3	98002	32,35	71,21	65,89	74,36	0,016	no data	88,35
AB4	55702	17,89	75,84	71,62	81,55	0,022	75,44	73,89
AB5	44213	5,78	56,08	48,45	60,07	0,031	31,42	61,78
AB6	31360	4,16	53,08	47,10	58,05	0,022	43,63	60,16
AB7	32234	3,79	49,76	44,98	55,53	0,022	46,44	59,79
AC	217858	11,79	57,27	53,47	62,50	0,022	59,23	67,79
B	45334	15,50	70,94	64,69	75,96	0,023	62,09	71,50
C	68337	19,39	70,93	67,12	74,02	0,016	60,23	75,39
D	123035	2,92	47,99	42,85	62,56	0,039	42,85	58,92
F	116042	1,53	53,87	53,87	53,87	0,000	no data	57,53
G	52053	3,21	64,34	52,36	70,78	0,025	64,54	59,21
H1	64374	6,92	63,95	59,11	59,11	0,025	57,80	62,92
H2	99350	9,93	63,33	58,61	67,82	0,025	68,12	67,82
I1	50014	8,64	61,71	55,57	66,65	0,026	67,03	62,29
I2	231902	12,50	68,07	63,26	73,71	0,024	69,62	68,50
J	3449	12,30	72,33	69,23	77,00	0,018	72,78	81,31
K	98812	15,33	66,00	62,99	70,26	0,015	60,41	71,33
L	61890	14,71	73,43	70,85	77,65	0,016	70,64	70,71
M	59767	16,40	71,86	67,73	79,93	0,021	68,39	72,40
N1	210197	16,33	68,78	65,44	73,16	0,017	64,86	72,33
N2	220864	16,51	69,87	64,71	73,90	0,023	65,50	72,51
N3	191040	17,74	73,39	68,30	76,69	0,019	60,34	73,74
N4	205287	18,07	74,10	69,68	79,70	0,019	69,51	74,07
O2	33494	18,19	68,72	64,40	72,78	0,019	64,26	74,19
P	107117	28,20	73,84	69,62	77,99	0,020	no data	84,20
Q	99681	46,45	67,64	57,13	76,43	0,067	no data	75,35
R	114350	66,28	75,11	73,54	77,06	0,009	no data	75,44
R2	199246	66,56	74,28	72,61	77,28	0,011	no data	75,11
R3	43577	29,93	72,78	65,28	76,08	0,019	no data	85,93
S	173799	77,03	72,71	71,30	75,05	0,010	no data	77,89
S2	111983	38,14	66,98	62,74	72,42	0,027	no data	94,14
S3	104443	38,38	66,69	61,32	71,97	0,035	no data	94,38
S4	62183	18,09	75,75	70,63	80,76	0,020	71,18	74,09
S5	55307	9,57	64,72	62,60	66,38	0,008	62,78	65,57
S6	38027	6,39	67,03	63,09	69,35	0,022	57,46	62,39
T1	72861	14,92	69,48	65,79	75,13	0,020	66,52	72,08
T2	65697	14,37	66,76	61,82	70,71	0,020	70,25	73,19
T3	42182	18,53	73,45	67,24	80,22	0,030	73,19	75,73
T4	56403	17,65	69,42	65,16	73,43	0,020	61,24	79,47
U	49990	14,15	71,08	66,72	76,01	0,023	61,36	70,15
V1	9703	17,70	72,53	69,15	76,61	0,017	70,73	73,70
V2	4852	17,79	70,69	66,30	75,06	0,020	63,96	73,79
W	50052	47,80	71,61	68,42	76,08	0,022	no data	73,22
X	128486	18,93	76,46	72,10	81,58	0,020	83,26	74,93
Y	593525	33,10	72,00	67,38	77,33	0,020	61,51	89,10
Z	47969	9,71	73,18	71,20	77,56	0,014	75,23	65,71
Ø			66,83	62,42	71,16	0,02	63,75	72,60

Figure 5.8: Results from 51 models over 50 attempts per model

of hand made packings, but this value is questionable as we only refer to 6 real packed models, the other are estimated from the continuous volume. We computed the σ-value as the square root of the variance, and weighted this value by the continuous volume to get an idea on the relative spread of about 2%. Former experiments as well as our theoretical examination in section 5 show that good real world coverages lie between 70% and 75% depending on the model. With a medial value of more than 71% our best results we lie within these bounds.

5.4 Scenario 3: An experimental attempt to estimate the runtime on a specialized model

To estimate the time, the algorithm needs to reach a certain quality depending on the input data, we took a model with 3445 triangles. Then we constructed new models from this by successively doubling the number of triangles. We did this by simply subdividing each triangle at its longest side into two triangles. With this method we produced 8 models describing the same geometry, but with different number of triangles starting from $3k$ triangles up to over $400K$ triangles.

As the runtime of the simulation is given in advance (in our experiments usually $4h$), we are interested in the time the algorithm needs to reach a certain quality, given by the coverage of the packing compared to the continuous volume, which obviously is the same for all 8 models. It is clear that not every attempt reaches 75%, so we also look at 50%, 60% and 70% as well. As a polynomial coherence would not be a surprise we choose logarithmic scales in our diagrams shown in figure (5.9). A polynomial function $y = a \cdot x^b$ becomes a linear function $Y = \log(a) + b \cdot X$ under logarithmic scale. To get an impression we also draw this polynomial function. The values a and b had been chosen such that it follows the gradient of our data points the best.

What we see in figure ((5.9)) is that for 50% or 60% coverage, there seems to be a polynomial correlation between runtime and number of triangles, and the polynomial function seems to lie in $O(x^{\frac{3}{2}})$. For 70% coverage, a dependency on the input data seems not to exist, the highest density we find at the end of the computation, where the optimization and `H`-Box adding phase starts. The high quality packings of 75% and more are only reached at the end of the computation during the optimization.

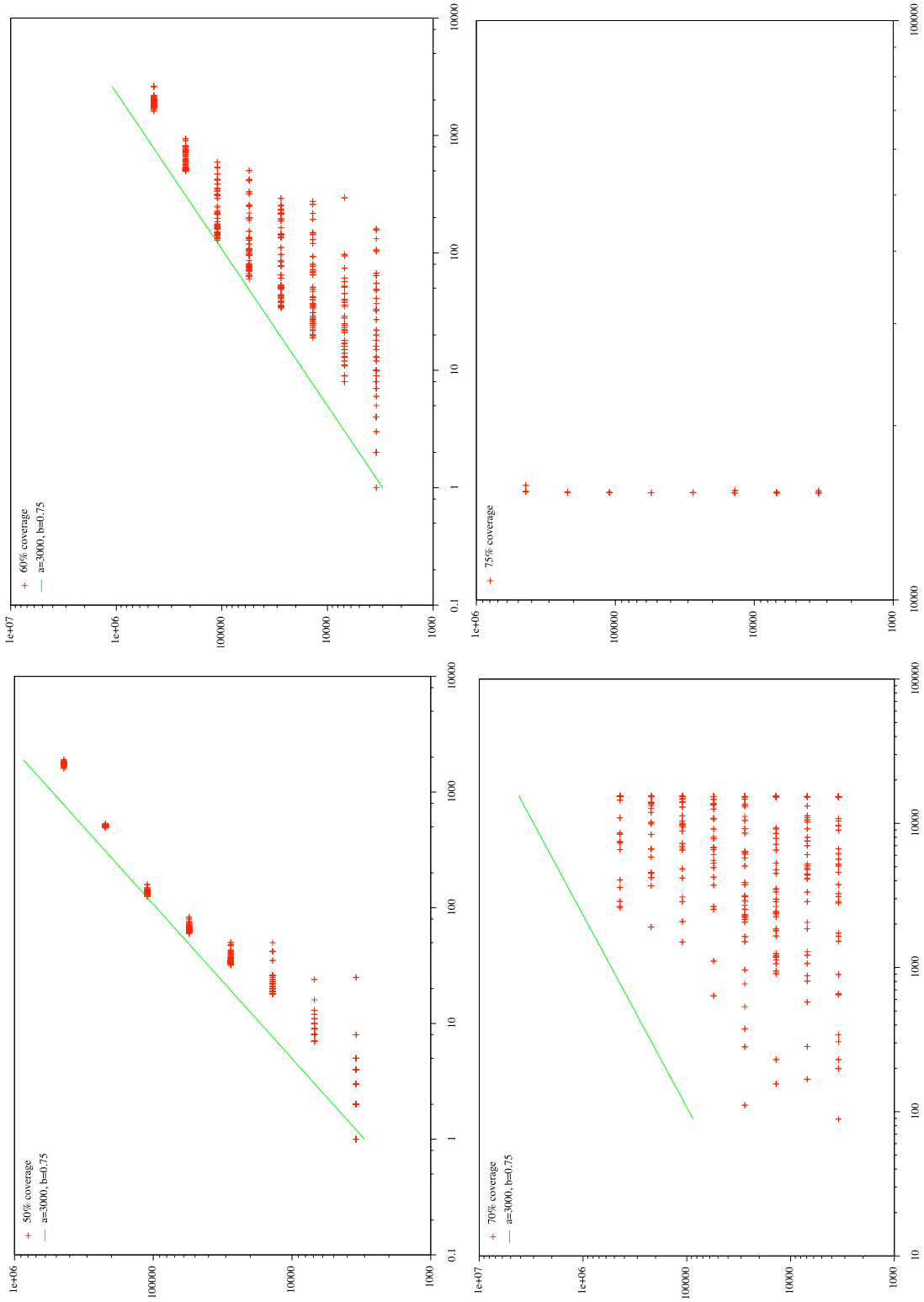


Figure 5.9: The algorithm's time to reach a certain quality depending on the input data (#TRI)

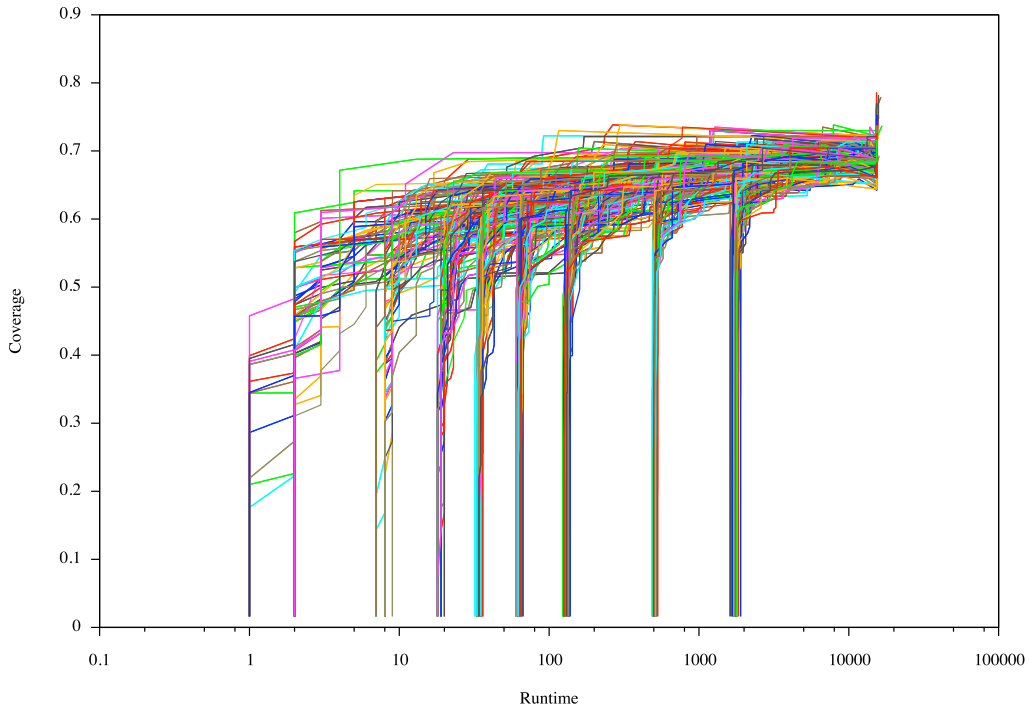


Figure 5.10: The algorithm's time to reach a certain quality depending on the input data (#TRI)

In figure 5.10 we see a diagram on the progress of the packing over time for all 7 models and 30 attempts per model. We observe that in the beginning the packing improves very fast. We also see the pre-computation phase, which is tremendously depending on the input data. This is the reason for the polynomial dependency between input size and packing quality for small coverages. The seven *bars* we witness reflect the seven instances of our trunk where every model i has twice as many triangles as model $i+1$. We also see that the really high coverages are taken at the very end of the computation.

6 Conclusion and future work

We have presented a continuous packing scheme based on a Monte Carlo simulation as a top level implementation and a streamlined physics engine as a bottom layer for the SAE variant of the trunk packing problem. We have presented several third party solutions and showed that our technique produces better or at least comparable results. We designed and presented novel and easy-to-implement algorithms for penetration depth, Euclidean distance, MTD, ε-distance and intersection volume for box/box, box/triangle and box/point. We also showed that our implementations are faster and more robust than existing state-of-the-art approaches.

The combination of Monte Carlo packing algorithms and contact simulation of moving rigid bodies proves to be a good concept, and many other top layer packing heuristics are imaginable for different problem instances. In this context evolutionary approaches like Tiwari et al. (2010) may benefit from our contact solver when it is used in the layout phase.

Also different packing scenarios are imaginable for this technique. Currently we are extending our approach to pack arbitrarily polyhedra into a trunk. Those polyhedra describe real objects like suitcases, bicycles, baby carriages or golf bags. Whereas the majority of moves can be adopted, the contact solver has to be adjusted. The challenge here is to design a fast penetration depth computation for arbitrarily triangle meshes. This is not straight forward to implement because the higher complexity of the objects effects the complexity of the implementation. Our hope lies in a method called *pointshell voxmap* presented in McNeely et al. (2005). Concerning the physical resolving we also experiment with heuristics that lack physical correctness for the sake of runtime improvement and complexity reduction (compare Erbes et al. (2013)). A free space detection as we used it for the H-boxes is harder to implement for triangle meshes. Minkowski sum computation for polygons exit (compare Hachenberger (2007)). Another imaginable technique is to sample the free space of the trunk and the inner of the items with dense point clouds and use the *iterative closest point* algorithm (ICP) to find an optimal translation and rotation to fit the item into the free space.

The DIN instance of the trunk packing problem is another field of application we plan to use our approach for. In contrast to our problem the DIN version defines equally sized boxes of $200mm \times 100mm \times 50mm$. Due to this small size, common packings of DIN boxes consist of several hundred boxes, whereas our method is suited for a small number of different sized boxes (usually less than 30). To make our approach suitable for this problem, we glue DIN boxes together such that they become boxes again of size $(k \cdot 200mm) \times (l \cdot 100mm) \times (m \cdot 50mm)$. We get different box types similar to those we are used to in the SAE variant. Every type is defined by a tuple (k, l, m), and the smallest type $k = l = m = 1$ is the equivalent of H-boxes in the SAE variant. We again have our move set - volume increasing, volume decreasing, perturbation - we also have a pre-packing phase and a fine tuning phase where free space detection is used to fill the remaining space with boxes of smallest type. From this idea we expect appropriate results that fulfill two requirements: optimal coverage and dense cores. Especially the second requirement is important for our industrial partner as holes are not desired a packing. As these parts are hopefully filled with the bigger box types, which represent dense DIN-box layers, the core is expected to be dense and the outer part of the package is manly filled with single boxes.

Especially in the field of real world packing strategies with three dimensional items of least 7 degrees of freedom the combination of physical modeling and stochastic optimization turned out to be very promising. The full potential has not been tapped at all yet, existing approaches have to be improved and new methods to be discovered. The physical resolving hopefully profits from faster and parallel hardware like nowadays graphics cards. In the field of stochastic methods a vast number of alternatives exist like evolutionary strategies that might be a good substitute for the SA approach used in this work. We therefore encourage other scientists to experiment in this manifold field of research and to find new promising combinations to produce high class packings.

List of Figures

Bibliography

Abam, M. A. and de Berg, M. (2005). Kinetic sorting and kinetic convex hulls. In *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, pages 190–197, New York, NY, USA. ACM Press.

Agarwal, P. K., de Berg, M., Gudmundsson, J., Hammar, M., and Haverkort, H. J. (2001). Box-trees and r-trees with near-optimal query time. In *SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry*, pages 124–133, New York, NY, USA. ACM Press.

Althaus, E., Baumann, T., Schömer, E., and Werth, K. (2007). Trunk Packing Revisited. In Demetrescu, C., editor, *Experimental Algorithms: Proceedings of the 6th International Workshop, WEA 2007*, volume 4525 of *Lecture Notes in Computer Science*, pages 420–432. Springer.

Basch, J., Guibas, L. J., and Hershberger, J. (1997). Data structures for mobile data. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 747–756, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Baumann, T., Schömer, E., and Werth, K. (2007). Solving geometric packing problems based on physics simulation. Submitted for ESA'07.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.

Bentley, J. L. (1990). K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197, New York, NY, USA. ACM Press.

Bruss, I. and Frick, A. (1996). Fast interactive 3-D graph visualization. *Lecture Notes in Computer Science*, 1027:99–??

BULLET (2010). *BULLET, Open Source Physics Library.* Game Physics Simulation. `http://bulletphysics.org/`.

Cagan, J. and Ding, Q. (2003). Automated trunk packing with extended pattern search. *Virtual Engineering, Simulation & Optimization*, SP-1779:33–41.

Cagan, J., Shimada, K., and Yin, S. (2002). A survey of computational approaches to three-dimensional layout problems. *Computer-Aided Design*, 34(8):597–611.

Cameron, S. A. and Culley, R. K. (1986). Determining the minimum translational distance between two convex polyhedra. In *IEEE International Conference on Robotics and Automation*, pages 591–596.

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition.

Dickerson, M., Duncan, C. A., and Goodrich, M. T. (2000). K-d trees are better when cut on the longest side. In *Proceedings of the 8th Annual European Symposium on Algorithms*, ESA '00, pages 179–190, London, UK, UK. Springer-Verlag.

Eberly, D. (2001). *3D Game Engine Design: A Practical Approach to Real Time Computer Graphics*. Morgan Kaufmann.

Eberly, D. H. (2006). *3D Game Engine Design, Second Edition: A Practical Approach to Real-Time Computer Graphics (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Eisenbrand, F., Funke, S., Karrenbauer, A., Reichel, J., and Schömer, E. (2005). Packing a trunk: now with a twist! In *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, pages 197–206, New York, NY, USA. ACM Press.

Eisenbrand, F., Funke, S., Karrenbauer, A., Reichel, J., and Schömer, E. (2007). Packing a truck - now with a twist! *Int. J. Comput. Geometry Appl.*, 17(5):505–527.

Eisenbrand, F., Funke, S., Reichel, J., and Schömer, E. (2003). Packing a trunk. In Di Battista, G. and Zwick, U., editors, *Algorithms - ESA 2003: 11th Annual European Symposium*, volume 2832 of *Lecture Notes in Computer Science*, pages 618–629, Budapest, Hungary. Springer.

Erbes, R., von Dziegielewski, A., Schoemer, E., and Wolpert, N. (2013). Efficient planning of disassembly paths in highly constrained environments. In *ICRA*, 2013 IEEE International Conference on Robotics and Automation (ICRA). submitted to ICRA October 2012.

Foley, J. D., van Dam, A., Feiner, S., and Hughes, J. F. (1990). *Computer graphics - principles and practice, 2nd Edition*. Addison-Wesley.

Franklin, W. R. (1987). Polygon properties calculated from the vertex neighborhoods. In *Proceedings of the third annual symposium on Computational geometry*, SCG '87, pages 110–118, New York, NY, USA. ACM.

Franklin, W. R. and Kankanhalli, M. (1992). Volumes from overlaying 3-d triangulations in parallel.

Fruchterman, T. M. J. and Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164.

Geometrictools (2006). Wild magic real-time 3d graphics engine. `http://www.geometrictools.com/`. Geometric Tools.

Gottschalk, S. (1998). *Collision Queries using Oriented Bounding Boxes*. PhD thesis.

Gottschalk, S., Lin, M. C., and Manocha, D. (1996). OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, pages 171–180. Proc. SIGGRAPH'96.

GTS (2006). The gnu triangulated surface library. `http://gts.sourceforge.net/`.

Hachenberger, P. (2007). Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces. In *ESA*, pages 669–680.

Hubbard, P. (1993). Interactive collision detection. In *In Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31.

Kamada, T. and Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15.

Karrenbauer, A. (2004). Packing boxes with arbitrary rotations. Diploma thesis, Universität des Saarlandes, Saarbrücken.

Kim, Y. J., Otaduy, M. A., Lin, M. C., and Manocha, D. (2002). Fast penetration depth computation for physically-based animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 23–31, New York, NY, USA. ACM Press.

McNeely, W. A., Puterbaugh, K. D., and Troy, J. J. (2005). Six degree-of-freedom haptic rendering using voxel sampling. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA. ACM.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

Mirtich, B. (1996a). Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2):31–50.

Mirtich, B. (1997). Efficient algorithms for two–phase collision detection. `citeseer.ist.psu.edu/article/mirtich97efficient.html`.

Mirtich, B. and Canny, J. (1995). Impulse-based simulation of rigid bodies. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 181–ff., New York, NY, USA. ACM Press.

Mirtich, B. V. (1996b). Impulse-based dynamic simulation of rigid body systems. Technical report.

Neumann, U. (2006). Optimierungsverfahren zur normgerechten Volumenbestimmung von Kofferräumen im europäischen Automobilbau. Diploma thesis, Technische Universität Braunschweig, Braunschweig.

NVIDIA (2009). *PhysX, a powerful physics engine enabling real-time physics in leading edge PC games.* NVIDIA. `http://www.nvidia.com/object/physx_new.html`.

ODE (2006). Open dynamics engine. `http://ode.org/`.

Reichel, J. (2006a). *Combinatorial approaches for the Trunk packing problem.* PhD thesis, Saarbrücken.

Reichel, J. (2006b). *Combinatorial Approaches for the Trunk Packing Problem.* PhD thesis, Universität des Saarlandes.

Rieskamp, J. (2005). Automation and Optimization of Monte Carlo Based Trunk Packing. Diploma thesis, Universität des Saarlandes, Saarbrücken.

SAE (2001). SAE J1100, Motor Vehicle Dimensions.

Shellshear, E., Carlson, J. S., and Bohlin, R. (2012). A combinatorial packing algorithm and standard trunk geometry for iso luggage packing. to-be-published in 2012.

Tiwari, S., Fadel, G., and Fenyes, P. (2010). A fast and efficient compact packing algorithm for sae and iso luggage packing problems. *Journal of Computing and Information Science in Engineering*, 10(2):021010.

van Bergen, G. (2006). SOLID, software library for interference detection. `http://www.dtecta.com/`.

van den Bergen, G. (1998). Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13.

van den Bergen, G. (2001). Proximity queries and penetration depth computation on 3d game objects. In *Game Developers Conference*.

von Dziegielewski, A., Hemmer, M., and Schömer, E. (2012). High quality conservative surface mesh generation for swept volumes. In *ICRA*, pages 764–769.

Werth, K. (2004). ExUS: Exact Union of Spheres. Diploma thesis, Max Planck Institut für Informatik, Universität des Saarlandes, Saarbrücken, Germany.

Will, D. (2009). Normgerechte Volumenbestimmung von Kofferräumen im deutschen Automobilbau mit Hilfe von Kontaktsimulation. Diploma thesis, Johannes Gutenberg-Universität Mainz, Mainz.