Maximilian Kolbe Gymnasium 2021/-22

Kollisionserkennung: Separating Axis Theorem

Facharbeit im Fach Mathe (LK)

bei Frau Blasius

Florian Hirche

Stufe: Q1

E-Mail: flo.hirche@gmx.de

Inhaltsverzeichnis

EINLEITUNG

In der Spieleprogrammierung kommt man schnell zum Thema der Kollisionserkennung. Fast jede Aktion hat eine x berührt y Bedingung, um ausgeführt zu werden. Bei einfachen Formen ist diese Abfrage noch sehr einfach: z.B. bei zwei Vierecken, deren Seiten parallel zur x- und y-Achse sind, muss man nur die x- und y-Werte der Eckpunkte miteinander vergleichen.

Wenn jetzt aber das Viereck gedreht wird, sodass die Seiten nicht mehr parallel zu den Koordinatenachsen sind, dann kann man nicht mehr einfach die Koordinaten vergleichen.

Ziel dieser Facharbeit ist es herauszufinden, ob zwei Körper kollidieren. Um dieses Ziel zu erreichen, möchte ich zeigen, wie man dieses Problem mithilfe des „**Separating Axis Theorem**“ (SAT) lösen kann.

Zu Beginn werde ich die Funktionsweise des Algorithmus anhand von Bildern erläutern. Danach gehe ich auf die mathematischen Formeln hinter dem Algorithmus ein. Im Anschluss wird ein Problem des Algorithmus vorgestellt und eine passende Lösung dazu erläutert.

Als Letztes zeige ich meine Implementation des Algorithmus in der Programmiersprache „Typescript“. Das Ergebnis ist unter http://florian.hirche.eu/website/public/school/Collision2d/index.html zu erreichen. (Falls ich den Link ändere, ist http://florian.hirche.eu die Hauptseite)

1. FUNKTIONSWEISE

Um zu überprüfen, ob zwei Körper miteinander kollidieren, versucht der SAT Algorithmus eine gerade Linie zwischen den beiden Körpern zu finden. Wenn nach der Überprüfung keine Gerade zwischen den beiden Körper gefunden wurde, heißt das, dass die Körper miteinander kollidieren. [1]

[1] Vgl. (Separating Axis Theorem kein Datum)

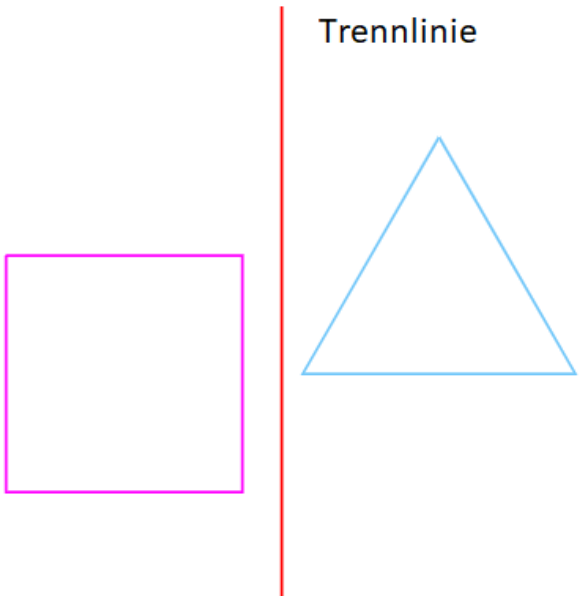

ABBILDUNG 1[2]

Um eine Trennline zu bilden, wird zu Beginn eine Seite zur Überprüfung ausgewählt. Rechtwinklig zu dieser Seite wird eine Gerade erstellt. Dabei ist es egal, wo sich diese Gerade im Raum befindet. [3]

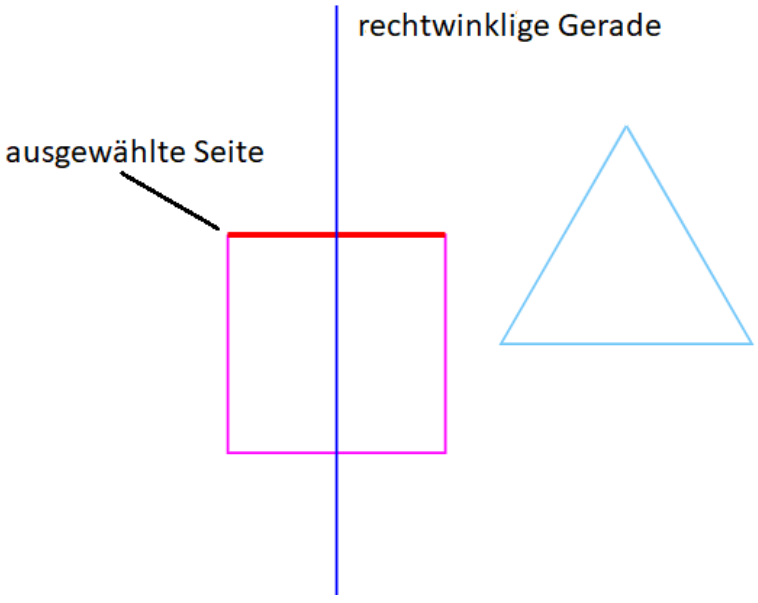

ABBILDUNG 2[4]

[2] Eigene Darstellung
[3] Vgl. (SAT (Separating Axis Theorem) 2010)
[4] Eigene Darstellung

Nun werden alle Eckpunkte der beiden Körper auf diese Gerade projektiert. Das heißt: Die Punkte werden so auf die Gerade verschoben, dass die „Verschiebungslinie“ zwischen der alten und der neuen Position des Punktes im rechten Winkel zur vorher erstellten Geraden steht. [5]

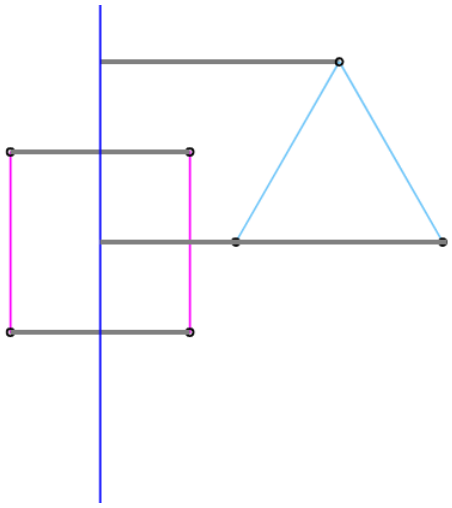

ABBILDUNG 3[6]

Jetzt befinden sich alle Punkte auf einer Linie. Man verbindet die Punkte eines Körpers, die am weitesten auseinanderliegen und ignoriert den Rest. Es entstehen zwei Linien.[7]

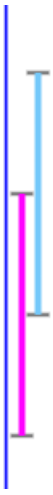

ABBILDUNG 4[8]

[5] Vgl. (Chong 2012)
[6] Eigene Darstellung
[7] Vgl. (Huynh 2008)
[8] Eigene Darstellung

Oben im Bild sieht man, dass die Linien übereinander liegen, es gibt also keine Lücke zwischen beiden Körpern entlang der überprüften Achse. Dies heißt aber noch nicht, dass die beiden Körper überlappen, sondern nur, dass die Körper keine „Trennlinie“ parallel zur zu überprüfenden Achse haben.[9]

Wenn aber eine Lücke zwischen den beiden Linien zu erkennen ist, heißt das, dass die Körper eine „Trennlinie“ parallel zur überprüften Achse haben und folglich auch nicht kollidieren. Man kann ab diesem Punkt die Überprüfung abbrechen, da die „Separating Axis“ gefunden wurde.[10]

ABBILDUNG 5[11]

Um eine „Trennlinie“ zu finden, müssen also maximal so viele Überprüfungen durchgeführt werden, wie die Anzahl aller Seiten beider Körper zusammen. Wurde bis dahin keine gefunden, berühren sich die beiden Körper.

[9] Vgl. (javidx9 2019)
[10] Vgl. (Pikuma 2021)
[11] Eigene Darstellung

2. MATHEMATIK

2.1. NORMALVEKTOR

Um eine rechtwinklige Gerade zu einer Seite zu finden, muss man zuerst den Vektor zwischen beiden Punkten der Seite kennen. Dann kann man nämlich einen Vektor rechtwinklig zu diesem Vektor bilden (siehe unten), dieser wird „Normalvektor“ genannt. Wenn man jetzt diesen Vektor unendlich weit verlängert, entsteht die gesuchte Gerade.

Um den Vektor zwischen zwei Punkten zu finden, muss man die Koordinaten der beiden Punkte kennen.

Der Vektor von Punkt A zu Punkt B kann wie folgt bestimmt werden:

$$\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA}$$

Der Ortsvektor eines Punktes P(x, y) ist:

$$\overrightarrow{OP} = \begin{pmatrix} x \\ y \end{pmatrix}$$

Nach einsetzen in die Formel ergibt sich für die Strecke $\overline{AB}$ der Vektor $\overrightarrow{AB}$:

$$\overrightarrow{\mathrm{AB}} = \begin{pmatrix} \mathrm{x_B} \\ \mathrm{y_B} \end{pmatrix} - \begin{pmatrix} \mathrm{x_A} \\ \mathrm{y_A} \end{pmatrix} \Rightarrow \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

Jetzt kann man den Normalvektor zu diesem Vektor bilden und erhält den rechtwinkligen Vektor, der später die Gerade angibt. Den Normalvektor bildet man, indem man die Koordinaten des Vektors vertauscht und bei einem der beiden Werten das Vorzeichen wechselt:

$$\vec{n} = \begin{pmatrix} -\Delta y \\ \Delta x \end{pmatrix}$$

oder:

$$\vec{n} = \begin{pmatrix} \Delta y \\ -\Delta x \end{pmatrix}$$

Nun hat man den Vektor, durch den die gesuchte Gerade angegeben wird. Als nächstes müssen alle Eckpunkte beider Körper auf die Gerade projektiert werden.

2.2. PROJEKTION MIT SKALARPRODUKT

Die Projektion von Punkten auf eine Gerade kann durch das „**Skalarprodukt**“ (*eng. Dot Product*) erreicht werden.

Das **Skalarprodukt** ist eine Form der Multiplikation von Vektoren. Man bildet es, indem alle Werte des ersten Vektors mit jeweils dem Wert an der gleichen Stelle des zweiten Vektors multipliziert werden. Durch das Aufaddieren aller Produkte erhält man dann das Skalarprodukt.[12]

$$\vec{a} \cdot \vec{b} = x_1 \cdot x_2 + y_1 \cdot y_2$$

Das Ergebnis dieser Multiplikation von Vektoren liefert keinen Vektor als Ergebnis, sondern einen Wert, einen **Skalar**. Dieser Wert ist gleich dem Produkt aus der Länge von $\vec{b}$ und der Länge $\vec{a}_{\vec{b}}$, wobei $\vec{a}_{\vec{b}}$ die Komponente von $\vec{a}$ ist, welche in die Richtung von $\vec{b}$ zeigt.[13]

$$\vec{a} \cdot \vec{b} = \left|\vec{a}_{\vec{b}}\right| \cdot \left|\vec{b}\right|$$

Um nun auf die Länge von der Komponente $\vec{a}_{\vec{b}}$ zu kommen, muss man das Skalarprodukt also nur durch die Länge von $\vec{b}$ teilen:

$$|\vec{a}_{\vec{b}}| = \frac{\vec{a} \cdot \vec{b}}{\left|\vec{b}\right|}$$

Unsere Punkte werden durch ihre Ortsvektoren angegeben. Da dieser im Ursprung startet und beim Punkt endet, ist die Länge $\left|\vec{a}_{\vec{b}}\right|$ die Länge der Strecke

[12] Vgl. (Mathematrix kein Datum)
[13] Vgl. (Cuemath kein Datum)

zwischen dem projektierten Ursprung und dem projektierten Punkt. Bei dem Skalarprodukt kann aber auch ein negatives Ergebnis entstehen.

Als Beispiel mit den Vektoren $\vec{a} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$ und $\vec{b} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$:

$$\left|\vec{a}_{\vec{b}}\right| = \frac{\begin{pmatrix} 2 \\ -3 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 3 \end{pmatrix}}{\left|\begin{pmatrix} 1 \\ 3 \end{pmatrix}\right|} = \frac{2 \cdot 1 + (-3) \cdot 3}{\sqrt{1^2 + 3^2}} = \frac{-7}{\sqrt{10}} \approx -2{,}22$$

Eine Länge kann aber nicht negativ sein. In diesem Fall bedeutet das Minus, dass der projektierte Punkt auf der anderen Seite vom Startpunkt aus liegt als der Vektor $\vec{b}$.

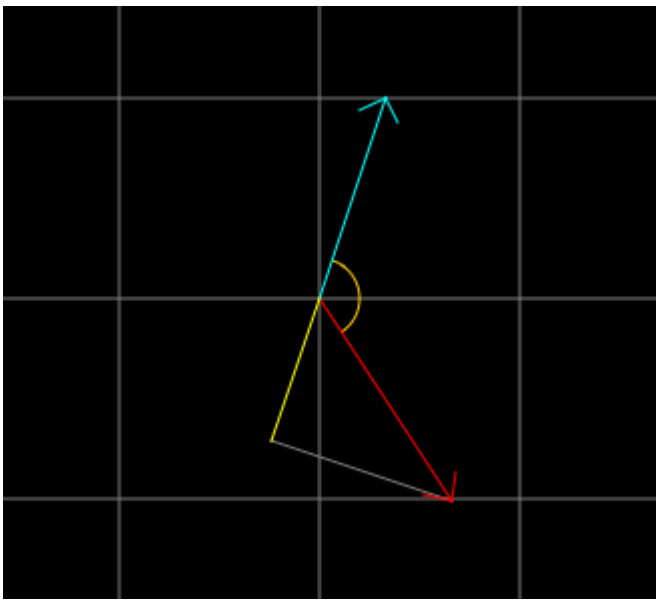

ABBILDUNG 6[14]

Im Bild sieht man die oben ausgerechnete Projektion graphisch dargestellt. Der rote Vektor ist der Vektor $\vec{a}$ und der blaue ist der Vektor $\vec{b}$. Die gelbe Linie zeigt den projektieren Vektor $\vec{a}$. Diese Linie hat, wie oben ausgerechnet eine Länge von 2,22 Einheiten und man sieht, dass die Linie in die entgegengesetzte Richtung von Vektor $\vec{b}$ geht. Dies wird durch das Minus angegeben.

Wenn man jetzt für jeden Punkt das Skalarprodukt gebildet hat, dann hat man eine Reihe von Werten.

[14] (Hanson kein Datum)

Diese Werte geben jetzt die Koordinate von allen Punkten auf der Geraden an. Je kleiner der Wert ist, desto weiter „links“ liegt er und je größer er ist, desto weiter „rechts“. „Links“ und „rechts“ stehen in Anführungszeichen, weil die Gerade nicht (immer) horizontal steht und es somit kein richtiges links oder rechts gibt. Wenn man den Blickwinkel auf die Gerade aber so verändert, dass die Gerade waagerecht steht, ergeben diese Bezeichnungen wieder Sinn.

Nun muss nur noch die Linie vom jeweils kleinsten zum jeweils größten Wert beider Körper gezogen werden. Im letzten Schritt überprüft man, ob die beiden Linien überlappen.

Wenn:

$$min\, x_1 < max\, x_2$$

und:

$$min\, x_2 < max\, x_1$$

dann wurde keine „Trennlinie“ gefunden, andernfalls wurde eine Lücke zwischen beiden Körpern gefunden. Somit kollidieren die beiden Körper nicht.

3. NICHT-KONVEXE FORMEN

3.1. DAS PROBLEM

Wie oben beschrieben versucht der Algorithmus eine Gerade zwischen beiden Körpern zu finden. Dies funktioniert aber nur, wenn beide Körper konvex sind. Ein Körper ist konvex, wenn eine gerade Linie durch den Körper, maximal einen Eintritts- und Austrittspunkt haben kann.

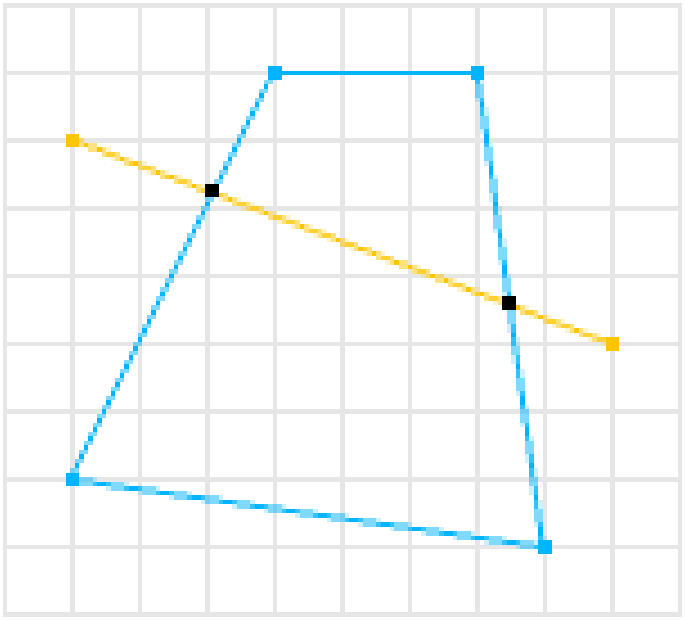

ABBILDUNG 7[15]

Wenn eine gerade Linie mehr als einen Eintritts- und Austrittspunkt haben kann, ist der Körper nicht konvex.

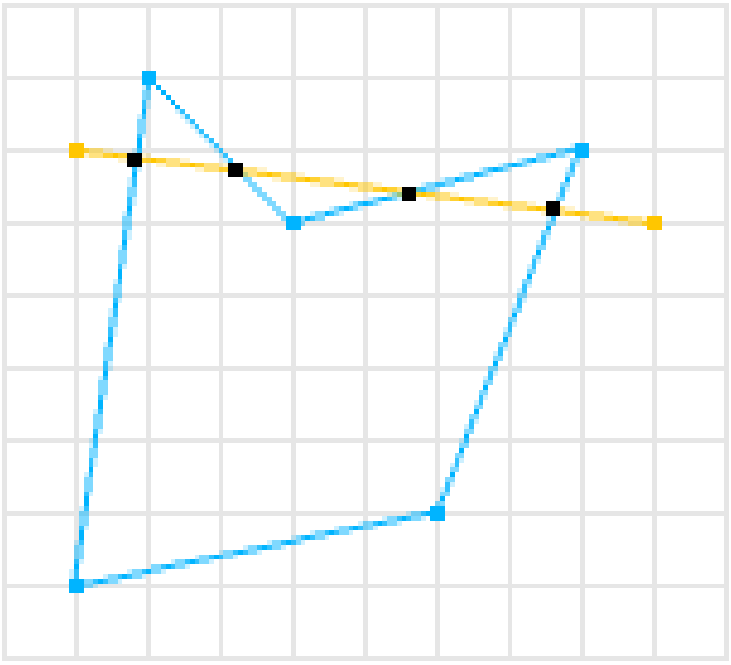

ABBILDUNG 8[16]

Sobald mindestens einer der beiden Körper nicht konvex ist, kann eine Lücke entstehen, welchen nicht mehr durch eine Gerade beschrieben werden kann.

[15] (SAT (Separating Axis Theorem) 2010)
[16] (SAT (Separating Axis Theorem) 2010)

ABBILDUNG 9[17]

Um dieses Problem zu lösen, kann man die nicht-konvexe Form in konvexe Teile zerlegen. Diese Teile kann man dann einzeln, wie normale konvexe Formen, mit dem anderen Körper überprüfen.

Da die Teile konvex sein sollen, bietet es sich, an den nicht-konvexen Körper in Dreiecke zu teilen. Dreiecke sind nämlich immer konvex.

Dieses Verfahren, des Zerlegen einer Form in Dreiecke, nennt man „**Polygon Triangulation**“.[18]

Um einen Körper jetzt in die einzelnen Dreiecke zu zerlegen, gibt es viele verschiedene Methoden. Ich habe den „**Ear clipping**“ Algorithmus benutzt.

3.2. EAR CLIPPING ALGORITHMUS[19]

Dieser funktioniert so, dass alle Eckpunkte des Körpers mit oder gegen den Uhrzeigersinn nummeriert werden.

[17] (Triangulation and Convex Hull 2018)
[18] (Triangulation and Convex Hull 2018)
[19] (Two-Bit 2021)

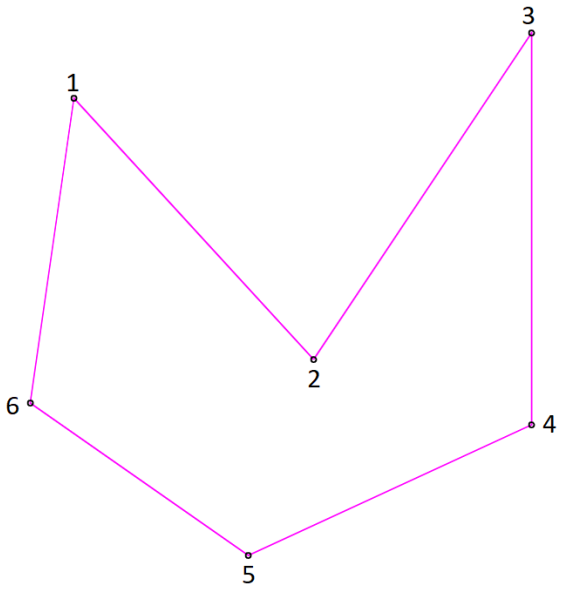

ABBILDUNG 10[20]

Dann werden drei aufeinanderfolgende Punkte A, B und C aus der nummerierten Liste gewählt und es wird ein Dreieck mit diesen Punkten als Eckpunkte gebildet. Der Algorithmus überprüft nun, ob das Dreieck ein „**ear**“ ist. Ein „**ear**“ ist ein Dreieck, von welchem zwei Seiten vom nicht-konvexen Körper sind, also Außenseiten, und die dritte Seite komplett im nicht-konvexen Körper liegt.

Das erste „ear“ der obigen Form wäre:

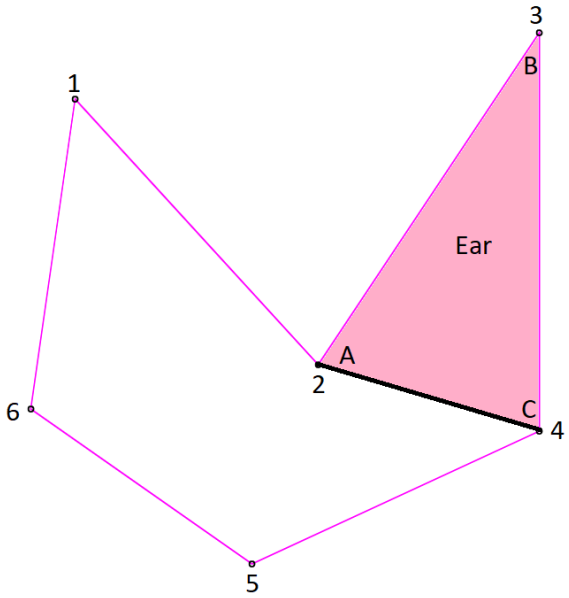

ABBILDUNG 11[21]

[20] Eigene Darstellung
[21] Eigene Darstellung

Dann wird der B Punkt des „ear“ aus dem nicht-konvexen Körper entfernt, sodass sich das „ear“ vom nicht-konvexen Körper abspaltet und ein neuer Körper entsteht. In diesem Fall ist der Punkt 3 der B Punkt des „ears“ und wird deswegen aus der Liste entfernt, sodass folgende Form entsteht.

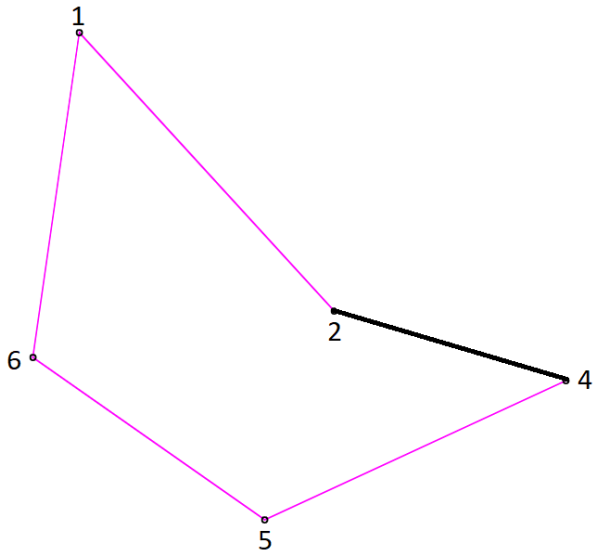

ABBILDUNG 12[22]

Dieser Vorgang wird so lange wiederholt, bis der Körper komplett in Dreiecke zerlegt ist. Das nächste „ear“ wäre:

Usw.

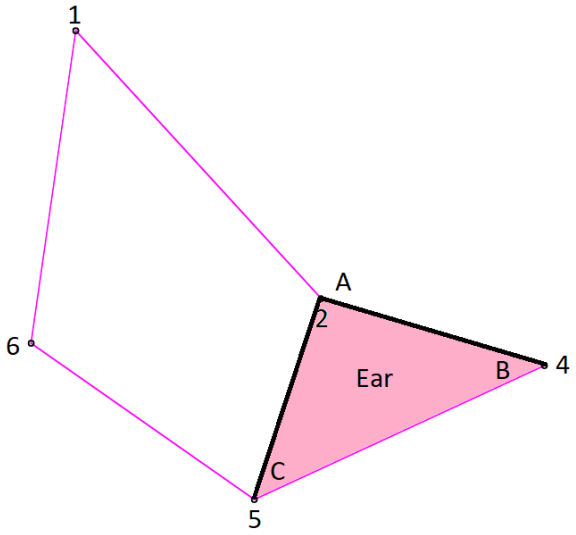

ABBILDUNG 13[23]

[22] Eigene Darstellung
[23] Eigene Darstellung

3.3. MATHEMATIK

Um zu schauen, ob ein gewähltes Dreieck A, B, C ein „ear“ ist, wird zuerst überprüf, ob jenes Dreieck überhaupt in der Form liegt.

Dies geschieht über das „**Kreuzprodukt**“ (*eng. Cross Product*) der Vektoren $\overrightarrow{BA}$ und $\overrightarrow{BC}$.

Das **Kreuzprodukt** ist ebenfalls, wie das oben erklärte **Skalarprodukt**, eine Form der Multiplikation von Vektoren. Anders als beim Skalarprodukt werden aber nicht immer die Werte an der gleichen Stelle multipliziert, sondern die Werte an der anderen Stelle (x mit y und umgekehrt). Anschließend werden die Produkte auch nicht zusammenaddiert, sondern subtrahiert. Um diese beiden Formen der Multiplikation zu unterscheiden, wird beim Kreuzprodukt statt dem **Malpunkt** „·“ das **Malkreuz** „×“ verwendet. (Daher auch die englischen Namen „*Dot Product*“ und „*Cross Product*“)

$$\vec{a} \times \vec{b} = x_1 \cdot y_2 - x_2 \cdot y_1$$

Das Ergebnis der Kreuzprodukts ist auch wieder ein **Skalar**. Durch diesen kann man ablesen, wie die beiden Vektoren zueinanderstehen. Für alle Winkel unter 180° gibt das Kreuzprodukt einen positiven Wert, für alle Winkel größer als 180° einen negativen. Bei genau 0° oder 180° ist das Kreuzprodukt 0.

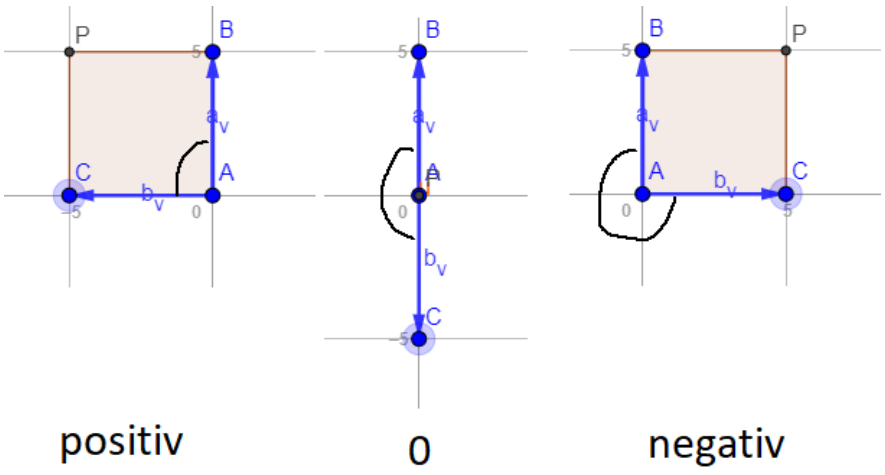

ABBILDUNG 14[24]

[24] (Babcock und Wall kein Datum)

Uns interessiert, ob der Winkel zwischen BA und BC kleiner als 180° ist. Wie unten in den Bildern zu sehen ist, liegt das Dreieck außerhalb der Form, sobald der Winkel größer ist als 180°.

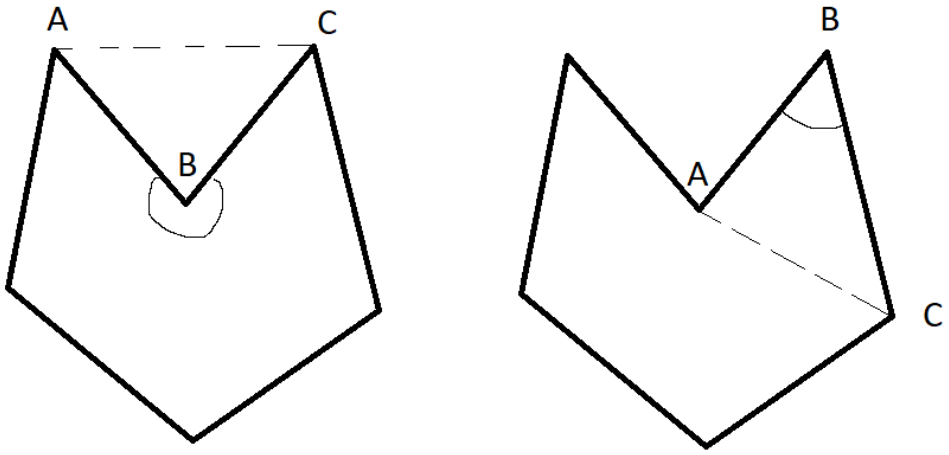

ABBILDUNG 15[25]

Durch das Minus in der Gleichung ist es, anders wie beim Skalarprodukt, wichtig, in welcher Reihenfolge die beiden Vektoren in die Gleichung gegeben werden. Durch das Vertauschen der Vektoren, wird nämlich mit dem Winkel in die andere Richtung gerechnet.

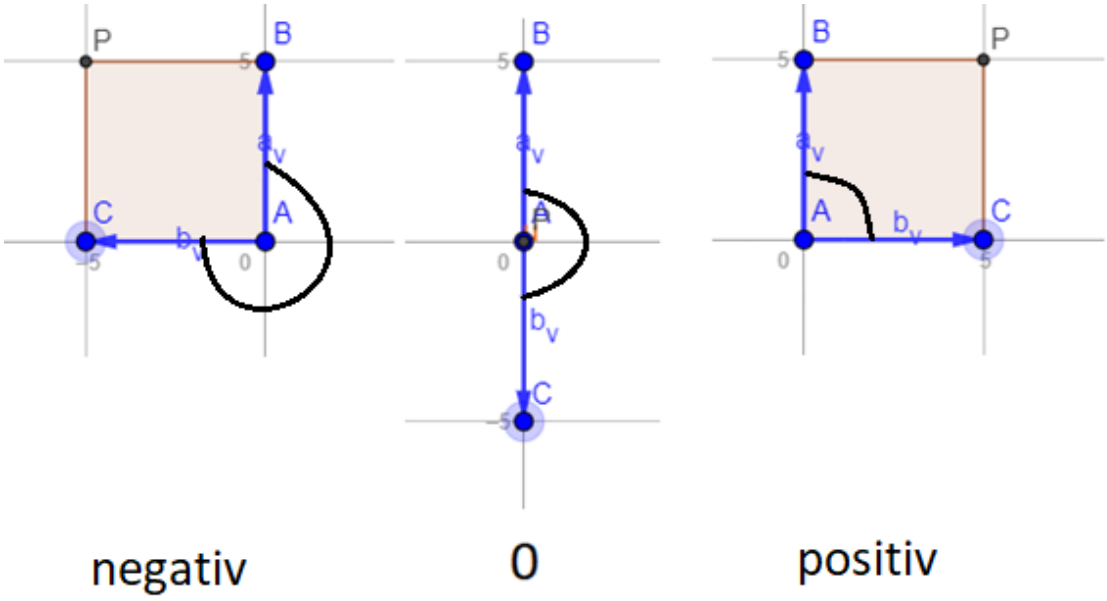

ABBILDUNG 16[26]

[25] Eigene Darstellung
[26] (Babcock und Wall kein Datum)

Nehmen wir an, dass alle Punkte im Uhrzeigersinn nummeriert wurden, dann muss das Kreuzprodukt einen positiven Wert haben. Wenn dies nicht der Fall ist, dann liegt das Dreieck außerhalb der Form. Das betrachtete Dreieck kann also kein „ear“ sein und man wiederholt diesen Test mit den nächsten 3 aufeinanderfolgenden Punkte.

Im zweiten Schritt wird überprüft, ob sich andere Eckpunkte der Form im gebildeten Dreieck befinden. Wenn sich ein Eckpunkt im Dreieck befinden würde, dann würde das Dreieck die Formgrenze schneiden und ein Teil des Dreiecks wäre außerhalb der Form.

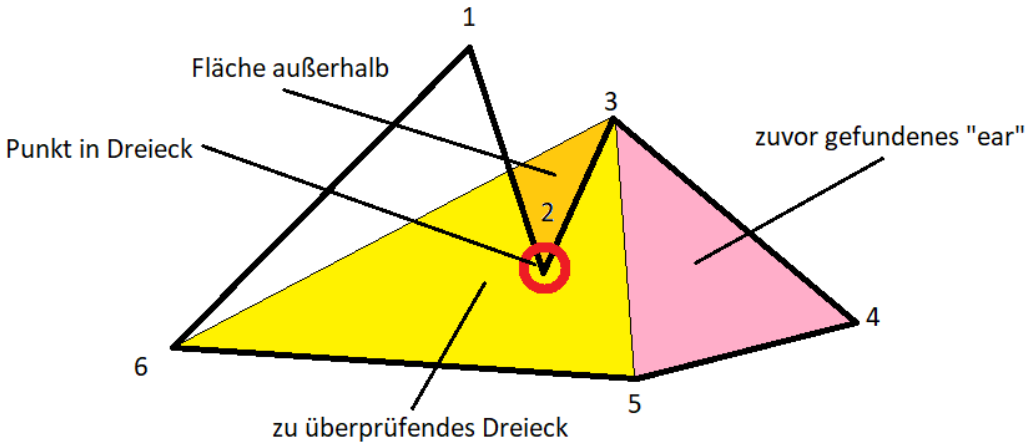

ABBILDUNG 17[27]

Als erstes nummeriert man die Eckpunkte im Uhrzeigersinn und bildet die Vektoren zwischen ihnen. Dann errechnet man die Vektoren von jedem Eckpunkt zum zu überprüfenden Punkt.

Durch das Kreuzprodukt kann jetzt überprüft werden, ob der Winkel zwischen dem Vektor von Eckpunkt1 zu Eckpunkt2 und dem Vektor von Eckpunkt1 zum Punkt kleiner ist als 180°. Wenn dies für alle Eckpunkte der Fall ist, dann liegt der Punkt im Dreieck.

[27] Eigene Darstellung

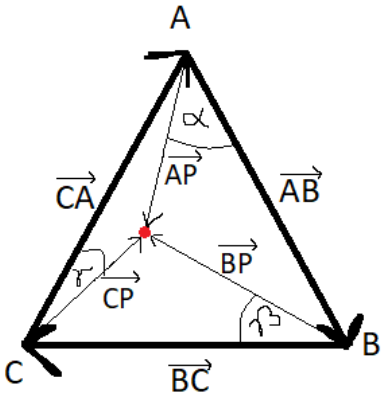

ABBILDUNG 18[28]

Sobald mindestens einer der Winkel mehr als 180° beträgt, liegt der Punkt außerhalb. Bei genau 180° liegt der Punkt exakt auf der Kante des Dreiecks und wird auch als außerhalb gewertet. (Hier Winkel γ)

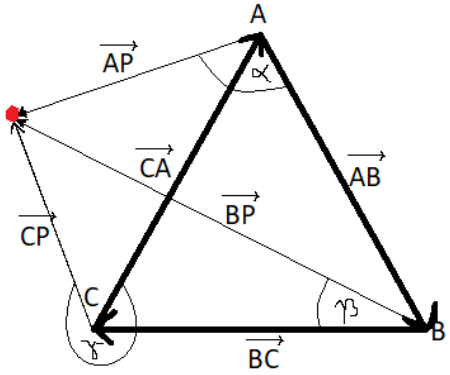

ABBILDUNG 19[29]

Nun wissen wir, dass ein gewähltes Dreieck ein „ear“ ist. Wir speichern dieses und entfernen Punkt B jetzt aus der Liste von Punkten. Wenn diese Liste dann nur noch drei Punkte enthält, ist das Dreieck dieser Punkte standardmäßig ein „ear“. Wir können dieses ebenfalls Abspeichern und sind mit der Triangulation fertig. Wenn aber noch mehr als drei Punkte vorhanden sind, wird diese Überprüfung mit der neuen Liste an Punkten wiederholt.

[28] Eigene Darstellung
[29] Eigene Darstellung

4. IMPLEMENTIERUNG

4.1. FORMEN ERSCHAFFEN

Um den Algorithmus im 2-dimensionalen implementieren zu können, brauchen wir erst einmal Formen.

Zum Erstellen von Formen starte ich mit zwei Werten: einen Radius r für einen Kreis und eine Anzahl n von Eckpunkten der Form. Nun erschaffen wir einen Kreis mit diesem Radius und verteilen die Punkte gleichmäßig auf diesem Kreis. Um dies zu schaffen, teilen wir die 360° eines Kreises durch die Anzahl der Eckpunkte und erhalten somit einen Gradwert, der den Winkel zwischen den Punkten angibt.

$$gradwert = \frac{360^0}{n} \cdot punktnummer$$

Jetzt bewegen wir die Punkte mit ihrer zugeteilten Gradzahl von der Kreismitte um r Einheiten.

ABBILDUNG 20[30]

Man sieht: alle Punkte liegen perfekt auf dem Kreis mit dem Radius r.

Mit zufälligen Radien und Anzahl von Eckpunkten erhalten wir solche Formen:

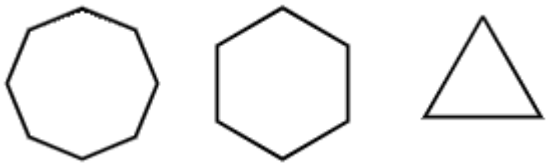

ABBILDUNG 21[31]

[30] Eigene Darstellung
[31] Eigene Darstellung

Um die Formen noch zufälliger zu machen, können wir für den Radius einen zufälligen Vorfaktor einbauen, sodass sich jeder Punkt unterschiedlich weit vom Mittelpunkt entfernt.

ABBILDUNG 22[32]

Nun können wir zufällige Formen an zufälligen Positionen erschaffen.

Als code Implementation:

```
static zufälligeForm(radius: number, numVertices: number, irregularity: number) {
  let form = new Array();
  for(let i = 0; i < numVertices; i++) {
    let mittelPunkt = new Vector2(0, 0);
    let gradWert = (360 / numVertices) * i;
    let distanz = (Util.randomBetween(-1, 1) * irregularity * radius) + radius
    form.push( Formeln.moveDirection(
      mittelPunkt,
      gradWert,
      distanz,
    ))
  }
  return form;
}
```

ABBILDUNG 23[33]

(Hier ist auch noch der „irregularity“ Wert mit eingebaut, mit dem ich beeinflussen kann, wie stark sich der Vorfaktor auswirkt)

4.2. SAT-IMPLEMENTATION

[32] Eigene Darstellung
[33] Eigene Darstellung

```
static testKollision(körper1: ICollideable, körper2: ICollideable) {
  let punkteKörper1 = körper1.translatePoints();
  let punkteKörper2 = körper2.translatePoints();

  let lezterPunkt = punkteKörper1[punkteKörper1.length - 1];

  for (let i = 0; i < punkteKörper1.length; i++) {
    let punkt = punkteKörper1[i];

    let normalVektor = lezterPunkt.vectorTo(punkt).getNormal();

    // Projektion Körper 1

    let min1 = Infinity;
    let max1 = -Infinity;

    punkteKörper1.forEach(point => {
      let skalarProdukt = point.skalarProdukt(normalVektor) / normalVektor.getLänge();
      min1 = Math.min(min1, skalarProdukt);
      max1 = Math.max(max1, skalarProdukt);
    })

    // Projektion Körper 2

    let min2 = Infinity;
    let max2 = -Infinity;

    punkteKörper2.forEach(point => {
      let skalarProdukt = point.skalarProdukt(normalVektor) / normalVektor.getLänge();
      min2 = Math.min(min2, skalarProdukt);
      max2 = Math.max(max2, skalarProdukt);
    })

    if (!(max2 >= min1 && max1 >= min2)) return false;

    lezterPunkt = punkt;
  }
  return true;
}
```

ABBILDUNG 24[34]

Diese Implementierung führt nur eine Überprüfung für alle Seiten von Körper 1 aus. Um den Algorithmus richtig auszuführen, muss die Methode noch einmal aufgerufen werden, diesmal aber mit Körper 1 und Körper 2 vertauscht. (In meinem Programm wird dies an anderer Stelle gemacht und ist deswegen nicht in der Methode implementiert)

[34] Eigene Darstellung

4.3. OPTIMIERUNG

Da die Kollisionsberechnung bei jeder Aktualisierung ausgeführt werden muss, werden bei vielen Körpern sehr viele unnötige Berechnungen durchgeführt. Bei Körpern, die sehr weit auseinander liegen, ist es, performancetechnisch, sehr verschwenderisch immer noch diese Methode zu benutzen.

Ein Weg, um dies zu vermeiden, ist es, einen Kreis um beide Körper zu zeichnen. Zu berechnen, ob sich zwei Kreise berühren, ist viel leichter und performance-freundlicher als SAT auf beiden Körpern anzuwenden. Dabei liegt der am weitesten außenliegende Punkt des Körpers auf dem Kreis und die Mitte des Körpers bildet die Kreismitte.

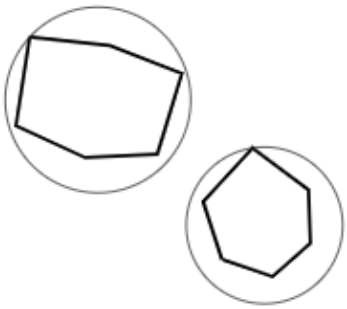

ABBILDUNG 25[35]

Um zu berechnen, ob sich zwei Kreise überschneiden, muss man nur die Distanz der beiden Kreismitten mit der Summe beider Radien vergleichen. Wenn die Distanz der beiden Kreismitten größer als die Summe beider Radien ist, dann gibt es keine Überschneidung, wenn aber die Summe beider Radien größer oder gleich der Distanz der Mitten ist, dann überschneiden sich beide Kreise und somit ist eine Kollision der Körper möglich.

Die Distanz zwischen zwei Punkten $A(x_A|y_A)$ und $B(x_B|y_B)$ kann so bestimmt werden:

$$d(A,B) = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

[35] Eigene Darstellung

Man hat jetzt zwei Kreise, mit den Mitten M_1 und M_2, mit den Radien r_1 und r_2.

Wenn also:

$$r_1 + r_2 \geq d(M_1, M_2)$$

Dann gibt es eine Überlappung und wenn:

$$r_1 + r_2 < d(M_1, M_2)$$

Dann gibt es keine Überlappung.

Programmiert sieht das Ganze dann so aus:

```
static potentialCollision(körper1: ICollideable, körper2: ICollideable) {
  [körper1, körper2].forEach(obj => obj.translatePoints());

  let distanzMitten = Formeln2.distance(körper1.pos, körper2.pos);

  let radius1 = Formeln2.distance(körper1.pos, Formeln2.farthestPoint(körper1.pos, körper1.translatePoints()));
  let radius2 = Formeln2.distance(körper2.pos, Formeln2.farthestPoint(körper2.pos, körper2.translatePoints()));

  let kollidieren = (distanzMitten < (radius1 + radius2));
  return kollidieren;
}
```

ABBILDUNG 26[36]

SCHLUSSBETRACHTUNG

Die Berechnung von Kollision ist für Spiele unerlässlich. Ohne Berührungsabfragen funktionieren nicht einmal die einfachsten Spiele, wie zum Beispiel Tic Tac Toe. Dort muss auch bestimmet werden, in welches Feld gesetzt wurde.

Der SAT Algorithmus ist einer der Wege, diese Kollision zu bestimmen. Jedoch wird der Algorithmus, gerade für nicht konvexe Körper, schnell unpraktisch, da für jeden Teilkörper mehrere Überprüfungen stattfinden müssen.

[36] Eigene Darstellung

Um den Algorithmus etwas mehr performance-freundlicher zu machen, habe ich einen Weg zur Optimierung aufgezeigt, durch eine erste Überprüfung mit Kreisen.

Die vorliegende Facharbeit beschäftigt sich ausschließlich mit der Frage, ob zwei Körper kollidieren. Ausbauende könnte man noch herausfinden, wo diese Kollision den überhaupt stattfindet. Dies ist nämlich ein wichtiger Schritt, um die Reaktion auf die Kollision zu gestalten.

Außerdem könnte es spannend sein, wie sich der Algorithmus verändern müsste, um auf teils kreisförmigen Körper richtig zu funktionieren.

LITERATURVERZEICHNIS

Babcock, Kara, und Wolfe Wall. *Cross Product and Area Visualization*. kein Datum. https://www.geogebra.org/m/psMTGDgc (Zugriff am 24. April 2022).

Center of Math. *Linear Algebra: Projection onto a Line*. 27. Juni 2014. https://www.youtube.com/watch?v=GnvYEbaSBoY (Zugriff am 24. April 2022).

Chong, Kah Shiu. *Collision Detection Using the Separating Axis Theorem*. 6. August 2012. https://gamedevelopment.tutsplus.com/tutorials/collision-detection-using-the-separating-axis-theorem--gamedev-169 (Zugriff am 24. April 2022).

Cuemath. kein Datum. https://www.cuemath.com/algebra/dot-product/ (Zugriff am 24. April 2022).

Hanson, Bob. kein Datum. https://www.falstad.com/dotproduct/ (Zugriff am 24. April 2022).

Huynh, Johnny. *Separating Axis Theorem for Oriented Bounding Boxes*. 16. Dezember 2008. https://www.jkh.me/files/tutorials/Separating%20Axis%20Theorem%20for%20Oriented%20Bounding%20Boxes.pdf (Zugriff am 24. April 2022).

javidx9. 2. Februar 2019. https://www.youtube.com/watch?v=7Ik2vowGcU0 (Zugriff am 24. April 2022).

Math Insight. kein Datum. https://mathinsight.org/definition/magnitude_vector#:~:text=The%20magnitude%20of%20a%20vector,are%20derived%20in%20this%20page. (Zugriff am 24. April 2022).

Mathematrix. kein Datum. http://www.mathematrix.de/skalarprodukt/ (Zugriff am 24. April 2022).

Ounsworth, Mike. *Algorithm to generate random 2D polygon*. 13. August 2014. https://stackoverflow.com/questions/8997099/algorithm-to-generate-random-2d-polygon (Zugriff am 24. April 2022).

Pikuma. *Math for Game Developers: Collision Detection with SAT*. 14. Juli 2021. https://www.youtube.com/watch?v=-EsWKT7Doww (Zugriff am 24. April 2022).

SAT (Separating Axis Theorem). 1. Januar 2010. https://dyn4j.org/2010/01/sat/ (Zugriff am 24. April 2022).

Separating Axis Theorem. kein Datum. http://programmerart.weebly.com/separating-axis-theorem.html (Zugriff am 24. April 2022).

talamundi. *Film 03 Projektion mit Skalarprodukt*. 19. Januar 2018. https://www.youtube.com/watch?v=1Agn8FfiE7g (Zugriff am 24. April 2022).

Triangulation and Convex Hull. 8. November 2018. https://www.uio.no/studier/emner/matnat/ifi/INF4130/h18/slides/forelesning-11---triangulering-og-convex-hull.pdf (Zugriff am 24. April 2022).

Two-Bit Coding. *Polygon Triangulation [2] - Ear Clipping Implementation in Code*. 7. Januar 2021. https://www.youtube.com/watch?v=hTJFcHutIs8 (Zugriff am 24. April 2022).

SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die im Literaturverzeichnis angegebenen Hilfsmittel verwendet habe. Alle genutzten Internetquellen wurden kenntlich gemacht. Sofern sich - auch zu einem späteren Zeitpunkt - herausstellen sollte, dass die Arbeit oder Teile davon nicht selbstständig verfasst wurden, die Zitationshinweise fehlen oder Teile aus dem Internet entnommen wurden, so wird die Arbeit auch nachträglich mit 0 Punkten gewertet.

Köln 24.04.2022 Florian Hirche