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.

Weil ich keinen richtigen Ansatz hatte, welche Engine die richtige für mich ist schloss ich zuerst aus einem bauch Gefühl alle ausser Unity und Unreal Engine aus meiner Entscheidung aus, weil ich von diesen Engines schon das meiste positive gehört hatte. In den unterschiedlichen vergleichen wurde auch klar, dass auf dem Niveau, auf welchem ich die Funktionen benötigen werde, keine einen richtigen Nachteil hatte. Deshalb entschied ich mich wieder grössten Teils nach Bauch Gefühl am Ende für Unity. Ein weiterer Punkt, welche in diese Entscheidung hineinspielte, war das ich bei meinen vorherigen versuchen die Unreal Engine zu installieren jedes Mal Probleme hatte. Jetzt im Nachhinein kann ich sagen das es eine gute Entscheidung war, weil ich den Überblick über das neue interface schnell gefunden habe und mich jetzt gut darin zurechtfinde.

Als erstes fing 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 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.

# Die Spiele-Engine Unity

Unity wurde Unity Technologies am 8. Juni 2005 veröffentlicht und ist eine Spiele-Engine und ein Framework welche viele unterschiedliche Spieleplattformen unterstützt wie zum Beispiel PCs auch Spielkonsolen, mobile Geräte und Webbrowser. [1] Sie beinhaltet einen Assetstore für Grafiken, Audio und Erweiterungen für Unity welche leicht heruntergeladen werden können. Ausserdem wird im Unity Manual jede Funktion erklärt. Unity wurde in C++ und verwendet Microsofts .NET Framework, um Anwendungen innerhalb von Unity zu programmieren wird C# verwendet. [2] Die Spiele-Engine Unity besteht aus mehreren Komponenten: den drei Engines für Grafik, Physik und Audio sowie Komponenten für Ein- und Ausgabe, Netzwerk und Ressourcenverwaltung. Auf die Physik-Engine werde ich später noch genauer eingehen.

## Frameworks

< Ein Framework ist ein Rahmenwerk für die Softwareentwicklung und Programmierung, das die Grundstruktur und das Programmiergerüst für die zu erstellende Software vorgibt. Es erleichtert den Programmierern die Arbeit und unterstützt objekt- und komponentenorientierte Entwicklungsansätze.> [3]

Im einfachsten sinn bietet ein Framework dem Entwickler der Anwendung von Anfang an Grundlegende Funktionen und Rahmenbedingungen, welche bei der Entwicklung der Software hilfreich sind, aber auch einschränken. Je nach Framework sind diese unterschiedlich und deswegen ist es wichtig sich das richtige Framework für das Projekt auszusuchen. Während das Framework den Rahmen und die regelmässig benötigten Funktionen zu Verfügung stellt damit diese nicht jedes von Grund auf an neu programmiert werden müssen, helfen diese nur bei der Entwicklung der Anwendung auf Basis dieser Grundlagen.

## 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

<TODO>schicht edge radius

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.

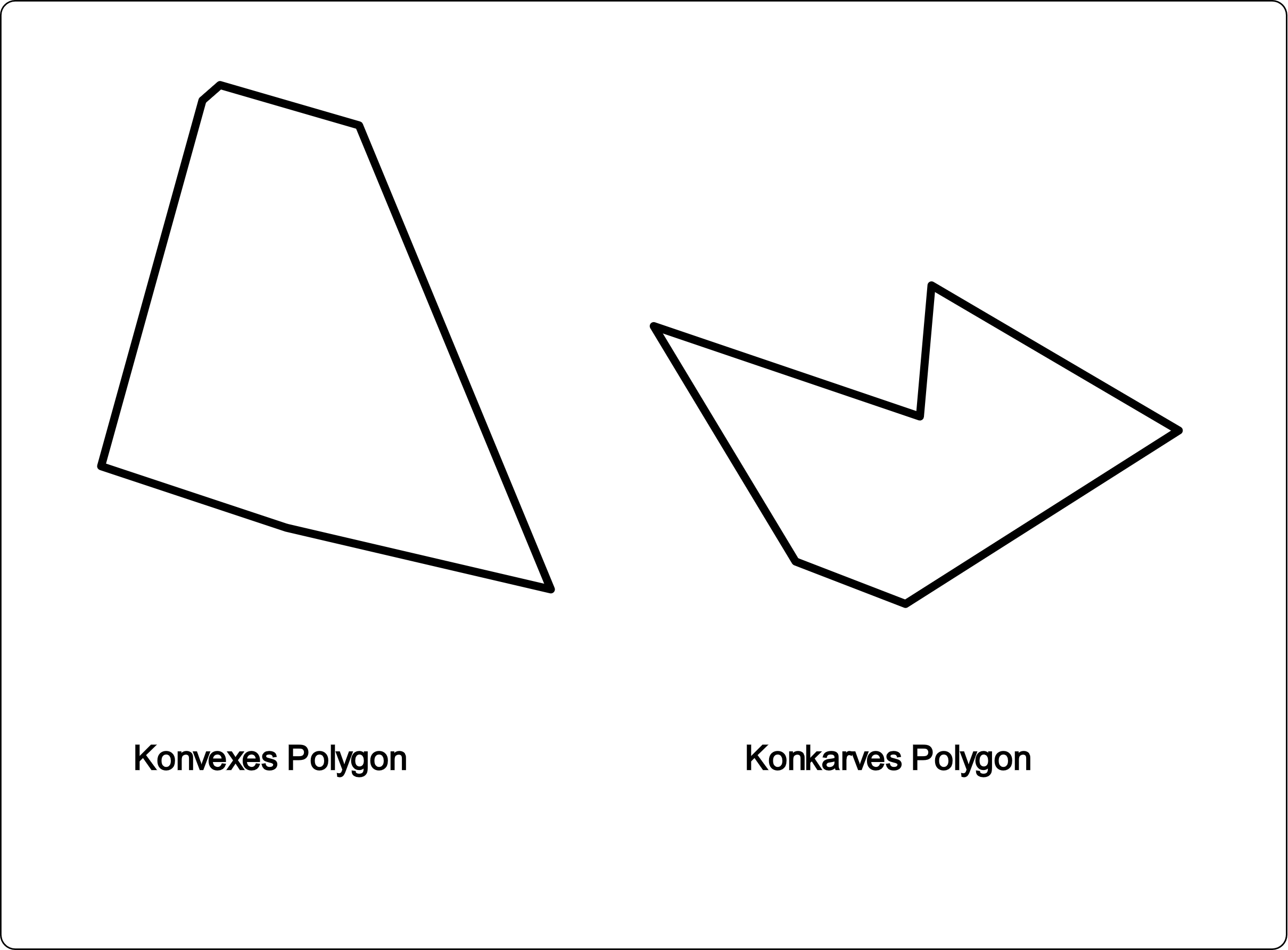


Abbildung :Grafik Polygone

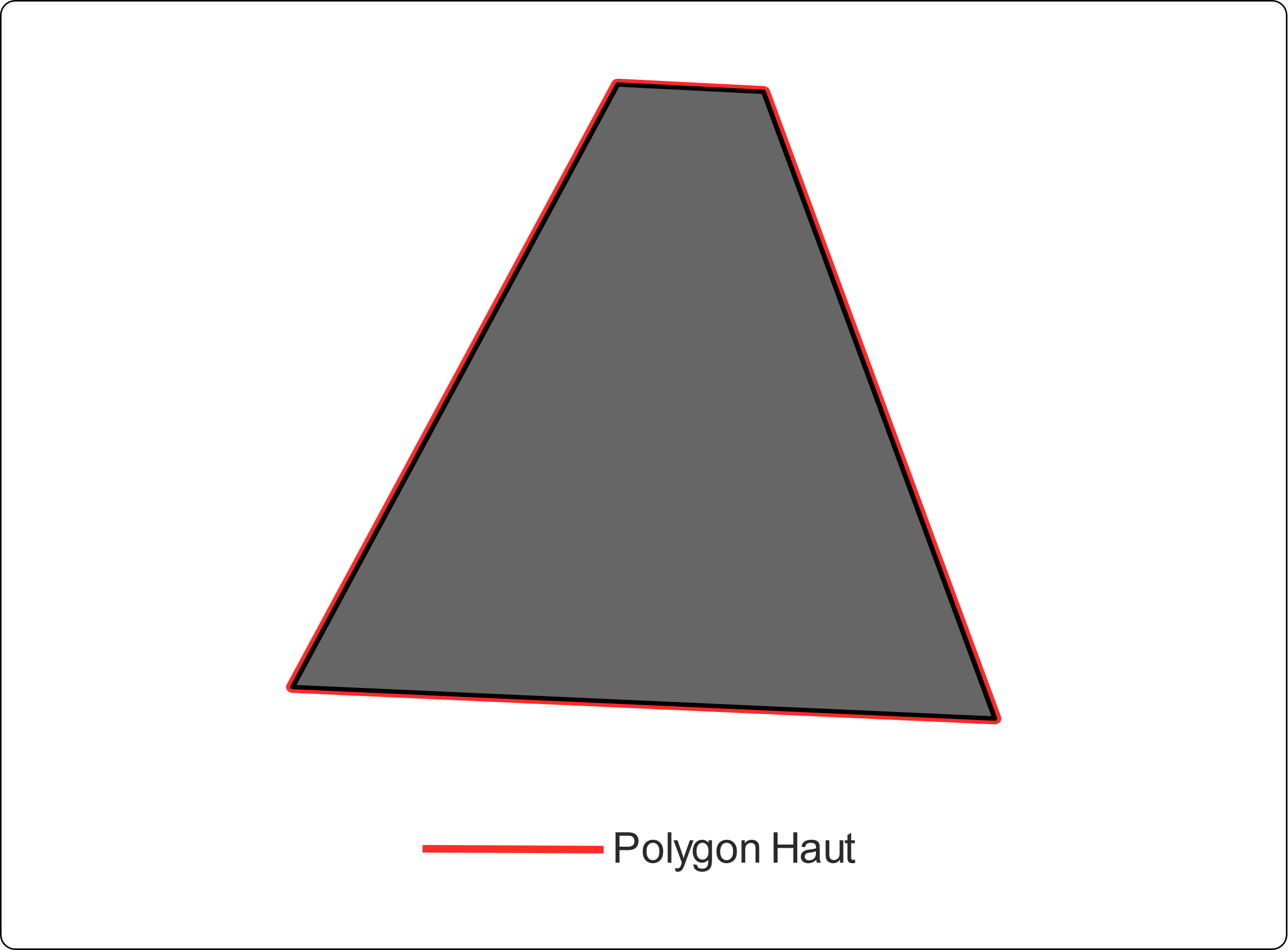


Abbildung :Polygon Haut

#### Edge Shapes

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

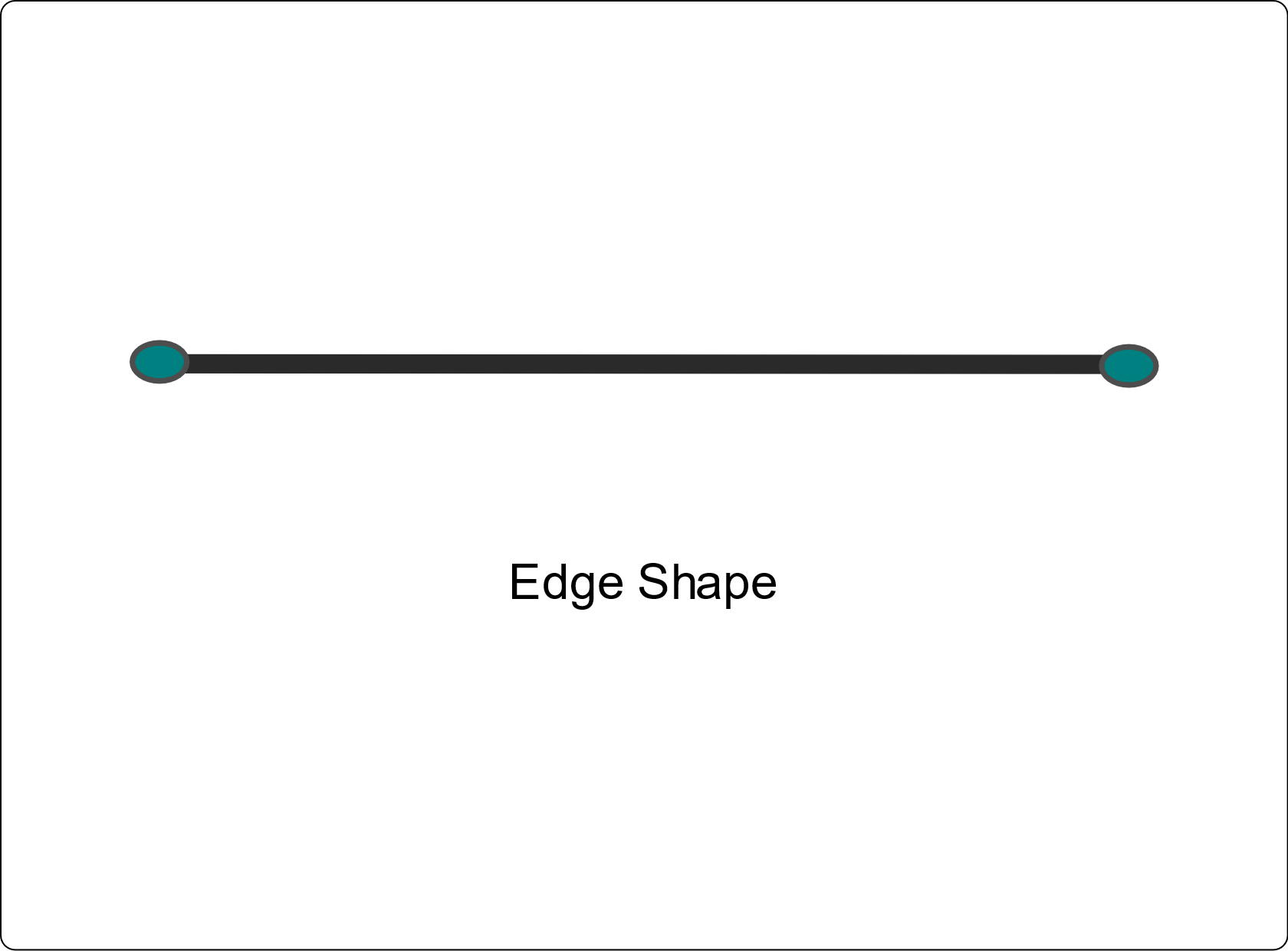


Abbildung :Edge Shape

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.

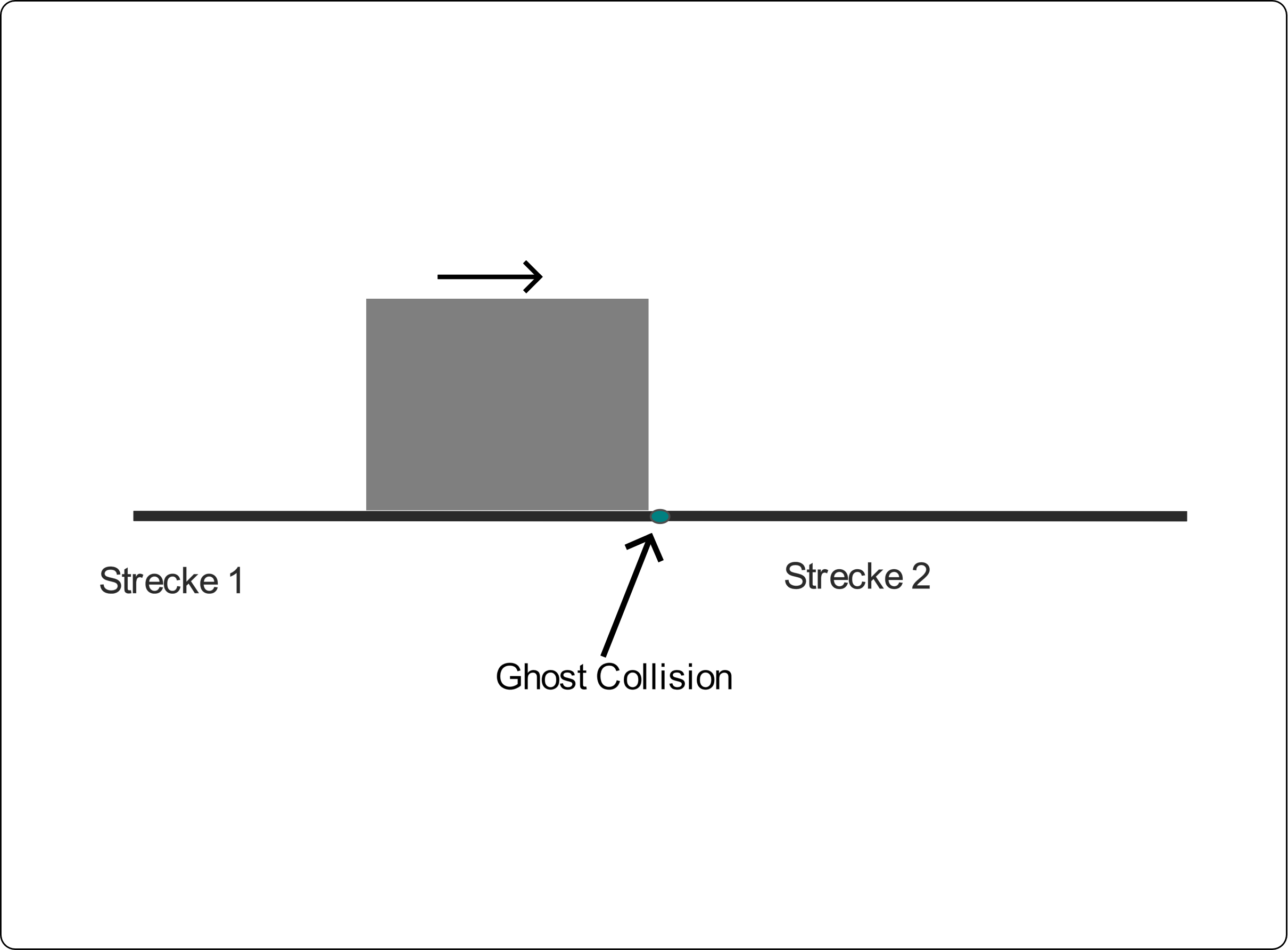


Abbildung :ghost collisions

#### 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.

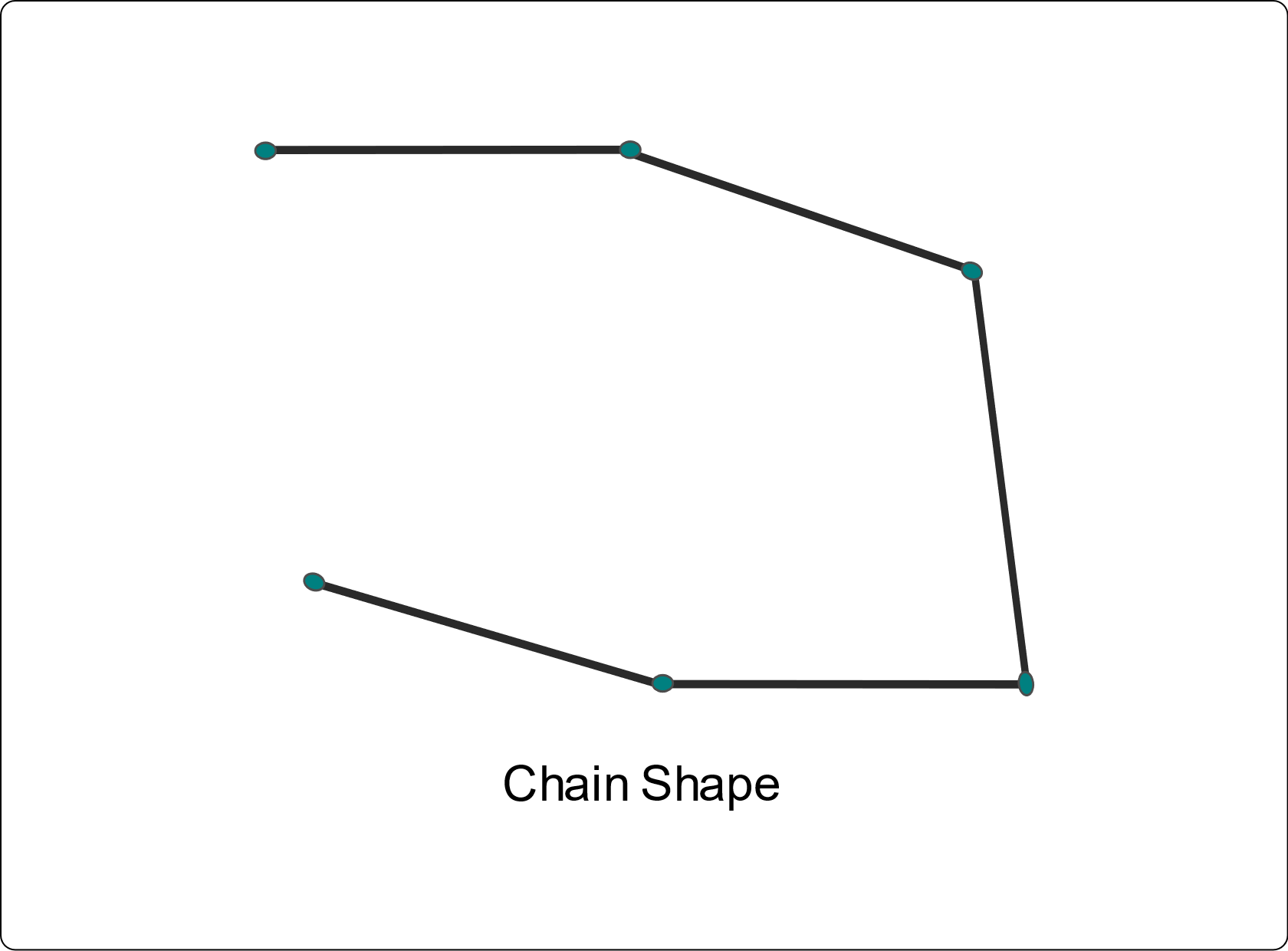
<TODO> Beispiel für Verwendung

Abbildung :Chain Shape

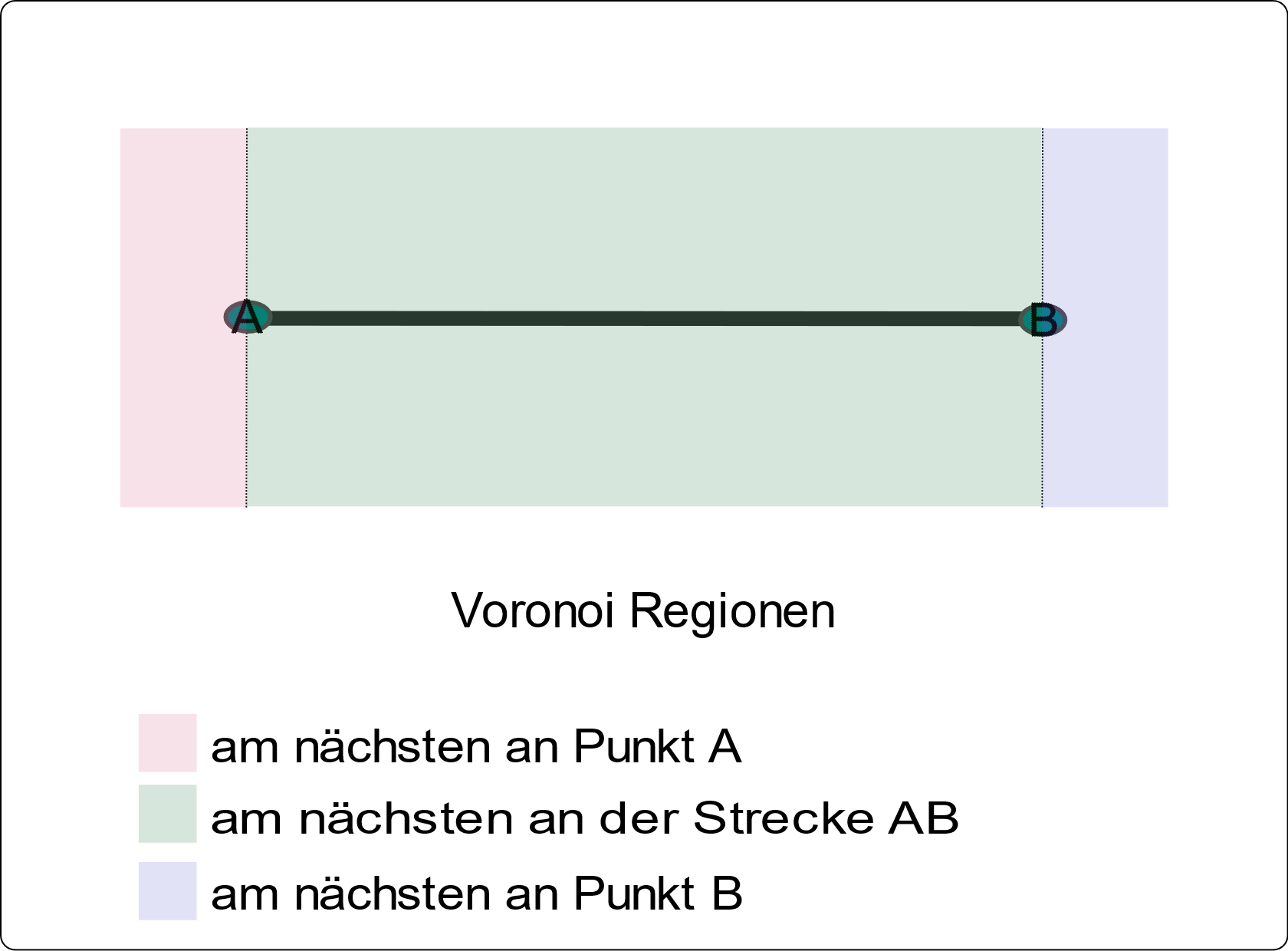
### Pairwise Functions

#### Contact manifolds

#### Distance

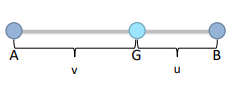
##### Voronoi regions

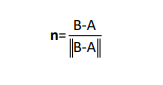
Um den Punkt auf einer Strecke zu finden, welcher am nächsten an einem anderen Punkt liegt, können Voronoi Regionen verwendet werden. Die Bereich um die Strecke wird in drei Teile eingeteilt je nach dem welchem Teil der Strecke ein Punkt in diesem Gebiet am nächsten ist.

<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

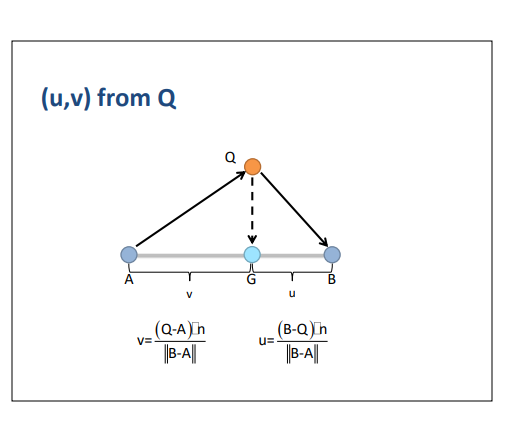


Danach kann man mit dem Vektorprodukt u und v berechnen

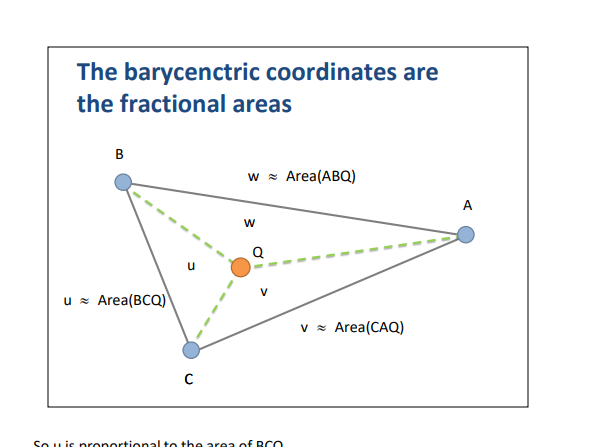
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

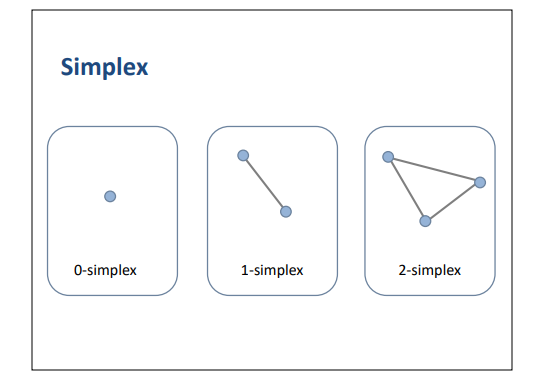
Weil sich das Vektorprodukt nicht verändert, wenn anstatt dem punkt auf der gerade der Datenabfragepunkt (Query point) verwendet wird kann auch direkt mit dem Datenabfragepunkt gerechnet werden.



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.

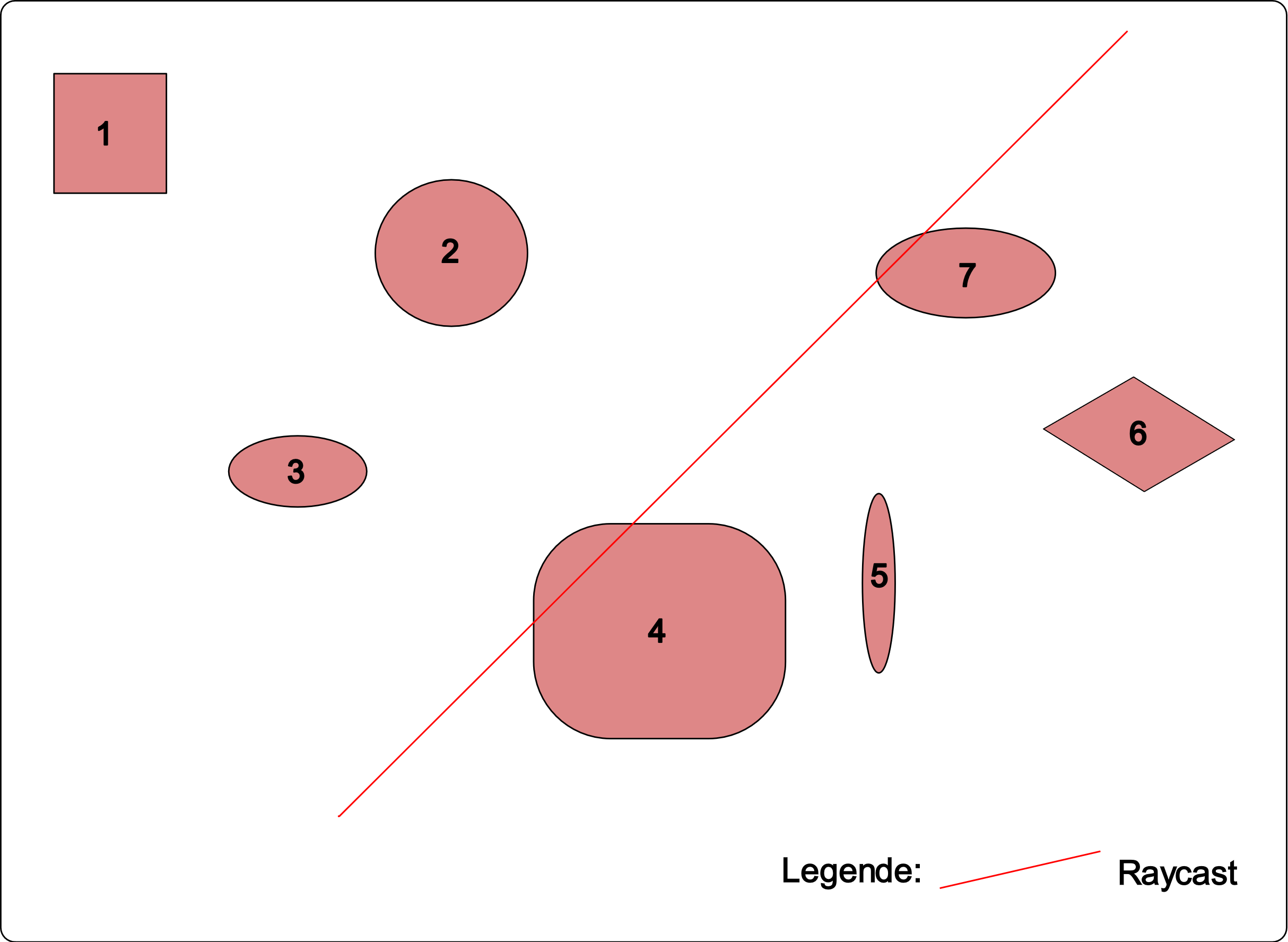


Abbildung : Grafik mit Objekten und einem Raycast

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.

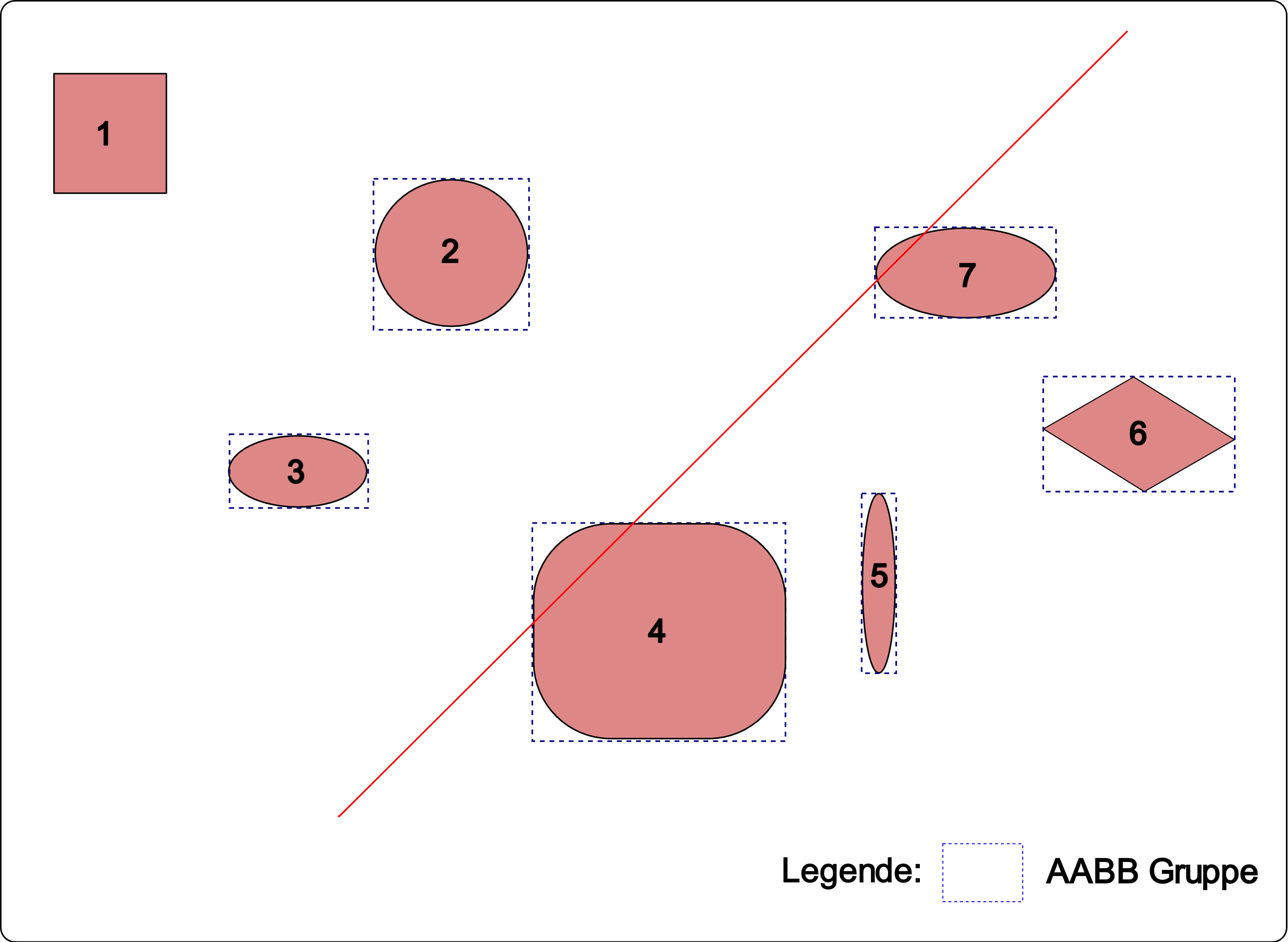


Abbildung : Graphik mit Objekten, deren AABBs und einem Raycast

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.

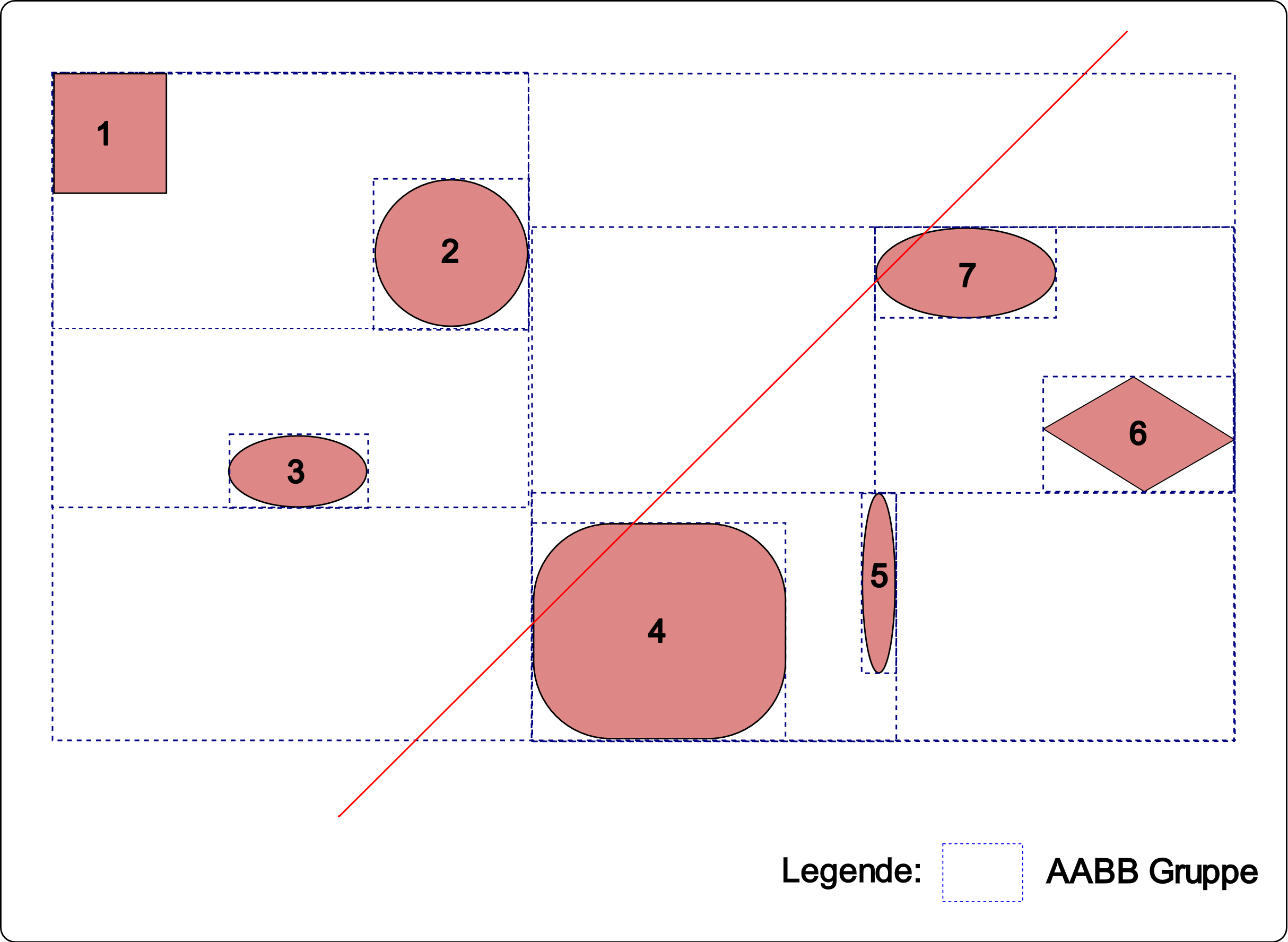


Abbildung : Grafik der AABB-Gruppen

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.

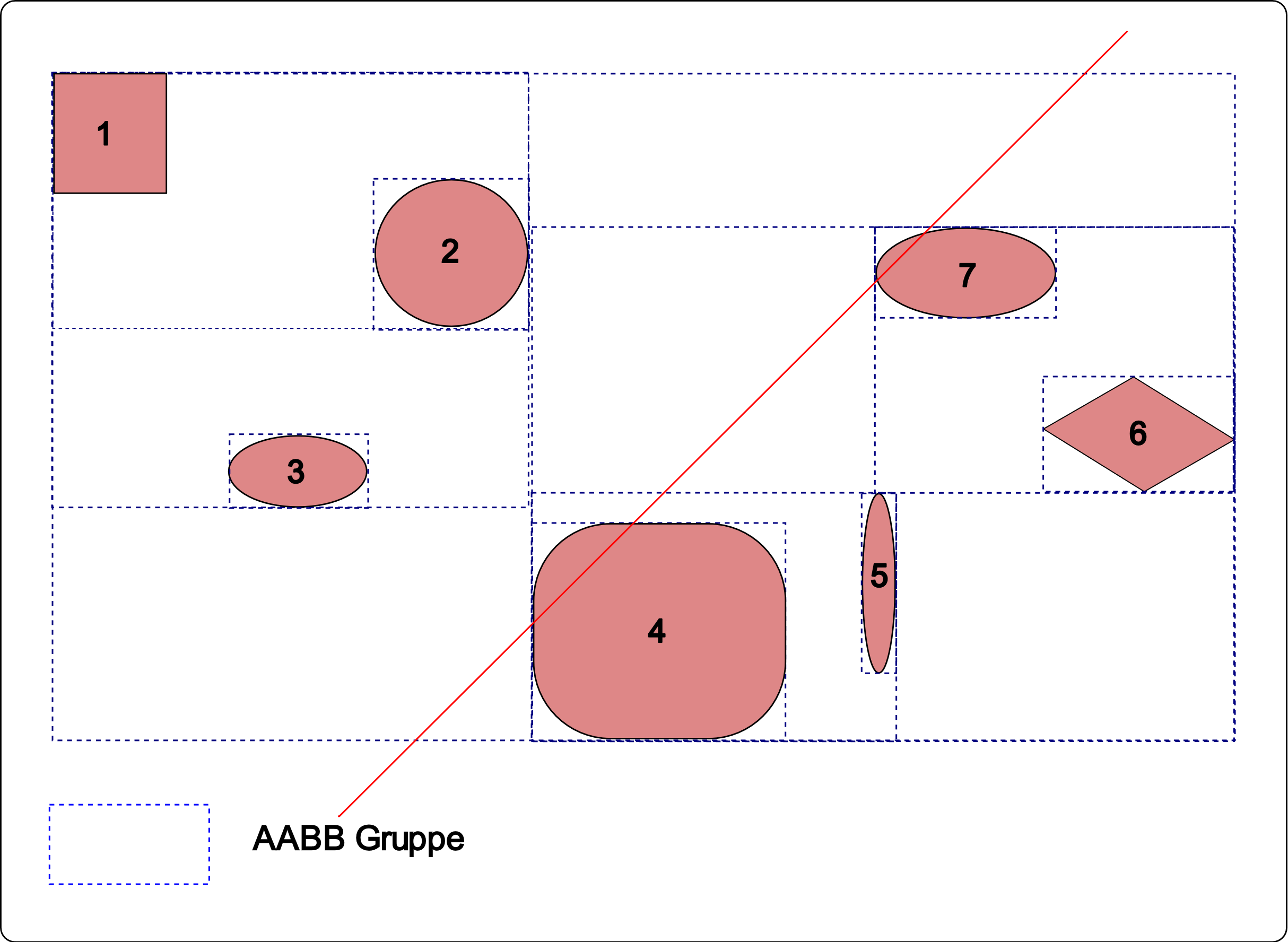
Bei dem Top Down Verfahren wird zuerst die übergreifende AABB erstellt, die alle Objekte beinhaltet. Danach wird die übergreifende Gruppe in zwei Gruppen geteilt. Für die Teilung gibt es zwei Möglichkeiten. Die AABB-Gruppe wird entweder in der Mitte aufgeteilt oder es werden die zwei AABBs mit der geringsten 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.

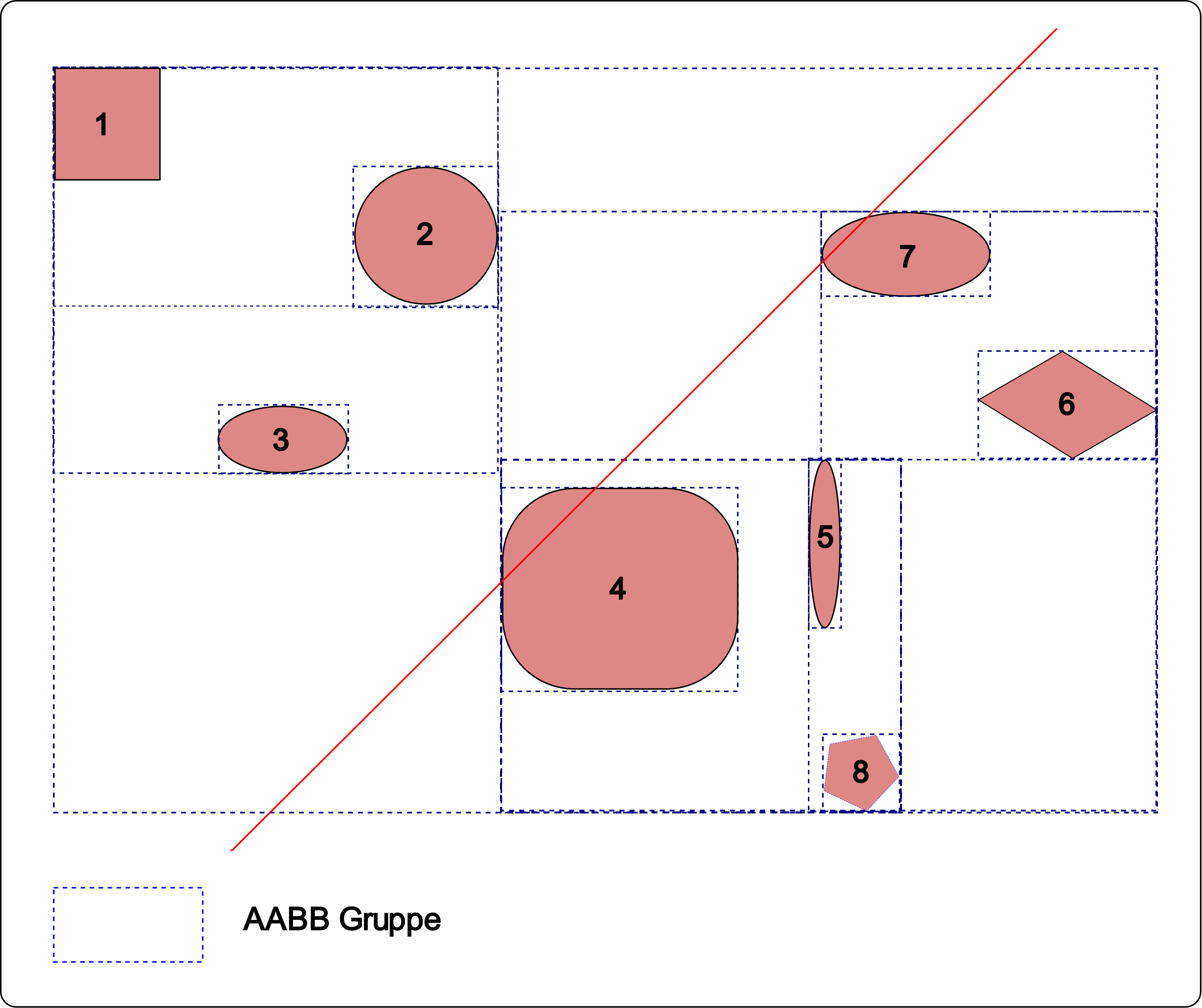
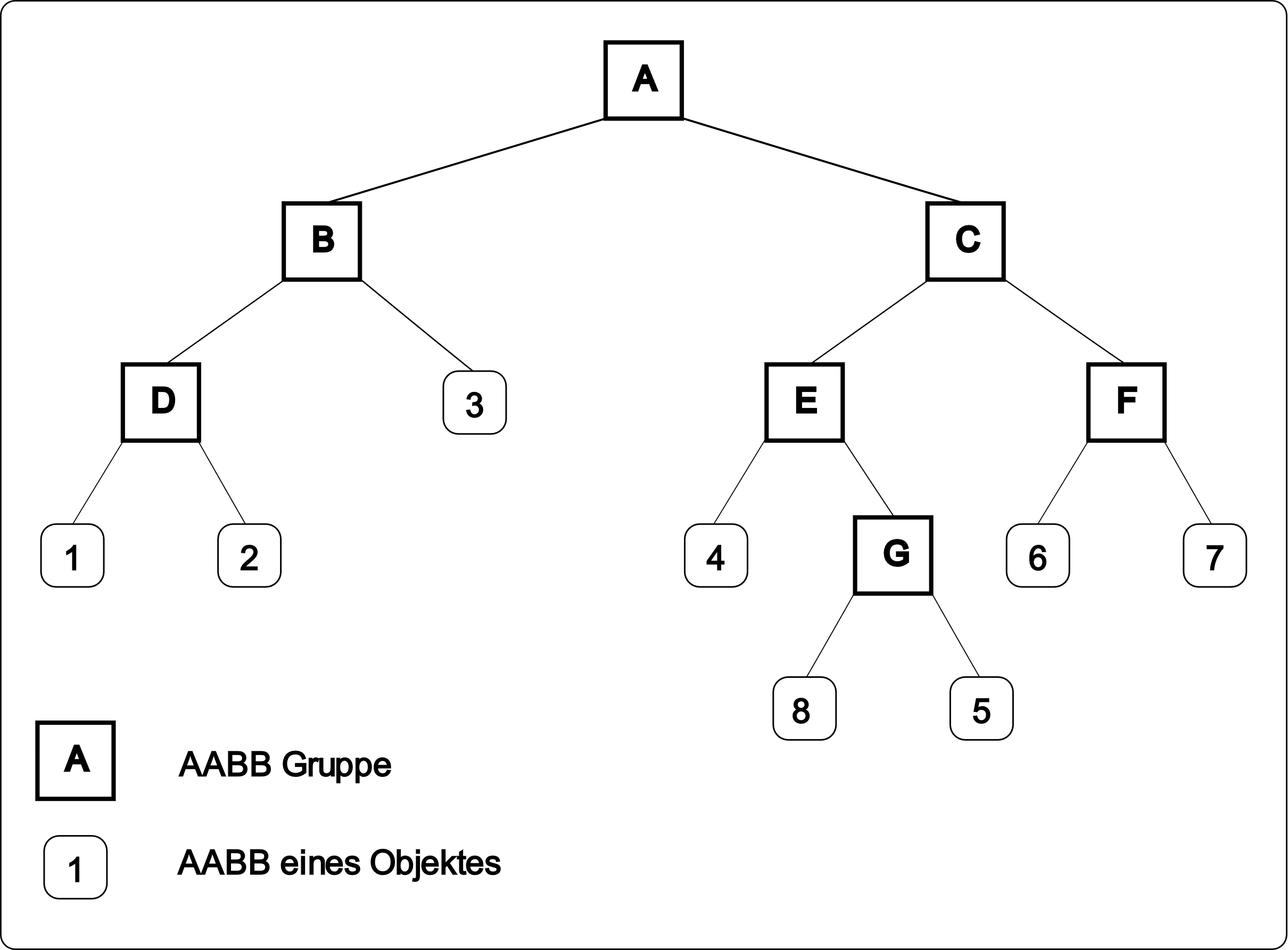
 

Abbildung : Grafik vor und nach dem Einfügen eines neuen Objektes

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:

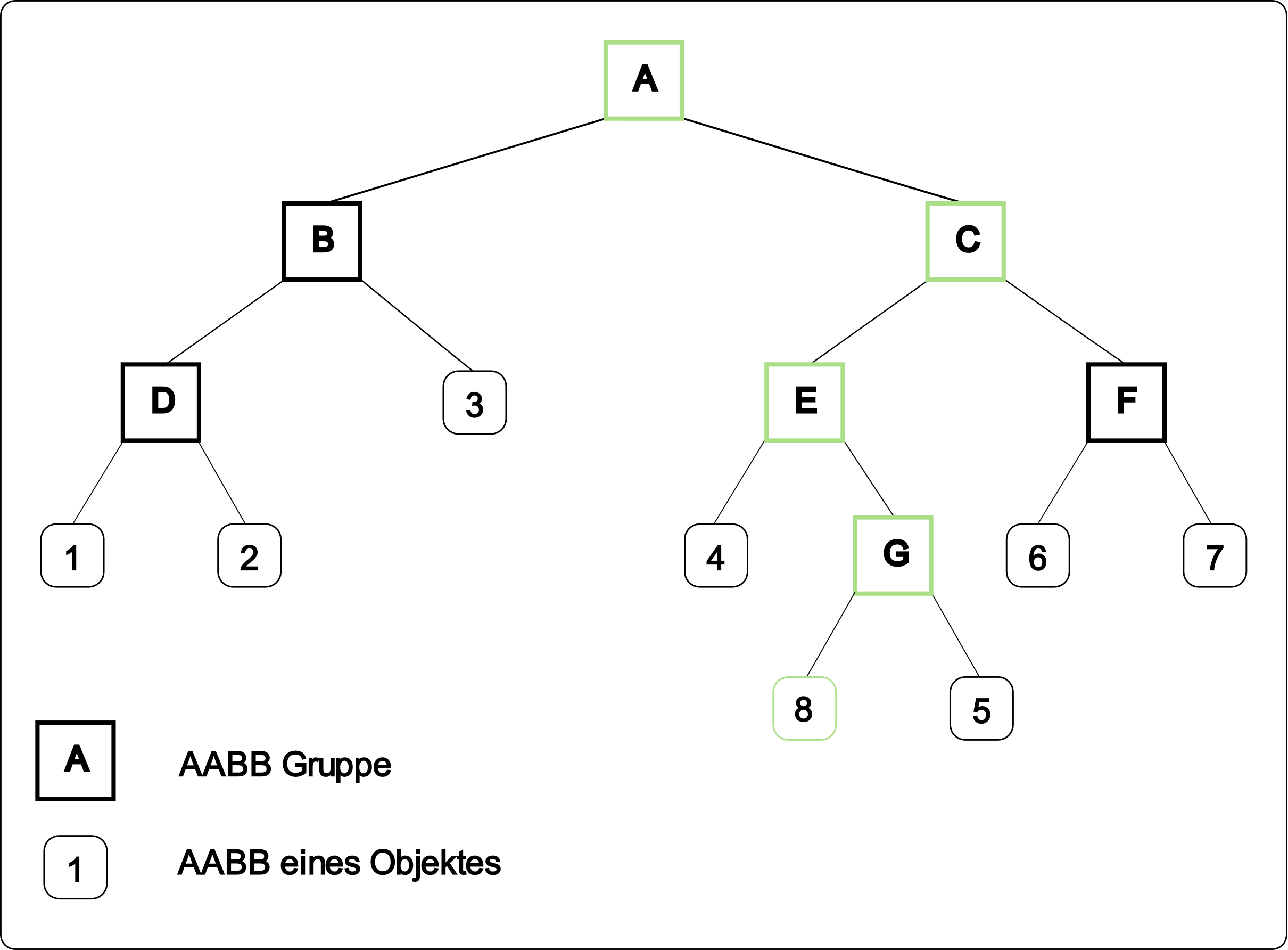


Abbildung : Graphische Darstellung der SAH Einfügekosten

# Die Entwicklung von «Das Spiel»

### Einarbeitung in Unity

* Basics von c#lernen
* Noch keine Erfahrung mit unity und c#
* Als aller erstes unity und vs2022 installieren
* Versuchen ersten überblick über ui zu bekommen
* Weil die Befehle innerhalb von unity unitz spezifisch sind zuerst mit tutorial mit c# in unity zurecht zu finden
* Gutes tutorial suchen in welchem jeder Schritt erklärt wird und kein vorgeschriebener Code verwendet werden soll
* Weil unity und vs intellisense nicht synchronisieren wurde das bis icb gemerkt habe das die Funktion existiert sehr erschwert
* Grafiken suchen
* Mit der zeit famd ich mich besser zurecht musste aber für neue sachen wie ui, Einstellungen, playerprefs und Speichen in binärdateien Tutorials folgen
* Surch das zusammensetzen unterschiedlicher Tutorials mit meinem selbst geschrieben code entstanden fehler dadurch ein grosser teil des Codes zumindest teilweise verändert
* Diese veränderungen entstanden aucb bei erweiterung mit neuen Funktionen welche Verbindungen mit den vorherigen haben
* Durch das unity manual fand ich was ich suchte nach ungefähr der Hälfte der zeit auch alleine was aber viel länder dauert.um den richtigen befehl und die richtige Verwendung herauszufinden.
* GitHub

### Entwicklung von Spielideen

* Viele ideen
* Die meisten aber sehr unrealistisch
* Deshalb auf platformer festgelegt welchen ich zu Rouge like ausbauen kann falls Ich komplizierte Elemente benötige
* Vllt Erklärung Rouge like , platformer
* Danach habe ich mit den grundlegenden spielerbewegungen angefangen und diese erweitert( springen, wände, wandsprünge, Doppelsprung, dash)
* Und auch andere mechaniken und funktionen( Hauptmenü mit Einstellungen, speichern und laden, neue Hindernisse, Checkpoints)
* Diese Funktionen habe ich hinter collectibles verschlossen bis der spieler den richtigen Punkt erreicht hat.

### Einbindung von speziellen Features

* Bei dee Einbindung von ton wie auch grafiken war die suche nach den passenen dateien das schwierigste
* Durch autotiling erweiterung wurde das erstellen der karte angenehmer
* Grafiken waren nachdem man den Kniff dahinter gefumden hat erstaunlich einfach
* Unitz übernimmt dabei auch einiges an arbeit
* Bei der conteller einbindung musste ich mich zuerst zwischen zwei Erweiterungen entscheiden wobei die neuere komplizierter aber auch besser ist, leider hatte diese weniger Anleitungenen weshalb icb anfangs ein bisschen herumprobieren musste aber schlussendlich hat ea funktioniert weshalb teile des codes verändern musste damach gab es noch ein Problem das über den Controller werte von 0-1 und auf der Tastatur nur entweder das eine oder das amdere was obwohl es sehr einfach war Probleme auslöste
* Audio

# «Das Spiel»

Das ziel des Spieles ist es mit unterschiedlichen Bewegung Mechaniken wie normalem laufen, Sprüngen, wand Sprüngen, Doppelsprüngen und dashes die Hindernisse des Spielkarte zu überwinden und das ende des Spieles zu erreichen. Während die einfachsten Hürden, Sprünge von Plattform zu Plattform sind, werden diese im verlauf des Spieles durch Plattformen, welche bei kontakt herunterfallen oder Stalaktiten welche versuchen den Spieler zu erschlagen erschwert.

<TODO>Grafiken aus Spiel

Die meisten Fähigkeiten des Spielers sind Anfangs noch nicht nutzbar sie werden aber im verlauf des spiels durch das aufsammeln von objekten freigeschaltet.

# Nachwort

# Quellenverzeichnis

|  |  |
| --- | --- |
| [1] | Unity Technologies, «Unity,» 2022. [Online]. Available: https://unity.com/. |
| [2] | Unity Technologies, «Unity Documentation,» 2 12 2022. [Online]. Available: https://docs.unity3d.com/Manual/unity-architecture.html. |

Probeme

Kein prototyp weshalb ich ohne konkretere idee der mechaniken angefangen habe das spiel zu programmieren

Als ich dann angefangen habe gezielter über die Mechaniken nachzudenken wurde mir schnell klar das ich das die meissten meiner ideen sehr stark vor anderen Spielen der gleichen art inspiriert waren

Angangs setzte ich meien fokus asuch zu stark auf die Grafik was in zwar hilfreich war weil das spiel endlich nach einem spiel aussah und nicht mehr nach schwarzen und grauen rechtecken. Das war der gefühlt erste richtige fortschritt zu einem spiel aber dadurch wurde ich wiederum vun den mechaniken abgelenkt.

Der scwerste teil der ganzen Maturaarbeit war nicht wie erwartet der Informatik Anteil sondern das Gamedesign. Während ich immer noch die längste Zeit damit verbrachte die neuen Mechaniken zu programmieren, hatte ich in der zwischen zeit keine neuen Ideen welche für mein programmierkönnen und meine grafischen Möglichkeiten realistisch waren.

Gleichzeitig habe ich das gefühl das ich in diesem jahr zwar sehr viel über Unity und die funktionen verstanden habe aber trotzdem nicht viel mehr als die basics gelernt habe. Als komplletter Anfänger in c#,unity und im Gamedesign hatte die anfängliche fehleischätzung der schwierigkeiten doch starkere folgen gleichzeiitig verbringe ich auch wenn ich schon eine idde für einen raum habe und alle mechaniken schon programmiert sind immernoch eine stunde bis alles funktionniert wie ich es geplant habe, je nach raum sogar länger.

Lieblingsfehler ; oder 2d vergessen