Maturarbeit

Entwicklung eines Computerspiels mit Unity

# Einleitung

Ich wollte schon seit längerer Zeit irgendetwas funktionierendes mit einer grösseren Anwendung Programmieren. Aber weil ich jedes Mal nach einigen Wochen Unterbrechungen in meinem lernfortschritt hatte und dadurch wieder von vorne hätte anfangen müssen, wurde daraus nie mehr als eine Idee. Als ich angefangen habe nach einem Thema für meine zu suchen, überlegte ich mir zuerst Themen in Chemie aber hatte auch Informatik im Hinterkopf. Schliesslich erinnerte ich mich wieder an meine vorherigen Ideen und überlegte, welche davon realistische Maturaarbeitsthemen waren. Am Ende entscheid ich mich für Die Entwicklung eines Computerspiels, weil die Komplexität je nach Ansatz sehr variabel ist. Weil ich vor meiner Maturarbeit nur sehr wenig Programmiererfahrung in Python hatte, war es sehr schwer den Umfang der Arbeit am Anfang einzuschätzen. Zusätzlich musste ich mich Zwischen den unterschiedlichen Sprachen und Spiele-Engines entscheiden. Am Ende entschied ich mich Für die Engine Unity in welcher man nur mit C# Programmieren kann. Über Diese Wahl schreibe ich in einem späteren Kapitel noch mehr. Durch diese aufgrund der fehlenden Erfahrung vor allem auf meinem Bauchgefühl basierten Entscheidung fing ich als erstes damit an die Grundlagen von c# lernen, um den umfang meines Zieles besser einschätzen zu können. Die grösste Schwierigkeit war, dass ich davor noch nie mit Unity, C# und Microsoft Visual Studio, in letzterem schreibe ich den Code, Gearbeitet habe. Weshalb ich anfangs eher ziellos anfing die Grundlagen auszuprobieren, bis ich ein besseres Gefühl für die neue Sprache und Programme entwickelt hatte. Direkt am Anfang nach der Installation gab es einen Fehler in der Automatischen Vervollständigung, welchen ich bis ein bisschen später nicht entdeckte, zu welchem ich zwar einige Lösungsvorschläge fand aber keiner wirklich funktionierte. Glücklicherweise löste sich das Problem irgendwann und gab keine anderen Fehler zwischen Unity und Microsoft Visual Studio mehr.

# Die Spiele-Engine Unity

## Frameworks

## Programmiersprachen

## Die Unity Physik-Engine

Die Physik-Engine Box2D umfasst Module, für die das Verhalten der zweidimensionalen Formen implementieren. Die Physik-Engine ist sehr umfangreich, folgende Module möchte ich detaillierter beschreiben:

* Objects and Shapes
* Pairwise Functions
* Dynamic Tree

### Objects and Shapes

<TODO> Erklärung Objekt

<TODO> Erklärung Shapes, Unterscheidung gefüllte und nicht gefüllte Shapes

Polygone und Kreise sind gefüllt und haben eine Masse während Edges nur Kanten zur Abgrenzung sind, an denen Objekte kollidieren können.

Die Formen in Box2D werden in Circle Shapes und Polygon Shapes eingeteilt.

#### Circle Shapes

Kreise haben eine Position und einen Radius.

#### Polygone Shapes

Polygone bestehen aus Strecken welche zwei Eckpunkte verbindet. Es gibt eine zusätzliche Unterteilung in konvexe und konkave Polygone. Um Polygone herum hat es eine dünne Schicht welche dich Polygone voneinander getrennt hält. Diese Schicht hilft bei «continuous collision» und verhindert, dass die Polygone sich überlappen. Dadurch entstehen aber kleine Abstände zwischen den Formen.

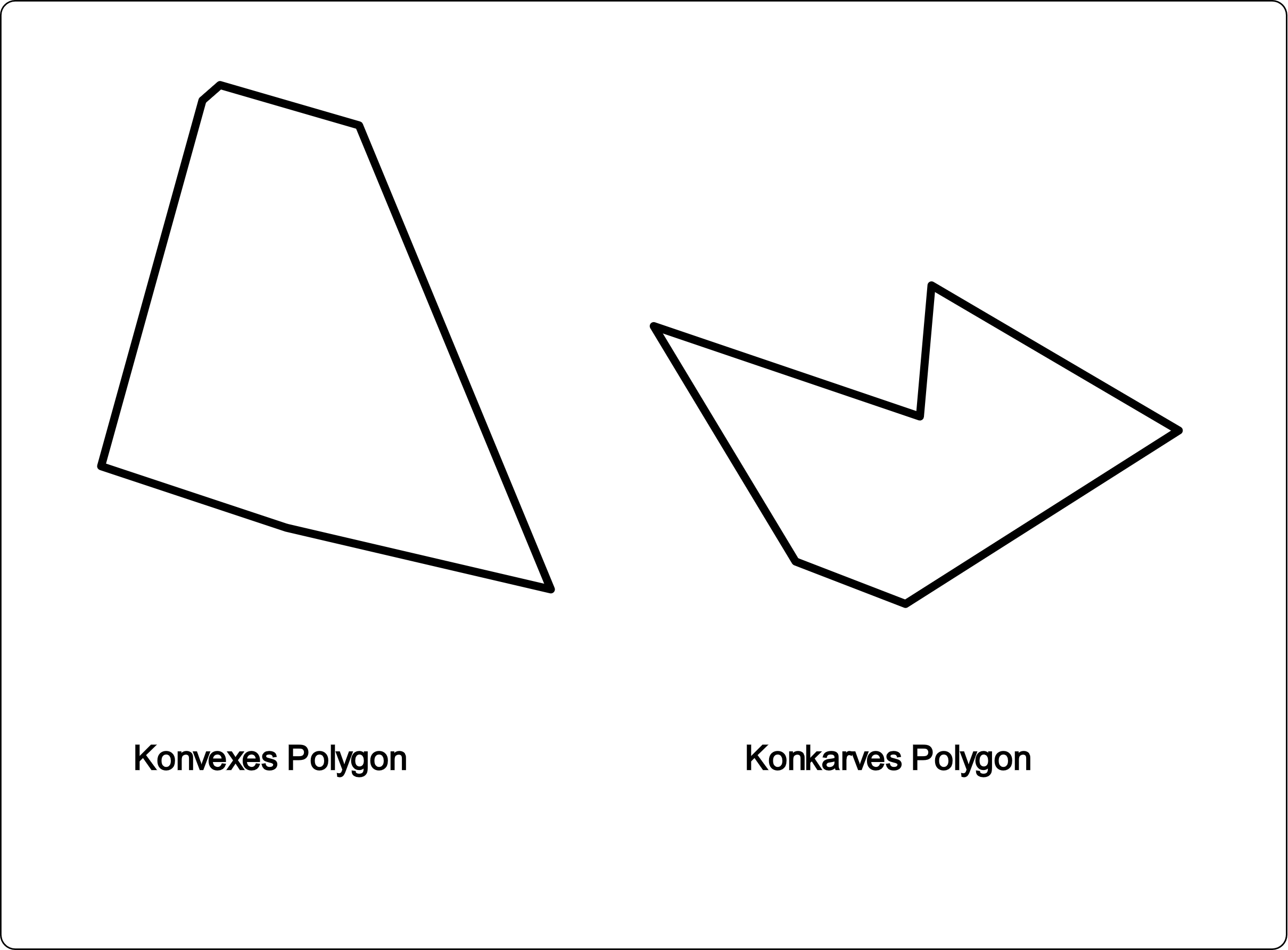


Abbildung :Grafik Polygone

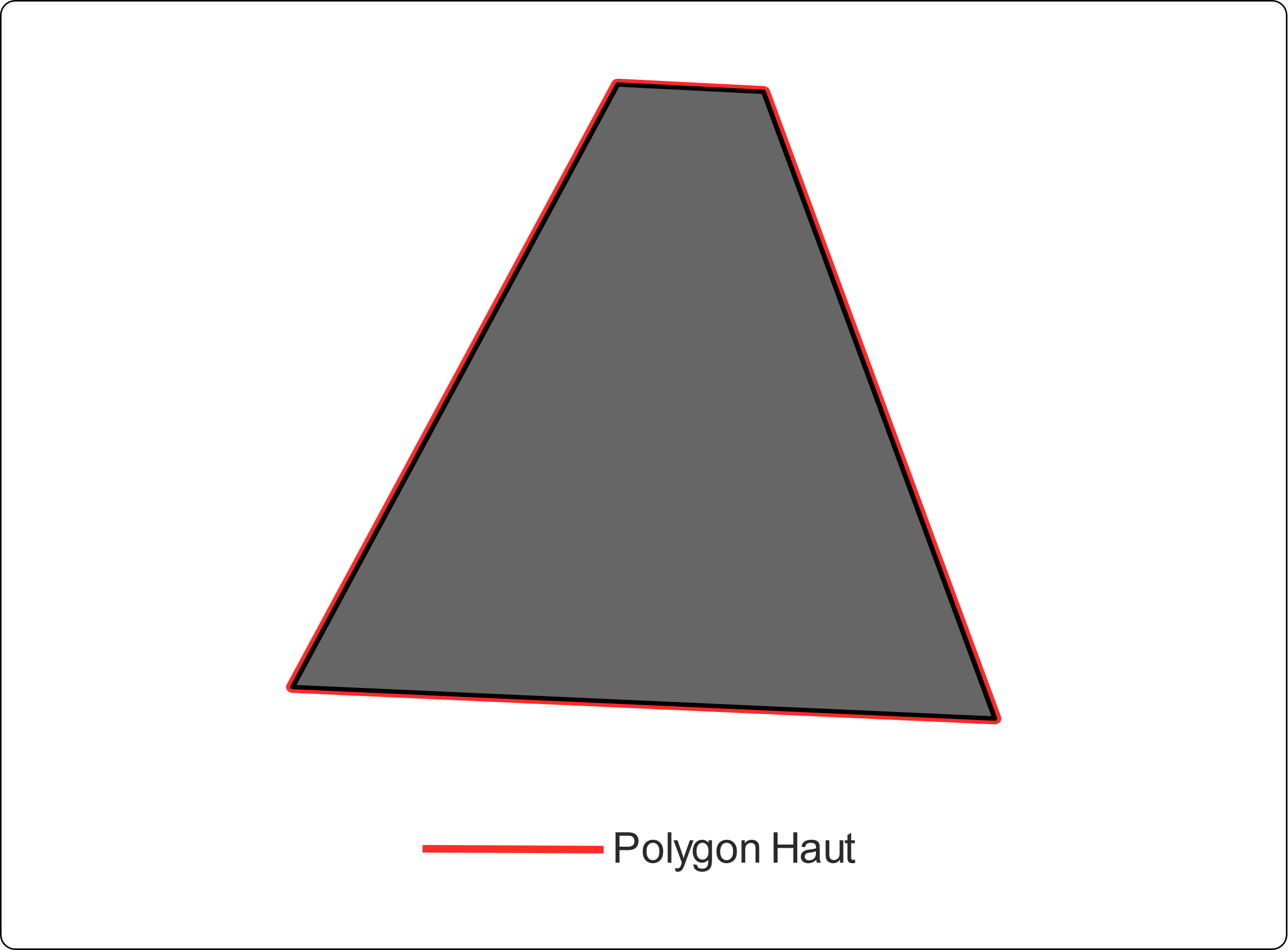


Abbildung :Polygon Haut

#### Edge Shapes

Edge Shapes sind Formen, welche nur mit anderen Objekten, welche keine Edge Shapes sind, zusammenstossen können. <TODO> Beispiel für Verwendung

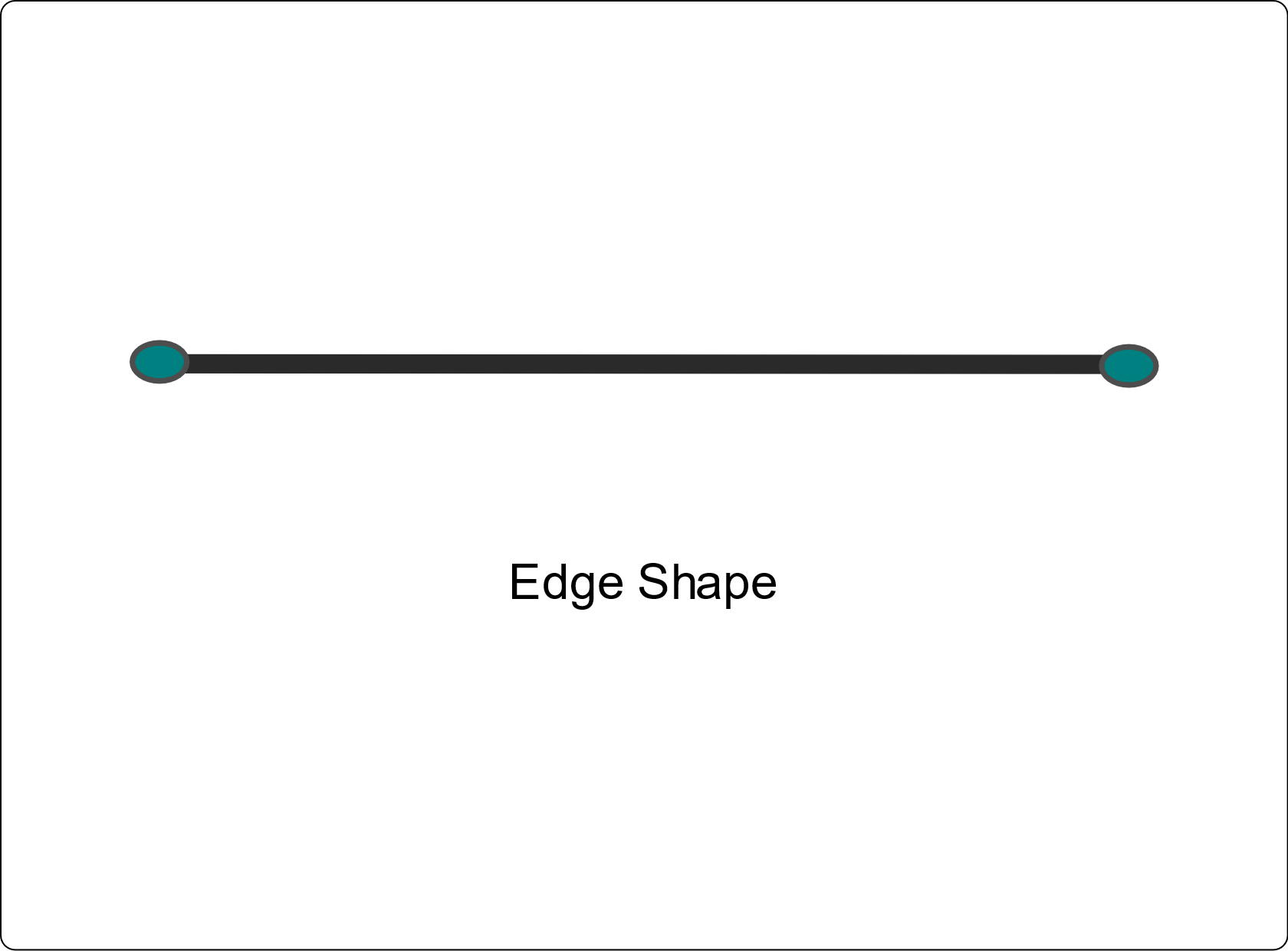


Abbildung :Edge Shape

Die Aneinanderreihung von mehreren Edge Shapes führt zu Zusammenstössen mit den Eckpunkten, sogenannte «ghost collisions». Diese können von Box2D behoben werden aber der Algorithmus funktioniert nur auf einer Seite der Linie.

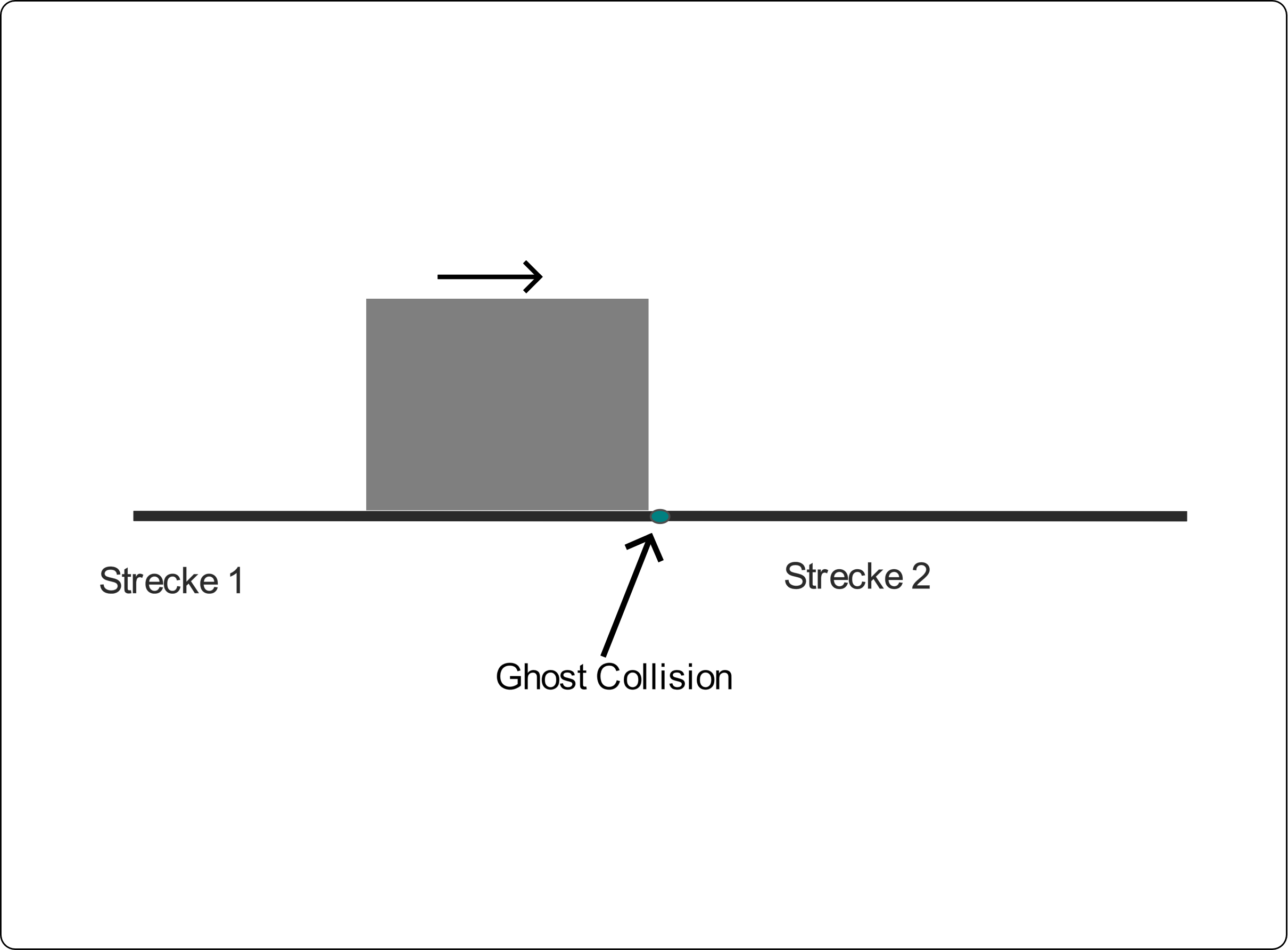


Abbildung :ghost collisions

#### Chain Shapes

Mehrere Strecken werden zu einer Chain Shape aneinandergereiht. Chain Shapes haben nur einseitige Kollision. Diese einseitige Kollision schützt vor ghost collisions. Sie werden vor allem als statische Objekte für die Spielewelt verwendet.

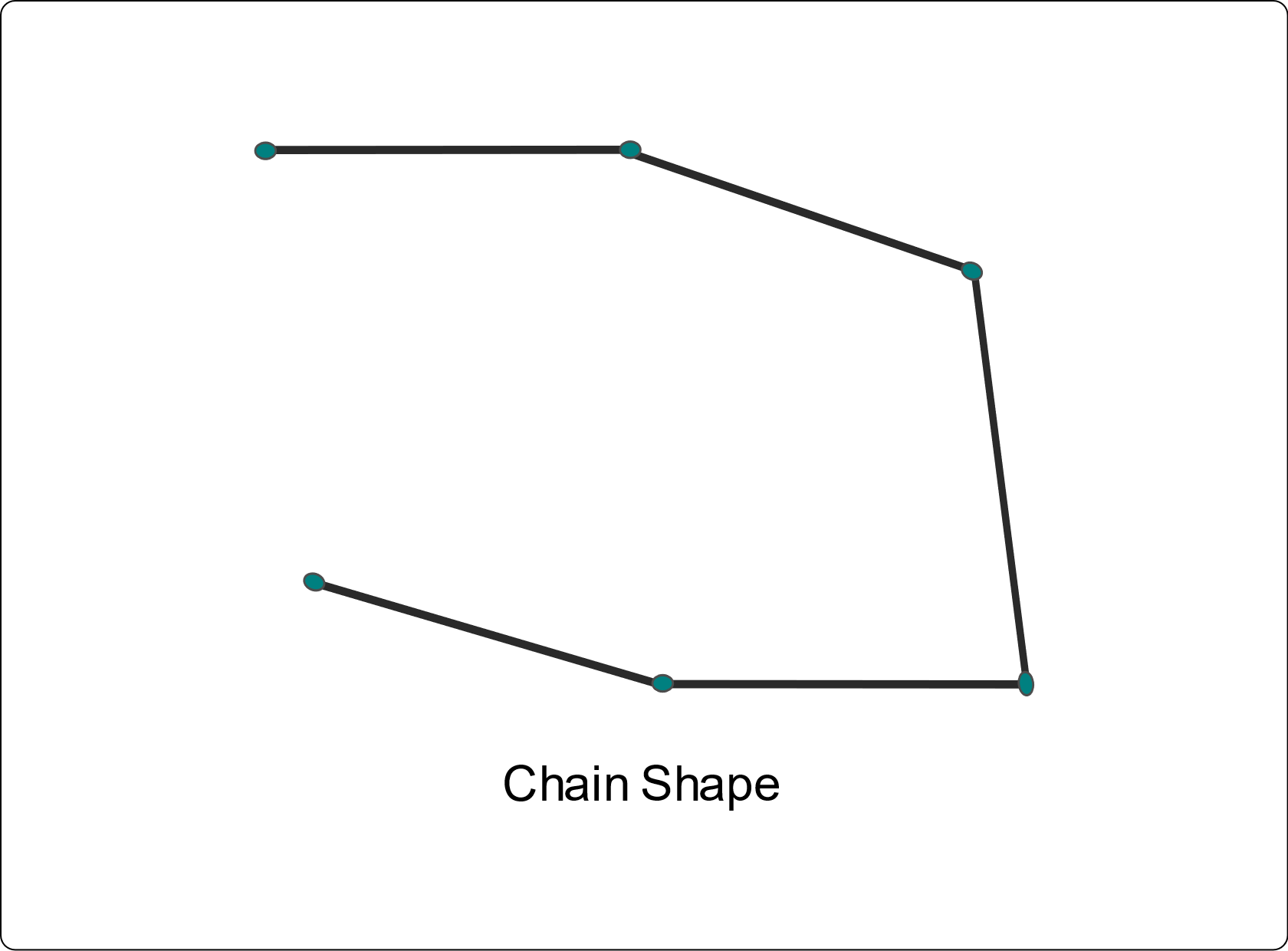
<TODO> Beispiel für Verwendung

Abbildung :Chain Shape

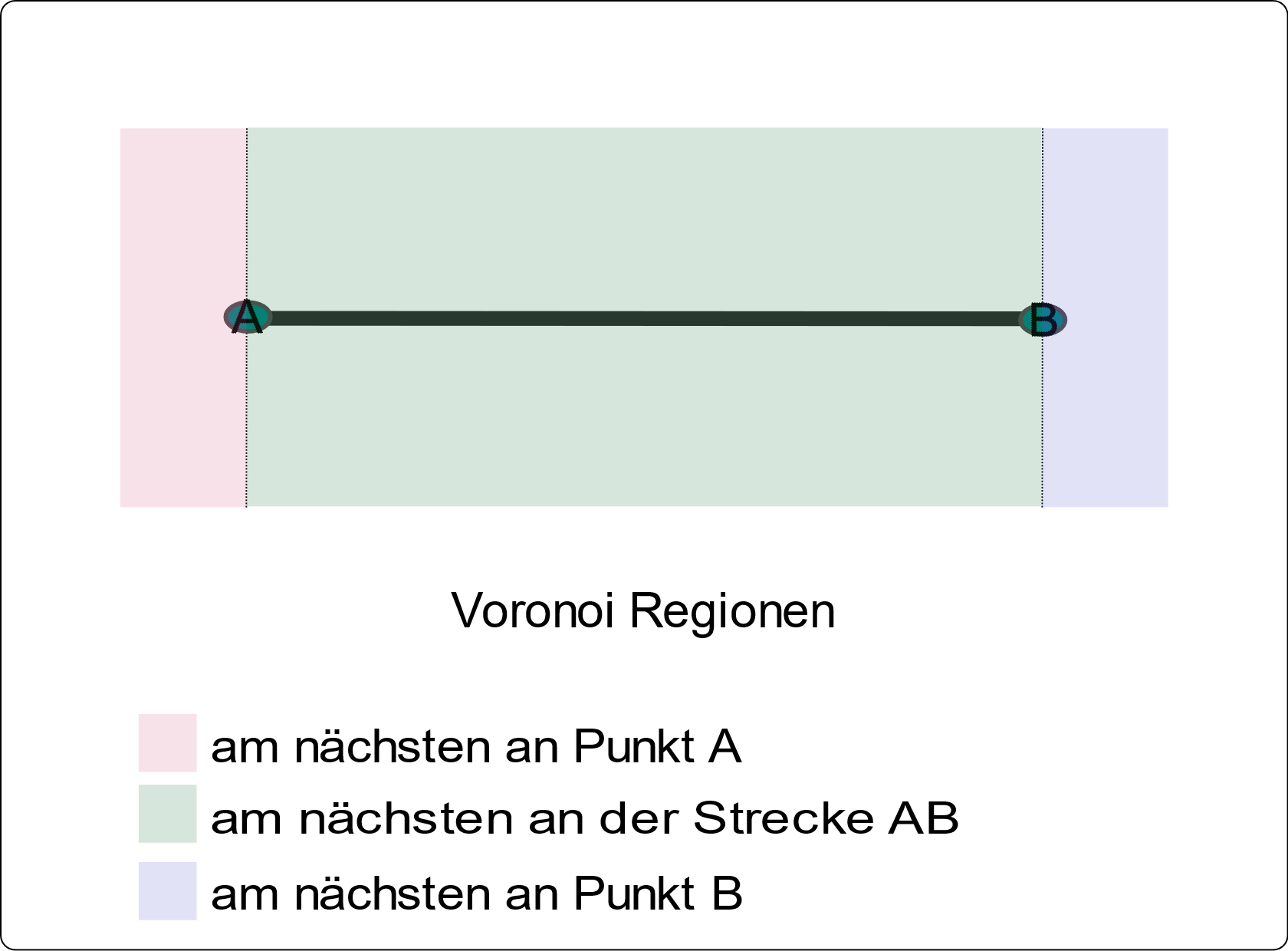
### Pairwise Functions

#### Contact manifolds

#### Distance

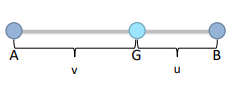
##### Voronoi regions

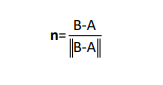
Um den Punkt auf einer Strecke zu finden, welcher am nächsten an einem anderen Punkt liegt, können Voronoi Regionen verwendet werden. Die Bereich um die Strecke wird in drei Teile eingeteilt je nach dem welchem Teil der Strecke ein Punkt in diesem Gebiet am nächsten ist.

<TODO>Grafik 

##### Barycentric coordinates

Baryzentrische Koordinaten definieren den Punkt auf der Strecke Welche dem Abfragepunkt am nächsten liegt genauer, indem sie den Bruchteil der Entfernung von den Eckpunkten angeben. Wenn man die beiden Koordinaten der Strecke addiert, erhält man immer 1.

  
Die Koordinaten u und v können bestimmt werden indem zuerst ein Einheitsvektor n ausgerechnet wird

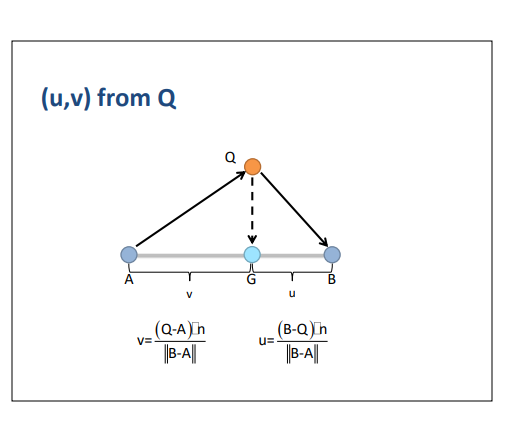


Danach kann man mit dem Vektorprodukt u und v berechnen

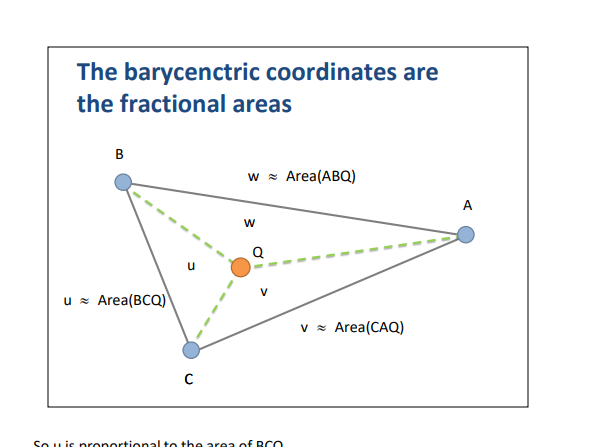
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

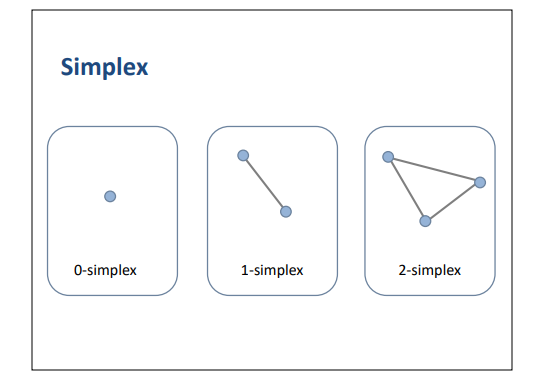
Weil sich das Vektorprodukt nicht verändert, wenn anstatt dem punkt auf der gerade der Datenabfragepunkt (Query point) verwendet wird kann auch direkt mit dem Datenabfragepunkt gerechnet werden.



Sobald die strecke zu einem Dreieck erweitert wird, entstehen neue Probleme. Mit dem jetzigen Vorgehen kann nur die Position der Datenabfragepunkte in der Nähe der Eckpunkte bestimmt werden und es hat keine Informationen über die Restlichen Möglichkeiten. Deshalb muss eine dritte Koordinate zu den Baryzentrischen Koordinaten hinzugefügt werden.



##### GJK distance algorithm



Um Polygone mit mehr als 3 ecken zu berechnen wird der GJK-Entfernungsalgorithmus verwendet. Dieser fängt mit einem 0-Simplex an und berechnet den Richtungsvektor mit dem Datenabfragepunkt. Mit dem Richtungsvektor wird ein Hilfspunkt ausgewählt. Diese beiden Eckpunkte werden zu einem 1-Simplex. Es wird wieder der Richtungsvektor des am nächsten an dem Datenabfragepunkt liegenden Punkt auf der 1-simplex und dem Datenabfragepunkt gesucht. Mit diesem Vektor wird ein zweiter Hilfs punkt ausgesucht und eine 2-Simplex gebildet. Der am weitesten von dem Datenabfragepunkt entfernte punkt kann entfernt werden, um die Rechnung zu vereinfachen. Mit der Verbleibenden 1-Simplex kann die kleinste Distanz zwischen dem Datenabfragepunkt und dem Objekt berechnet werden.

##### Minkowski difference

<TODO>

### Das Dynamic Tree Modul

Ein Spiel besteht in der Regel aus sehr vielen Objekten. Für den Spielverlauf muss häufig die Beziehung zwischen Objekten geprüft werden, zum Beispiel zur Prüfung:

* ob zwei Objekte miteinander kollidieren,
* ob ein Objekt aus Perspektive der Spielfigur sichtbar ist,
* oder zum Finden des Bodens.

Für die Berechnung werden sogenannte Casts verwendet. Die Casts kann man sich als Linien (Raycast) oder Flächen (Boxcast) vorstellen. Für 3D-Spiele gibt es auch räumliche Casts (Spherecast), was aber nicht Teil diese Arbeit ist. Mit den Casts wird geprüft, ob sich diese mit den Objekten schneiden. Die Längen der Casts sind konfigurierbar, um das Verhalten der Objekte zu bestimmen.

Die folgende Grafik illustriert einen Cast und sieben Objekte, wobei zwei der Objekte vom Cast geschnitten werden.

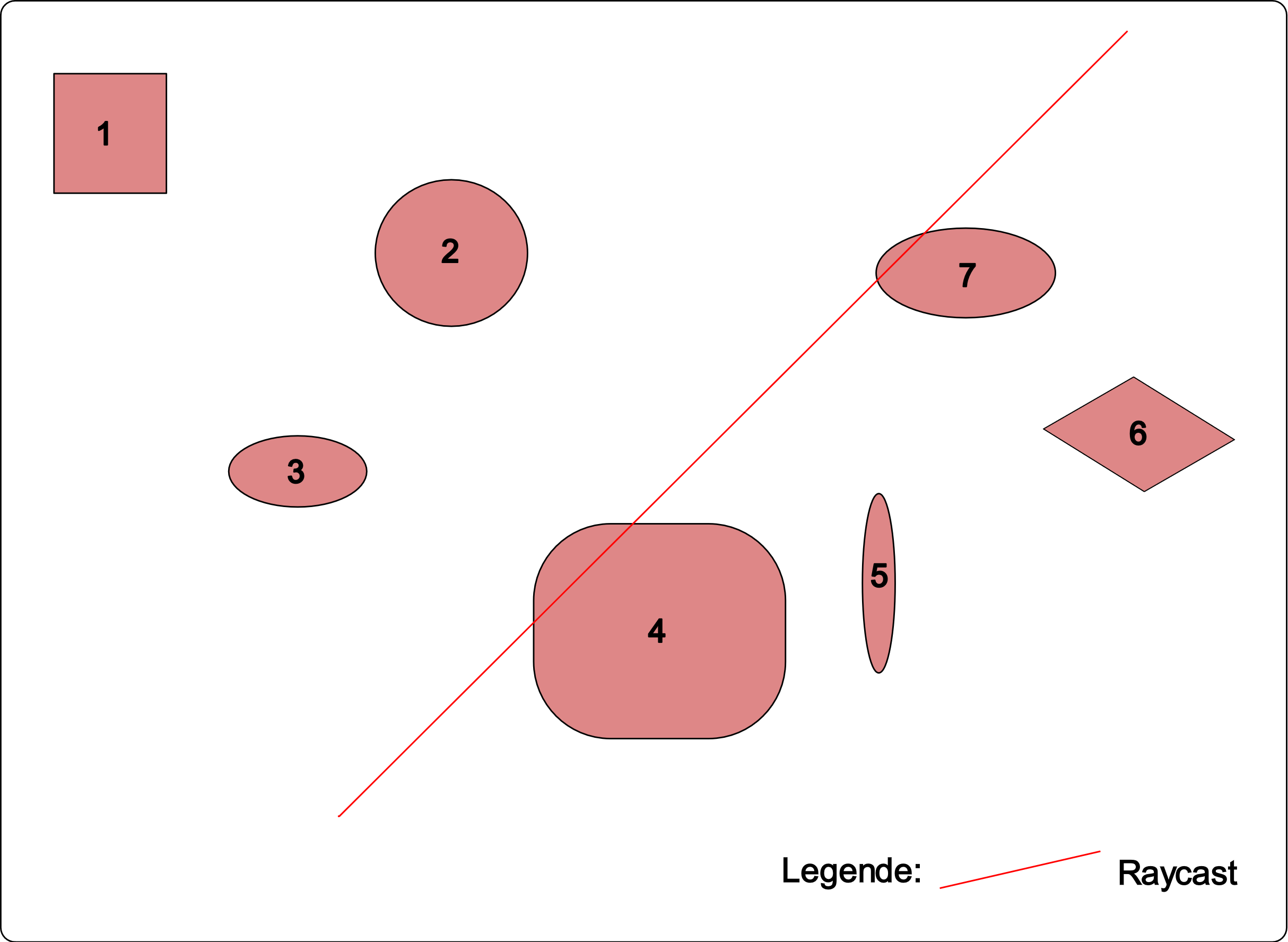


Abbildung : Grafik mit Objekten und einem Raycast

Die Berechnung von Schnittpunkten komplizierter Objekt-Formen wäre sehr langsam, daher werden die Objekte durch rechteckige Rahmen eingegrenzt. Für jedes Objekt wird mit den oberen und unteren Ecken sogenannte Axis Aligned Bounding Boxes (AABB) berechnet, wie am Beispiel der folgenden Grafik zu sehen ist.

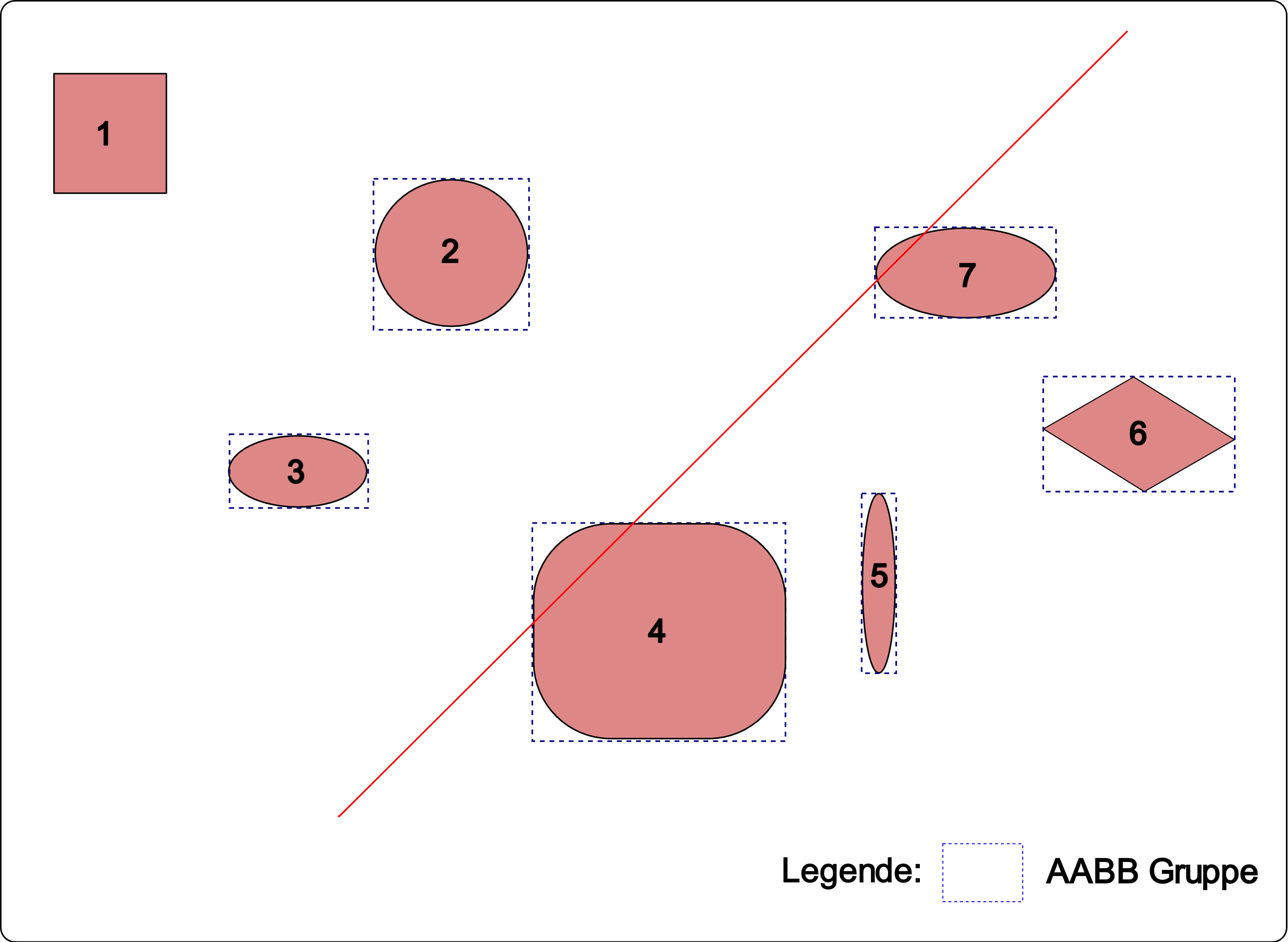


Abbildung : Graphik mit Objekten, deren AABBs und einem Raycast

Je mehr Objekte vorhanden sind, desto länger würde die Prüfung dauern, wenn jedes Objekt einzeln berechnet wird, was man Brute Force Vorgehen nennt. Um die Effizienz der Berechnung zu verbessern, werden die AABBs der Objekte in Gruppen zusammengefasst. Wenn ein Raycast eine AABB-Gruppe trifft, wird die Berechnung innerhalb dieser Gruppe fortgeführt, wodurch insgesamt weniger Berechnungen benötigt werden.

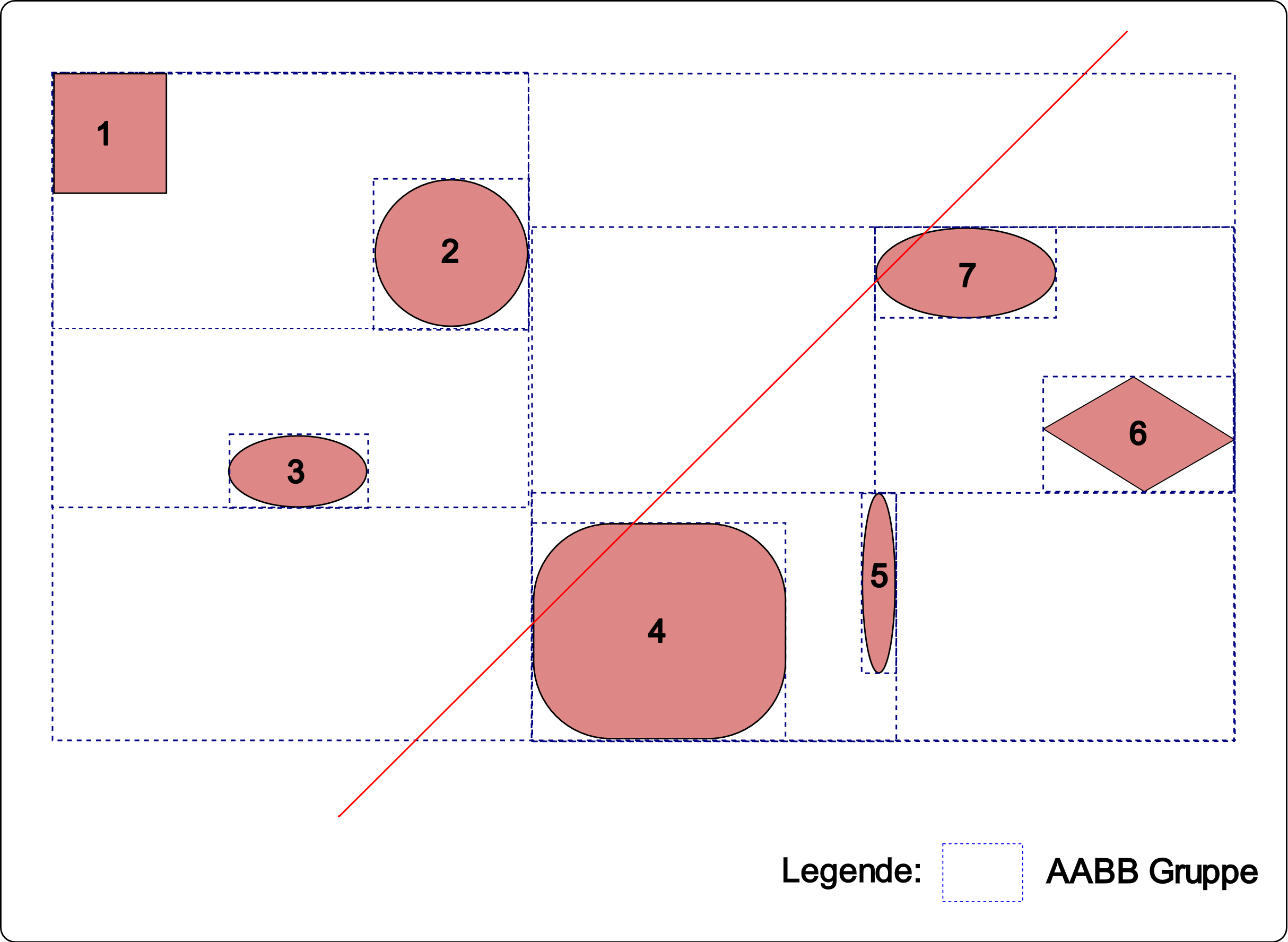


Abbildung : Grafik der AABB-Gruppen

Für die Erstellung der AABB-Gruppen werden die AABBs von jeweils zwei Objekten in ein neues AABB zusammengefasst. Dieser Vorgang wird wiederholt, bis eine AABB-Gruppe übrig ist, die alle Objekte beinhaltet. Die AABBs der Objekte und der Objektgruppen werden in einem Binärbaum, einer sogenannten Bounding Volume Hierarchy (BVH) gespeichert.



Abbildung :Bounding Volume Hierarchy

#### Such-Algorithmus mit Bounding Volume Hierarchy

Durch die Bounding Volume Hierarchy können zuerst die grössten AABBs kontrolliert werden, wodurch bereits viele Objekte ausgeschlossen werden können, falls eines der AABBs an der Spitze der Hierarchie nicht vom Raycast getroffen wird. Dieser Vorgang wird wiederholt bis die AABBs nur noch einzelne Objekte enthalten. Anschliessend wird die Berechnung für die tatsächliche Objektform durchgeführt.

#### Erstellung der Bounding Volume Hierarchy

Es gibt drei Varianten für die Erstellung der Bounding Volume Hierarchy:

* Bottom Up
* Top Down
* Incremental

Bei dem Bottom Up Vorgehen werden die Objekte mit dem jeweiligen Nachbar-Objekt in eine Gruppe zusammengefasst. Danach wird dieser Vorgang mit den AABBs der neu erstellten Gruppen wiederholt, bis nur noch eine AABB übrig ist. Mit dem Bottom Up Vorgehen werden alle Objekte gleichmässig in der Bounding Volume Hierarchy verteilt.

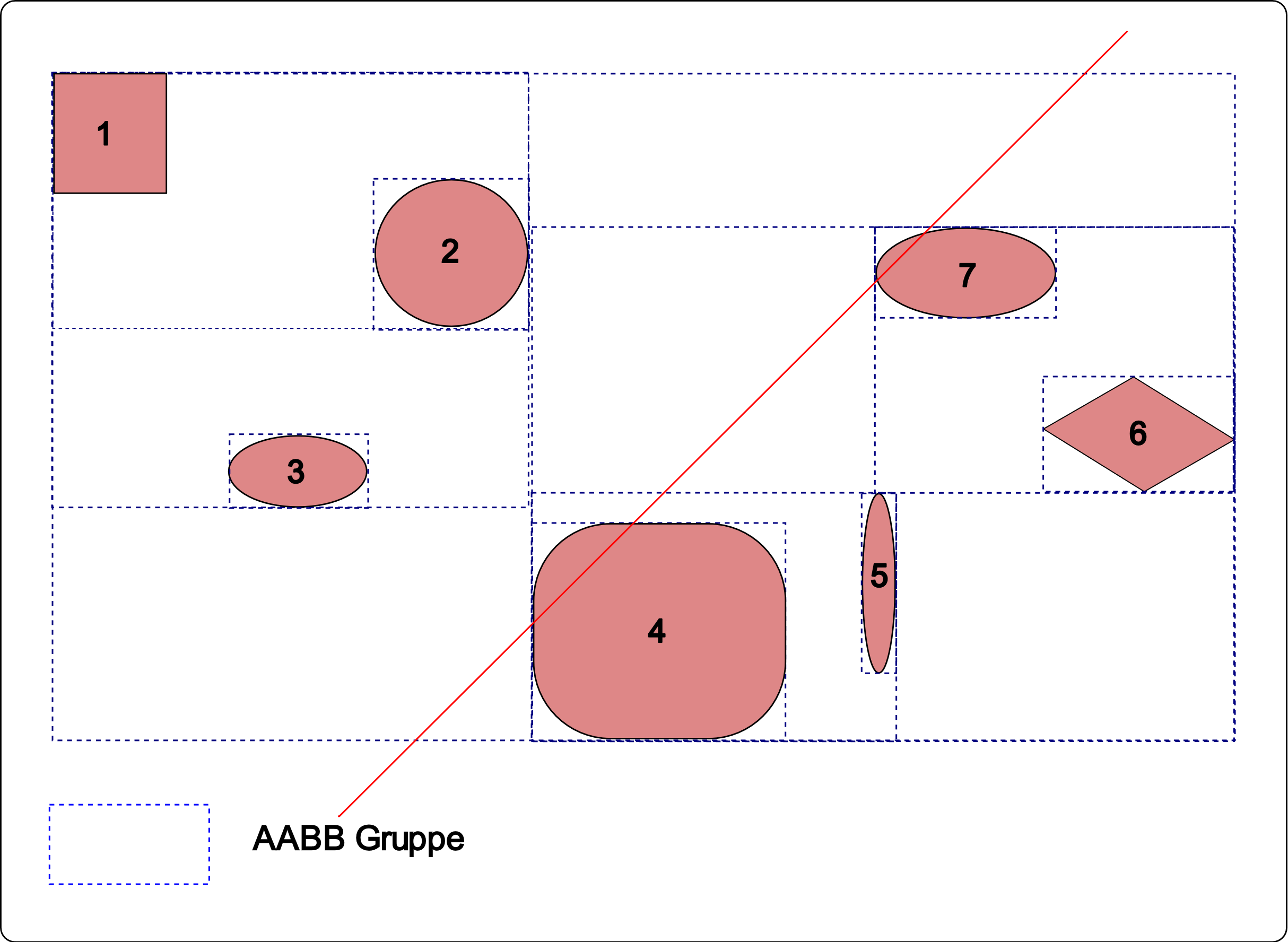
Bei dem Top Down Verfahren wird zuerst die übergreifende AABB erstellt, die alle Objekte beinhaltet. Danach wird die übergreifende Gruppe in zwei Gruppen geteilt. Für die Teilung gibt es zwei Möglichkeiten. Die AABB-Gruppe wird entweder in der Mitte aufgeteilt oder es werden die zwei AABBs mit der geringsten Oberfläche gesucht. Bei der ersten Variante kann die Unterteilung durch die einfachere Berechnung effizienter durchgeführt werden, die dadurch entstehenden Gruppen sind jedoch sehr unterschiedlich. Bei der zweiten Variante dauert das Erstellen der Bounding Volume Hierarchy länger, die entstehenden Gruppen sind jedoch bei der späteren Berechnung effizienter.

Das inkrementale Verfahren fängt mit einem einzelnen Objekt im Bounding Volume Hierarchy an und es werden die weiteren Objekte eingefügt und mit einem anderen Objekt in eine AABB zusammengefasst. Der dadurch entstehende Baum ist meistens ineffizient und kann durch einen Algorithmus verbessert werden.

Je kleiner die Flächen der AABBs sind, desto geringer ist die Wahrscheinlichkeit, dass ein Raycast das AABB trifft. Daher ist das Verfahren für die Erstellung der Bounding Volume Hierarchy sehr wichtig für die Effizienz der Berechnungen und damit für die Reaktionsgeschwindigkeit des Spieles.

#### Algorithmus zum Einfügen weiterer Objekte

Wenn ein neues Objekt in die Bounding Volume Hierarchy eingefügt werden soll, wird mit einem der anderen Objekte eine neue AABB-Gruppe gebildet, die beide Objekte enthält. Die darüberliegenden AABB-Gruppen werden so angepasst, dass sie die neue AABB-Gruppe einschliessen.

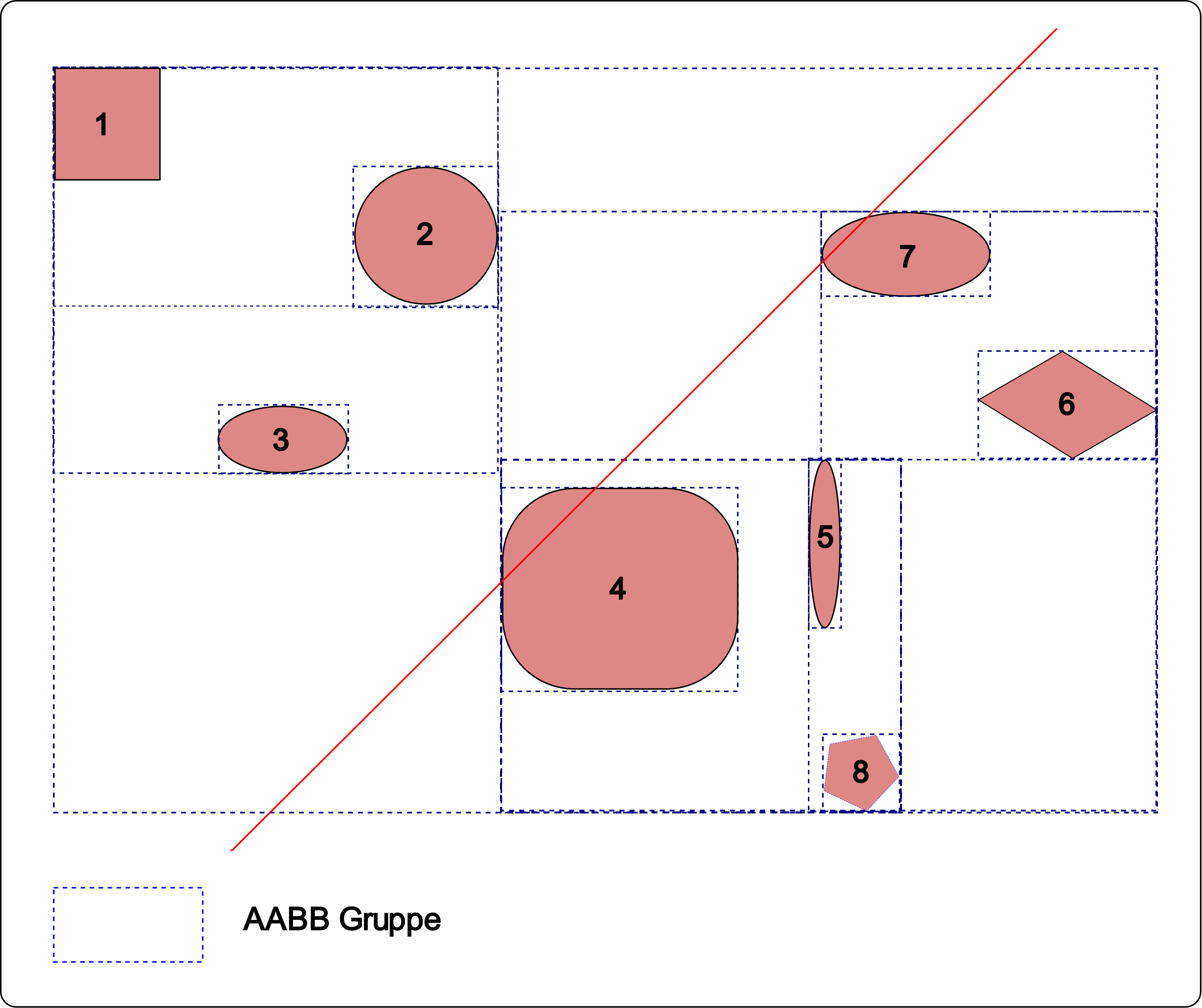
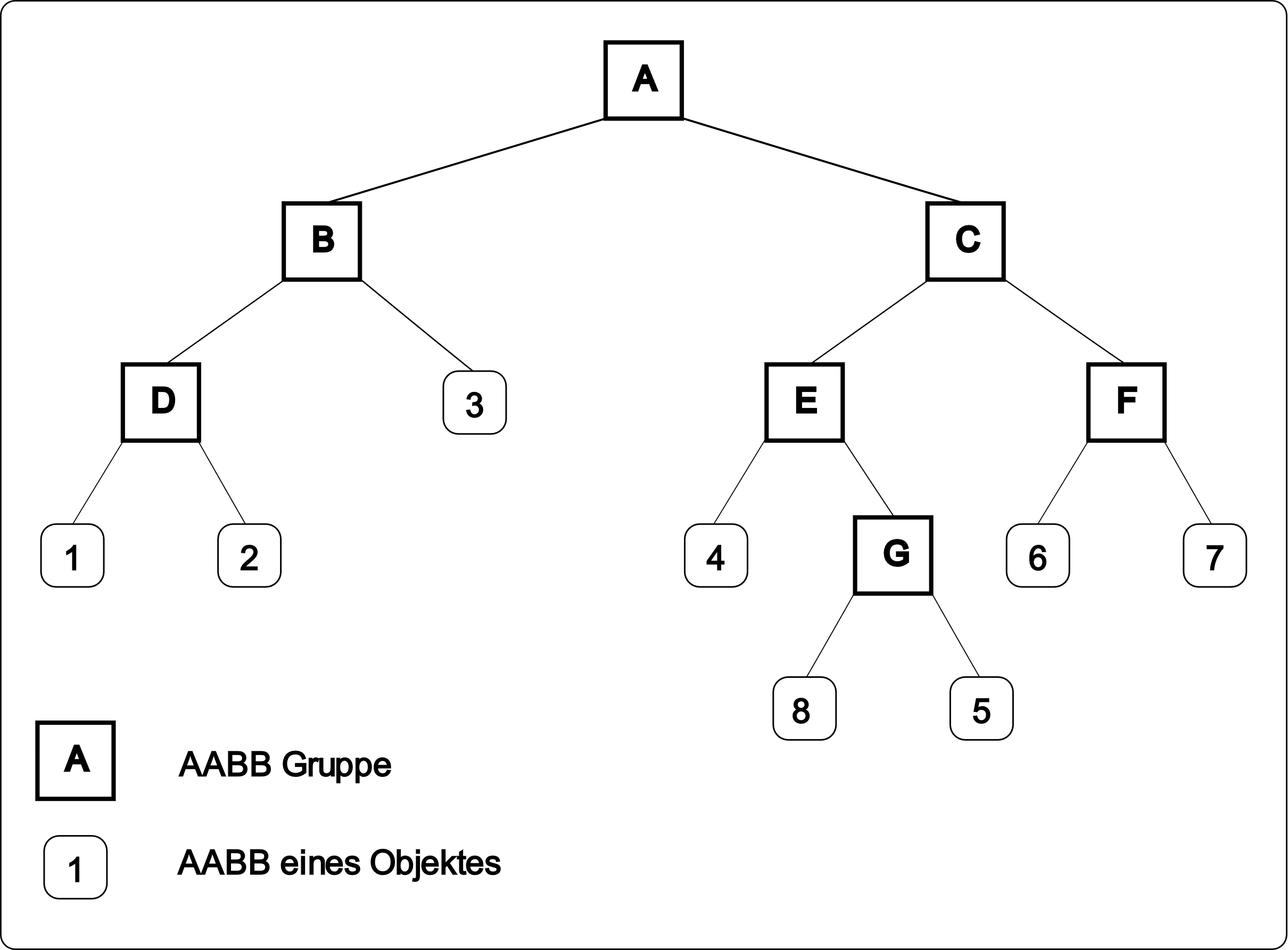
 

Abbildung : Grafik vor und nach dem Einfügen eines neuen Objektes

Für die Berechnung der besten Option für die neue AABB-Gruppe werden die Einfügekosten des neuen Objektes in den bereits vorhandenen Gruppen verglichen. Unter Einfügekosten versteht man die Summe der Zunahmen aller Flächen der betroffenen AABB-Gruppen.

Kostenfunktion einer Bounding Volume Hierarchy:

C(T) =

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Abbildung : Kostenfunktion als C++-Code

Auch hier gilt es, einen effizienten Weg für die Berechnung zu finden, ohne die Kosten aller Möglichkeiten berechnen zu müssen. Mit Hilfe der Surface Area Heuristic (SAH) wird der Vergleich pro AABB-Gruppe durchgeführt und die beste Option wird weiterverfolgt.

Die Einfügekosten setzen sich aus der Fläche der neuen AABB-Gruppe und der Summe der Flächenvergrösserung der darüber liegenden AABB-Gruppen zusammen.

Im oben genannten Beispiel würde sich folgende Formel ergeben:

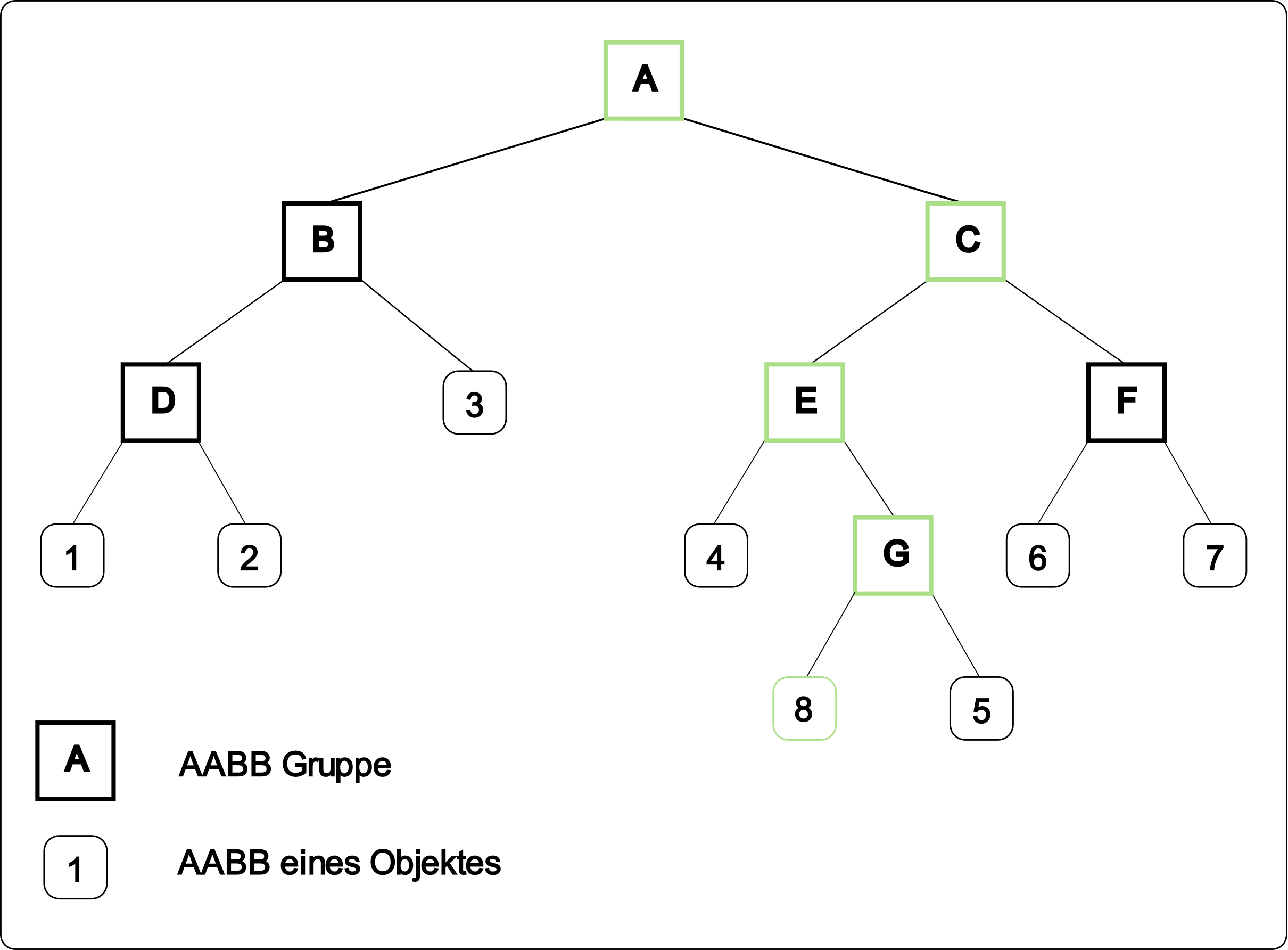


Abbildung : Graphische Darstellung der SAH Einfügekosten

# Die Entwicklung von «Das Spiel»

### Einarbeitung in Unity

### Entwicklung von Spielideen

### Einbindung von speziellen Features

# «Das Spiel»

# Nachwort

# Quellenverzeichnis