FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

# Effiziente Kollisionserkennung für konvexe Körper in parallelen Simulationen

Tobias Leemann

Bachelorarbeit

# Effiziente Kollisionserkennung für konvexe Körper in parallelen Simulationen

Tobias Leemann

Bachelorarbeit

| | |
|---|---|
| Aufgabensteller: | Prof. Dr. Ulrich Rüde |
| Betreuer: | Sebastian Eibl, M. Sc. |
| Bearbeitungszeitraum: | 1.7.2017 – 1.12.2017 |

**Erklärung:**

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Bachelorarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 27. November 2017 .....

**Abstract**
In dieser Arbeit wird eine Implementierung der Kollisionsberechnung für allgemeine konvexe Körper in Starrkörpersimulationen vorgestellt. Sie verwendet den GJK-Algorithmus (*Gilbert-Johnson-Keerthi*) zur Detektion von Kollisionen und den EPA (*Expanding Polytope Algorithm*) zur Bestimmung von Kontaktdaten. Es werden die Stärken und Schwächen dieser und weiterer Algorithmen abgewogen und Lösungsansätze für Probleme, die bei der Erstellung auftraten, präsentiert. Besonderes Augenmerk lag auf der Robustheit und Präzision des Codes, die in gründlichen Tests belegt wurden. Um festzustellen, welche Genauigkeit und Geschwindigkeit von einer allgemeinen Kollisionserkennung erreicht werden können, wurden verschiedene Messungen durchgeführt und Richtwerte ermittelt.

# Inhaltsverzeichnis

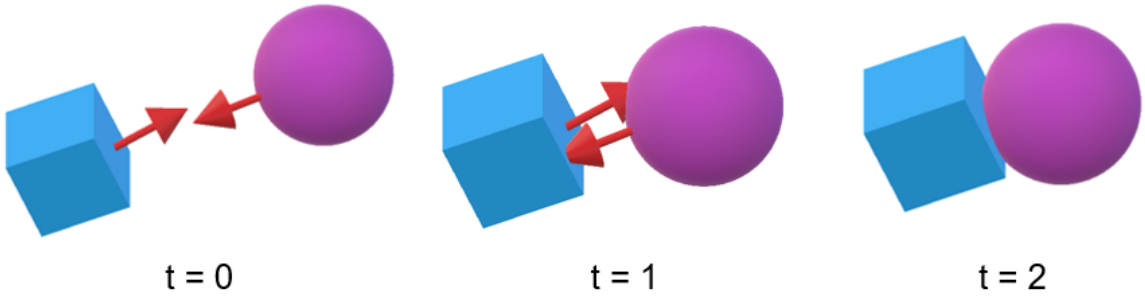



Abbildung 1: Zeitschritte in einer typischen Starrkörper-Simulation. Bei $t = 2$ kommt es zur Kollision, die erkannt werden muss.

# 1 Problemstellung

Seit der Einführung der klassischen Mechanik kann die Bewegung von starren, als nicht deformierbar angenommenen Körpern berechnet werden. Die kontinuierlichen Verbesserungen der Computertechnik ermöglichen heutzutage auch die exakte Simulation von Systemen, die aus extrem vielen Körpern bestehen, wie zum Beispiel granularen Strömungen [13]. Das Verhalten solcher, aus einzelnen Partikeln bestehender Stoffe, ist noch immer nicht vollständig verstanden und Methoden zur Simulation solcher Partikel sind Thema vieler Forschungsarbeiten [16, 22]. Granulare Mischvorgänge spielen bei der Herstellung von Kosmetik- und Pharmaprodukten [11], im Bergbau und der Weiterverarbeitung von Mineralien eine entscheidende Rolle.
Einen zentralen Teil der meisten Simulationsumgebungen für granulare Materialen, die jeden Partikel als einzelnen Körper betrachteten, stellt die *Kollisionserkennung* dar. Sie sorgt dafür, dass Objekte nicht ohne Auswirkung aufeinander treffen können. Wenn zwei Körper in Kontakt sind, üben sie Kräfte aufeinander aus und beeinflussen die gegenseitige Bewegung. Jedem Beobachter ist klar, dass Gegenstände nicht einfach durcheinander hindurch fallen können. Diese Bedingungen umzusetzen, stellt in komplexen Zuständen aber eine große Herausforderung dar.
Um den Einsatz der Kollisionserkennung besser zu verstehen, betrachten wir den typischen Ablauf einer Simulation mit diskreten Zeitschritten (Abb. 1). Hierbei wird zur Bestimmung der Geschwindigkeit numerisch über die Beschleunigung integriert. Aus nochmaliger Integration erhält man schließlich den Weg $\Delta x$ und den Winkel $\Delta\varphi$, um welchen der Körper verschoben bzw. verdreht werden muss. Dies wird für alle Körper durchgeführt und deren Positionen und Rotationszustände entsprechend angepasst.
Nun kann es aber sein (wie in Zeitschritt 2 der Abbildung), dass zwei Körper inzwischen in Kontakt sind. Deshalb kommt nach jedem Verschiebungsschritt die Kollisionserkennung zum Einsatz, deren Aufgabe es ist, alle Kontakte zu bestimmen. Außerdem sind ausreichend Daten über diese zu sammeln, damit die Kollision aufgelöst, das heißt die weitere Bewegung der beiden Kollisionspartner bestimmt werden kann.
Es wird nun für alle möglichen Paare von Körpern $A$ und $B$ ermittelt, ob sie kollidieren. Mathematisch können $A$ und $B$ als Teilmengen des $\mathbb{R}^3$ aufgefasst werden, die alle Punkte des jeweiligen Körpers enthalten. Ein Kollisionszustand liegt genau dann vor, wenn diese nicht disjunkt sind, also

$$A \cap B \neq \emptyset$$

Die Anzahl der möglichen Kontaktpaare steigt proportional zum Quadrat der Anzahl Körper im System. Deshalb ist es sehr sinnvoll, möglichst viele Paare, die auf keinen Fall eine

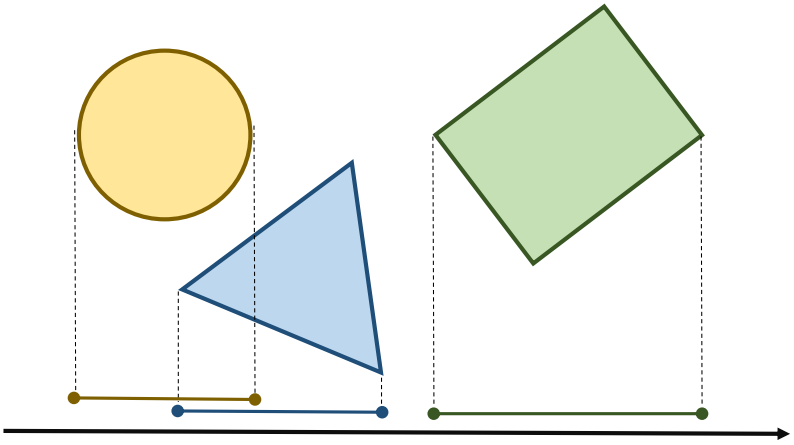


Abbildung 2: Anfangs- und Endpunkte dreier Körper, wie sie in einem *Swipe-and-Prune*-Verfahren gespeichert werden, in 2D. Die Punkte werden aufsteigend sortiert, sodass Überlappungen in linearer Laufzeit gefunden werden können. Von drei möglichen Kollisionspaaren können hier bereits zwei durch die CCD ausgeschlossen werden.

Schnittmenge haben können, schnell auszusortieren. Diesen Schritt bezeichnet man als *Coarse Collision Detection* (CCD). Die übrig gebliebenen Paare werden dann der *Fine Collision Detection* (FCD) zugeführt, die eine Kollision zweifelsfrei feststellen kann.
Wichtig ist, dass die CCD eine niedrigere Laufzeitordnung als $O(n^2)$ bei $n$ Körpern im System hat. Nur mit diesen schnellen Ausschlussverfahren ist es überhaupt möglich, Simulationen in extremer Größe durchzuführen.
Eine Möglichkeit hierzu ist das *Swipe-and-Prune* Verfahren, bei dem eine Liste der Anfangs- und Endpunkte der Projektion des Körpers auf eine Achse gespeichert und sortiert wird (Abb. 2). Die sortierte Liste wird von vorne durchgegangen und alle Überlappungen gesucht. Dies kann für mehrere Achsen, zum Beispiel alle drei Koordinatenachsen wiederholt werden. Ein Durchlauf der Liste, sowie auch das Sortieren (unter der Annahme, dass die sortierte Liste aus dem letzten Schritt vorhanden ist und dass sich Körper pro Zeitschritt nur wenig bewegen) ist in linearer Laufzeit $O(n)$ möglich.
Unsere Umgebung verwendet *Hierarchische Hash-Grids* [19], ein Verfahren, bei dem die Körper Gitterzellen zugewiesen werden und Körper, die in nicht benachbarten Zellen sind, aussortiert werden können. Auch die *Hash-Grids* erreichen im Mittel die Laufzeitordnung $O(n)$. Diese und andere Methoden der *Coarse Collision Detection* sind auf verschiedenste Körpertypen anwendbar und in vielen Implementierungen vorhanden. Für die weitere Arbeit soll deshalb ausschließlich die *Fine Collision Detection* betrachtet werden.

## 1.1 Analytische Kollisionserkennung

Bei gegebenen Körpertypen, Positionen und Rotationszuständen kann *analytisch* berechnet werden, ob eine Kollision stattfindet. Hierbei wird je nach den beteiligten Grundkörpern (zum Beispiel ein Quader und ein Zylinder) eine unterschiedliche Berechnungsfunktion aufgerufen. Daher ist es nötig, für jede Kombination zweier Grundkörper einen Lösungsalgorithmus bereitzustellen.
Verwendet eine Software eine Auswahl von $n$ Grundkörpern, so sind $\frac{n*(n+1)}{2} \in O(n^2)$ verschiedene Algorithmen erforderlich. Abgesehen von einem enorm steigenden Implementierungsaufwand (fügt man einen $k$-ten Grundkörper hinzu sind auch $k$ neue Funktionen nötig) ist auch

die Fehleranfälligkeit beträchtlich.
Bei einer korrekten Implementierung sind die Geschwindigkeit und Genauigkeit der Berechnung allerdings nicht zu übertreffen. Deshalb kommt die analytische Kollisionserkennung trotzdem in vielen Anwendungen zum Einsatz. Besonders bei wenigen simplen Grundkörpern ist eine analytische Kollisionserkennung durchaus sinnvoll.

### 1.2 Kollisionserkennung für allgemeine Körper

Es existieren auch Verfahren, die auf eine Vielzahl von Körpertypen anwendbar sind und deshalb als *allgemein* bezeichnet werden. Neue Geometrien können mit sehr geringem Aufwand ebenfalls hinzugefügt werden. Die bekannten Algorithmen für diesen Zweck sind allerdings auf *konvexe* Körper beschränkt. Da ein *konkaver* Körper (Definitionen und ausführliche Erläuterungen in Abschnitt 2.1) aus konvexen Teilen zusammengesetzt werden kann, sind sie trotzdem allgemein verwendbar.
Aufgrund der schrittweisen Approximation der analytischen Lösung spricht man auch von *iterativer* Kollisionserkennung.
In der Regel ist die Laufzeit einer iterativen Methode länger und das Ergebnis etwas ungenauer als bei einer analytischen Rechnung. Werden viele oder komplexere Körpertypen in der Simulation benötigt, macht es trotzdem Sinn eine allgemeine Kollisionserkennung zu verwenden.
Es existieren außerdem Formen, für die eine analytische Lösung nicht vorhanden ist, sodass eine iterative Kollisionserkennung die einzige Option darstellt.

### 1.3 Ziel und Aufbau der Arbeit

Die Anforderungen an Genauigkeit und Ausführungsgeschwindigkeit sind, abhängig von der Anwendung, sehr unterschiedlich. Für Animationen in interaktiven Unterhaltungsmedien wird versucht, ein möglichst realistisches Aussehen in kurzer Rechenzeit zu erreichen. In physikalischen Simulationen steht die Qualität der Berechnung im Vordergrund.
Ziel dieser Arbeit ist es, eine robuste und präzise allgemeine Kollisionserkennung für parallele Simulationen zu erstellen, die auch in der Lage ist, die benötigten Kontaktdaten zu ermitteln.
Zuerst sollen die verfügbaren Algorithmen vorgestellt und bewertet werden. Die Methode mit dem größten Potential kommt für die Implementierung zum Einsatz (Abschnitt 2).
Anschließend wird die allgemeine Kollisionserkennung in den am Lehrstuhl entwickelten *pe Physics Engine* [12] integriert. Diese Simulationsumgebung ist Teil des *waLBerla*-Framework [10] und ist für die Simulation von Starrkörpern auf massiv parallelen Architekturen optimiert. Das gesamte Projekt ist quelloffen. Der Code kann auf der Website [2] heruntergeladen werden.
Die Implementierung soll im Folgenden gründlich auf eventuelle Schwächen analysiert und optimiert werden (Abschnitt 3).
Ausführliche Tests stellen die Funktionsfähigkeit sicher. Zudem werden die Genauigkeit und die Geschwindigkeit bestimmt (Abschnitt 4).

## 2 Bekannte Algorithmen

### 2.1 Geometrische Grundlagen

Zu Beginn dieses Kapitels sollen einige Grundlagen erklärt werden. Ausführlichere und gut verständliche Beschreibungen finden sich beispielsweise in [26] und [6]. Die Zeichnungen aus diesen Quellen dienen zudem als Vorlage für die Abbildungen in diesem Abschnitt.
Ein Körper wird im Folgenden immer als eine Menge aller seiner Punkte im $\mathbb{R}^3$ aufgefasst. Die Begriffe "Körper" und "Menge" werden daher synonym verwendet.

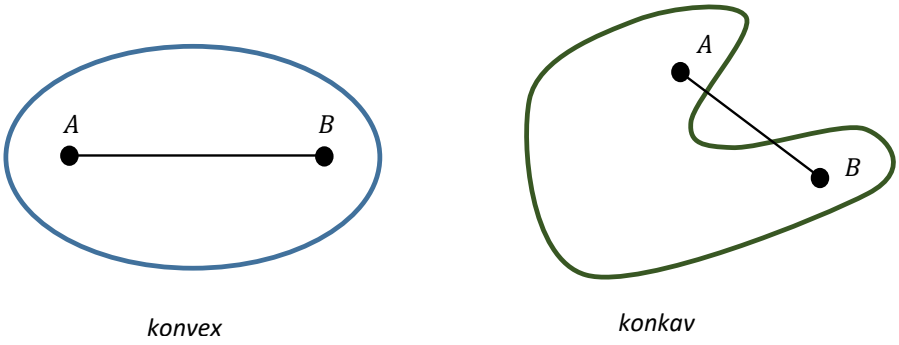



Abbildung 3: Ein konvexes und ein konkaves Objekt. Das konvexe Objekt enthält alle Verbindungslinien.

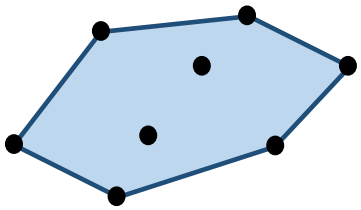


Abbildung 4: Konvexe Hülle einer Menge von Punkten in einer Ebene. Die *konvexe Hülle* (blau) ist die kleinste konvexe Menge, die alle Punkte enthält.

### 2.1.1 Körpertypen

**Konvexe Körper**
Ein Körper wird als *konvex* bezeichnet, wenn er alle Strecken, die zwei beliebige seiner Punkte verbinden, enthält. Ist dies nicht der Fall, bezeichnet man den Körper als *konkav*. Diese beiden Eigenschaften sind in Abb. 3 dargestellt. Viele der gängigen Kollisionserkennungsalgorithmen sind nur auf konvexe Körper anwendbar. Dies ist allerdings für die meisten Anwendungen ausreichend, da es möglich ist, ein konkaves Objekt in mehrere konvexe Teile zu zerlegen und diese anschließend einzeln zu betrachten. Auch diese Arbeit ist auf die Betrachtung von konvexen Körpern beschränkt.

**Konvexes Polytop**
Als *konvexes Polytop* bezeichnet man die *konvexe Hülle* einer Menge von Punkten $A$. Die konvexe Hülle ist die kleinste konvexe Menge, welche alle Punkte aus $A$ enthält (Abb. 4).
Für einen Punkt enthält die konvexe Hülle ebenfalls nur diesen Punkt. Für zwei Punkte enthält sie genau die Verbindungslinie der beiden Punkte, da die Hülle sonst nicht konvex wäre. Die konvexe Hülle einer Punktmenge $A$ wird als $conv(A)$ notiert.

**Simplex**
Das Simplex stellt die einfachste Form eines konvexen Polytopes dar. Deshalb ist es ebenfalls die konvexe Hülle einer Menge von Punkten $A$. Die Elemente in $A$ müssen jedoch zusätzlich *affin unabhängig* sein, um ein Simplex zu bilden.
Laut Definition sind eine Anzahl von $k \in \mathbb{N}$ Punkten $\vec{v_0}, ..., \vec{v_k} \in \mathbb{R}^n$ affin unabhängig, wenn das Gleichungssystem mit Skalaren $\lambda_0, ..., \lambda_k$ als Unbekannten

$$\lambda_0 \vec{v_0} + ... + \lambda_k \vec{v_k} = 0$$

$$\lambda_0 + ... + \lambda_k = 0$$

als einzige Lösung $\lambda_0 = ... = \lambda_k = 0$ annimmt.
Alternativ kann die affine Unabhängigkeit auch durch die lineare Unabhängigkeit der Vektoren

$$\{\vec{v_1} - \vec{v_0}, ..., \vec{v_k} - \vec{v_0}\}$$

gezeigt werden.
Eine Menge aus einem Punkt ist immer affin unabhängig. Zwei Punkte sind es, wenn sie nicht identisch sind; drei, wenn sie nicht auf der gleichen Gerade liegen und vier, wenn sie nicht in der gleichen Ebene enthalten sind.
Im $\mathbb{R}^3$ können nicht mehr als vier Punkte affin unabhängig sein. Deshalb gibt es vier Arten von Simplices, je nachdem wie viele Punkte in der erzeugenden Menge enthalten sind und das Simplex aufspannen: Den Punkt (auch als 1-Simplex bezeichnet), die Strecke (2-Simplex), das Dreieck (3-Simplex) und das Tetraeder (4-Simplex).

### 2.1.2 Minkowski-Summe und Differenz

Ein weiteres benötigtes Konzept ist die *Minkowski-Addition* zweier Mengen. Die *Minkowski-Summe* der Körper $A$ und $B$ ist definiert als

$$A + B = \{\vec{x} + \vec{y} : \vec{x} \in A, \vec{y} \in B\}$$

Dies entspricht der Menge aller Punkte, die als Summe eines Punktes aus $A$ und einem aus $B$ dargestellt werden können. Ein einfaches Beispiel stellt die Summe eines Kreises und eines Rechteckes dar (Abb. 5). Das Resultat ist ein größeres Rechteck mit abgerundeten Ecken. Dies erhält man, wenn man den Ursprung des Kreises gedanklich über jeden Punkt des Rechtecks $A$ legt.
Die *Minkowski-Differenz* entsteht aus der Subtraktion der Mengen und ist passend zur Minkowski-Summe definiert als

$$A - B = A + (-B) = \{\vec{x} - \vec{y} : \vec{x} \in A, \vec{y} \in B\}$$

Sie kann auch als die Minkowski-Summe des Körpers $A$ und der *Reflektion* $-B$ des Körpers $B$ aufgefasst werden. Die Reflektion ist gegeben durch

$$-B = \{-\vec{y} : \vec{y} \in B\}$$

Die Minkowski-Differenz ist die Menge aller Vektoren von Punkten in $B$ zu Punkten in $A$ und kann auch so konstruiert werden: Sucht man in jede Richtung den längsten und kürzesten Vektor von $B$ zu $A$ und trägt diese in einem Koordinatensystem ein, erhält man Randpunkte der Minkowski-Differenz.
Alternativ kann diese eben über die Reflektion bestimmt werden, wie in Abb. 6 zu sehen ist. Hier wird die Konstruktion der Minkowski-Differenz eines Rechtecks und eines Dreiecks durchgeführt.
Zu erkennen ist ein entscheidender Punkt: Haben die beiden Körper mindestens einen gemeinsamen Punkt, so enthält die Minkowski-Differenenz den Ursprung. Es gilt folgende Äquivalenz:

$$A \cap B \neq \emptyset \iff \vec{0} \in A - B$$

Die Minkowski-Differenz der Körper $A$ und $B$ wird synonym auch als *Configuration Space Obstacle* (CSO) von $A$ und $B$ bezeichnet.

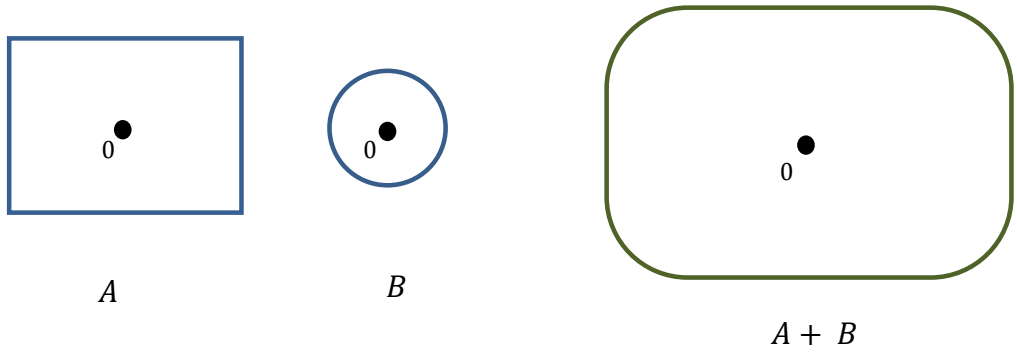



Abbildung 5: Minkowski-Summe eines Kreises und eines Rechtecks.

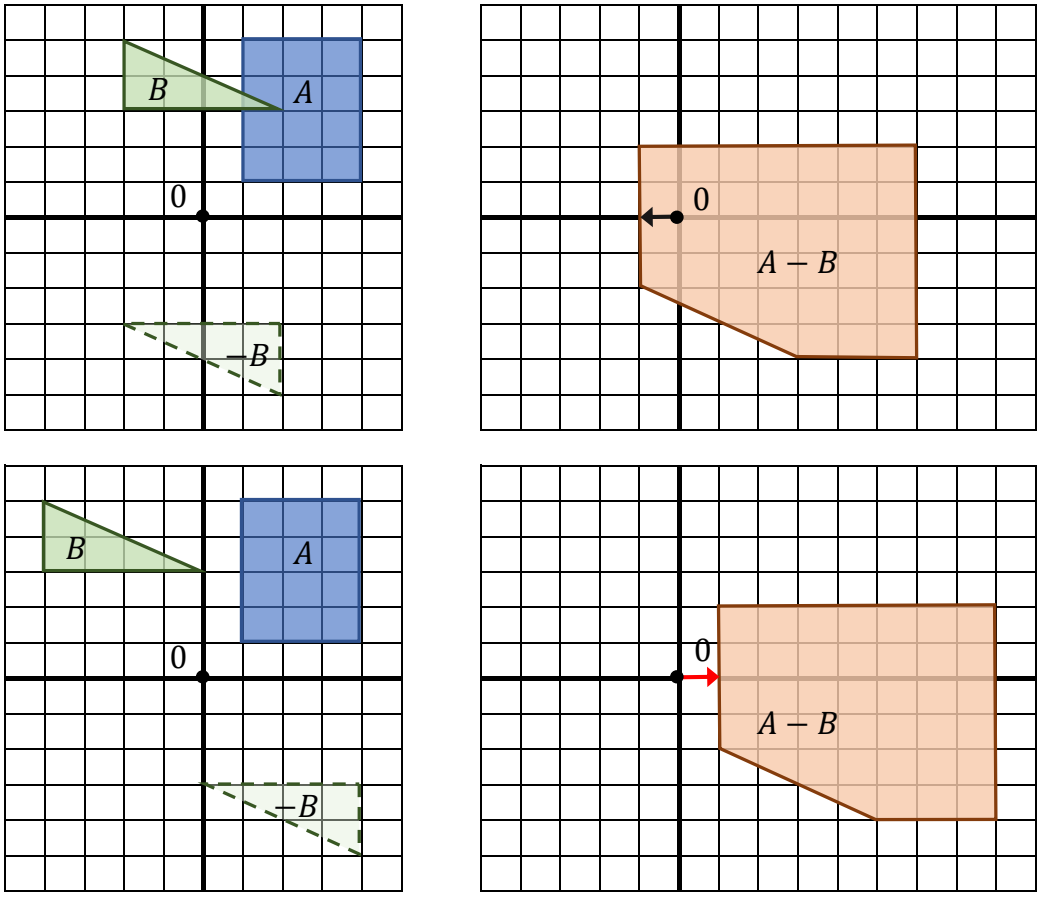



Abbildung 6: Konstruktion der Minkowski-Differenz. Diese enthält den Ursprung, sobald die Körper in Kontakt sind. Der Abstandvektor (rot, im Bild unten) definiert den Abstand und der Penetrationsvektor (schwarz, oben) die Eindringtiefe.

Bei einer Verschiebung des Körpers $A$, während $B$ an seiner Position bleibt, verändert sich die Form ihres CSO nicht. Es verschiebt sich allerdings genau um den Translationsvektor, welcher auf den ersten Körper angewandt wurde. Verschiebt man $B$, wandert das CSO in die inverse Richtung der Verschiebung.
Aus dem CSO auf die Form und Position der ursprünglichen Körper zu schließen, ist nicht ohne weiteres möglich.
Mithilfe der Minkowski-Differenz können einige Grundbegriffe definiert werden. Die hier verwendeten Definitionen entsprechen den von van den Bergen [26, S.36].

**Abstand**
Falls sich zwei Objekte nicht in Kontakt befinden, trennt sie ein *Abstand* $d(A, B) > 0$. Dieser kann mithilfe ihres CSO folgendermaßen ausgedrückt werden:

$$d(A, B) = min\{\|\vec{x} - \vec{y}\| : \vec{x} \in A, \vec{y} \in B\} = min\{\|\vec{z}\| : \vec{z} \in A - B\}$$

Der Abstand entspricht also der kürzesten Distanz, welche einen Punkt aus $A$ mit einem Punkt aus $B$ verbindet. Im CSO ist der Abstand die Entfernung des nächsten Punktes zum Ursprung. Der Vektor vom Ursprung zu eben diesem Punkt wird als *Abstandsvektor* bezeichnet (Abb. 6, unten). Verschiebt man $B$ um den Abstandsvektor, befinden sich die Körper in Berührkontakt.

**Penetrationstiefe**
Gegenteilig dazu haben zwei Objekte, die sich in Kontakt befinden, eine *Penetrationstiefe* $p(A, B) \geq 0$. Diese ist gegeben durch

$$p(A, B) = inf\{\|\vec{z}\| : \vec{z} \notin A - B\}$$

Im CSO ist das die Länge des kürzesten Vektors, der außerhalb liegt (Abb. 6, oben). Genauer gesagt eine größte untere Schranke hierfür.
Geometrisch ist die Penetrationstiefe die Länge des kleinsten Vektors, um den Körper $B$ verschoben werden muss, um die beiden Objekte in Berührkontakt zu bringen. Dieser Vektor wird als *Penetrationsvektor* bezeichnet und ist nicht immer eindeutig.
Der Penetrationsvektor $\vec{p}$ fasst zwei Größen zusammen: Die Penetrationstiefe $p$, die durch die Länge oder den Betrag des Vektors gegeben ist, sowie die *Kontakt-* oder *Stoßnormale* $\vec{n}$. Diese wird durch Normierung des Penetrationsvektors berechnet. Es gilt also folgender Zusammenhang:

$$\vec{p} = p\vec{n}$$

**Kontaktpunkt**
Einen Kontaktpunkt gibt es nur bei Kollisionen mit Berührkontakt (Penetrationstiefe = 0). Enthält die Schnittmenge $A \cap B$ nur einen Punkt, so ist er der eindeutige Kontaktpunkt. Sollte die Menge mehrere Punkte, etwa in einer Fläche enthalten, so sind alle Punkte Kontaktpunkte.
Kollisionen mit größerer Penetrationstiefe und Penetrationsvektor $\vec{p}$ müssen auf eine Kollision mit Berührkontakt zurückgeführt werden. Dies kann etwa durch Verschiebung des Körpers $B$ um $\frac{1}{2}\vec{p}$ und des Körpers $A$ um $-\frac{1}{2}\vec{p}$ erreicht werden. Die Kontaktpunkte dieser Konfiguration können dann verwendet werden.

Zur physikalisch korrekten Berechnung der Bewegung der Körper nach der Kollision (*Kollisionsauflösung*) sind drei Größen von Bedeutung:

- Penetrationstiefe $p$

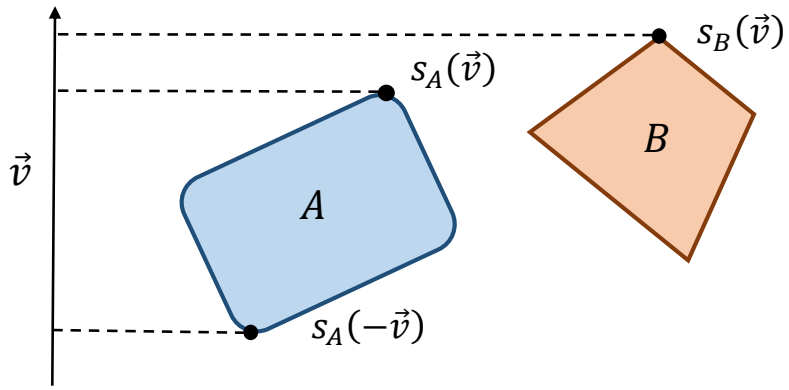



Abbildung 7: Supportpunkte zweier Körper. Der Supportpunkt $\vec{s_A}(\vec{v})$ liegt weiter in Richtung $\vec{v}$, als jeder andere Punkt des Körpers.

- Stoßnormale $n$
- Kontaktpunkt

Diese Daten zu ermitteln, sollte neben der reinen Kollisionserkennung auch Teil jeder Kollisionsberechnung sein.

### 2.1.3 Supportfunktionen

Als letztes soll nun noch auf die *Supportfunktion* eingegangen werden. Mit dieser kann ein konvexer Körper vollständig beschrieben werden.
Die Supportfunktion $\vec{s_A}(\vec{v})$ eines Körpers $A$ in eine Richtung $\vec{v}$ liefert einen Punkt, sodass folgendes gilt:

$$\vec{s_A}(\vec{v}) \cdot \vec{v} = max\{\vec{x} \cdot \vec{v} : \vec{x} \in A\}$$

Ergebnis der Funktion ist also der äußerste Punkt des Körpers in Richtung $\vec{v}$. Genauer gesagt darf es keinen Punkt geben, dessen Projektion auf $\vec{v}$ einen größeren Wert annimmt. Dieses Ergebnis muss nicht immer eindeutig sein. Ein beliebiger Punkt, für den die Defintion erfüllt ist, gilt als korrekter Rückgabewert und wird auch als *Supportpunkt* des Körpers $A$ bezeichnet (Abb. 7).

**Beispiele für Supportfunktionen**
Für eine **Kugel** $A$ mit Mittelpunkt $\vec{C}$ und Radius $r$ ist die Supportfunktion kurz anzugeben. Sie lautet für $\vec{v} \neq \vec{0}$:

$$\vec{s_A}(\vec{v}) = \vec{C} + r\frac{\vec{v}}{\|\vec{v}\|}$$

Die Supportfunktion eines **Quaders**, dessen Kantenrichtungen den Koordinatenachsen entsprechen, kann ebenfalls bestimmt werden. Sei $A$ ein Quader mit Kantenlängen $h_x$, $h_y$ und $h_z$ am Punkt $\vec{C}$, so lautet die entsprechende Supportfunktion in Richtung $\vec{v} = (v_1, v_2, v_3)^T \neq \vec{0}$

$$\vec{s_A}(\vec{v}) = \vec{C} + \frac{1}{2}\begin{bmatrix} sgn(v_1)h_x \\ sgn(v_2)h_y \\ sgn(v_3)h_z \end{bmatrix}$$

**Supportfunktionen von Minkowski-Summen und Differenzen**
Auch für Minkowski-Summen und -Differenzen von Körpern lässt sich die Supportfunktion recht einfach bestimmen.
Die Supportfunktion einer Minkowski-Summe $A + B$ der Körper $A$ und $B$ kann durch die Supportfunktionen der Einzelobjekte ausgedrückt werden. Einsetzen in die Definition

$$\vec{s}_{A+B}(\vec{v}) \cdot \vec{v} = max\{(\vec{x} + \vec{y}) \cdot \vec{v} : \vec{x} \in A, \vec{y} \in B\}$$

$$= max\{\vec{x} \cdot \vec{v} : \vec{x} \in A\} + max\{\vec{y} \cdot \vec{v} : \vec{y} \in B\} = (\vec{s_A}(\vec{v}) + \vec{s_B}(\vec{v})) \cdot \vec{v}$$

führt zu

$$\vec{s}_{A+B}(\vec{v}) = \vec{s_A}(\vec{v}) + \vec{s_B}(\vec{v})$$

Ähnlich sind auch Supportpunkte der Minkowski-Differenz, des CSO der Körper $A$ und $B$ zu bestimmen. Wieder wird die Definition eingesetzt:

$$\vec{s}_{A-B}(\vec{v}) \cdot \vec{v} = max\{(\vec{x} - \vec{y}) \cdot \vec{v} : \vec{x} \in A, \vec{y} \in B\} = max\{\vec{x} \cdot \vec{v} : \vec{x} \in A\} - min\{\vec{y} \cdot \vec{v} : \vec{y} \in B\}$$

$$= max\{\vec{x} \cdot \vec{v} : \vec{x} \in A\} - max\{\vec{y} \cdot (-\vec{v}) : \vec{y} \in B\} = (\vec{s_A}(\vec{v}) - \vec{s_B}(-\vec{v})) \cdot \vec{v}$$

Der Supportpunkt einer Minkowski-Differenz kann also durch

$$\vec{s}_{A-B}(\vec{v}) = \vec{s_A}(\vec{v}) - \vec{s_B}(-\vec{v})$$

angegeben werden.
Wie wir bereits gesehen haben, können alle relevanten Kollisionsdaten aus dem CSO abgeleitet werden, das als Ganzes allerdings nicht trivial zu berechnen ist. Supportpunkte zu ermitteln ist mit diesem Wissen aber einfach, was die folgenden Algorithmen nutzen, um Rückschlüsse auf das Aussehen des CSO zu ziehen.

## 2.2 Der GJK-Algorithmus

Der bekannteste und verbreiteteste Algorithmus für allgemeine Kollisionserkennung ist der nach seinen Autoren Gilbert, Johnson and Keerthi benannte *GJK*-Algorithmus. Obwohl dieser in der Erstveröffentlichung [9] nur für eine eingeschränkte Menge von Formen anwendbar war, wurde das Potential schnell erkannt und der Algorithmus für beliebige konvexe Körper verallgemeinert [8].

**Funktionsweise**
Ziel des GJK ist es, den Abstand zweier Objekte zu bestimmen. Alle Berechnungen finden im CSO $A - B$ der Körper statt, wo der Abstandsvektor angenähert wird.
Zu Beginn wird der GJK-Algorithmus mit einer beliebigen Suchrichtung $\vec{v_0}$ gestartet. Es wird ein Supportpunkt des CSO in Gegenrichtung $\vec{w_0} = \vec{s}_{A-B}(-\vec{v_0})$ bestimmt. Die *GJK-Simplexmenge* $W$ ist zu Beginn leer ($W_0 = \emptyset$). Diese Simplexmenge wird immer maximal vier, affin unäbhängige Punkte enthalten. Ihre konvexe Hülle bildet deshalb ein Simplex.
Nach dieser Initialisierung wird die folgende Iteration für $i = 1, 2, ...$ durchgeführt:

- Der letzte berechnete Supportpunkt wird gedanklich in die Simplexmenge aufgenommen. $W_i^* = W_{i-1} \cup \vec{w}_{i-1}$
- Der ursprungsnächste Punkt aus der konvexen Hülle der Simplexmenge (des Simplices) wird ermittelt und als neue Suchrichtung $\vec{v_i}$ gespeichert. $\|\vec{v_i}\| = min\{\|\vec{x}\| : \vec{x} \in W_i^*\}$
- Die aktualisierte Simplexmenge $W_i$ wird die kleinste mögliche Teilmenge von $W_i^*$, in deren konvexen Hülle $\vec{v_i}$ enthalten ist. Es werden also alle Punkte entfernt, ohne die $\vec{v_i}$ immer noch im aufgespannten Simplex der verbliebenen Punkte enthalten wäre.

- Zuletzt wird ein neuer Supportpunkt $w_i$ entgegen der Suchrichtung berechnet. $w_i = \vec{s}_{A-B}(-\vec{v_i})$

Abbildung 8 zeigt Initialisierung und drei Iterationen des GJK-Algorithmus.

**Abbruchkriterium**
Die Iterationen werden solange wiederholt, bis ein Abbruchkriterium erfüllt ist.
Es wird abgebrochen, sobald der Ursprung selbst im Simplex enthalten ist und der ursprungsnächste Punkt $\vec{v_i} = \vec{0}$. Dies ist insbesondere in $d$ Dimensionen immer der Fall, wenn die verkleinerte Simplexmenge am Ende der Iteration $d+1$ Punkte enthält. In 3D kann die Simplexmenge also maximal vier Punkte enthalten, die ein Tetraeder bilden.
Dieses schließt den Ursprung dann sicher ein. Hier wird der Abstand null als Ergebnis zurückgegeben, da aufgrund der Konvexität von $A-B$ der Ursprung auch sicher im CSO liegt, wenn er Teil des Simplex ist.
Liegt der Ursprung außerhalb, wird die Approximation abgebrochen, sobald eine gewisse Genauigkeit erreicht wurde. Hierzu werden eine obere und eine untere Schranke für den Abstand bestimmt. Die aktuelle Suchrichtung $\vec{v_i}$ ist ein Punkt im CSO. $\|\vec{v_i}\|$ ist daher eine obere Schranke für den Abstand. Der Supportpunkt $\vec{w_i}$ liegt am nächsten in Richtung $\vec{v_i}$ am Ursprung. Der Körper liegt also vollständig hinter der Ebene mit Normalen $\vec{v_i}$, in der $\vec{w_i}$ liegt (gestrichelte Linie in Abb. 8). Der Abstand zu dieser *Supportebene* stellt eine untere Schranke dar. Er ist durch

$$\frac{\vec{w_i} \cdot \vec{v_i}}{\|\vec{v_i}\|}$$

gegeben.
Sind diese beiden Grenzen nah genug, kann die Iteration beendet werden und der ermittelte angenäherte Abstand $\|\vec{v_i}\|$ zurückgegeben werden.
Es existiert zusätzlich ein Beweis für Konvergenz des Vektors $\vec{v_i}$ gegen den Abstandsvektor [26, S. 123 ff].

Der GJK-Algorithmus ist auf alle Körper, für die eine Supportfunktion angegeben werden kann, anwendbar: Er bestimmt zuverlässig den Abstand und ob eine Kollision vorliegt. Für die Berechnung des Penetrationsvektors, der zur Kollisionsauflösung benötigt wird, ist er allerdings nicht geeignet. Hierfür wird ein weiterer Algorithmus benötigt.

## 2.3 Expanding Polytope Algorithm (EPA)

Die wohl exakteste Methode zur Bestimmung des Penetrationsvektors, des kürzesten Vektors außerhalb des CSO, ist der *Expanding Polytope Algorithm* (EPA) [26]. Auch er verwendet Supportfunktionen und rechnet im CSO.
Als Ausgangsbasis benötigt er vier Supportpunkte, die ein Tetraeder bilden. Dieses Tetraeder $P_0$ muss den Ursprung enthalten, damit EPA funktioniert. Eine gute Ausgangsbasis stellt das Tetraeder dar, mit dem eine GJK-Berechnung abbricht, wenn der Ursprung im CSO liegt.
Nun wird dieses zu einem Polytop zu erweitert, das die Oberfläche des CSO annähert. So soll der Penetrationsvektor bestimmt werden. Das Polytop in Iteration $i$, $P_i = conv(W_i)$, wird durch eine Menge von Supportpunkten $W_i$ definiert, wobei zu Beginn die vier bekannten Punkte die Menge $W_0$ bilden.
Zur Approximation werden folgende Schritte für $i = 1, 2, ...$ wiederholt:

- Der Punkt $\vec{v_i}$ auf dem Rand $\partial P_i$ des Polytopes, welcher dem Ursprung am nächsten ist, wird bestimmt. $\|\vec{v_i}\| = min\{\|\vec{x}\| : \vec{x} \in \partial P_i\}$
- Es wird ein neuer Supportpunkt $\vec{w_i}$ in Richtung $\vec{v_i}$ bestimmt. $\vec{w_i} = \vec{s}_{A-B}(\vec{v_i})$

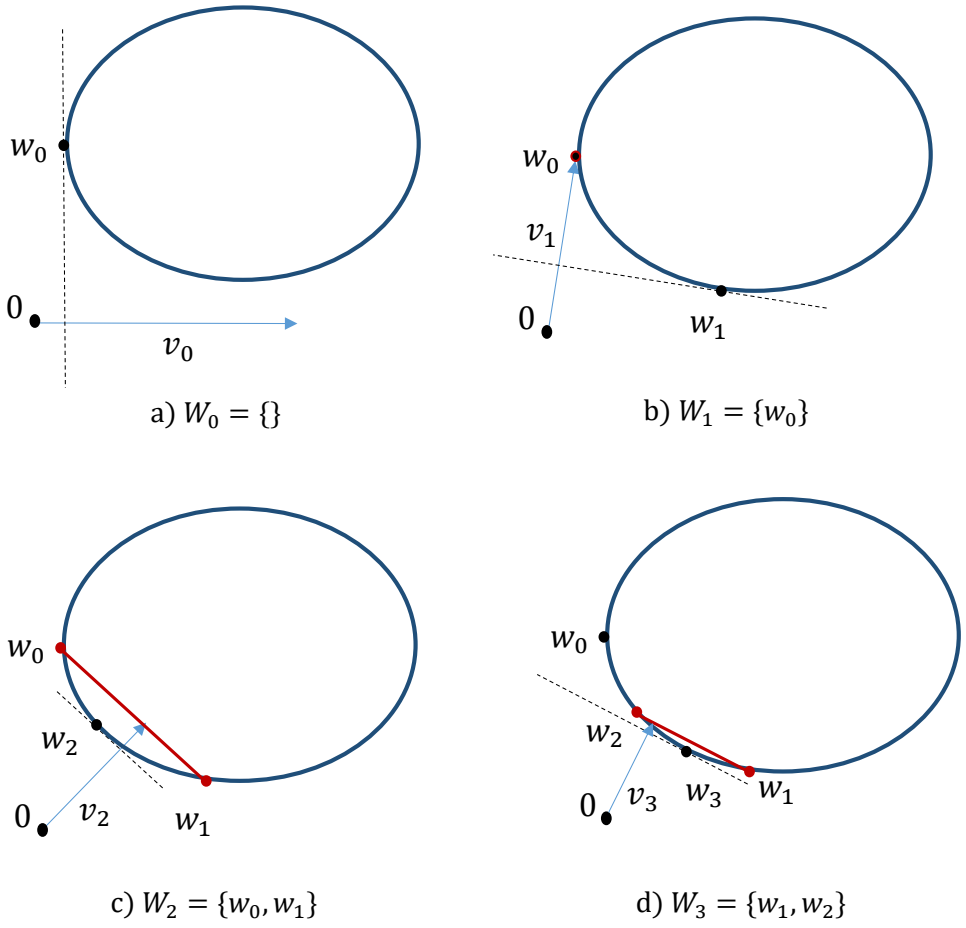



Abbildung 8: Vier GJK-Iterationen am CSO (blau) in 2D. Nach der Initialisierung in Bild a) werden drei Durchläufe ausgeführt. $W_i$ gibt die Simplexmenge nach Entfernung der überflüssigen Punkte an. Nach Schritt c) wird die Simplexmenge beispielsweise erst um $w_2$ erweitert, sodass $W_3^* = \{w_0, w_1, w_2\}$. Das Simplex aus den Punkten $w_1, w_2$ enthält den nächsten Punkt $v_3$ allerdings auch, weshalb $w_0$ überflüssig und nicht Teil der neuen Simplexmenge $W_3$ ist.

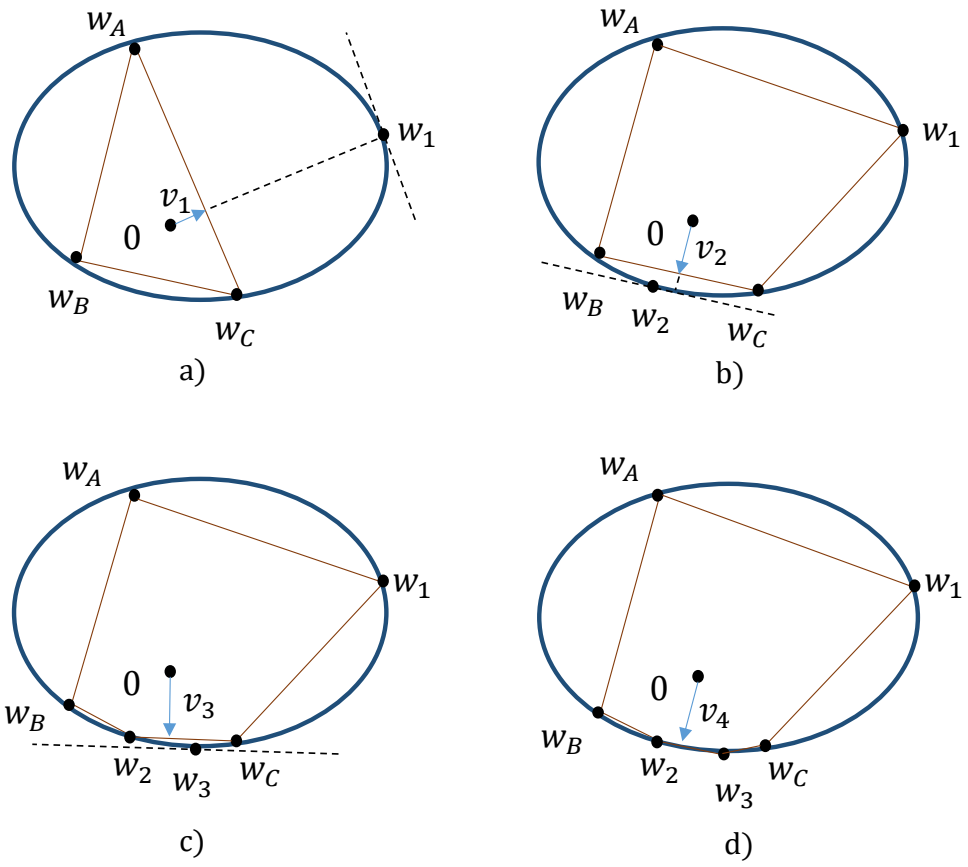



Abbildung 9: Vier Schritte des *Expanding Polytope Algorithm* in 2D. Ausgangsbasis ist hier ein Dreieck aus Supportpunkten $w_A, w_B, w_C$, das den Ursprung enthält. Der ursprungsnächste Punkt $v_i$ wird in jedem Schritt durch den blauen Pfeil markiert.
Zur Berechnung des nächsten Punktes wird in 2D eine Liste aller Kanten geführt. Bei Hinzukommen eines neuen Punktes wird die entsprechende Kante in der Liste durch ihre zwei Nachfolgerkanten ersetzt.

- Der Punkt $\vec{w_i}$ wird in die Punktmenge $W_i$ aufgenommen und das Polytop $P_i$ aktualisiert. $W_i = W_{i-1} \cup \vec{w_i}, P_i = conv(W_i)$

Abbildung 9 zeigt vier solche Iterationen.
Der am schwierigsten zu implementierende Schritt ist die effiziente Bestimmung des nächesten Punktes auf dem Rand des Polytopes. Hierzu ist es nötig, eine Auflistung aller Seitenflächen des Polytopes mit dem Abstand ihres jeweils ursprungsnächsten Punktes in einer Prioritätswarteschlange (Heap) zu führen.
Kommt ein neuer Punkt hinzu, müssen durch ihn neu erzeugte Facetten hinzugefügt und die nicht mehr aktuellen Flächen verworfen werden. Im Regelfall wird die Dreiecksfläche, welche den nächsten Punkt $\vec{v_i}$ enthält, durch drei neue Dreiecksflächen mit dem berechneten Supportpunkt ersetzt (Abb. 10, oberes Bild).
Wie in der Abbildung führt dies aber nicht immer zur korrekten Darstellung der konvexen Hülle. Es ist nötig, alle vom Punkt $w_i$ aus sichtbaren Knoten mit diesem zu verbinden. Die Verbindungslinie dieser Knoten wird in diesem Zusammenhang als *Silhouette* bezeichnet. Die Dreiecke innerhalb der Sihouette (und damit auch deren Kanten) sind aus der Liste zu entfernen. Weitere Details zur Implementierung und den benötigten Datenstrukturen sind in der

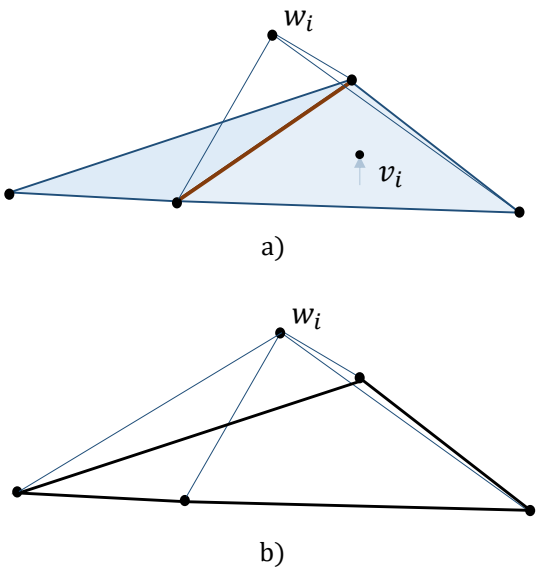



Abbildung 10: Hinzufügen eines Supportpunktes $w_i$ zum EPA-Polytop. Bild a) deutet die einfache Methode an, bei der die Fläche mit dem nächsten Punkt $v_i$ entfernt und ihre Kanten mit $w_i$ verbunden werden. So entstehen drei neue Flächen.
Dieses Vorgehen bildet die konvexe Hülle allerdings nicht richtig ab, die rote Kannte ist fehlerhaft. Für eine richtige Verarbeitung, wie in Bild b) ist es notwendig die Silhouette (schwarz) zu bestimmen. Dann wird $w_i$ mit allen Punkten auf der Silhouette verbunden.

Publikation ausführlich beschrieben [26, S. 147ff].

**Abbruchkriterium**
Wie GJK benötigt auch EPA ein Abbruchkritierium. Die Schranken des GJK sind hier ebenfalls anwendbar, mit dem Unterschied, dass $\|\vec{v_i}\|$ nun die untere Schranke und der Abstand zur Supportebene $\frac{\vec{w_i} \cdot \vec{v_i}}{\|\vec{v_i}\|}$ nun die obere Schranke für die Penetrationstiefe darstellen.

EPA ist wie GJK auf alle konvexen Körper mit Supportfunktion anwendbar. Der Algorithmus konvergiert gegen den (oder einen von mehreren) Penetrationsvektor und liefert somit präzise Kontaktinformationen.

## 2.4 Minkowski Portal Refinement (MPR)

Eine relativ neue Methode zur Kollisionserkennung stellt das *Minkowski Portal Refinement* (MPR) dar. Dieser Algorithmus wurde 2007 von Gary Snethen erstmals unter dem Namen *XenoCollide* beschrieben [20,21]. Auch hier finden alle Berechnungen im CSO statt. Zusätzlich zur Supportfunktion des Körpers wird eine *Centerfunktion* benötigt, die den geometrischen Mittelpunkt oder einfach nur einen Punkt nahe diesem liefert. Subtrahiert man die beiden Mittelpunkte der Körper, erhält man einen Punkt in der Mitte des CSO.

**Beschreibung des Algorithmus in 2D**
Wichtigstes Konzept des MPR ist der *Ursprungsstrahl*. Diese Halbgerade beginnt im Mittelpunkt des CSO, `v0`, und geht von dort aus durch den Ursprung und weiter (Abb. 11(b)). Befindet sich der Ursprung im CSO, so passiert der Ursprungstrahl diesen vor der Oberfläche des CSO.

Der erste Schritt des MPR wird als *Portal Discovery* bezeichnet. In 2D ist ein *Portal* eine Strecke zwischen zwei Punkten auf der Oberfläche des CSO, die der Ursprungsstrahl kreuzt (Die Strecke `V1-V2` in Abb. 11(f) ist ein solches Portal).
Um ein Portal zu finden, werden folgende Schritte gewählt:

- Ein Supportpunkt `V1` in Richtung des Ursprungstrahles wird berechnet (Abb. 11(b, c)).
- `V0` und `V1` werden verbunden. Die senkrechte Richtung $\vec{n}$ zur Strecke, welche in Richtung des Ursprungs weist, wird bestimmt (Abb. 11(d)).
- Ein Supportpunkt `V2` in diese senkrechte Richtung $\vec{n}$ wird berechnet (Abb. 11(d)).
- Die Punkte `V0` und `V2` werden verbunden (Abb. 11(e)).
- Liegt der Ursprung zwischen den Geraden `V0-V1` und `V0-V2`, so ist die Strecke `V1-V2` ein Portal, da der Ursprungstrahl sie sicher kreuzt (Abb. 11(f)).
  - Liegt der Ursprung nicht zwischen den enstandenen Geraden, geht man folgendermaßen vor:
  - Der vorletzte Punkt (Hier: `V1`) wird verworfen.
  - Ein neuer Supportpunkt in Richtung der zum Ursprung zeigenden Senkrechten der verbliebenen Strecke (Hier: `V0-V2`) wird generiert.
  - Dieses Vorgehen wird wiederholt, bis ein Portal gefunden wurde

Wurde ein Portal gefunden, wird der *Portal Refinement* Schritt durchgeführt.

- Zuerst wird geprüft, ob sich der Ursprung vor oder hinter der Portalfläche befindet. Hierzu wird die nach außen zeigende Portalnormale berechnet ($\vec{n}$ in Abb. 11(g)) und das Portal sowie der Ursprung auf diese projiziert.
  - Liegt der Ursprung vor dem Portal (also in Richtung des Punktes `V0`), liegt er auch im Dreieck welches durch `V0` und die Portalstrecke `V1-V2` gebildet wird. Er liegt also auch im konvexen CSO. Die Kollision wird detektiert und der Algorithmus wird an dieser Stelle beendet.
- Ist dies nicht der Fall, muss das Portal verfeinert werden. Dazu wird ein neuer Supportpunkt `V3` in Richtung der Portalnormale bestimmt. (Abb. 11(g))
- Dieser Punkt `V3` wird auf die Portalnormale projiziert (gestrichelte Linie). Liegt er vor dem Ursprung, so kann der Algorithmus ebenfalls beendet werden. In diesem Fall gibt es keine Kollision.
- Je nachdem auf welcher Seite der Strecke `V0-V3` sich der Ursprung befindet, wird der Supportpunkt auf der Gegenseite (hier `V2`) verworfen. (Abb. 11(h))
- Die verbliebenen beiden Supportpunkte (hier `V1` und `V3`) bilden ein neues Portal. Es wird wieder mit dem ersten Punkt des *Portal Refinement* begonnen. (Abb. 11(i))

**Anwendung in drei Dimensionen**
Der Algorithmus kann mit wenigen Anpassungen auf drei Dimensionen verallgemeinert werden. Statt einer Strecke ist das Portal nun ein Dreieck und bildet zusammen mit dem Mittelpunkt ein Tetraeder.
Für eine ausführliche Beschreibung wird an dieser Stelle auf die Veröffentlichung in *Game Programming Gems 7* verwiesen [21].

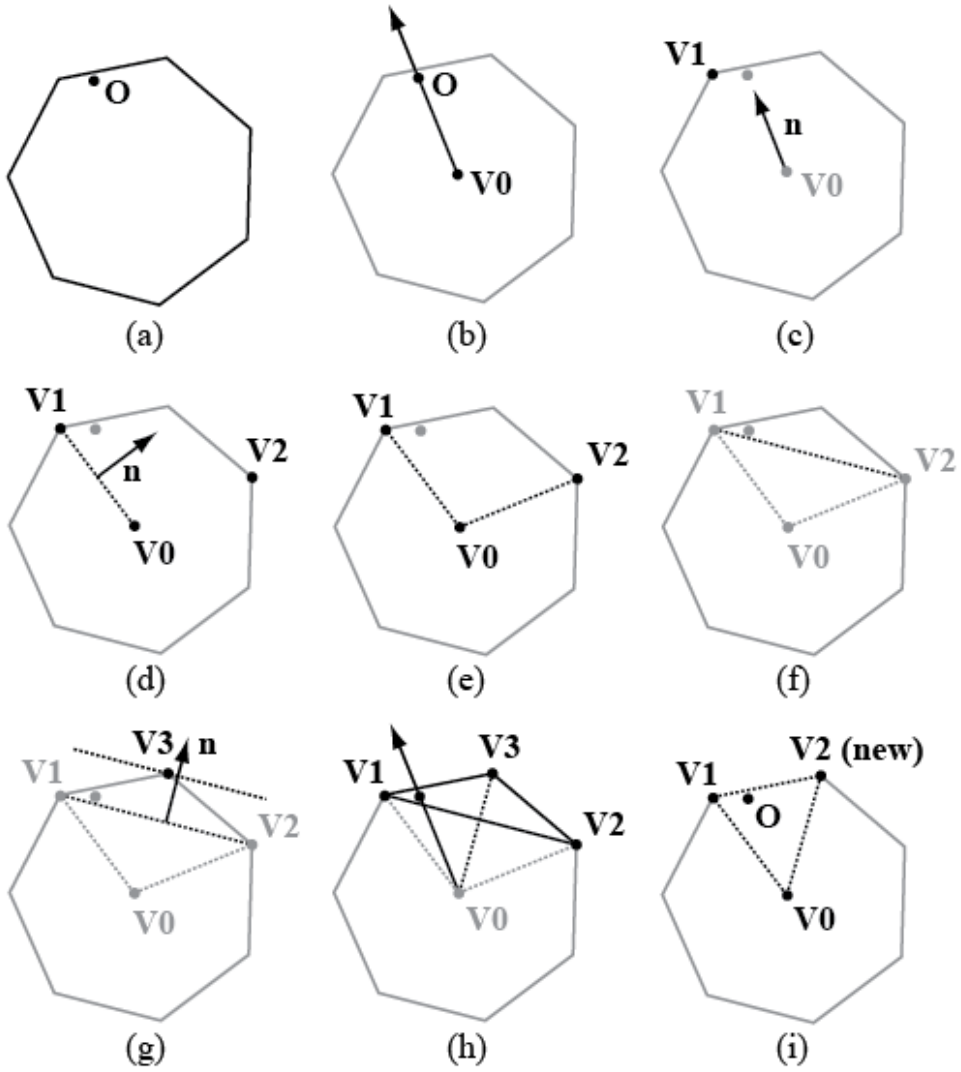



Abbildung 11: Schritte des MPR-Algorithmus in 2D. (Quelle: [20])

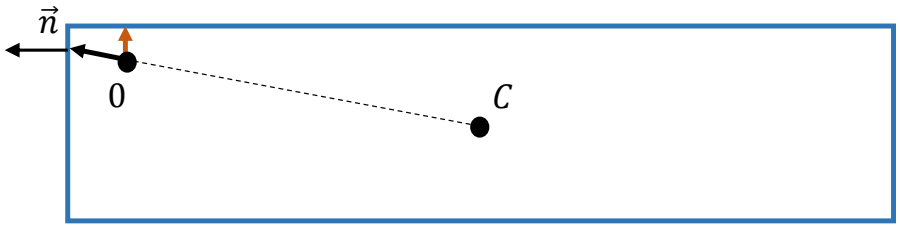



Abbildung 12: Beispiel des CSO zweier Rechtecke, für die MPR eine falsche Kontaktnormale liefern würde. Der analytische Penetrationsvektor ist rot markiert, der MPR-Vektor schwarz.

**Kontaktdatenbestimmung mittels MPR**
Verfahren zur Bestimmung eines Penetrationsvektors mit MPR werden in diesem Werk ebenfalls beschrieben.
Hierfür wird der Algorithmus, sobald die Kollision erkannt wurde, nicht abgebrochen, sondern fortgesetzt, bis sich das Portal der Oberfläche des CSO bis auf eine bestimmte Genauigkeit angenähert hat. Der Vektor vom Ursprung bis zum Portal kann als Penetrationsvektor verwendet werden.
Eine weitere Möglichkeit besteht darin, die gesamte Berechnung nach Erkennung der Kollision noch einmal für einen alternativen Ursprungsstrahl durchzuführen. Dieser geht nicht von dem Mittelpunkt aus zum Ursprung, sondern verläuft vom Ursprung in Richtung der relativen Bewegung des Körpers A zu B. Dieser Strahl wird ebenfalls durch ein Portal bis an die Oberfläche angenähert und kann anschließend als Penetrationsvektor verwendet werden. Dies führt zu besseren Resultaten bei dynamischen Kontakten. Der Vektor zeigt allerdings immer in die Richtung der relativen Bewegung.

**Stärken und Schwächen**
Wie GJK und EPA funktioniert auch MPR für alle konvexen Formen. Es wird nur die entsprechende Supportfunktion benötigt. Er ist also extrem vielseitig einsetzbar und stellt die einzige funktional gleichwertige Alternative zum GJK dar.
Ziel des MPR war es, eine weniger mathematische Methode zu entwickeln, die für viele Programmierer einfacher zu implementieren ist. Dies ist auch der Fall, da MPR leicht zu visualisieren ist und nicht mit Determinanten und Ähnlichem arbeitet.
Einen Vergleich von MPR und GJK-EPA führt Joshua Newth durch [15]. Auch er kommt zu dem Schluss, dass MPR der verständlichere Algorithmus ist und auch deutlich weniger Fallunterscheidungen enthält. Zur reinen Erkennung von Kollisionen stellt das Verfahren daher eine Verbesserung gegenüber GJK dar.
Die Schwäche des Algorithmus liegt in der Berechnung von Kontaktdaten. Der Vektor des Ursprungsstrahls auf die Oberfläche liefert in vielen Fällen eine gute Annäherung des Penetrationsvektors, in anderen, wie in Abb. 12, allerdings eine gänzlich andere Richtung.
Eine vom Autor vorgeschlagene Verbesserung ist es, die ermittelte Portalnormale (entspricht der angenäherten Oberflächennormalen $\vec{n}$) als Kontaktnormale zu verwenden. Auch diese liefert im abgebildeten Fall nicht das korrekte Ergebnis.
Eine passende Kontaktnormale ist für die physikalisch korrekte Kollisionsauflösung unverzichtbar. Sie zu bestimmen ist mit MPR nicht zuverlässig möglich.

## 2.5 Weitere Algorithmen

Es existieren noch weitere Algorithmen, die zwar für eine Vielzahl von Formen, aber nicht für alle konvexen Körper einsetzbar sind. Zwei bekannte Algorithmen, die beide auf konvexe Polytope beschränkt sind, werden hier noch kurz vorgestellt.

**Chung-Wang-Algorithmus**
Der Algorithmus von Chung und Wang beschreibt eine Methode, für zwei konvexe Polytope eine trennende Achse (*Seperating Axis*) zu finden [4]. Ein Vektor $\vec{v}$ ist trennende Achse zweier Körper $A$ und $B$, wenn für alle $\vec{x} \in A, \vec{y} \in B$ gilt:

$$\vec{v} \cdot \vec{x} > \vec{v} \cdot \vec{y}$$

Eine Ebene mit dieser Normale $\vec{v}$ kann so zwischen den Objekten plaziert werden, dass eines vollständig auf der positiven und das Andere auf der negativen Seite liegt.
Existiert so eine trennende Achse, können die Körper offensichtlich nicht in Kontakt sein. So kann der Chung-Wang-Algorithmus bestimmen, ob zwei Polytope kollidieren.
Zur Beschreibung des Polytopes wird nur die Supportfunktion benötigt. Deshalb klingt es einfach, den Algorithmus auf konvexe Nicht-Polytope zu verallgemeinern. Diese Möglichkeit wurde in [26, S. 111ff] untersucht. Die Methode von Chung und Wang geht ebenfalls iterativ vor, der Aufwand steigt allerdings mit jeder Iteration. Insgesamt wurde festgestellt, dass $k$ Iterationen einen Aufwand in $O(k^2)$ haben und deshalb nur bei geringer Iterationszahl eine akzeptable Rechenzeit möglich ist.
Die maximal mögliche Iterationszahl für die Berechnung hängt von der Anzahl möglicher Supportpunkte ab. Für Polytope mit $m$ und $n$ Punkten (diese sind die möglichen Supportpunkte) ergibt sich das Maximum zu $mn$ Iterationen. Für kontinuierliche Körper gibt es unendlich viele mögliche Supportpunkte. Deshalb ist eine unbegrenzte Iteration möglich. Die Konvergenz kann zudem sehr langsam sein.
Anders als beim GJK-Algorithmus sind für allgemeine Körper keine sicheren Aussagen zur Laufzeit möglich, was einen Einsatz in interaktiven Anwendungen ausschließt.

**Lin-Canny-Algorithmus**
Die Algorithmus von Lin und Canny berechnet die sich nächsten Grundelemente (Eckpunkte, Kanten, Facetten) zweier konvexer Polytope [14]. Die Methode beginnt mit einem beliebigen Element. Ist dies nicht das nächste, wird ein näheres angrenzendes Element ausgewählt und untersucht. Aufgrund der Konvexität kann die Entfernung mit jedem Schritt verringert werden, bis die nächsten Elemente (und die nächsten Punkte) gefunden sind. So wird der Abstand bestimmt.
Dieser Algorithmus kam zuerst in der I-COLLIDE-Bibliothek [5] zum Einsatz. Inzwischen sind viele verbesserte Versionen im Umlauf, die aber auf derselben Idee basieren.
Der Einsatz dieses Verfahrens ist aber auf konvexe Polytope beschränkt, da nur deren Oberfläche vollständig durch Ecken, Kanten und Flächen dargestellt werden kann.

## 2.6 Auswahl eines geeigneten Verfahrens

Nur zwei Algorithmen erfüllen die Anforderung, für beliebige konvexe Körper einsetzbar zu sein: Eine Kombination aus GJK zur Erkennung und EPA zur Berechnung des Penetrationsvektors, sowie MPR.
Nachdem die Generierung von Kontaktdaten mit MPR nicht immer belastbar funktioniert, bleibt EPA für diesen Zweck als einzige Option.
Ohne größere Anpassungen ist es nicht möglich, MPR zur Kollisionserkennung und EPA zur

Kontaktberechnung einzusetzen. Das aus Portal und Mittelpunkt geformte Tetraeder (als Ergebnis von MPR) kann nicht anstelle des GJK-Simplex als Startpolytop für EPA verwendet werden. Die Punkte dieses Polytopes müssen alle auf der Oberfläche des CSO liegen, was für den Mittelpunkt nicht der Fall ist.
Da EPA in Kombination mit GJK zudem ein erprobtes und belastbares Verfahren darstellt, ist es wohl die beste Alternative und soll für die Implementierung verwendet werden.

# 3 Durchgeführte Verbesserungen

## 3.1 Ausgangslage

Als Grundlage der Arbeit soll eine bereits bestehende Implementierung des Algorithmus verwendet, in den *pe Physics Engine* integriert und für diesen optimiert werden. Hierzu wurden die folgende Bibliotheken betrachtet:

**dyn4j**
Das *dyn4j*-Paket enthält sowohl Kollisionserkennung als auch Physik-Engine [1]. Zur Kollisionserkennung werden GJK und EPA, sowie das Seperating-Axis-Theorem verwendet (nur bei Polygonen). Diese ist allerdings in Java implementiert und ohne größeren Aufwand nicht in den *pe Physics Engine*, der in C++ geschrieben wurde, zu integrieren. Aufgrund der hohen Verbreitung von Java kommt *dyn4j* oft zum Einsatz und ist durchaus eine Betrachtung wert.

**libccd**
Eine bekannte Open-Source-Bibliothek für iterative Kollisionserkennung stellt die *libccd* [7] dar. Sie enthält eine Implementierung der GJK-Algorithmus und des EPA. Zusätzlich ist die libccd laut dem Entwickler die einzige Open-Source-Software, in der auch der MPR-Algorithmus verfügbar ist. Die ähnlichen Signaturen ermöglichen einen einfachen Vergleich von GJK-EPA und der MPR. Der Code wurde in C implementiert. Nun soll die Implementierung analysiert werden, um festzustellen, ob diese im *pe Physics Engine* zum Einsatz kommen könnte.

**SOLID**
Die SOLID-Bibliothek geht auf den renommierten Spielephysik- und Graphikexperten Gino van den Bergen zurück. Auch sie verwendet den GJK, enthält aber bereits diverse Optimierungen [24, 25]. Zudem ist sie in der Programmiersprache C++ implementiert, weshalb der Code direkt in den *pe Physics Engine* eingebunden werden kann. Tests in den Veröffentlichungen zu SOLID weisen eine gute Performance aus. So beschreibt der Autor ein Experiment [25], in dem etwa die fünffache Geschwindigkeit der *I-COLLIDE*-Bibliothek erreicht wurde, obwohl diese nicht auf allgemeine Körper anwendbar ist.

**ReactPhysics3D**
Eine Implementierung des GJK und EPA ist auch in *ReactPhysics3D* vorhanden [3]. Das Open-Source-Projekt beherrscht die Simulation einer Vielzahl von Körpern und verfügt auch über eine Coarse Collision Detection. Implementierungssprache ist hier ebenfalls C++. Der Code ist dem von *SOLID* sehr ähnlich und enthält auch dessen Optimierungen.

Von den oben genannten Software-Bibliotheken sind die *libccd* und eine Implementierung, welche auf dem Code des *SOLID*-Frameworks basiert, wohl am besten geeignet. *SOLID* selbst wurde hier aufgrund der Dokumentation in vielen Publikationen dem *ReactPhysics3D*-Code vorgezogen. Die Implementierungen sollen nun analysiert und optimiert werden.

## 3.2 Analyse der *LibCCD*-Implementierung

Zuerst wurde die *libccd* untersucht. Für einen ersten Eindruck wurden einfache Testfälle konstruiert und ausgeführt (eine genaue Beschreibung der Fälle findet sich in Abschnitt 4.1.1). Hierbei kamen beim EPA deutliche Schwächen zum Vorschein.

**Durchführung des Kollisionserkennungstests**
Als Erstes wurde die reine Kollisionserkennung (ohne Kontaktdaten) getestet. Hierbei wurde ermittelt, ab welchem Abstand eine Kollision erkannt wird (*Detektionsabstand*). Es wurde ein Verfahren verwendet, das dem der binären Suche ähnelt (siehe Algorithmus 1). Die Körper werden in Berührkontakt gebracht und die Kollisionserkennung aufgerufen.
Wird die Kollision dann erkannt, werden die Körper um eine Distanz $\Delta s$ auseinander, wird sie nicht erkannt, zusammengeschoben. Die Distanz $\Delta s$ wird halbiert und es wird wieder von vorne begonnen.
Liegt der Detektionsabstand im Intervall $[-1; 1]$ (-1 entspricht einer Eindringtiefe von 1; 0 entspricht dem Berührkontakt), so kann zu Beginn jedes Schleifendurchlaufes in Algorithmus 1 der Detektionsabstand im Intervall $[abstand - 2\Delta s; abstand + 2\Delta s]$ verortet werden. $\Delta s$ wird solange verrringert, bis das Intervall exakt genug ist. So kann die Erkennungsschwelle genau bestimmt werden.
Der *libccd* können als Parameter zusätzlich noch eine maximale Iterationszahl für den GJK und ein Toleranzwert für den MPR übergeben werden. Wählt man beide ausreichend hoch (max. 100 GJK-Iterationen und $10^{-5}$ als Toleranz für MPR) so ist der Betrag des Detektionsabstandes in allen Fällen und mit beiden Algorithmen kleiner als $2 \times 10^{-10}$. Dies ist sehr exakt.

**Algorithmus 1** Messung des Detektionsabstandes

**Require:** Körper A, B
1: **procedure** MESSEDETEKTIONSABSTAND(A, B)
2: *setze die Körper A und B in Berührkontakt.*
3: $\Delta s \leftarrow 0,5$
4: $abstand \leftarrow 0$
5: **while** $s > 1 \times 10^{-10}$ **do** ▷ Fahre fort, bis Detektionsabst. genau genug
6: **if** DOCOLLIDE(A,B) **then**
7: schiebe die Körper um $\Delta s$ auseinander ▷ Detektionsabst. ist größer
8: $abstand \leftarrow abstand + \Delta s$
9: **else**
10: schiebe die Körper um $\Delta s$ zusammen ▷ Detektionsabst. ist kleiner
11: $abstand \leftarrow abstand - \Delta s$
12: **end if**
13: $\Delta s \leftarrow \Delta s/2$
14: **end while**
15: **return** abstand
16: **end procedure**

**Analyse der generierten Kontaktdaten**
Zusätzlich zum Erkennungstest wurden die erhaltenen Kontaktdaten, bestehend aus Eindringtiefe, Normalenvektor und Kontaktpunkt, ausgewertet. Hier kam es auch bei hohen Iterations- und kleinen Toleranzwerten zu falschen Normalen (Abweichung bis zu 65°) und auch die Kontaktpunkte waren sehr ungenau. Die Implementierung machte insgesamt keinen verlässlichen

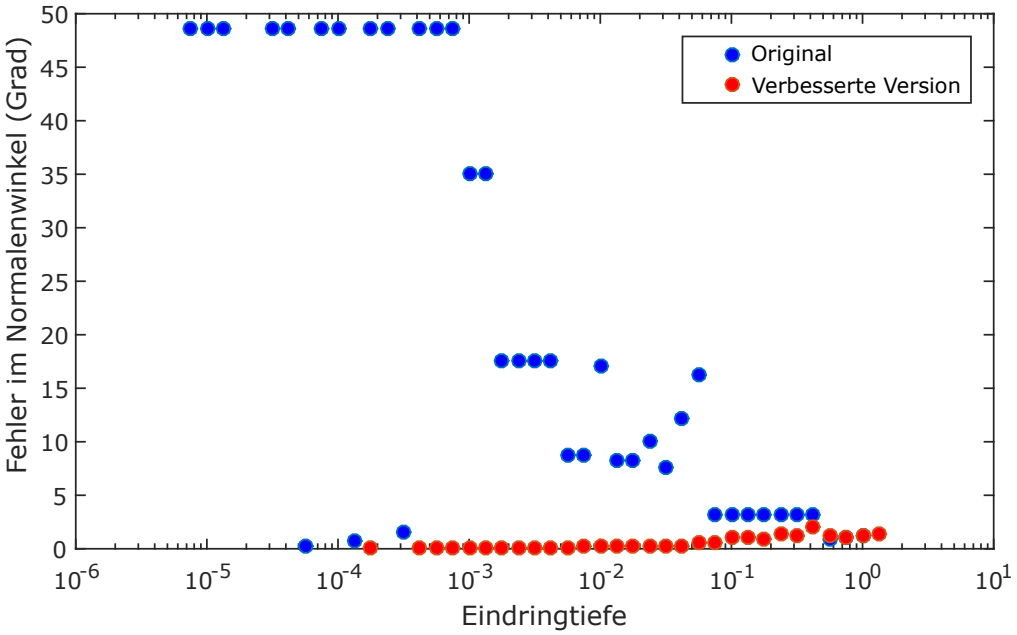



Abbildung 13: Fehler des von *libccd* berechneten Normalenwinkel bei verschiedenen Eindringtiefen. Die Verbesserte Version enthält alle in diesem Kapitel beschreibenen Optimierungen.Für Tiefen ohne rote Punkte wurde die Kollision, wegen des erhöhten Toleranzwertes, nicht erkannt.

Eindruck. So schwankt der Normalenvektor beispielsweise in einem Testfall, bei dem eine Kugel auf die Ecke eines Quaders trifft (genaue Darstellung in Abschnitt 4.1.1 und Abb. 19, Test 1), abhängig von der Penetrationstiefe sehr stark (Abbildung 13, Original). Die Ursachen hierfür sollen nun kurz untersucht werden. Zudem wurden einige Laufzeitexplosionen beobachtet, die auf schlechte Konvergenz hinweisen.

### 3.2.1 Durchgeführte Optimierungen

**Relatives Abbruchkriterium für den EPA**

Es konnte ermittelt werden, dass es in einigen Fällen im EPA zu verfrühtem Abbruch kam, weshalb der Winkel der Normalen nicht nah genug am richtigen Ergebnis liegt. Der Grund hierfür liegt im gewählten Abbruchkriterium. In der *libccd* wird nach Erhalt eines neuen Supportpunktes $\vec{w_i} = s_{A-B}(\vec{v_i})$ in Richtung des bisher nächsten, im Polytop enthaltenen Punktes $\vec{v_i}$ folgendes Kriterium ausgewertet:

$$\frac{\vec{w_i} \cdot \vec{v_i}}{\|\vec{v_i}\|} - \|\vec{v_i}\| < \epsilon_{abs}$$

Dies entspricht, wie in Abschnitt 2.3 beschrieben, dem Abstand des nächsten Punktes innerhalb des EPA-Polytopes zur Supportebene. Der Wert von $\epsilon_{abs}$ kann durch den Benutzer vorgegeben werden, ein Beispiel auf der Website verwendet den Wert $10^{-4}$. Es kommt zu zwei Problemen: Für große Objekte kann der maschinenbedingte Rundungsfehler die Größenordnung von $\epsilon_{abs}$ erreichen und dies zu einem ungewollten Abbruch oder auch zu einer Endlositeration führen.
Für kleine Eindringtiefen im Bereich von $\epsilon_{abs}$ ist der erlaubte Fehler allerdings viel zu groß und verhindert eine bessere Approximation.

Dieses Problem lässt sich durch Einführung einer relativen Fehlerschranke [26] beheben. Der absolute Fehler wird durch

$$\epsilon_{abs} = \epsilon_{rel}\|\vec{v}\|$$

ersetzt und die Abbruchbedingung ergibt sich zu

$$\frac{\vec{w}\cdot\vec{v}}{\|\vec{v}\|} - \|\vec{v}\| < \epsilon_{rel}\|\vec{v}\|$$

$$\leftrightarrow \frac{\vec{w}\cdot\vec{v}}{\|\vec{v}\|} < (1+\epsilon_{rel})\|\vec{v}\|$$

$$\leftrightarrow \vec{w}\cdot\vec{v} < (1+\epsilon_{rel})\|\vec{v}\|^2$$

Im letzten Schritt wurde, um das Ziehen der Wurzeln (bei der Berechnung der Norm) zu vermeiden, noch mit $\|\vec{v}\|$ multipliziert.
Dieses Kriterium wurde in der *libccd* eingesetzt und führte zu besseren Resultaten (Abbildung 13, Verbesserte Version).

**Kontaktpunktbestimmung mit Linearkombinationen**
Als Nächstes wurde das Augenmerk auf die Berechnung der Kontaktpunkte gerichtet. Die "Rückwärtsrechnung" vom CSO in das normale Koordinatensystem ist nicht einfach möglich. Es hilft, für jeden Supportpunkt des CSO $\vec{w_i}$ die ursprünglichen Punkte $\vec{p_i} \in A$ und $\vec{q_i} \in B$, aus welchen sich dieser ergibt ($\vec{w_i} = \vec{p_i} - \vec{q_i}$) zu speichern. Dies tut die *libccd*. Anschließend wird folgendes Vorgehen gewählt:

- Alle Supportpunkte $\vec{w_i}$ des EPA-Polytop werden nach ihrem Abstand zum Ursprung sortiert.
- Die hintere Hälfte der Punkte wird verworfen.
- Zu allen Supportpunkten aus der verbliebenen Hälfte (!), werden die gespeicherten Punkte $\vec{p_i}$ und $\vec{q_i}$ der Originalkörper betrachtet.
- Der Mittelwert dieser wird als Ergebnis verwendet.

Dieses Verfahren hat jedoch keine geometrische oder mathematische Grundlage und liefert nur einen Punkt, der ungefähr in die Richtung des realen Kontaktpunktes zeigt. Für die Genauigkeitsanforderungen unserer Physiksimulation ist dies nicht tolerierbar.
Als Verbesserung wurde das für zwei Dimensionen in [26, S. 151] beschriebene Verfahren in 3D implementiert. Es geht folgendermaßen vor: Am Ende des EPA gibt es einen Punkt $\vec{v}$ im CSO, welcher dem Ursprung am nächsten ist. Dieser entspricht dem angenäherten Penetrationsvektor und ist Teil eines Dreiecks, welches von den Supportpunkten $\vec{w_1}$, $\vec{w_2}$ und $\vec{w_3}$ aufgespannt wird. $\vec{v}$ lässt sich als Linearkombination dieser ausdrücken:

$$\vec{v} = \lambda_1\vec{w_1} + \lambda_2\vec{w_2} + \lambda_3\vec{w_3}$$

$$\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}^0_+, \lambda_1 + \lambda_2 + \lambda_3 = 1$$

Wie im ursprünglichen Verfahren der *libccd*, speichert man nun zu jedem Supportpunkt $\vec{w_i} = \vec{p_i} - \vec{q_i}$ die zugehörigen Punkte $\vec{p_i} \in A$ und $\vec{q_i} \in B$. Nun kann man Punkte $\vec{a} \in A$ und $\vec{b} \in B$ berechnen.

$$\vec{a} = \lambda_1\vec{p_1} + \lambda_2\vec{p_2} + \lambda_3\vec{p_3}$$
$$\vec{b} = \lambda_1\vec{q_1} + \lambda_2\vec{q_2} + \lambda_3\vec{q_3}$$

Es kann durch Einsetzen dieser Gleichungen gezeigt werden, dass $\vec{a} - \vec{b} = \vec{v}$ gilt. Die beiden Punkte $\vec{a}$ und $\vec{b}$ haben also den Abstand einer Länge des Penetrationsvektors und stellen den jeweiligen Berührpunkt dar, falls einer der Körper in bzw. entgegen der Richtung des Pentetrationsvektors verschoben wird. Da im *pe Physics Engine* wie auch in der *libccd* pro Kollision nur ein Kontaktpunkt $\vec{c}$ geliefert werden soll, wird der Mittelpunkt der Strecke hierfür verwendet.

$$\vec{c} = \frac{1}{2}(\vec{a} + \vec{b})$$

**Toleranzwert für Double-Precision**
Die *libccd* führt sämtliche Vergleiche von Gleitkommazahlen mit einem bestimmten Toleranzwert $\epsilon$ durch. Zwei Zahlen $Z_1$ und $Z_2$ werden als gleich betrachtet, wenn $|Z_1 - Z_2| < \epsilon$.
Als Zahlenwert für $\epsilon$ werden die C-Konstanten `FLT_EPSILON` bzw. `DBL_EPSILON` für einfache oder doppelte Genauigkeit verwendet.
Diese Konstante enthält den kleinsten Wert $x$, sodass

$$1 + x \neq 1$$

gilt. Der Wert gibt also an, wie genau eine Zahl, die Eingabedatum einer Funktion ist, darstellbar ist.
In der *libccd* wird dieser Toleranzwert beispielsweise auch verwendet, um zu überprüfen, ob ein Punkt in einem Dreieck enthalten ist. Hierzu ist er aber deutlich zu klein. Um das Punkt-In-Dreieck-Problem zu lösen, müssen mehrere Kreuz- oder Skalarprodukte berechnet werden. Mit jedem Rechenschritt vervielfacht sich der Rundungsfehler, sodass das Endergebnis mit einem deutlich größeren Fehler behaftet ist, als die Eingabedaten. Besonders für länglich geformte Dreiecke ist diese Berechnung numerisch schlecht konditioniert. Die Determinante ist fast gleich Null, und es kommt zu sogenannten *Auslöschungseffekten*. Dies führte auch zu einigen unerwarteten Resultaten der *libccd*.
Die Kondition kann verbessert werden, indem zum Beispiel auf eine Berechnung von Ergebnissen mit Determinanten und der Cramerschen Regel, wo es möglich ist, vermieden wird.
Zudem wurde der Toleranzwert testweise etwas erhöht, was in einer kleinen Verbesserung, aber auch in einigen nicht erkannten Kollisionen resultierte. Dies allein ist also auch keine Lösung. Einige Berechnungen der *libccd* sind numerisch nicht besonders stabil, und müssten überarbeitet werden.

**Vereinigung von Dreiecken im EPA**
Wie im Abschnitt 2.3 beschrieben, kann ein neuer Supportpunkt in das EPA-Polytop nicht immer durch Aufteilen des aktuellen Dreiecks in drei weitere Dreiecke, welche jeweils mit diesem verbunden sind, eingefügt werden. Dies führt in einigen Fällen zu einem nicht-konvexen Polytop (und damit zur Ungültigkeit vieler im Algorithmus verwendeter Annahmen). Bei diesem Verfahren bleiben zudem alle einmal hinzugekommen Kanten erhalten.
Während die Supportpunkte sicher auf der Oberfläche des CSO liegen, können Kanten auch quer hindurch verlaufen. Werden solche Kanten, die weit von der Oberfläche entfernt liegen, nicht ersetzt, behindern sie die Approximation stark.
Die *libccd* verwendet diese einfache, aber ungeeignete Methode anstelle der aufwändigeren Suche nach der Silhouette.

### 3.2.2 Fazit

Die *libccd* ist mit einigen mehr oder weniger gravierenden Fehlern behaftet. Besonders schwer wiegt die nicht implementierte Silhouettensuche, für die auch neue Datenstrukturen erforderlich wären. Ohne diese ist eine einwandfreie Funktion nicht möglich. Da die SOLID-Bibliothek

all diese Schwächen nicht aufweist, wurde entschieden, dass diese eine bessere Grundlage für die Verwendung im *pe Physics Engine* darstellt.

## 3.3 Optimierung der *SOLID*-basierten Version

Nun soll eine Implementierung auf Basis der *SOLID*-Bibliothek in den *pe Physics Engine* integriert und auf eventuelle Schwächen analysiert werden.
Begonnen wurde bereits in einer Diplomarbeit [18]. Es fehlt aber noch die Implementierung einiger Schnittstellen. Zudem müssen die Robustheit und Genauigkeit dieser Implementierung deutlich verbessert werden.

### 3.3.1 Fehlertoleranz und Robustheit

Die Implementierung wurde mit denselben Testfällen wie die *libccd* untersucht. Dabei machte sie einen deutlich stabileren Eindruck. Allerdings traten bei einigen Fällen noch gravierende Fehler auf.

**Nicht-Erkennung von Kollisionen im EPA**
In einigen Fällen, besonders in solchen mit kleiner Überlappung, wurde keine Kollision erkannt, obwohl sich die Objekte in Kontakt befanden. Dies konnte auf eine unpassende Ausnahmebehandlung zu Beginn des EPA zurückgeführt werden. Dieser sollte mit einem Tetraeder des GJK, welches den Ursprung enthält, starten. In manchen Fällen liegt der Ursprung aber fast exakt auf einer Dreieckfläche des Tetraeders. Dies muss nicht zwangsläufig etwas über das Aussehen des CSO aussagen, sondern kann einfach durch die Lage der beteiligten Supportpunkte begründet sein.
In der vorliegenden Implementierung führt dies zum Abbruch des EPA und zur Rückgabe, dass keine Kollision stattfindet. Dieser Abbruch ist nicht nötig, er führt sogar zu falschen Resultaten. Nach der Entfernung dieser Ausnahmebehandlung muss allerdings beachtet werden, dass der EPA später eine Fläche mit Abstand von fast null vom Ursprung zur Erweiterung des Polytopes erhalten kann. Diese neue Suchrichtung, welche im Regelfall diejenige des ursprungsnächsten Punktes in der Dreiecksfläche wäre, ist hier nicht definiert oder extrem schlecht konditioniert.
Eine gute Alternative stellt allerdings die Richtung der nach außen zeigenden Normale der Dreiecksfläche dar. Diese entspricht der Richtung des nächsten Punktes bei einem Abstand größer null, da dieser Punkt auch genau senkrecht zur Oberfläche zu finden ist. In der Implementierung wird nun diese Richtung verwendet, falls es erforderlich sein sollte.
Durch diese Maßnahmen konnte das Problem behoben werden.

**Alternative Suche des Starttetraeders des EPA**
Es kann durchaus vorkommen, dass der GJK als Endsimplex eine Linie oder ein Dreieck liefert. Für den EPA wird zum Start der Erweiterung aber mindestens ein Tetraeder benötigt.
So ist ein Verfahren nötig, um aus Stecken oder Dreiecken ein Tetraeder oder ein Polytop mit mehr als vier Eckpunkten (mehr als vier Punkte zu Beginn sind kein Problem) als Ausgangsbasis des EPA zu formen. In der Theorie ist dies nicht schwierig, da das Simplex, mit dem der GJK terminiert, immer den Ursprung enthält. Ist am Ende also beispielweise ein Dreieck vorhanden, ist klar, dass der Ursprung in dieser Fläche liegt. Es ist somit nur ein Polytop zu erzeugen, welches das GJK-Dreieck vollständig enthält. Dies ist zum Beispiel durch Hinzufügen zweier Supportpunkte $w_4$ und $w_5$, einmal in Richtung der Flächennormale $\vec{n}$ und einmal entgegen dieser möglich (Abb. 14). Da das Dreieck $w_1, w_2, w_3$ den Ursprung enthält, ist dieser auch Teil des erzeugten Hexahedrons.

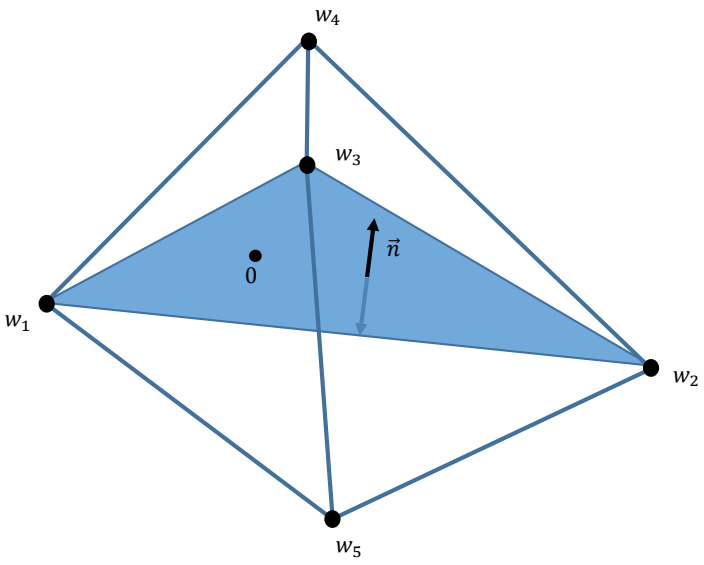



Abbildung 14: Konstruktion eines Hexahedrons aus einem Dreieck, welches den Ursprung enthält.

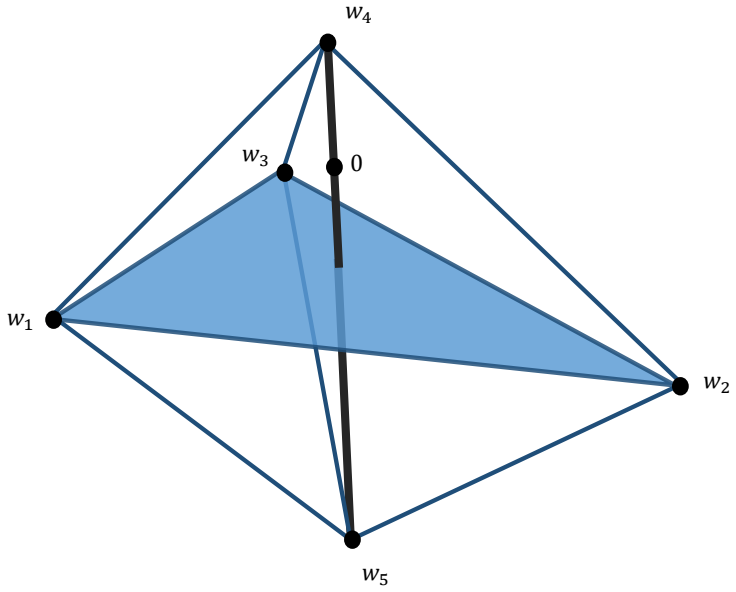



Abbildung 15: Konstruktion eines Hexahedrons aus einem Liniensegment, welches den Ursprung enthält.

Ein ähnliches Verfahren existiert für den Fall, dass das GJK-Simplex eine Strecke sein sollte. Hier werden ebenfalls drei Punkte um diese Linie herum hinzugefügt. Zur Strecke $w_4, w_5$ in Abb. 15 würden dann Punkte, in einer Lage ähnlich zu der von $w_1, w_2, w_3$ verwendet. So entsteht ebenfalls ein Hexahedron, das den Ursprung enthält.
In der Praxis liegt der Ursprung allerdings nicht ganz exakt auf dem Dreieck oder dem Liniensegment, da nicht mit unendlicher Genauigkeit gerechnet werden kann. Deshalb kann es auch vorkommen, dass das konstruierte Hexahedron, welches als Basis des EPA dient, den Ursprung nicht enthält. Besonders bei sehr kleinen Eindringtiefen konnte dieser Umstand und ein Versagen des Algorithmus beobachtet werden. Es muss unbedingt sichergestellt werden, dass der Urspung im Ausgangspolytop liegt.
Eine alternative Methode wurde ausgewählt und implementiert. Diese geht von einem beliebigen Tetraeder aus Supportpunkten aus, das nicht zwangsläufig den Ursprung enthalten muss. Hier kann zum Beispiel der obere oder untere Teil des Hexahedrons verwendet werden. Es wird versucht, daraus iterativ ein Tetraeder zu erzeugen, das den Nullpunkt enthält. Der Algorithmus ähnelt dem *Discover Portal*-Schritt des MPR. Algorithmus 2 beschreibt das Vorgehen in grobem Pseudo-Code. Die beste Implementierung hängt stark von den verwendeten Datenstrukturen ab.
Abbildung 16 stellt das Vorgehen in 2D dar. Das blaue Dreieck wird als Ausgangsbasis verwendet. Zu Beginn wird für jede Seite des Dreiecks geprüft, ob der Ursprung außerhalb (im Sinne von: In Richtung der nach außen zeigenden Normalen) liegt. Dies ist für die Strecke $(w_2, w_3)$ der Fall, weshalb ein neuer Supportpunkt $w_4 = s_{A-B}(v_1)$ in Richtung ihrer Normalen berechnet wird. Der gegenüberliegende Punkt $w_1$ wird durch den Punkt $w_4$ ersetzt. Dieses Vorgehen wird wiederholt, bis der Ursprung enthalten ist.
Nachteil dieses Verfahrens ist, dass eventuell einige Supportpunkte berechnet werden müssen. Trotzdem sorgt es für stabilere Ergebnisse, diese Methode der Berechnung zusätzlich zu den vorgestellten einfachen Methoden zu verwenden, wenn diese kein korrektes Tetraeder liefern.

**Algorithmus 2** Suche nach einem Tetraeder, das den Ursprung einschließt

```
1:  procedure searchTetrahedron
2:      while Ursprung nicht im Tetraeder enthalten do
3:          for all Flächen im Tetraeder do
4:              if Ursprung liegt auf der Außenseite der Fläche then
5:                  Suche neuen Supportpunkt in Richtung der äußeren Flächennormale
6:                  Ersetze gegenüberliegenden Punkt der Fläche durch den Supportpunkt
7:                  verlasse For-Schleife
8:              end if
9:          end for
10:     end while
11: end procedure
```

**Schlechte Kondition des EPA für kleine Eindringtiefen**
Sind die Eindringtiefen der Körper sehr klein, ist die numerische Kondition des EPA schlecht [26, S. 166]. Abhilfe schafft es, die Körper um einen gewissen Randwert (*margin*) zu vergößern. Es wird die Minkowski-Summe aus dem eigentlichen Objekt und einer Kugel, deren Radius dem Randwert entspricht, betrachtet.
Die Supportfunktion $s_{A+Kugel}(\vec{v})$ dieses, um den Randwert $m$ vergrößerten Körpers, ist einfach aus der Supportfunktion des Körpers $s_A(\vec{v})$ abzuleiten

$$s_{A+Kugel}(\vec{v}) = s_A(\vec{v}) + s_{Kugel}(\vec{v}) = s_A(\vec{v}) + m\vec{v}$$

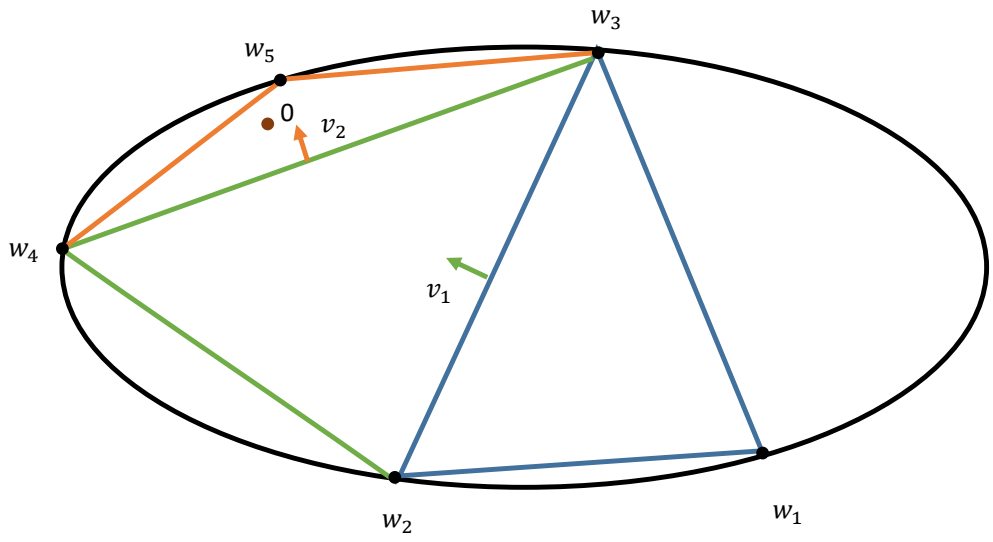



Abbildung 16: Zwei Schritte des Algorithmus um ein Dreieck zu finden, das den Ursprung enthält. Das Dreieck $w_1, w_2, w_3$ wird in Richtung $v_1$ durch einen Supportpunkt erweitert, dann nochmals in Richtung $v_2$.

Diese Berechnung ist für jeden Körpertyp gleich und erfordert nur eine einmalige Implementierung. Wie sonst auch, wird zuerst der GJK-Algorithmus mit den vergrößerten Körpern aufgerufen. Liefert dieser einen Abstand größer als null, so sind die Körper sicher separiert. Ist dies nicht der Fall, wird der EPA gestartet. Von der Eindringtiefe, die berechnet wurde, wird am Ende $2m$ subtrahiert, um die Eindringtiefe der ursprünglichen Körper zu erhalten. Ist diese groß genug (der *pe Physics Engine* erfordert einen Abstand kleiner als `pe::contactThreshold`) so wird ein Kontakt als Ergebnis generiert, andernfalls wird die Disjunktheit der Körper vermeldet.
Zur Bestimmung des Kontaktpunktes werden allerdings weiterhin die Punkte auf der Oberfläche der unvergrößerten Körper verwendet. Dies hat den Vorteil, dass Berührkontakte in einem Bereich berechnet werden können, der numerisch stabil ist. Unstabilitäten werden im Voraus vermieden. Auch diese Verbesserung wurde implementiert und getestet.

**Abbruchkriterium des EPA**
Wie bei der *libccd* wurde auch hier das Abbruchkriterium des EPA noch einmal untersucht. Die Implementierung enthielt bereits das relative Kriterium aus Abschnitt 3.2.1. Der relative Fehler war jedoch im Code auf einen festen Wert gesetzt. Er kann in der überarbeiteten Version vom Benutzer übergeben werden.
Ist dieser Wert zu klein gewählt, kommt es aufgrund von Rundungsfehlern zu keinem regulären Abbruch, wählt man ihn zu groß, verliert man jedoch Genauigkeit. Es wurde deshalb getestet, bis zu welchem relativen Fehler das Ergebnis sich verbessert. Eine genauere Untersuchung der Auswirkung dieses Parameters wird in Abschnitt 4.3 durchgeführt.

### 3.3.2 Interpolation von Flächen durch Kugelschalen

Da der EPA-Algorithmus versucht, die Oberfläche des CSO durch ein Polytop darzustellen, kommt es bei kontinuierlich gekrümmten Oberflächen zu einer schlechten Annäherung. Die beiden Grundformen Kugel und Capsule des *pe Physics Engine* enthalten allerdings solche Oberflächen. Es ist deshalb wichtig, die Approximation der Oberfläche für diese Formen zu verbessern.
Ein implementierter Ansatz ist die Interpolation von Oberflächen durch eine Kugelschale anstelle einer ebenen Fläche. Vier Punkte im Raum sind ausreichend, um eine Kugel zu bestimmen, wenn diese ein Tetraeder mit einem Volumen größer als 0 aufspannen. Durch die Kugelfläche soll die Oberfläche besser approximiert werden.

**Effiziente Berechnung**
Mathematisch kann eine Kugel mit Mittelpunkt $\vec{M} = (m_x, m_y, m_z)^T$ und Radius $r$ durch die Kugelgleichung beschrieben werden

$$(x - m_x)^2 + (y - m_y)^2 + (z - m_z)^2 = r^2$$

Setzt man die vier bekannten Punktkoordinaten in diese Gleichung ein und betrachtet $m_x, m_y, m_z$ und $r$ als Unbekannte, führt dies zu vier quadratischen Gleichungen. Diese wären mathematisch nur mit relativ hohem Aufwand lösbar. Sie lassen sich allerdings in ein lineares System umwandeln.
Die Berechnung des Mittelpunktes des Kreises kann auf den Schnitt dreier Ebenen im Raum zurückgeführt werden. Nennen wir die gegebenen Punkte $A, B, C$ und $D$. Vom Kreismittelpunkt sollen alle vier Punkte den gleichen Abstand haben. Betrachtet man nun zum Beispiel nur die Punkte $A$ und $B$, so liegen alle Punkte, welche zu beiden den gleichen Abstand haben auf einer Ebene. Diese Ebene hat die Flächennormale $\vec{A} - \vec{B}$ und geht durch den Mittelpunkt der Strecke $\overline{AB}$ gegeben durch $\frac{1}{2}(\vec{A} + \vec{B})$. Der Mittelpunkt $\vec{M}$ liegt auf der Ebene, und es gilt

$$\vec{M} \cdot (\vec{A} - \vec{B}) = \frac{1}{2}(\vec{A} + \vec{B}) \cdot (\vec{A} - \vec{B}) = \frac{1}{2}(\|\vec{A}\|^2 - \|\vec{B}\|^2)$$

Benutzt man zusätzlich die Punkte A zusammen mit $C$ und $D$ führt dies zu einem linearen Gleichungssytem der folgenden Form

$$\begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix} \cdot \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \|\vec{A}\|^2 - \|\vec{B}\|^2 \\ \|\vec{A}\|^2 - \|\vec{C}\|^2 \\ \|\vec{A}\|^2 - \|\vec{D}\|^2 \end{bmatrix} := \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$\vec{v_1} = \vec{A} - \vec{B}, \vec{v_2} = \vec{A} - \vec{C}, \vec{v_3} = \vec{A} - \vec{D}$$

Zuerst prüfen wir die Lösbarkeit des Systems. Diese ist gegeben falls

$$\vec{v_1} \cdot (\vec{v_2} \times \vec{v_3}) \neq 0$$

Um das System schnell zu lösen bietet es sich an, die in der *pe* optimierten Vektoroperationen Skalar- und Kreuzprodukt zu verwenden. Ausgehend von der Cramerschen Regel lautet die Lösung mit Vektorprodukten ausgedrückt

$$\vec{M} = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \frac{1}{\vec{v_1} \cdot (\vec{v_2} \times \vec{v_3})}(d_1 * (\vec{v_2} \times \vec{v_3}) + d_2 * (\vec{v_3} \times \vec{v_1}) + d_3 * (\vec{v_1} \times \vec{v_2}))$$

Zusammengezählt erfordert die Berechnung der Kugel weniger als 20 Vektor-Operationen und kein Wurzelziehen. Sie ist daher relativ kostengünstig durchzuführen.

**Einsatz im EPA-Algorithmus und Annahmekriterien**
Im EPA-Algorithmus wird die Kugelberechnung durchgeführt, nachdem ein Dreieck zur Verfeinerung ausgewählt und ein Supportpunkt in dessen Richtung berechnet wurde. Für die Rechnung werden die drei Eckpunkte des Dreiecks und der neu hinzugekommene Supportpunkt herangezogen.
Kann nun eine Kugel berechnet werden, muss geprüft werden, ob diese Approxmation die Oberfläche des CSO gut beschreibt und für das Endergebnis verwendet werden soll (Abb. 17). Die Penetrationstiefe $p$ kann einfach berechnet werden:

$$p = r - \|\vec{M}\|$$

Zuerst wird verglichen, ob die auf diese Weise gefundene Tiefe im Bereich zwischen der unteren und der oberen Schranke, die durch den EPA gegeben sind, liegt. Ist dies nicht der Fall, wird normal weitergearbeitet.
Obwohl der Abstand zur berechneten Kugel innerhalb der Schranken liegt, kann es trotzdem sein, dass Oberfläche nicht gut durch eine Kugel angenähert werden kann. Ein weiterer Test ist erforderlich. Es wird der ursprungsnächste Punkt $\vec{P}$ auf der Kugelschale bestimmt, der auf einer Gerade durch den Kugelmittelpunkt $\vec{M}$ und den Ursprung zu finden ist. Er ist durch

$$\vec{P} = -p\frac{\vec{M}}{\|\vec{M}\|}$$

gegeben. Handelt es sich bei der Oberfläche tatsächlich um ein Kugelsegment, kann man einen Supportpunkt in Richtung dieses Punktes $\vec{P}$ berechnen und erhält erneut $\vec{P}$ als Resultat. Gilt also

$$\vec{s}_{A-B}(\vec{P}) = \vec{P}$$

werden die Werte, die mit Hilfe der Kugelschale bestimmt wurden, als Ergebnis zurückgeliert. Dies verbessert die Darstellung des CSO nicht nur bei Kollision zweier Kugeln, sondern auch für bestimmte Kontakte von Kugeln oder Körpern mit Kugelteilen mit anderen Körpern, da das CSO in all diesen Fällen kugelförmige Flächen aufweißt.

## 3.4 Implementierung von Ebenen und Vereinigungen

**Ebenen**
Einen ersten Spezialfall stellt die Geometrie der Ebene dar. Durch sie wird ein Halbraum abgetrennt. Da eine Ebene keine endliche Supportfunktion besitzt, sind die iterativen Algorithmen nicht anwendbar und Kollisionen zwischen Körpern und Ebenen müssen gesondert behandelt werden. Abbildung 18 gibt einen Überblick über die zur Berechnung verwendeten Punkte und Abmessungen.
Durch einen Punkt $\vec{P}$ und einen Normalenvektor $\vec{n}$ kann eine Ebene $E$ folgendermaßen definiert werden:

$$E := \{\vec{x} \in \mathbb{R}^3 : \vec{x} \cdot \vec{n} = \vec{P} \cdot \vec{n} = \delta\}$$

$\delta$ bezeichnet die Verschiebung und entspricht dem Abstand der Ebene vom Ursprung, wenn der Vektor $\vec{n}$ die Länge 1 hat, wovon wir im Folgenden ausgehen werden.
Eine Kollision zwischen einem Körper $A$ mit Supportfunktion $\vec{s_A}$ und der Ebene $E$ ist nun genau dann vorhanden, wenn

$$\vec{s_A}(-\vec{n}) \cdot \vec{n} \leq \delta$$

Der Supportpunkt $\vec{s_A}(-\vec{n})$ ist der Punkt, welcher der Ebene am nächsten kommt, bzw. bei Durchdringung am weitesten hinter ihr liegt. Über ihn ist es auch möglich, die Penetrationstiefe $p$ zu bestimmen. Sie berechnet sich durch

$$p = \vec{s_A}(-\vec{n}) \cdot \vec{n} - \delta$$

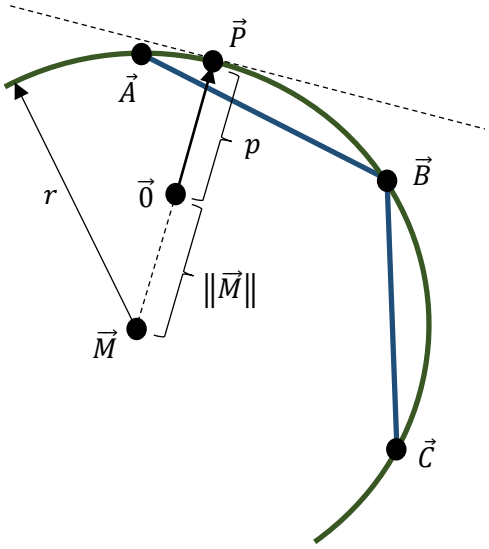



Abbildung 17: Kugelinterpolation und Annahme in 2D: Alternativ zu einer geraden Verbindung (blau) kann die Form zwischen den Punkten $\vec{A}$, $\vec{B}$ und $\vec{C}$ durch ein Kreissegment mit Radius $r$ (grün) angenähert werden. Zur Überprüfung der Kreisform des CSO wird ein Supportpunkt in Richtung des ursprungsnächsten Punktes $\vec{P}$ auf dem Segment berechnet. Handelt es sich beim betrachteten Teil des CSO tatsächlich um ein Kreissegment, so ist der erhaltene Supportpunkt genau wieder der Punkt $\vec{P}$ und die Kreisapproxmation wird verwendet.

Als Stoßnormalenvektor wird immer der Vektor $-\vec{n}$ bzw. $\vec{n}$ (abhängig davon, ob die Ebene erster oder zweiter Körper ist) verwendet. Zuletzt muss noch ein Kontaktpunkt $\vec{C}$ generiert werden. Im Idealfall berühren sich die beiden Körper nur und der Kontaktpunkt ist gleich dem Supportpunkt $s_A(-\vec{n})$. Wenn dies nicht zutrifft, stellt der Mittelpunkt der kürzesten Strecke von $\vec{s_A}(-\vec{n})$ an die Ebene (dünne Linie in Abb. 18) eine gute Wahl dar und kann folgendermaßen bestimmt werden:

$$\vec{C} = \vec{s_A}(-\vec{n}) + \frac{1}{2}p\vec{n}$$

Alternativ kann der entsprechende Punkt in der Ebene

$$\vec{C} = \vec{s_A}(-\vec{n}) + p\vec{n}$$

verwendet werden. Dies ist sinnvoll, wenn ohne Penetration gerechnet werden soll. Kollisionen von zwei Ebenen werden im *pe Physics Engine* nicht behandelt.

**Vereinigungen von Körpern**

Eine weitere Anforderung an die iterative Kollisionserkennung ist die Behandlung von zusammengesetzten Objekten, die aus mehreren der Grundkörpern bestehen und in einer festen Konfiguration miteinander verbunden sind.
Das hier angewandte Verfahren prüft für jede Kombination aus Teilkörpern der Vereinigungen, ob es zu einer Kollision kommt. Wie in Algorithmus 3 dargestellt, kann mit Hilfe der Funktion `hasSubbodies` unterschieden werden, ob es sich bei einem Testgegenstand um eine Vereinigung handelt.

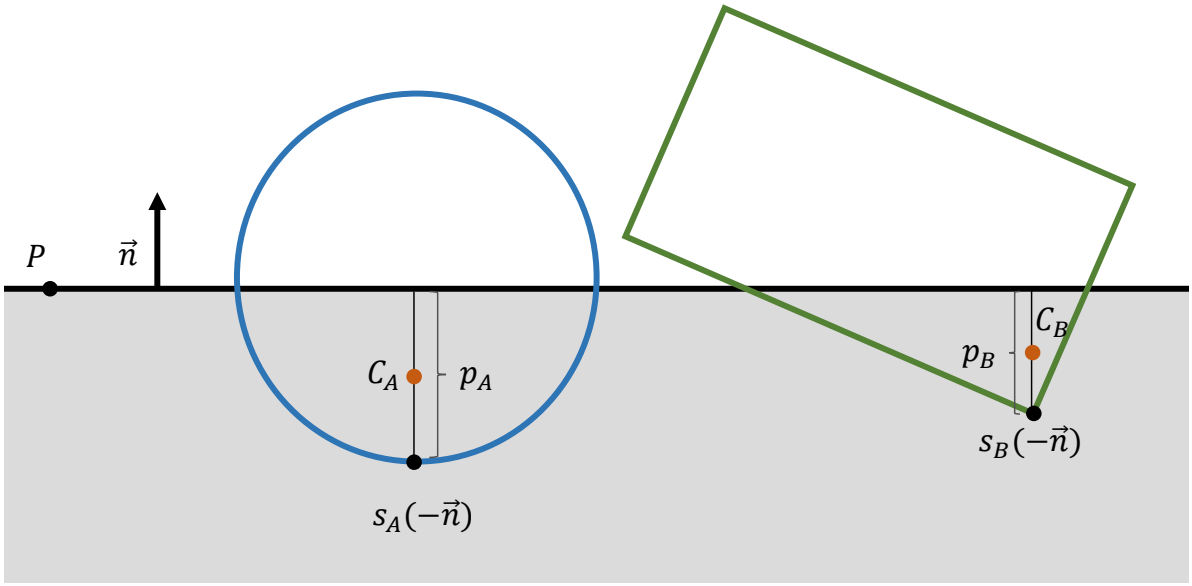



Abbildung 18: Kontakte von Körpern mit einer Ebene

**Algorithmus 3** Detektion einer Kollision von Vereinigungen

**Require:** Körper A, B

```
1:  procedure doCollide(A, B)      ▷ Prüft, ob die zusammengesetzten Körper kollidieren
2:      if A.hasSubbodies( ) then
3:          for all subBodyA in A.getSubbodies( ) do
4:              if doCollide(subBodyA, B) then                    ▷ Rekursiver Aufruf
5:                  return true
6:              end if
7:          end for
8:      else
9:          if B.hasSubbodies( ) then
10:             for all subBodyB in B.getSubbodies( ) do
11:                 if doCollide(A, subBodyB) then                ▷ Rekursiver Aufruf
12:                     return true
13:                 end if
14:             end for
15:         else
16:             return iterativeDoCollide(subBodyA, subBodyB)  ▷ Eigentliche Rechnung
17:         end if
18:     end if
19:     return false
20: end procedure
```

# 4 Messergebnisse und Tests

Zum Beleg der Funktionalität des implementierten Codes wurden verschiedene Formen von Tests durchgeführt. Die *lokalen Vergleiche* betrachten das Ergebnis jeder Berechnung einzeln. Sie sorgen dafür, dass die Daten für einen bestimmten Fall übereinstimmen.
Im Gegensatz dazu betrachten die *globalen Vergleiche* das Gesamtergebnis einer Simulation, das aus dem Zusammenspiel vieler Kontaktberechnungen entsteht. Sie stellen sicher, dass die mit der iterativen Kollisionserkennung gewonnen Simulationsendergebnisse ebenfalls überzeugend sind.

## 4.1 Lokale Vergleiche

### 4.1.1 Black-Box-Test

Einige analytisch gut lösbare Fälle von Kollisionen, bei denen alle möglichen Körpertypen Kugel, Würfel, Capsule und Ellipsoid vorkommen, wurden explizit programmiert. Diese Fälle können automatisiert geprüft werden.
Beim Test werden die benötigten Körper erzeugt und so positioniert, dass sie sich genau berühren. Dann wird ein Körper mit verschiedenen Penetrationstiefen in den anderen hinein geschoben. Für jede Tiefe wird die Kollisionserkennung doppelt aufgerufen. Beim zweiten Aufruf wird die Reihenfolge der Körper in der Argumentenliste vertauscht. Dies ist notwendig, da einige Fehler beim anfänglichen Test der *libccd* nur bei einer von beiden Argumentreihenfolgen auftraten. Insgesamt wird die iterative Kollisionserkennung auf diese Weise über 100 Mal aufgerufen und die gelieferten Kollisionsdaten mit einer analytisch ermittelten Lösung verglichen. Dies stellt die Basisfunktionalität der Implementierung sicher.
Hierbei kommt ein Fehlertoleranzwert zum Einsatz. Beim Winkel der Normalen erreicht die Implementierung auch bei großen Eindringtiefen eine maximale Abweichung von unter $0.1°$. Diese Schranke sollte aber als absolutes Maximum gesehen werden und wird nur in einem Testfall mit Ellipsoiden annähernd erreicht. Für alle anderen Testfälle ist eine Toleranz von $0.01°$ auch ausreichend. Der große Unterschied zwischen diesen beiden Werten ist wohl mit der kontinuierlich gekrümmten Oberfläche des Ellipsoids zu begründen. Diese ist nur schlecht durch ein Dreiecksnetz anzunähern.
Abbildung 19 bis Abbildung 21 sollen einen Überblick über die implementierten Testfälle geben. Zudem findet sich unter ihnen eine kurze Beschreibung, welches Szenario durch den entsprechenden Fall abgedeckt wird. Der orange Pfeil gibt die Richtung an, in der der zweite Körper in den ersten geschoben wird. Die verwendeten Tiefen reichen von einem Abstand von $10^{-5}$ bis zu einer Eindringtiefe von 1. Die gestrichelte Linie soll verdeutlichen, wenn die Bewegungsrichtung gleich der Verbindunglinie der Mittelpunkte ist.

### 4.1.2 Brute-Force-Test

Mit dem *pe Physics Engine* können mehrere Milliarden nicht-kugelförmige Partikel simuliert werden [17]. Hierbei wird eine gewaltige Menge an Kollisionsberechnungen durchgeführt. Die Idee hinter dem Brute-Force-Test ist, jede dieser Rechnungen als Testfall zu verwenden. Eine analytische Lösung für einfache Typen ist im *pe Physics Engine* implementiert und kann als Test- und Vergleichswert benutzt werden. Dabei soll herausgefunden werden, ob und wie gut der Algorithmus und die Implementierung in der Praxis funktionsfähig sind. Zudem kann eine obere Schranke für die maximale Abweichung von den analytischen Daten angenähert werden.

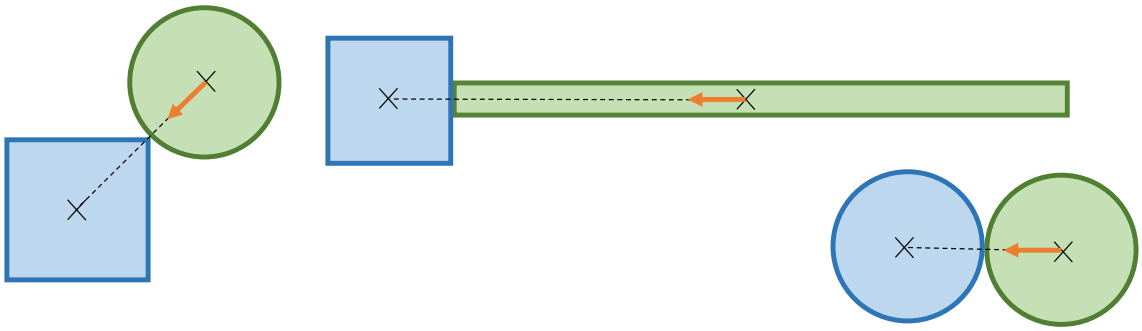


Abbildung 19: Testfälle 1 - 3 (v. l. n. r.)

**Test 1:** Eine Kugel trifft auf einen Würfel. Die Position und Verschiebungsrichtung sind so gewählt, dass (hier in der Draufsicht nicht zu sehen) die Ecke des Würfels als erstes auf die Kugelfläche trifft. Dieser Test prüft die Kollision von einer Ecke und einer runden Oberfläche.
**Test 2:** Der Würfel wird nun von einem länglichen Quader an der Seite getroffen. Bestätigt die Erkennung einer Kollision zweier ebener Flächen und die Funktionalität mit sehr unterschiedlich dimensionierten Objekten.
**Test 3:** Kollision zweier Kugeln. Prüft den Kontakt zweier runder Flächen.

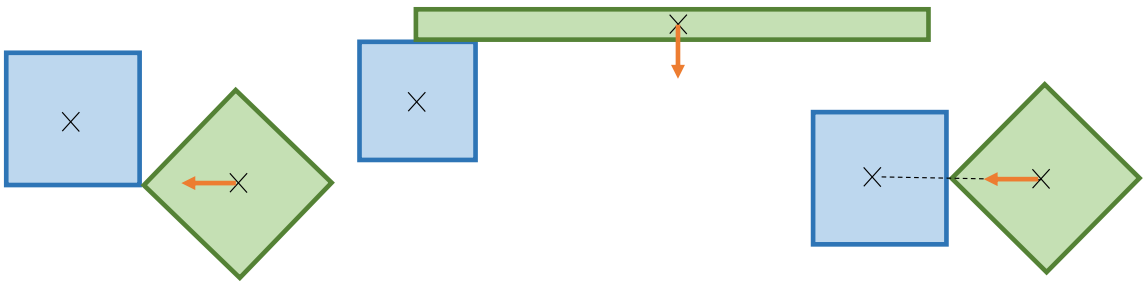


Abbildung 20: Testfälle 4 - 6 (v. l. n. r.)

**Test 4:** Ein Würfel trifft auf einen anderen Würfel. Der Kontakt entsteht zuerst entlang einer Kante. Die Eindringtiefe bei diesem Test ist zudem nicht gleich der Verschiebung des grünen Körpers, sondern um den Faktor $\frac{1}{\sqrt{2}}$ reduziert. Testet die Kontaktbildung bei einem Aufeinandertreffen zweier Kanten und die richtige Bestimmung der Eindringtiefe.
**Test 5:** Nicht zentrische Kollision eines Würfelw und eines länglichen Quaders. Da die Stoßnormale in fast allen Fällen gleich der Verbindung der Schwerpunkte ist, soll dieser Test prüfen, ob die Kontaktdaten auch korrekt sind, wenn dies nicht der Fall ist. Zusätzlich interessant ist der Kontaktpunkt, der nicht in der Mitte der Oberflächen liegt.
**Test 6:** Kollision zweier Würfel. Hier trifft der Würfel nur mit einer Ecke auf die Seitenfläche eines anderen Würfels. Testet die Kontaktgenerierung bei Kollision einer geraden Fläche mit einer Ecke. Das Aufeinandertreffen von Ecke und Fläche ist bei Kollisionen zweier Quader in freier Bewegung der wahrscheinlichste Fall und wird hier noch einmal gesondert getestet.

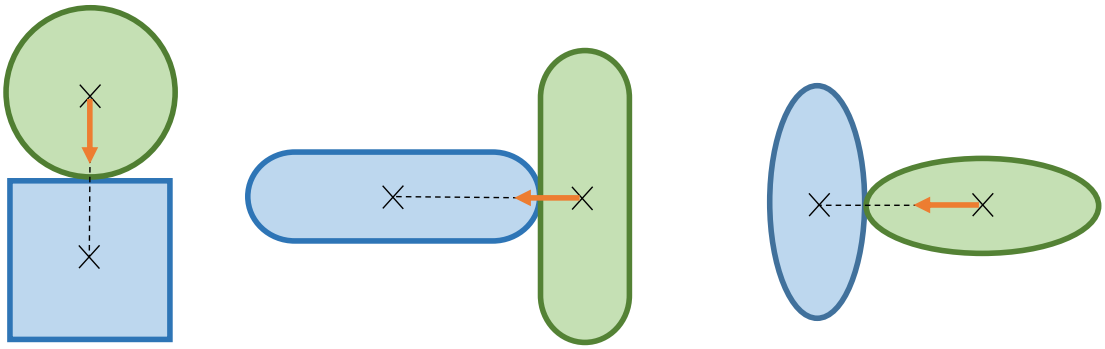


Abbildung 21: Testfälle 7 - 9 (v. l. n. r.)

**Test 7:** Eine Kugel trifft auf die Seitenfläche eines Würfels. Test des Kontaktes einer Kugelfläche mit einer ebenen Fläche.
**Test 8:** Test zweier Capsules. Da der obere und untere Teil identisch mit Kugelflächen sind, die bereits getestet wurden, werden die Capsules am Zylinder-Teil zur Kollision gebracht.
**Test 9:** Kollision zweier länglicher Ellipsoide. Test des Ellipsoid-Types. Zum Test wurde ein Fall, bei dem ein kleiner und ein größerer Krümmungsradius aufeinandertreffen, ausgewählt.

**Vorgehen**

Um eine möglichst realistische Testumgebung zu erhalten, wird eine Simulation mit verschiedenen Körpern, die gleichmäßig in einem würfelartigen, durch 6 Ebenen begrenzten Volumen enthalten sind, erzeugt. Die Körper erhalten eine zufällig gewählte Anfangsgeschwindigkeit.
Der Ablauf des Tests gleicht dem einer realen Simulation. Die Körper bewegen sich durch den Raum und kollidieren. Zur Auflösung von Kollisionen wird die bereits vorhandene analytische Kollisionserkennung verwendet. Zusätzlich zur Analytischen werden allerdings alle möglichen Kontaktpaare auch mit der iterativen Kollisionserkennung berechnet. Deren Ergebnisse haben zwar keinen Einfluss auf das Fortschreiten der Simulation (an deren physikalischen Ergebnis man ja nicht interessiert ist), werden aber mit denen der analytischen Kollisionserkennung verglichen.
Sollten Kollisionen nur von einer der beiden Erkennungen bemerkt werden oder die Abweichung der Kollisionsdaten über einem gewissen Grenzwert liegen, wird dies als *Incident* mit allen Daten der beteiligten Körper zur Nachverfolgung und zur Fehlersuche gespeichert. Nach Beendigung der Simulation werden die Incidents nach ihrer Relevanz sortiert und können genau analysiert werden.
Es wurden Versuche für alle Kombinationen der drei im *pe Physics Engine* implementierten Grundkörper Kugel, Quader (hier als Würfel) und Capsule durchgeführt. Bei Tests, welche Würfel und Capsules enthielten, fiel allerdings ein Fehler in der analytischen Berechnung auf, sodass hier kein Vergleich möglich war. Dies verdeutlicht aber die Effizienz dieser Testmethode.
Einen Überblick über die durchgeführten Tests gibt Tabelle 1. Es wurden jeweils 150 Körper erzeugt und ein elastischer Löser verwendet. Die Simulation lief über 100.000 Zeitschritte mit einem Zeitschritt von $\Delta t = 2 \times 10^{-4}$, sodass gesamt 20 reelle Sekunden berechnet wurden.
Die Körper haben jeweils etwa die selbe Abmessung (so entspricht beispielsweise die Kantenlänge der Würfel dem Durchmesser der Kugeln). Bei der Kombination verschiedener Körper wird immer abwechselnd ein Körper des ersten Typs neben einem des zweiten Typs platziert. Insgesamt wurden auf diese Art über 6 Mio. Kollisionen automatisiert "nachgerechnet". Die beiden folgenden Kapitel geben einen Überblick über die Ergebnisse.

| Verwendete Objekte | Anzahl Berechnungen | Erkannte Kollisionen |
|---|---|---|
| Kugeln | 834.565 | 109.929 |
| Kugeln u. Würfel | 601.704 | 68.158 |
| Würfel | 403.287 | 47.591 |
| Kugeln u. Capsules | 1.563.083 | 146.817 |
| Capsules | 2.859.619 | 215.498 |
| **Gesamt** | 6.262.258 | 587.993 |

Tabelle 1: Durchgeführte Brute-Force-Tests. In "Anzahl Berechnungen" wurden alle von der CCD gelieferten möglichen Kontaktpaare gezählt. Die Anzahl detektierter Kollisionen findet sich in der Spalte "Erkannte Kollisionen", hier gab es keine Abweichung zwischen analytischer und iterativer Erkennung.

### 4.1.3 Erkennung von Kollisionen

Die reine Erkennung von Kollisionen funktioniert sehr gut.
Die Black-Box-Testfälle aus Abschnitt 4.1.1 enthalten zusätzlich eine Überprüfung, ob die Kollision überhaupt detektiert wurde. Alle Tests prüfen, ob bei einem Abstand von $10^{-5}$ noch keine Kollision erkannt wird. Anschließend wird bei allen Eindringtiefen, die ebenfalls bei $10^{-5}$ beginnen, getestet ob der Kontakt erkannt wird. Wie eine Ausführung der Tests bestätigt, gibt es hier keine Probleme.
Auch beim Brute-Force-Test kam es nicht zu fälschlicherweise detektierten oder nicht erkannten Kontakten.

### 4.1.4 Genauigkeit der Kollisionsdaten

In diesem Abschnitt soll versucht werden, die Güte der gelieferten Kollisionsdaten zu quantifizieren. So wird ermittelt, welche Genauigkeit von der iterativen Kollisionserkennung zu erwarten ist.
Einen Überblick über die maximalen bzw. mittleren Abweichungen der im Brute-Force-Test gesammelten gegenüber den analytischen Daten liefern Tabelle 2 und Tabelle 3.

| Verwendete Objekte | Max. Abweichung des Winkels (Grad) | Max. Abweichung der Eindringtiefe | Max. Abweichung der Position (in % der Körpergröße) |
|---|---|---|---|
| Kugeln | $1,39 \times 10^{-10}$ | $2,14 \times 10^{-14}$ | $4,10 \times 10^{-13}$ % |
| Kugeln u. Würfel | 0,42 | $9,65 \times 10^{-6}$ | 0,97 % |
| Würfel | $4,01 \times 10^{-4}$ | $2,76 \times 10^{-6}$ | 5,16 % |
| Kugeln u. Capsules | 0,67 | $3,50 \times 10^{-5}$ | 0,05 % |
| Capsules | 0,95 | $4,10 \times 10^{-5}$ | 1,01 % |
| **Gesamtmaximum** | 0,95 | $4,10 \times 10^{-5}$ | 5,16 % |

Tabelle 2: Maximale Abweichungen im Brute-Force-Tests

**Winkel der Kontaktnormalen**
Ein großes Problem der ersten Implementierung im *pe Physics Engine* und der *LibCCD* war die genaue Bestimmung einer Kontaktnormale, welche für die Kollisionsauflösung von erheblicher Bedeutung ist. Sie bestimmt den Winkel, unter dem die Stoßkräfte wirken. Auf diesem Wert lag deshalb ein besonderes Augenmerk.

| Verwendete Objekte | Mittlere Abweichung des Winkels (Grad) | Mittlere Abweichung der Eindringtiefe | Mittlere Abweichung der Position (in % der Körpergröße) |
|---|---|---|---|
| Kugeln | $3,45 \times 10^{-12}$ | $4,15 \times 10^{-15}$ | $8,77 \times 10^{-14}$ % |
| Kugeln u. Würfel | $6,90 \times 10^{-3}$ | $3,08 \times 10^{-7}$ | 0,093% |
| Würfel | $1,21 \times 10^{-5}$ | $1,96 \times 10^{-7}$ | 0,070% |
| Kugeln u. Capsules | 0.0117 | $9,66 \times 10^{-7}$ | $4,62 \times 10^{-4}$ % |
| Capsules | 0.0151 | $1,20 \times 10^{-6}$ | $1,85 \times 10^{-3}$ % |
| **Gesamtmittel** | $9,26 \times 10^{-3}$ | $7,34 \times 10^{-7}$ | 0,017% |

Tabelle 3: Mittlere Abweichungen im Brute-Force-Tests

| Quantil | Abweichung des Winkels (Grad) | Abweichung der Eindringtiefe | Abweichung der Position (in % der Körpergröße) |
|---|---|---|---|
| 99% | 0,11 | $0,82 \times 10^{-5}$ | 0,30% |
| 99,9% | 0,25 | $1,33 \times 10^{-5}$ | 0,58% |
| 99,99% | 0,45 | $2,08 \times 10^{-5}$ | 0,96% |

Tabelle 4: Quantile der Abweichungsverteilungen
Lesebeispiel: Die Abweichung des Winkels liegt in 99% der Fälle unter $0,11°$.

Wie die Messergebnisse zeigen, konnte diese Abweichung sehr klein gehalten werden. Der höchste im Test gemessene Wert lag bei $0,95°$. Im Mittel lag der Wert allerdings mit ca. $0,01°$ deutlich darunter.
Wie bei allen Werten fallen die großen Unterschiede je nach verwendetem Körpertyp auf. Bei den Kugeln ist aufgrund der quasi-analytischen Berechnung fast keine Abweichung erkennbar. Groß wird diese vor allem bei gerundeten Objekten wie Capsules oder auch den Ellipsoiden. Tests mit analytisch lösbaren Kollisionen von Ellipsen mit hohen Eindringtiefen zeigen, dass hier mitunter auch Abweichungen von bis zu $2°$ möglich sind.
Ein Blick in die Quantiltabelle der Verteilung (Tabelle 4) zeigt, dass 99,99% der gemessenen Kollisionen eine Abweichung von unter $0,45°$ haben. Anders gesagt: Nur in einem von 10.000 betrachteten Fällen kommt es zu einer größeren Differenz.
Eine ganz exakte Bestimmung ist allerdings oft nicht möglich, da – anders als bei der Eindringtiefe – der Fehler im Normalenwinkel nicht im Algorithmus bestimmbar ist. Ist die Eindringtiefe bereits sehr nah am korrekten Wert, so wird davon ausgegangen, dass die Normale ebenfalls gut angenähert ist. Dies trifft jedoch nicht immer zu.

**Eindringtiefe**
Zu erwarten war, dass die Eindringtiefe sehr exakt bestimmt wird – schließlich bricht die Iteration des EPA regulär dann erst ab, wenn ein bestimmter relativer Fehler unterschritten wird.
Die Messergebnisse bestätigen dies. Da der relative Fehler im Algorithmus beschränkt wird, ist bei kleinen Eindringtiefen zudem ein verringerter Fehler zu erwarten. Die Bestimmung der Eindringtiefe ist insgesamt sehr zuverlässig.

**Position des Kontaktpunktes**
Die größte Schwierigkeit stellt die Bestimmung der Position des Kontaktpunktes dar. Diese ist nämlich aus dem CSO der beiden Körper, in dem alle Berechnungen stattfinden, nicht ersichtlich. Verwendet wird die in Abschnitt 3.2.1 beschriebene Methode der Bestimmung, die

trotzdem relativ genaue Ergebnisse liefert.
Bei Körpern mit unstetigen Supportfunktionen (z.B. alle Körper mit geraden Flächen) oder besonders bei einer begrenzten Anzahl von möglichen Supportpunkten (z.B. bei einem konvexen Polyeder) kann die Interpolation allerdings ungenau werden.
Zudem ist der Kontaktpunkt oft nicht eindeutig, wie bei einem Kontakt von Flächen oder Kanten.
Die analytische Kollisionserkennung des *pe Physics Engine* hat die Fähigkeit, in diesem Fall mehrere Kontaktpunkte zurück zu liefern. Dies ist der iterativen Kollisionserkennung nicht so einfach möglich, da im CSO nicht ersichtlich ist, ob ein Kontakt eindeutig ist.
Bei den gemessenen Werten wurden nur Kollisionen mit einem eindeutigen Kontaktpunkt berücksichtigt. Wenn die analytische Erkennung mehrere Punkte lieferte, wurde der iterative Wert nicht in die Statistik aufgenommen. Einige Testfälle des Black-Box-Tests zeigen aber, dass der Kontaktpunkt in uneindeutigen Zuständen trotzdem innerhalb der erwarteten Fläche liegt, nur der genaue Ort nicht vorhersagbar ist.
Für eindeutige Kontaktpunkte liefert die Erkennung gute Werte (Abweichung unter 1% der Körpergröße), nur bei den Würfeln ist ein Ausreißer mit 5% Abweichung zu sehen. Wie oben bereits beschrieben, funktioniert die Bestimmung umso besser, je kontinuierlicher der Körper ist. Experimente mit Capsules und Kugeln liefern hier die besten (Mittel-)Werte.

**Zusammenfassung**
Im Brute-Force- und im Black-Box-Test konnten für Kugeln, Würfeln und Capsules mit hoher statistischer Sicherheit (Überschreitung in weniger als 1 von 10000 Fällen) in etwa folgende Genauigkeiten für die Kollisionsdaten ermittelt werden:

- Winkel der Kontaktnormalen: $0,5°$
- Position der Kontaktpunkte: 0,06 (entspricht ca. 1% der Größe der simulierten Körper)
- Abweichung der Eindringtiefe: $2,5 \times 10^{-5}$

## 4.2 Globale Vergleiche

Von übergeordneter Bedeutung sind die Daten, welche als Endergebnis einer Simulation gewonnen werden. Diese sind oft über mehrere Simulationsläufe, Körper oder Zeitschritte gemittelt. Wichtig ist hier, dass ein im Mittel korrektes Ergebnis errechnet wird oder dass Fehler sich im Mittel aufheben. Um zu überprüfen, ob die iterative Kollisionserkennung gemittelte Daten nicht verfälscht, wurden eine Schüttung und eine Simulation von Gasteilchen durchgeführt. In einem Referenzversuch wurde die analytische Kollisionserkennung verwendet. Anschließend wurde der gleiche Versuch mit der iterativen Kollisionserkennung wiederholt und die erhaltenen Daten mit ihren Referenzwerten verglichen.

### 4.2.1 Schüttungsversuche

**Versuchsumgebung**
Eines der Haupteinsatzgebiete des *pe Physics Engine* ist die Simulation von granularen Partikeln. So kann zum Beispiel der Inhalt eines Silos und der Austritt aus einer Öffung dieses Behälters simuliert werden.
Im durchgeführten Versuch wird eine Menge von 3000 Körpern zufällig aus unterschiedlichen Höhen in eine Box mit quadratischer Grundfläche fallen gelassen. Die Simulation wird beendet, sobald ein stabiler Endzustand erreicht wurde. Anschließend wird der Anteil des von den Körpern ausgefüllten Volumens, der *Feststoffvolumenanteil*, als Kenngröße berechnet. Der

Versuch wurde jeweils einmal nur mit Kugeln, mit einer 1:1-Mischung aus Kugeln und Würfeln und nur mit Würfeln durchgeführt. Zum Test der Ellipsoid-Implementierung wurden statt der Kugeln auch kugelförmige Ellipsoide verwendet, welche das selbe Resultat liefern sollten. Eine analytische Lösung für Ellipsoid-Kollisionen ist im *pe Physics Engine* nicht vorhanden. Deshalb gibt es hier nur ein iteratives Ergebnis.

| Geschüttete Körper | Wert iterative Kollisionsrechnung | Wert analytische Kollisionsrechnung | Referenzwert |
|---|---|---|---|
| Kugeln | 59,08 % | 59,28 % | 55 % - 71 % [27] |
| Kugelförmige Ellipsoide | 59,25 % | - | 55 % - 71 % [27] |
| Kugeln u. Würfel | 62,83 % | 62,23 % | - |
| Würfel | 78,96 % | 71,72 % | - |

Tabelle 5: Feststoffvolumenanteil der Schüttungen (ohne analytische Werte für Ellipsoide, da die Berechnung nicht implementiert ist)

**Ergebnisse**

Tabelle 5 zeigt die berechneten Feststoffvolumenanteile. Diese sind bei beiden Kollisionserkennungen ähnlich. Im Falle der Kugeln liegen sie auch in einem in Veröffentlichungen angegebenen Bereich. Die kugelförmigen Ellipsoide liefern wie erwartet in etwa den selben Wert wie die Kugeln. Ein größerer Unterschied ist bei der Schüttung der Würfel (diese befanden sich zu Beginn in einem zufälligen Rotationszustand) zu beobachten.

Um diese Unterschiede zu erklären und eventuell weitere zu identifizieren, lag auf den Eindringtiefen im erhaltenen Endzustand ein besonderes Augenmerk. Dieser Zustand, in dem jeder Körper mit mehreren anderen in Kontakt ist, stellt einen weiteren realistischen Test für die Kollisionserkennung dar.

Alle vorkommenden Kollisionen dieses Zustandes wurden mit jeder Erkennung erneut berechnet. Das heißt, eine Schüttung wurde beispielsweise einmal mit der analytischen Erkennung simuliert, der erhaltene Endzustand dann jeweils noch einmal mit der Analytischen und der Iterativen analysiert. Anschließend wurde die Simulation iterativ berechnet und der Endzustand wieder mit beiden Erkennungen betrachtet.

Das Histogramm in Abb. 22 zeigt die erkannten Kollisionstiefen im Endzustand (die Simulation wurde mit dem analytischen Löser durchgeführt) mit beiden Erkennungen am Beispiel der Kugelschüttung. Die Gesamtanzahl der erkannten Kollisionen ist hier exakt gleich. Im *pe Physics Engine* werden Kontakte schon ab einem Abstand von $1 \times 10^{-8}$ generiert, weshalb es auch Kontakte mit einem Abstand größer als null gibt (ein Abstand $< 0$ steht für eine entsprechende Penetrationstiefe). Die im Histogramm zu erkennenden Unterschiede (die analytische Erkennung erkennt etwas mehr Kontakte mit einem Abstand über null, die iterative unter null) sind bei einer genaueren Betrachtung auf maschinenbedingte Rundungsfehler zurückzuführen – der größte Unterschied der berechneten Tiefen betrug $1,4 \times 10^{-15}$.

Diese marginale Differenz kann aber ein gegensätzliches Vorzeichen verursachen und dafür sorgen, dass der Fall im anderen Intervall des Histogramms auftaucht. Besonders wenn es wie hier viele Eindringtiefen gibt, die – mit Maschinenpräzion betrachtet – fast null sind, kommt es häufig zu solchen Schwankungen.

Tabelle 6 zeigt die durchschnittliche Eindringtiefe für alle Schüttungsversuche im Endzustand. Die verwendete Kollisionserkennung zur Berechnung macht für den Durchschnitt keinen Unterschied. Die ermittelten Werte sind für die Kugeln sehr ähnlich. Sind Würfel beteiligt kommt es allerdings zu größeren Differenzen.

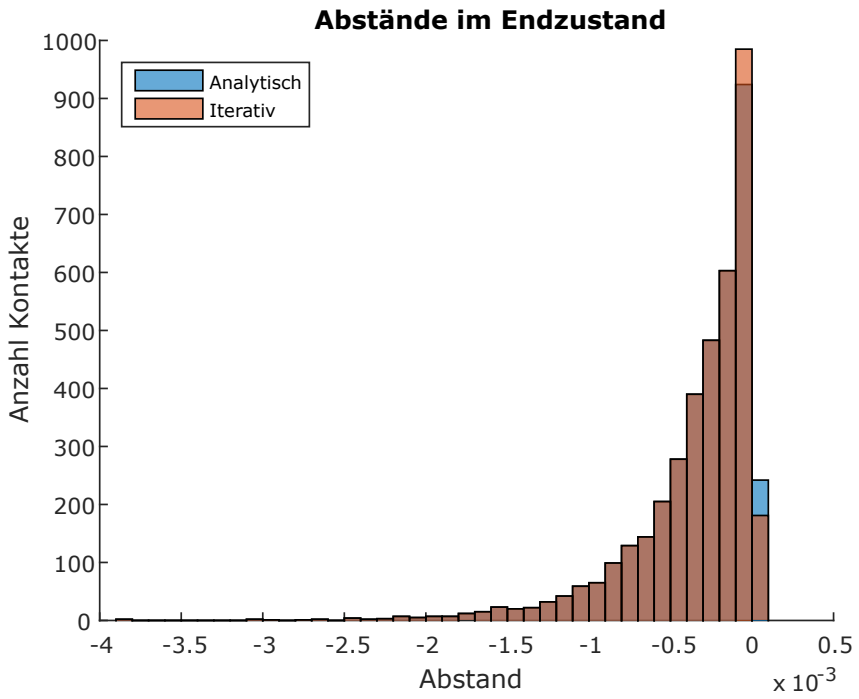



Abbildung 22: Histogramm der erkannten Eindringtiefen im Endzustand der Kugelschüttung

| Geschüttete Körper | Wert iterative Kollisionsrechnung | Wert analytische Kollisionsrechnung |
|---|---|---|
| Kugeln | 0,36 | 0,35 |
| Kugelförmige Ellipsoide | 0,37 | - |
| Kugeln u. Würfel | 0,57 | 0,39 |
| Würfel | 9,50 | 2,92 |

Tabelle 6: Durchschnittliche Eindringtiefe im Endzustand ($\times 10^{-3}$)

Der Unterschied im Festkörpervolumenanteil der Würfelschüttung ist allein mit den Eindringtiefen aber noch nicht vollständig zu erklären. Dies zeigt eine kurze Überschlagsrechnung: Die dichteste Packung der Würfel ist eine systematische Stapelung. Nimmt man hier für jede der sechs Kontaktflächen mit Kantenlänge $l_{Würfel}$ zusätzlich eine Überlappung um die mittlere Eindringtiefe $\bar{d}$ aus dem Iterativen Versuch an, kann man den neuen Feststoffvolumenanteil (der dann über 100% liegt) berechnen

$$\frac{V_{Würfel}}{V_{Würfel,verbraucht}} = \frac{l^3_{Würfel}}{(l_{Würfel} - 2\bar{d})^3} \approx 1,049$$

Selbst in dichtester Packung und nur bei Flächenkontakten sind also maximal 4,9 % mehr Volumenanteil durch die höheren Penetrationen möglich. In dieser Zufallsschüttung sind es wohl deutlich weniger.
Für einen weiteren Erklärungsansatz muss auf einen grundlegenden Unterschied zwischen der analytischen Kollisionsberechnung im *pe Physics Engine* und der Iterativen eingegangen werden. Während die Iterative für jedes Körperpaar nur einen Kontaktpunkt generiert, sind bei der Analytischen auch mehrere möglich. Grenzen zum Beispiel zwei Würfel auf einer Fläche aneinander, wird für jeden der vier Eckpunkte ein Kontakt erzeugt.

Eine Vermutung ist, dass dies zu stabileren Gleichgewicht führt, sodass gestapelte Quader nicht so leicht dazu neigen, in der Simulation umzukippen. Die mit der analytischen Erkennung generierten Kontaktpunkte neigen wahrscheinlich schneller zu einem statischen Gleichgewicht , wohingegen Stapelungen mit der iterativen Berechnung aufgrund des einen Kontaktpunktes eher instabiler zu sein scheinen.
Die Statik bilden mehrere Kontaktpunkte wohl zutreffender ab, allerdings ist dies mit den in dieser Arbeit verwendeten Algorithmen schwierig zu realisieren.
Tabelle 7 zeigt den Anteil der Kontaktpaare aus der Würfelschüttung, bei denen mehrere Kontakte generiert wurden (die Flächen-/Kantenkontakte). Bei der iterativen Erkennung gibt es davon anteilig etwas mehr und auch insgesamt mehr Kontaktpaare, was genau dadurch enstanden sein könnte, dass Strukturen, die mit der analytischen Erkennung stehen bleiben würden, in der Simulation mit der iterativen Erkennung noch einmal kippen. Dies könnte auch ein Grund für den höheren Feststoffvolumenanteil sein.

| | Wert iterative Kollisionsrechnung | Wert der analytischen Kollisionsrechung |
|---|---|---|
| Anzahl Kontaktpaare | 10397 | 9891 |
| Anteil Flächen-/Kantenkontakte | 37,2 % | 34,6 % |

Tabelle 7: Kontaktpaare der Würfelschüttung und Anteil der Flächen-/Kantenkontakte mit mehreren Kontaktpunkten in den Experimenten

### 4.2.2 Simulation von Gas-Teilchen

Ein weiterer Versuch soll zeigen, dass auch mit Hilfe der iterativen Kollisionserkennung physikalisch richtige Ergebnisse gewonnen werden können und dass sie sich auch in stark parallelen Simulation einsetzen lässt.
Hierfür wurde die Simulation der Bewegung von idealisierten Gasmolekülen ausgewählt. Diese bewegen sich frei durch den Raum und kollidieren mit anderen Molekülen. Das Resultat ist die *Brownsche Molekularbewegung.*

**Physikalischer Hintergrund**
In der kinetischen Gastheorie besteht ein Gas aus vielen Molekülen, die sich ungeordnet bewegen. Teilchen stoßen vollelastisch zusammen, üben aber sonst keine Kräfte aufeinander aus. Mithilfe der statistischen Mechanik kann ermittelt werden, dass sich die Wahrscheinlichkeiten, ein Teilchen mit einem bestimmten Geschwindigkeitsbetrag $v$ anzutreffen, durch die *Maxwell-Boltzmann-Verteilung* beschreiben lassen. Deren Dichtefunktion $f(v)$ ist durch

$$f(v) = \sqrt{\frac{2}{\pi}} * \frac{v^2 e^{-\frac{v^2}{2a^2}}}{a^3}, a \in \mathbb{R}_+$$

gegeben. Über den Parameter $a$ kann beim idealen Gas der Einfluss von Temperatur $T$ und Molekülmasse $m$ ausgedrückt werden. Es gilt

$$a = \sqrt{\frac{k_B T}{m}}$$

mit der Boltzmann-Konstante $k_B$. Das mittlere Geschwindigkeitsquadrat kann mit Hilfe der Dichtefunktion abhängig von $a$ bestimmt werden und ergibt sich zu

$$\overline{v^2} = 3 * a^2 = \frac{3 k_B T}{m}$$

**Modell und Durchführung**
Als Modell für die Teilchen wurden in dieser Simulation Kugeln verwendet, welche vollelastisch stoßen. Die Gesamtenergie bleibt also erhalten. Zu Beginn wurde eine Anzahl von 64.000 Atomen erzeugt. Diese befinden sich in einem würfelförmigen Volumen mit periodischen Randbedingungen (verlässt ein Teilchen durch eine Seite das betrachtete Volumen, wird es auf der gegenüberliegenden Seite wieder eingefügt). Dadurch wirkt es, als betrachte man einen Ausschnitt aus einer unendlichen Gasmenge.
Zu Beginn erhalten alle Moleküle eine Anfangsgeschwindigkeit $v_0$, welche aber in eine zufällige Richtung zeigt. Die mittlere kinetische Energie $\overline{E}$ eines Teilchens kann abhängig von dem mittleren Geschwindigkeitsquadrat $\overline{v^2}$ und seiner Masse $m$ wie in der klassischen Mechanik durch

$$\overline{E} = \frac{1}{2} m \overline{v^2}$$

bestimmt werden. Da mittlere Energie und Masse über die Zeit konstant bleiben, ist auch $\overline{v^2}$ zeitlich unverändert.
Der Verteilungsparameter $a$ kann deshalb für diesen Versuch über das mittlere Geschwindigkeitsquadrat zu Beginn, $v_0^2$, bestimmt werden:

$$a = \sqrt{\frac{1}{3} v_0^2}$$

Die zu erwartende Geschwindigkeitsverteilung ist also vollständig bekannt. Die anfängliche Verteilung, bei der alle Moleküle den gleichen Geschwindigkeitsbetrag haben, wandelt sich nach einer außreichend großen Anzahl von Zeitschritten annähernd in die Maxwell-Boltzmann-Verteilung um. Abbildung 23 zeigt diesen Prozess.
Die Simulation wurde mit 64 Prozessen auf 32 physischen Rechenkernen durchgeführt, wobei jedem Prozess einen Teil des zu simulierenden Raumes zugeteilt war.

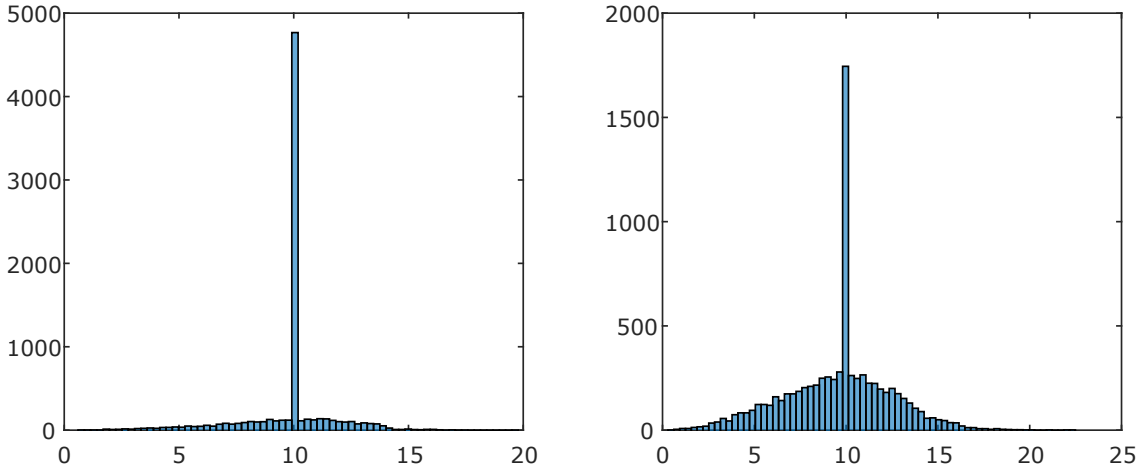
**Ergebnisse**
Mit dem bloßen Auge sind in den mit beiden Kollisionserkennungen erhaltenen Endverteilungen (Abb. 24) keine großen Unterschiede zu erkennen. Beide sind der mathematischen Verteilung sehr nah. Deshalb wurden in Tabelle 8 noch einige statistische Kennzahlen ermittelt.
Auch hier ist das Bild nicht eindeutig. Beide Verfahren liefern aber sehr genaue Werte.
Es konnte gezeigt werden, dass die iterative Kollisionserkennung auch parallel nutzbar ist und dass sie physikalisch korrekte Ergebnisse produziert.

| Kennzahl | Wert iterative Kollisionsrechnung | Wert analytische Kollisionsrechnung | Mathematischer Wert [23] |
|---|---|---|---|
| Mittelwert $\overline{v}$ | 9,2214 | 9,2093 | 9,2132 |
| Varianz | 15,0545 | 15,1049 | 15,1174 |
| Median | 8,8890 | 8.8698 | 8,8806 |
| Schiefe | 0.4836 | 0.4975 | 0,4857 |

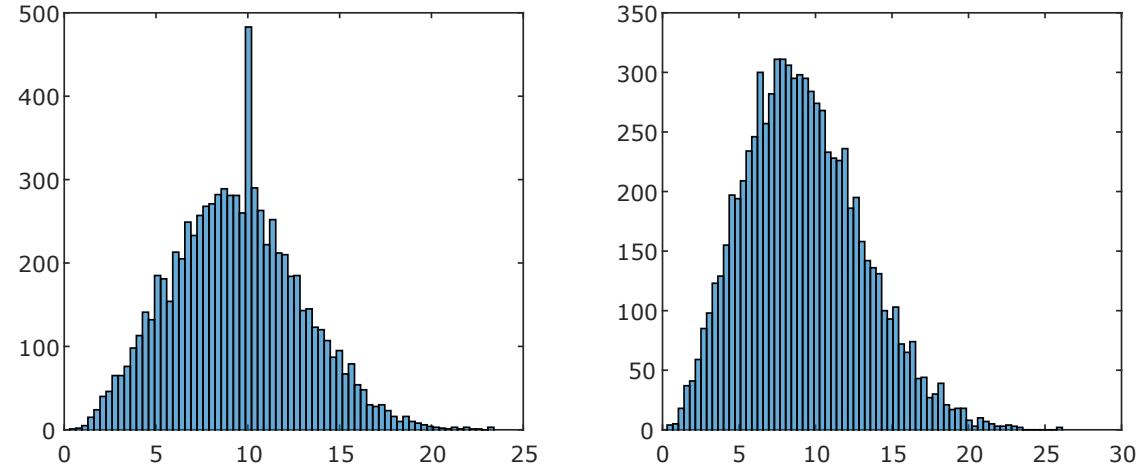
Tabelle 8: Kenndaten der ermittelten Geschwindigkeitsverteilungen

## 4.3 Performance und Wahl der Parameter

Zum Abschluss der Evaluation soll noch ein kurzer Vergleich der Performance (Rechenleistung) zwischen iterativer und analytischer Kollisionserkennung durchgeführt werden. Alle Messun-

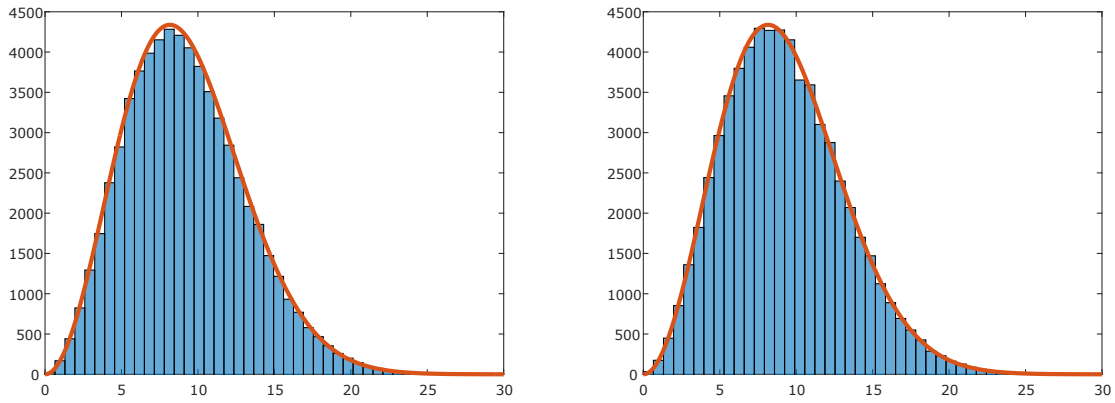



(a) $t = 20.000$ (b) $t = 50.000$




(c) $t = 100.000$ (d) $t = 240.000$

Abbildung 23: Entstehung der Maxwell-Boltzmann-Verteilung aus der Einheitsverteilung. Die Histogramme zeigen die Anzahl der Teilchen pro Geschwindigkeitsbereich (hier: $n = 8000$).



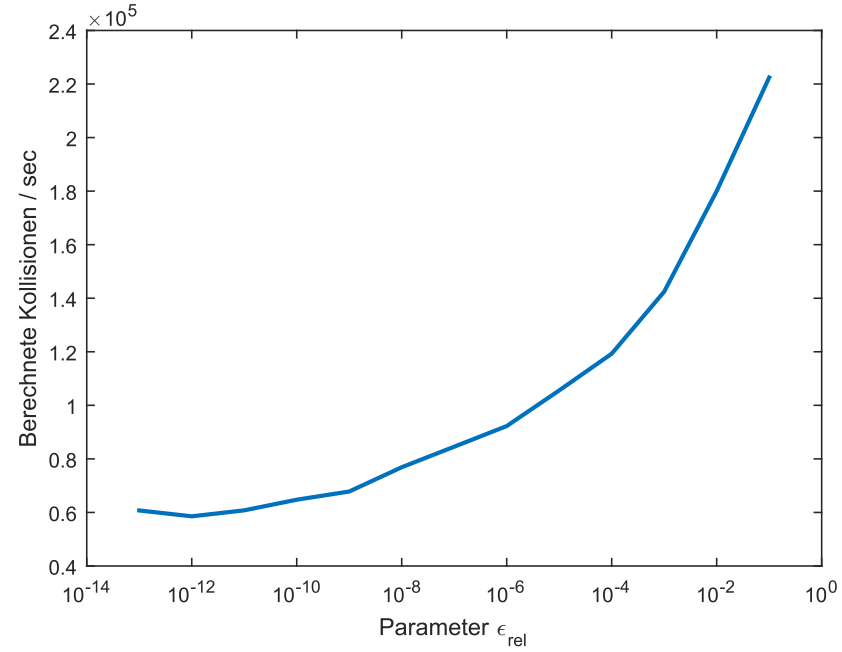

(a) Analytische Kollisionserkennung

(b) Iterative Kollisionserkennung

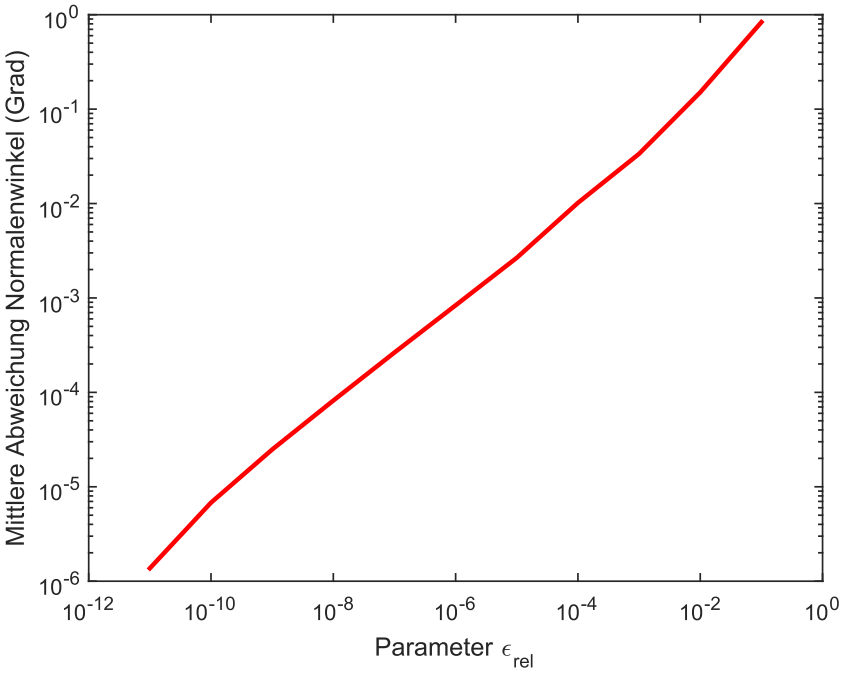
Abbildung 24: Histogramm und berechneter Dichtegraph der Maxwell-Boltzmann-Verteilung

gen wurden auf einem Kern eines Intel Xeon E5620 Prozessors durchgeführt. Die Taktung wurde dafür fest auf 2,4 GHz gesetzt, um Ungenauigkeiten durch automatische Frequenzanpassungen zu vermeiden.
Alle Berechnungen in diesem Abschnitt wurden mit doppelter Genauigkeit (*Double Precision*) durchgeführt.
Um möglichst realistische Eingabedaten für die Messung zu erhalten, wurden Konfigurationen aus dem Schüttungsversuch verwendet. Dazu wurde dieser noch einmal durchgeführt und nach jedem Zeitschritt die von der CCD gelieferten möglichen Kontaktpaare gespeichert. Diese Datensätze enthielten die Art der beteiligten Körper sowie deren exakte Positionen und Rotationszustände.
Für die Messung wurde zuerst eine Menge von Paaren gleichen Typs (z.B. Würfel mit Würfel) in den Speicher gelesen und berechnet.
Die erhaltenen Werte geben nur die Zeit der Kollisionsberechnung, abhängig von den betrachteten Objekten, Parametern und der Art der Kollisionsberechnung, wieder. Andere Teile einer Starrkörpersimulation, wie Integration oder Kommunikationsoverhead, haben keinen Einfluss auf die Messung.

**Wahl des Abbruchparameters**
Entscheidend für die Performance ist bei einer Beteiligung runder Formen die Wahl des Abbruchkriteriums. Deshalb wurde der Parameter des EPA, welcher den relativen Fehler bei der Eindringtiefe beschränkt, $\epsilon_{rel}$, für die erste Messung unterschiedlich gewählt.
Im Regelfall bricht eine Berechnung erst dann ab, wenn diese Genauigkeit auch wirklich erreicht wurde. Die Implementierung erkennt aber auch, ab wann eine weitere Approximation aufgrund numerischer Fehler nicht möglich ist. Dies ist zum Beispiel der Fall, wenn keine Dreiecksfläche des Polytopes mehr einen Abstand aufweist, der zwischen den Schranken, die in vorherigen Iterationen bestimmt wurden, liegt. Auch wenn das Polytop aufgrund numerischer Ungenauigkeiten nicht mehr konvex ist, wird dies bemerkt. Hier liefert eine Fortführung der Berechnung keine sinnvollen Ergebnisse mehr und es werden die Werte der letzten Iteration zurückgegeben.
Um den Einfluss des Parameters zu bestimmen, wurden Versuche mit Ellipsoiden durchgeführt. Kugeln sind nicht geeignet, da der Algorithmus auf diese Flächen besonders optimiert wurde. Die Quader stellen ebenfalls keine gute Wahl dar, da sich aufgrund ihrer wenigen Eckpunkte Flächen nach einigen Schritten exakt interpolieren lassen. Der Wert des Abbruchparameters




(a) Performance




(b) Mittlerer Fehler zum Normalenwinkel-Referenzwert in Grad

Abbildung 25: Auswirkungen der Wahl des Abbruchparameters auf die gemessene Performance (berechnete Kollisionen pro Sekunde) und Genauigkeit der Kontaktnormalen

| $\epsilon_{rel}$ (Parameter) | numerische Fehler | nicht erkannte Kollisionen |
|---|---|---|
| $10^{-1}$ | 0 | 5939 |
| $10^{-2}$ | 0 | 561 |
| $10^{-3}$ | 0 | 62 |
| $10^{-4}$ | 0 | 5 |
| $10^{-5}$ | 0 | 2 |
| $10^{-6}$ | 0 | 0 |
| $10^{-7}$ | 0 | 0 |
| $10^{-8}$ | 0 | 0 |
| $10^{-9}$ | 1 | 0 |
| $10^{-10}$ | 43 | 0 |
| $10^{-11}$ | 488 | 0 |
| $10^{-12}$ | 7015 | 0 |
| $10^{-13}$ | 15411 | 0 |

Tabelle 9: Anzahl der EPA-Berechnungen, die aufgrund numerischer Fehler abgebrochen wurden und Anzahl nicht erkannter Kollisionen. Variert wurde der Parameter $\epsilon_{rel}$, der den relativen Fehler bestimmt. (Es wurden 250.000 Ellipsoid-Paare betrachtet, von denen allerdings einige Kollisionen bereits durch den GJK-Algorithmus ausgeschlossen wurden. Die Daten beziehen sich auf die ca. 87.000 Fälle, in denen der EPA zur Anwendung kam.)

hat dann keine Auswirkung mehr. Ellipsoide können mit mehr Flächen und Punkten immer genauer approximiert werden, weshalb sie hier zum Einsatz kamen.
Tabelle 9 zeigt, dass die Zahl der bemerkten numerischen Fehler bei einer zu niedrigen Wahl des Abbruchkritierums rasant ansteigt. Eine weitere Verringerung führt in diesen Fällen zu keiner Verbesserung des Ergebnisses mehr. Des Weiteren ist ersichtlich, dass der Parameter nicht zu groß sein darf, damit alle Kollisionen zuverlässig erkannt werden.
Die eigentliche Performance sinkt wie erwartet mit steigender Genauigkeit (Abb. 25a). Ab einem bestimmten Niveau (ca. $10^{-10}$) stagniert die Laufzeit, da der Abbruch nun immer öfter aufgrund numerischer Fehler erfolgt.
Für die Eindringtiefe wird der gegebene relative Fehler in der Regel erreicht (außer falls ein Abbruch aufgrund numerischer Fehler erfolgt). Die Genauigkeit der Eindringtiefe ist daher direkt proportional zu $\epsilon_{rel}$.
Eine entscheidende Frage ist aber, wie sich eine Verringerung der Toleranz hier auf die Genauigkeit der Kontaktnormalen auswirkt, die ja für die Auflösung der Kollision von entscheidender Bedeutung ist. Da für Ellipsoide keine analytische Rechnung implementiert ist und eine Reduktion des Versuches auf einfache Fälle keine realistische Stichprobe darstellt, konnten keine analytischen Referenzwerte verwendet werden. Im Diagramm (Abb. 25b) ist daher die mittlere Abweichung von einer Normalen, die iterativ mit sehr geringem ($10^{-12}$) relativen Fehler bestimmt wurde, eingezeichnet. Es ist erkennbar, dass auch die Genauigkeit der Kontaktnormalen mit einer höheren Genauigkeit der Eindringtiefe stetig steigt. Der Graph ähnelt im doppelt logarithmischen Plot sogar einer Gerade, was auf einen Zusammenhang deutet, der sich durch eine Potenz- oder Wurzelfunktion beschreiben lässt. Tatsächlich kann man feststellen, dass die mittlere Abweichung des Normalenwinkels $\overline{\alpha}$ in etwa proportional zur Wurzel des vorgegebenen relativen Fehlers ist.

$$\overline{\alpha} \sim \sqrt{\epsilon_{rel}}$$

Für optimale Ergebnisse sorgt eine Wahl des Parameters zwischen $10^{-5}$ und $10^{-10}$, da so Kollisionen zuverlässig erkannt und numerische Fehler gering gehalten werden. Abhängig davon, ob die Genauigkeit oder die Performance bei der Anwendung zu priorisieren ist, sollte ein

niedriger oder größerer Toleranzwert aus diesem Intervall gewählt werden.

**Messwerte beider Kollisionserkennungen für verschieden Körperpaare**
Aufgrund der Ergebnisse des letzten Abschnittes wurde der relative Fehler der iterativen Methode für die folgenden Messungen fest auf $\epsilon_{rel} = 10^{-7}$ beschränkt und die Berechnung für verschiedene Körperkombinationen iterativ und analytisch durchgeführt.
In Tabelle 10 und in Abbildung 26 ist die Anzahl der Kontaktpaare, welche in einer Sekunde berechnet werden konnten, angegeben bzw. graphisch dargestellt.
Die allgemeine, iterative Kollisionserkennung ist um einen zweistelligen Faktor langsamer. Dies ist auch zu erwarten gewesen. Schließlich kann für die hier verwendeten Körpertypen (insbesondere die Kugeln) die Kollision extrem einfach berechnet werden. Hierzu wird nur eine Berechnung des Abstandes der beiden Mittelpunkte und eine Subtraktion der beiden Radii benötigt.
Die iterative Berechnung und die Approximation der Fläche durch den EPA ist natürlich deutlich rechenintensiver. Vor allem für die gerundeten Flächen der Kugel bedeutet dies einen sehr hohen Aufwand. Für eine Simulation, welche nur aus Kugeln besteht, ist deshalb der Unterschied am erheblichsten, da die analytische Rechnung sehr günstig ist, die iterative hingegen sehr rechenlastig. Es ergibt sich ein Faktor von über 50.
Werden die Körper etwas komplexer, steigt auch der Aufwand der analytischen Rechnung schnell an. In der Simulation, welche nur Würfel enthält, benötigt diese schon mehr als die vierfache Zeit verglichen mit der Simulation der Kugeln.
Hier hat die iterative Erkennung hingegen sogar noch den Vorteil, dass sich die Oberflächen des CSO nach einer endlichen Anzahl von Schritten perfekt durch Flächen annähern lassen. Deshalb ist sie hier mehr als doppelt so schnell wie bei den Kugeln. Gesamt liegt der Unterschied hier nur noch bei einem Faktor von ca. fünf.
Es ist davon auszugehen, dass sich dieser Trend mit noch komplexeren Körpern fortsetzten lässt. Würde man konvexe Dreiecksnetze kollidieren lassen, würde sich die Leistung der analytischen Berechnung nochmals deutlich verringern, wohingegen die der Iterativen in etwa konstant bleiben sollte. Hier könnte es durchaus möglich sein, dass beide Verfahren eine ähnliche Geschwindigkeit entfalten.
Ähnliches gilt eventuell auch für die Ellipsoide. Für die iterative Kollisionserkennung kann man mit einer Leistung wie bei Kugeln rechnen. Die analytische Berechnung ist für diese Formen allerdings sehr aufwändig.

| kollidierende Körper | iterative Kollisionsrechnung | analytische Kollisionsrechnung | Verhältnis Analytisch zu Iterativ |
|---|---|---|---|
| Kugel – Kugel | 1,61 | 89,29 | 55,6 |
| Kugel – Würfel | 2,91 | 56,82 | 19,5 |
| Würfel – Würfel | 4,16 | 21,74 | 5,2 |

Tabelle 10: Anzahl berechneter Kollisionen in einer Sekunde ($\times 10^5$)

**Hybride Methode**
Wenn beide Erkennungen implementiert sind stellt eine Fallunterscheidung wohl die beste Methode dar.
Handelt es sich um einen einfachen Körpertyp (Kugel, Würfel, etc.), ist die analytische Erkennung auf jeden Fall vorzuziehen und zu verwenden. Sie ist präziser und auch schneller. Für komplexe Objekte wie Dreiecksnetze oder Ellipsoide befindet sich die iterative Lösung jedoch in einer vergleichbaren Geschwindigkeitsgrößenordnung, und sollte zum Einsatz kommen.

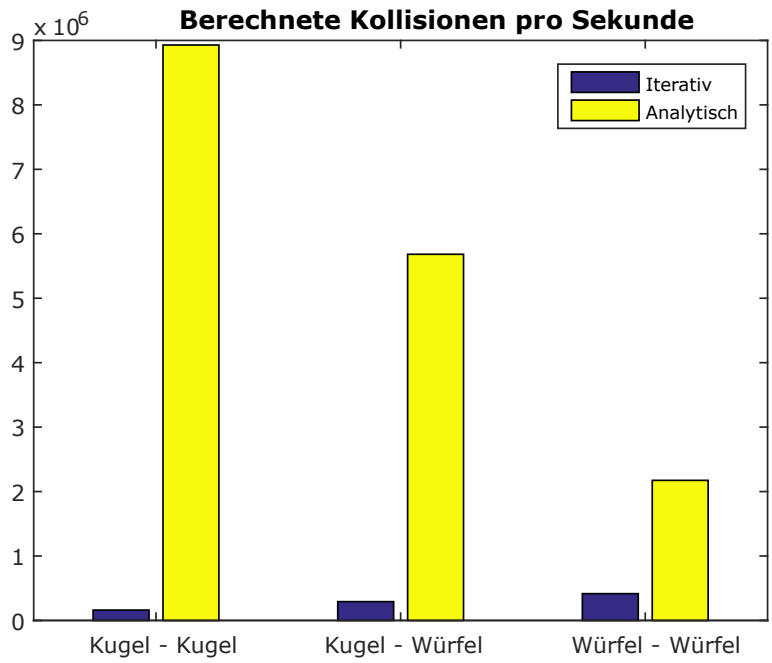



Abbildung 26: Gemessene Anzahl berechneter Kollisionen in einer Sekunde

Allerdings ist es auch nicht der Anspruch an eine allgemeine Kollisionserkennung, die Überschneidung zweier Kugeln so schnell zu berechnen, wie eine simple Formel – ihr Vorteil liegt vor allem darin, dass sie universell einsetzbar ist. Die Laufzeit spielt dabei eine Nebenrolle. Auch manche, für Implementierungsfehler anfälligen Kombinationen, wie die Kollision eines Zylinders mit einem Quader, sind deshalb mit der iterativen Erkennung gut behandelt.
Trotzdem ist es wichtig einen Blick auf die Kosten dieser Verallgemeinerung zu werfen und ein paar Abschätzungen zur Hand zu haben, wie sie dieser Abschnitt liefert.

# 5 Fazit und Ausblick

In dieser Arbeit wurde eine robuste Kollisionserkennung für konvexe Körper implementiert. Die Ergebnisse zeigen, dass eine iterative Kollisionsberechnung durchaus auch in Simulationen gut einsetzbar und die gewonnenen Werte belastbar sind.
Aus der Auswahl von betrachteten Algorithmen verfügen GJK und EPA über das größte Potential, da der funktional vergleichbare MPR-Algorithmus Schwächen bei der Berechnung der Kontaktnormalen aufweißt. Die Fähigkeiten und Schwachpunkte der einzelnen Verfahren wurden gründlich analysiert und verglichen. Anschließend wurden vorhandene Open-Source-Implementierungen bewertet und eine Implementierung auf Basis der *SOLID*-Bibliothek in den *pe Physics Engine* integriert. Die Simulationsumgebung ist hiermit in der Lage eine fast unbegrenzte Auswahl an Körpern zu verarbeiten. Eine Vielzahl neuer Anwendungsmöglichkeiten wurde erschlossen. So können die Partikel in granularen Stoffen beispielsweise durch Ellipsoide oder Dreiecksgitter noch genauer abgebildet werden.
Obwohl der EPA die beste, zurzeit bekannte Methode zur Berechnung von Penetrationsvektoren darstellt, stößt man bei der Implementierung auf einige Schwierigkeiten. Es kommt zu numerischen Fehlern, die erkannt und behoben werden müssen. Es wurden Methoden vorgestellt, die eine Anwendung des EPA ermöglichen, auch wenn der Startpunkt auf der Oberfläche

oder sogar ganz außerhalb des GJK-Ergebnissimplices liegt. So werden Abbrüche und Fehler verhindert. Durch eine Vergrößerung der betrachteten Körper um einen kleinen Rand finden die Berechnungen in einer numerisch sicheren Zone statt. Zudem wurde die exakte Darstellung von Kugelsegmenten ermöglicht, was bei vielen Kollisionen mit kugelartigen Formen zu Verbesserungen führt.
Durch gründliche Tests wurde die Funktion des Codes bewiesen und dokumentiert. Zu allen entwickelten Funktionen wurden Black-Box-Tests bereitgestellt, welche die Korrektheit bestätigen. Es wurden Millionen von einzelnen Fällen automatisiert geprüft, um eine maximale Robustheit sicherzustellen. Auch die Auswirkungen auf Endergebnisse einer Simulation wurden am Beispiel einer Schüttung oder der Bewegung von Gasteilchen untersucht. In einem parallelen Rechenumfeld ist der Einsatz ohne Probleme möglich.
Zusätzlich konnte eine genaue Abschätzung der Präzision einer iterativen Kollisionsberechnung ermittelt werden. Zusammen mit dem anschließend durchgeführten Rechenzeitvergleich liefern die Daten ein genaues Bild, was von solchen Verfahren zu erwarten ist. So kann nun besser beurteilt werden, für welche Anwendungsfälle es sinnvoll ist, eine Kollisionserkennung für allgemeine Körper zu verwenden. Auch die Kombination mit einer analytischen Methode für einfache Körpertypen ist eine gute Option, da sie Schnelligkeit und Vielseitigkeit vereint.

Auf einigen Gebieten bedarf es aber noch einer weiteren Untersuchung. Die Kollision zweier paralleler Flächen stellt für die Algorithmen insbesondere bei der Kontaktpunktbestimmung noch immer eine Schwierigkeit dar.
Obwohl in dieser Implementierung darauf geachtet wurde, effiziente Berechnungsverfahren zu verwenden, um niedrige Laufzeiten zu ermöglichen, lag der Fokus hier vor allem auf Robustheit und Korrektheit des Ergebnisses. Eine gründliche Analyse der Performance könnte deshalb zu einer weiteren Verbesserung führen.
Die Supportfunktion und die Datenstrukturen für eine Ellipsoidform wurden im Zuge dieser Arbeit implementiert, sodass Ellipsoide als Körpertyp nun verfügbar sind. Weitere Formen, wie zum Beispiel Dreiecksgitter, sollten in Zukunft noch hinzukommen. Durch die iterative Kollisionserkennung ist der Implementierungsaufwand aber beherrschbar geworden, da nur die entsprechenden Datenstrukturen mit Supportfunktion bereitgestellt werden müssen.

# Literatur


[1] dyn4j: Java collision detection and physics engine. http://www.dyn4j.org/.

[2] waLBerla framework. http://walberla.net.

[3] D. Chappuis. ReactPhysics3D – open-source physics engine. http://www.reactphysics3d.com/.

[4] K. Chung and W. Wang. Quick collision detection of polytopes in virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 125 – 131, 1996.

[5] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*, pages 189– 195, 1995.

[6] C. Ericson. *Real-time collision detection*. CRC Press, 2004.

[7] D. Fiser. libccd. https://github.com/danfis/libccd/. Github-Projektseite.

[8] E. G. Gilbert and C. P. Foo. Computing the distance between general convex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation*, 6(1):53–61, 1990.

[9] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[10] C. Godenschwager, F. Schornbaum, M. Bauer, H. Köstler, and U. Rüde. A framework for hybrid parallel flow simulations with a trillion cells in complex geometries. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM*, page 35, 2013.

[11] A. Hassanpour, H. Tan, A. Bayly, P. Gopalkrishnan, B. Ng, and M. Ghadiri. Analysis of particle motion in a paddle mixer using discrete element method (dem). *Powder Technology*, 206(1):189 – 194, 2011.

[12] K. Iglberger and U. Rüde. Large-scale rigid body simulations. *Multibody System Dynamics*, 25(1):81 – 95, 2010.

[13] K. Iglberger and U. Rüde. Massively parallel granular flow simulations with non-spherical particles. *Computer Science-Research and Development*, 25(1-2):105–113, 2010.

[14] M. C Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *Proceedings of the IEEE Interational Conference on Robotics and Automation*, pages 1008 – 1014, 1991.

[15] J. Newth. Minkowski portal refinement and speculative contacts in Box2D. 2013.

[16] T. Preclik, S. Eibl, and U. Rüde. The maximum dissipation principle in rigid-body dynamics with inelastic impacts. *Computational Mechanics*, pages 1–16, 2017.

[17] T. Preclik and U. Rüde. Ultrascale simulations of non-smooth granular dynamics. *Computational Particle Mechanics*, 2(2):173 – 196, 2015.

[18] T. Scharpff. Numerische Simulation scharfkantiger Teilchen in der pe Physikengine. https://www10.cs.fau.de/publications/theses/2014/Scharpff_DA_2014.pdf, 2014. Diplomarbeit.

[19] F. Schornbaum. Hierarchical hash grids for coarse collision detection. https://www10.cs.fau.de/publications/theses/2009/Schornbaum_SA_2009.pdf, 2009. Studienarbeit.

[20] G. Snethen. Xenocollide. http://xenocollide.snethen.com.

[21] G. Snethen. *Xenocollide: Complex collision made simple*, volume 7 of *Game Programming Gems*, chapter 2.5, pages 165–178. Course Technology PTR, 2008.

[22] A. Tasora and M. Anitescu. A convex complementarity approach for simulating large granular flows. *Journal of Computational and Nonlinear Dynamics*, 5(3):1–10, 2010.

[23] J. B. Tatum. Properties of gases. http://orca.phys.uvic.ca/~tatum/thermod/thermod06.pdf, 2015.

[24] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1 – 13, 1997.

[25] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7 – 25, 1999.

[26] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. The Morgan Kaufmann Series in Interactive 3D Technology. Elsevier, San Francisco, 2004.

[27] T. Zeiser. Simulation und Analyse von durchströmten Kugelschüttungen in engen Rohren unter Verwendung von Hochleistungsrechnern. https://opus4.kobv.de/opus4-fau/files/759/ThomasZeiserDissertation.pdf, 2008. Dissertation.

Alle Links wurden zuletzt am 27. November 2017 überprüft.