

Dokumentation
des
Projektstudiums

Physikalische Simulation starrer Körper

Betreuer: Prof. Dr. Dirk Eisenbiegler

Vorgelegt von: Daniel Wagner, 259256
Manuel Maringolo, 260683
Lukas Brausch, 260418
Sebastian Hoffmann, 259316
Dimitrios Stüber, 257744

Zeitraum: SS20 – WS20/10

Inhaltsverzeichnis

Inhaltsverzeichnis	II
1 Einleitung.....	1
2 Einarbeitung (1. Semester).....	2
3 Physikalische Grundlagen und Begriffe.....	7
3.1 Inertialsystem	7
3.2 Translatorische Bewegung	7
3.2.1 Kinematik der Translation	7
3.2.2 Masse	8
3.2.3 Kraft	8
3.2.4 Impuls	8
3.2.5 Kraftstoß	8
3.2.6 Translationsbewegungsenergie	9
3.2.7 Impulssatz.....	9
3.3 Punktmasse.....	9
3.4 Rotationsbewegung.....	9
3.4.1 Kinematik der Rotation.....	9
3.4.2 Trägheitsmoment	10
3.4.3 Drehmoment	10
3.4.4 Drehimpuls.....	10
3.4.5 Drehstoß	11
3.4.6 Rotationsbewegungsenergie	11
3.4.7 Drallsatz.....	11
3.5 Starre Körper.....	11
3.6 Stoß.....	12

3.6.1	Ideal elastischer Stoß	12
3.6.2	Ideal unelastischer Stoß	12
3.6.3	Realer Stoß	12
3.6.4	Exzentrische Stoß	12
3.7	Kräfte	14
3.7.1	Gewichtskraft	14
3.7.2	Hangabtriebskraft	14
3.7.3	Normalkraftkomponente	15
3.7.4	Rollreibungskraft	15
3.7.5	Haftreibung	16
3.7.6	Gleitreibung	16
3.7.7	Resultierende Kraft	17
4	Physikalische Simulation starrer Körper	18
4.1	Das physikalische System	18
4.2	Ein starrer Körper	18
4.2.1	Attribute starrer Körper	18
4.2.2	Berechnung der Energie	20
4.3	Mehrere starre Körper	21
4.3.1	Stöße	21
4.3.2	Rollen	27
4.3.3	Trockene Reibung	31
4.3.4	Von der Ebene Fallen	34
5	Spiele	36
5.1	Pong	39
6	Organisation des Projektes durch Dimitrios Stüber	40
6.1	Meine Aufgaben im 2. Teil des Projektes	40
6.1.1	Zu diesen Aufgaben zählen:	40

6.1.2	Weitere Aufgaben für meinen eigenen Teil des Projektes:	40
6.2	Logo für TdM	41
6.3	Videos/ Homepage	41
7	Fazit.....	43
8	Eidesstattliche Erklärung	45

1 Einleitung

Physikalische Simulationen sind in der heutigen Welt schon lange zum Standard geworden. Diese finden in jedem Computerspiel und jeder Animation einen Platz. Aber sie gehören auch in den Alltag von Naturwissenschaftlern und Ingenieuren, welche ihre Experimente in einer Simulation testen oder damit Prognosen treffen. Dafür ist eine physikalisch korrekt funktionierende Simulation erforderlich.

Im Projekt „Physikalische Simulation starrer Körper“ soll eine Komponentenbibliothek für die Simulationsumgebung des Physolators entwickelt werden. Diese soll im ersten Schritt zunächst Stöße von beliebigen starren Körpern simulieren können. Im nächsten Schritt soll diese Library zum Beispiel ebenso um Rollbewegungen von Kreisen oder um Effekte wie Haftreibung und Gleitreibung ergänzt werden. Im letzten Schritt sollen auf Basis der physikalischen Komponenten Spiele entwickelt werden, welche die physikalischen Effekte zur Schau stellen.

Der Physolator ist ein Framework des Projektbetreuers Professor Eisenbiegler zur physikalischen Simulation. Das Framework unterstützt eine objektorientierte Modellierung von physikalischen Komponenten und bietet unter anderem eine automatische, numerische Lösung von Differenzialgleichungen sowie eine Vielzahl anderer hilfreicher Features die bei der Simulation physikalischer Systeme helfen. (Physolator Homepage: <https://physolator.com/de/>)

2 Einarbeitung (1. Semester)

Der Physolator war zu Beginn des Projekts für uns eine neue Umgebung. Zuvor hatte noch niemand Berührungspunkte damit gehabt. Um uns mit dem Physolator vertraut zu machen, wurden uns einige Aufgaben zur Einarbeitung zur Verfügung gestellt. Die Aufgaben können unter anderem auf der Website des Physolators gefunden werden oder im Buch „Arbeitsbuch physikalische Simulation: Interdisziplinäre Aufgaben aus dem MINT-Bereich“ von Professor Eisenbiegler. Außerdem wurde als Nachschlagewerk das Buch "Objektorientierte Modellierung und Simulation physikalischer Systeme mit dem Physolator" ausgeteilt. Ziel dieser Aufgaben war es, uns Schritt für Schritt an den Physolator heranzuführen. Dadurch sollten wir seine Funktionsweise und Fähigkeiten kennenlernen. Die Einarbeitung sollte im Sommersemesters 2020 abgeschlossen werden, um im Anschluss mit der Aufgabe beginnen zu können.

Jede der insgesamt 11 Aufgaben hatte ein bestimmtes Thema, welches sie vertiefte. Alle diese Themen sind für unser Projekt relevant, darum war es wichtig, dass jeder die Aufgaben zur Einarbeitung so gewissenhaft bearbeitet, dass er die Funktionsweise und die dort angesprochenen Themen verstanden hat.

Die Aufgaben 1 und 2 standen unter dem Oberthema der Physik. Genauer betrachtet, sollten Federpendel simuliert werden. Hierzu haben wir das Hooksche Gesetz

$$F = D \cdot \Delta s$$

angewendet. Daraufhin haben wir die benötigten Variablen mit den dazugehörigen Ableitungen und Formeln in den Physolator implementiert. Dadurch kann der Physolator die Simulation physikalisch korrekt berechnen indem er die Differenzialgleichungen und deren Anfangswert Probleme numerisch löst und die Methode `f`, in welcher die Formeln implementiert werden, immer wieder aufruft. Im Anschluss daran wurde ein Graph durch die Methode `initPlotterDiscription()` erstellt, der die vertikale Auslenkung x und Geschwindigkeit v des Federpendels in Abhängigkeit der Zeit t grafisch darstellt.

Die nächsten beiden Aufgaben hatten ihren Fokus auf der Programmierung und dem Zeichnen in der grafischen Oberfläche des Physolators. Als Übung wurde von uns ein Haus mithilfe von Koordinaten, elementaren Zeichenbefehlen, Methoden mit Parameterübergaben und `for`-Schleifen gezeichnet. Im ersten Schritt sollten wir ein statisches Haus mit fixer Position und Größe in die Grafische Oberfläche des Physolators

zeichnen. Hierzu wurde eine Grafikkomponente *HausTVG* erzeugt. Diese kann mithilfe der Methode *paint()* Zeichenbefehle grafisch darstellen. Nun wurde durch das Einsetzen von Methoden und verschachtelten *for*-Schleifen eine Siedlung erstellt. Die Größe (also Anzahl der Häuser in vertikaler, sowie horizontaler Richtung) der Siedlung wurde durch *Parameter* und *Slider* für den User dynamisch anpassbar.

Daraufhin wurde ein Hauskonfigurator erstellt, der Höhe, Breite, Anzahl der Stockwerke und Größe des Kamins ebenso dynamisch darstellen kann. Wichtig war hierbei auch, wiederkehrende Funktionalitäten in eigene Methoden auszulagern, um Code-Dopplungen zu vermeiden.

Der Rollende Ball, Aufgabe 5, vereinte nun die zuvor gelernten Kenntnisse von Programmierung, Physik und Geometrie. Als neue Gegenstände wurden Rollreibung, Strömungswiderstand, Stoß, Trigonometrie, physikalische Ereignisse und Schwellwerte eingefügt. Ziel war es einen Programmcode zu entwickeln, der einen physikalisch korrekt rollenden Ball mit Rollreibung und Luftwiderstand berechnet, und in der Grafikkomponente darstellt. Ebenso sollten im Plotter alle relevanten Größen dargestellt werden.

Im ersten Schritt wurden wieder die physikalischen Größen initialisiert, deklariert und deren Ableitungen angegeben. Nun haben wir die Rollreibung F_r mit Hilfe der Formel

$$F_r = -m \cdot g \cdot c_r$$

berechnet. Das Minus-Zeichen ist wichtig, da die Rollreibung immer entgegen der Rollrichtung zeigt. Nur so wird der Ball physikalisch korrekt abgebremst. Würden wir das Minus-Zeichen weglassen, so würde der Ball hingegen beschleunigt werden, was aber nicht der Realität entsprechen würde.

In dieser Aufgabe wurde es zum ersten Mal erforderlich, dass die Grafikkomponente auf die „live“ berechneten physikalischen Größen zugreifen kann, um die Simulation darstellen zu können. Um dies möglich zu machen, haben wir die Methode

```
public void initGraphicsComponents(GraphicsComponents g) {  
    g.addTVG(new RollenderBallTVG(this));  
}
```

eingefügt. Dabei wird beim Konstruktoraufbau *RollenderBallTVG(this)* dem Konstruktor das physikalische System übergeben. Damit die Werte auch in der Grafikkomponente

verwendet werden können, wird das erzeugte Objekt im Konstruktor in einer Variable gespeichert

```
rb = rollenderBall;
```

.

Durch den Programmcode

```
public void g(double t, AfterEventDescription afterEventDescription) {
    if (v < 0)
        afterEventDescription.reportEvent(() -> {
            Fr = 0;
            v = 0;
        });
}
```

haben wir festgelegt, dass der Ball eine Geschwindigkeit von 0 und eine Rollreibung von 0 besitzen soll, wenn dessen Geschwindigkeit kleiner als 0 ist.

Zum Schluss haben wir noch analog der Rollreibung einen Luftwiderstand durch

$$F_L = \frac{1}{2} A \cdot c_w \cdot \rho \cdot v^2$$

hinzugefügt. Durch einige *if*-Konstrukte haben wir noch einen Stoß an einer Wand eingefügt, sodass der Ball nach dem Stoß in die entgegengesetzte Richtung rollt.

In Aufgabe 6 haben wir einen schiefen Wurf simuliert. Wir sind analog zu Aufgabe 4 vorgegangen. Zusätzlich haben wir noch einen Trigger eingefügt:

```
public ThresholdTrigger tr1 = new ThresholdTrigger(() -> vy < 0).setName("Hochpunkt").setInfo(() -> "x="+x + "y="+y).setDoPrint(true);
```

Dieser wird nur aktiv, wenn die vertikale Geschwindigkeit <0 ist. Das heißt, wir geben somit die Zeit bis zur Auslösung des Triggers und die Koordinaten des Hochpunktes des Schiefen Wurfs aus. In den nächsten Schritten haben wir mehrere Trigger eingefügt, die die Position zu verschiedenen Zeitpunkten ausgeben. Ein Beispiel:

```
public TimeTrigger tr4 = new TimeTrigger(0.27134, 0.4525, 1.273).setInfo(() -> "x=" + x + " y=" + y).setDoPrint(true)
```

Aufgabe 7 beschäftigt sich mit dem neuen Thema bewegte, zeitabhängige Grafiken. Dabei animieren wir eine Lokomotive, welche wir zuvor durch die Grafikkomponente gezeichnet haben. Die komplette Lokomotive soll sich in Abhängigkeit von der Zeit t

nach rechts bewegen. Zusätzlich sollen Räder und Speichen sich drehen und Achsen und Bolzen sich korrekt bewegen.

Hierbei wurde noch einmal insbesondere der Umgang mit trigonometrischen Funktionen, sowie deren Einsatz geschult. So kann die Drehbewegung am besten durch Matrixmultiplikation mit einer Drehmatrix dargestellt werden. Außerdem besitzt die Lock ihr eigenes Koordinatensystem. Dieses wird dann einfach verschoben. Die einzelnen Werte wie Position, Größe und Rotation von Teilkomponenten der Lock müssen dann nichtmehr angepasst werden.

Einzelne Aufgaben wie das Zeichnen eines Rades können in eigene Methoden ausgelagert werden. Parameter, wie z.B. die Anzahl der Speichen oder der Radius können dynamisch angepasst werden.

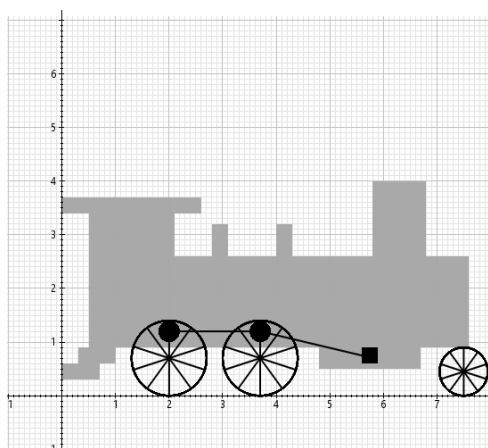


Abbildung 2 $t = 0s$

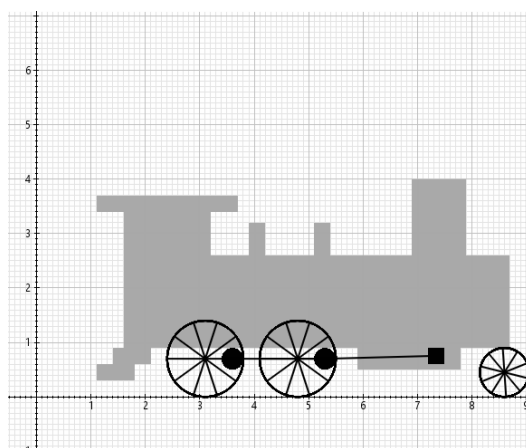


Abbildung 2 $t = 1s$

Die letzten beiden Aufgabenblätter, haben sich mit der Gravitation von punktförmigen Massen beschäftigt. Diese Massen sollen in beiden Aufgaben Satelliten darstellen, welche um die Erde (0|0) kreisen. Hierzu verwenden wir die Library *mechanics.jar*.

Wieder wurde ein physikalisches System mit Variablen, Ableitungen und Formeln erstellt, sodass sich das System physikalisch korrekt verhält. Durch Hinzunahme des Programmcodes

```
public ThresholdTrigger tr1 = new ThresholdTrigger(() -> y<0)
    .setName("Umrundungsdauer").setDoPrint(true);
```

haben wir eine Umrundungsdauer in der Konsole ausgeben lassen.

In Aufgabe 9 wurde das System eines Satelliten erweitert, indem wir diesen zu einem Objekt gemacht haben. Außerdem haben wir die Werte für die horizontale und vertikale Position (x, y) , Geschwindigkeit (v_x, v_y) und Beschleunigung (a_x, a_y) in Vektoren $((x, y), (v_x, v_y), (a_x, a_y))$ angegeben. Durch dieses Satelliten-Objekt konnten wir beliebig viele neue Satelliten mit unterschiedlichen Anfangswerten aber denselben Eigenschaften und Verhalten erzeugen. Diese wurden gleichzeitig in einer Grafikkomponente ausgegeben.

Dieses System nennt sich objektorientierte Programmierung und ist enorm wichtig und essenziell zur Erstellung von sauberen und effizienten Programmen.

Obwohl wir nur 9 der 11 Einarbeitungsaufgaben bearbeitet haben, reicht uns dieses Wissen, um unser Projekt umzusetzen. Alle bis hier hin kennengelernte Elemente werden wir in unserem Projekt benötigen. Sie helfen uns dabei unser Projekt korrekt umzusetzen.

Da jeder zu Beginn des Projekts unterschiedliche Kenntnisse in der Programmierung hatte, waren auch Hürden und Probleme sehr individuell. Im Folgenden wird jedes einzelnes Gruppenmitglied diese Punkte für sich speziell erläutern.

3 Physikalische Grundlagen und Begriffe

3.1 Inertialsystem

In der Physik ist ein Inertialsystem ein Bezugssystem, in dem jeder Körper, auf den keine Kräfte wirken, oder wo die Summe der Kräfte gleich Null ist, entweder in Ruhe verharrt oder sich unbeschleunigt geradlinig durch den Raum bewegt.

3.2 Translatorische Bewegung

Bei einer Translation erfahren alle Punkte eines Körpers dieselbe Verschiebung im Bezugssystem. Somit besitzt ein freier Körper auf der Ebene zwei Freiheitsgrade der Translation.

3.2.1 Kinematik der Translation

Die folgenden Ableitungsverhältnisse der Kinematik, der Lehre von geometrischer Bewegung ohne Kräfte, sind entscheidend für das Verständnis der physikalischen Simulation.

3.2.1.1 Ort

Der Ort eines Körpers wird durch den Vektor \vec{r} beschrieben und die Einheit ist der Meter (m). Das Weg-Zeit-Gesetz ist eine Funktion $\vec{r}(t)$, die jedem Zeitpunkt einen Ort zuordnet.

3.2.1.2 Geschwindigkeit

Die Geschwindigkeit \vec{v} ist die zeitliche Änderung des Ortes. Es gilt daher $\vec{v}(t) = \dot{\vec{r}}(t)$ wobei der Punkt eine gebräuchliche Kurznotation für $\frac{d\vec{r}}{dt}$ also eine Ableitung nach der Zeit ist.

Der Betrag in der Ebene ist $|\vec{v}| = \sqrt{v_x^2 + v_y^2}$

3.2.1.3 Beschleunigung

Die Beschleunigung \vec{a} ist die zeitliche Änderung der Geschwindigkeit. Es gilt daher $\vec{a}(t) = \dot{\vec{v}}(t) = \ddot{\vec{r}}(t)$

3.2.2 Masse

Die Masse ist in der klassischen Mechanik eine Erhaltungsgröße, man unterscheidet dabei zwischen Träger und schwerer Masse. Dabei kann davon ausgegangen werden, dass schwere und träge Masse dieselbe physikalische Größe sind, somit Betragsgleich sind. Die Einheit ist das Kilogramm (kg) und das Formelzeichen ein m.

In diesem Projekt ist insbesondere die träge Masse von Interesse da Gravitationskräfte zwischen Körpern nicht betrachtet werden sollen.

3.2.2.1 Trägheit (Träge Masse)

Aufgrund seiner Masse setzt ein Körper einer Kraft, die seine Geschwindigkeit ändert, einen Widerstand entgegen. Man kann somit die Trägheit auch als eine Art Beharrungsvermögen verstehen in einem Bewegungszustand zu verharren.

3.2.2.2 Gravitationsladung (Schwere Masse)

Zwei Körper ziehen sich gegenseitig an. Die Kraft wirkt entlang der Verbindungslinie und ist zu den Massen der Körper proportional.

3.2.3 Kraft

Die Kraft kann in der klassischen Mechanik als eine vektorielle Größe aufgefasst werden, welche einen Körper beschleunigt oder bremst. Das Formelzeichen ist ein F und die Einheit ist das Newton. Es gilt $1N = 1kg \cdot m \cdot s^{-2}$.

Nach dem Superpositionsprinzip gilt für einen Körper, auf den mehrere Kräfte wirken $\vec{F} = \vec{F}_1 + \dots + \vec{F}_n$.

3.2.4 Impuls

Der physikalische Impuls kann umgangssprachlich vielleicht als „Wucht“ aufgefasst werden. Somit ist der Impuls eines Körpers größer, je höher seine Geschwindigkeit oder seine Masse ist ($\vec{p} = m\vec{v}$). Das Formelzeichen ist ein p und die Einheit ist die Newtonsekunde. Es gilt daher $Ns = kg \cdot m \cdot s^{-1}$.

3.2.5 Kraftstoß

Eine Kraft, die über eine Dauer auf einen Körper wirkt, führt zu einer Impulsänderung, die man Kraftstoß nennt. Das Formelzeichen ist ein I und die Einheit natürlich äquivalent zum Impuls das Newtonmeter. Ist die Kraft über das Zeitintervall $\Delta t_{1,2}$ konstant gilt

somit $\vec{I} = \Delta \vec{p} = \vec{F} \cdot \Delta t_{1,2}$. Im Allgemeinen ist die Kraft aber Zeitabhängig und der Kraftstoß muss über Integration ermittelt werden.

3.2.6 Translationsbewegungsenergie

In der klassischen Mechanik ist die Translationsbewegungsenergie abhängig von Masse und Geschwindigkeit des Körpers. Die Formel zur Berechnung ist $E_t = \frac{1}{2}mv^2$.

Die Einheit ist das Joule $J = kg \cdot m^2 \cdot s^{-2} = Nm$.

3.2.7 Impulssatz

Die zeitliche Änderung eines Impulses entspricht der einwirkenden äußeren Kraft. Somit gilt $\vec{F} = \dot{\vec{p}}$ und im Falle einer konstanten Masse $\vec{F} = m \cdot \vec{a}$.

3.3 Punktmasse

Eine Punktmasse ist die größte Vereinfachung eines realen Körpers in der Physik. Die gesamte Masse ist dabei in einem Punkt, der Punktmasse, konzentriert.

Die Punktmasse besitzt translatorische Freiheitsgrade, aber aufgrund der fehlenden Eigenschaften wie Abmessungen, Volumen oder Form keine Rotationsfreiheitsgrade.

Zur Beschreibung werden daher nur die Masse, die Koordinaten und die Translatorischen Größen benötigt. Die Punktmasse bewegt sich wie durch die Kinematik festgelegt.

3.4 Rotationsbewegung

3.4.1 Kinematik der Rotation

Die Kinematik der Rotation ist parallel zur Kinematik der Translation aufzufassen.

3.4.1.1 Winkel

Der Drehwinkel Φ φ gibt die Drehung aus der Ausgangsposition des Körpers an. Im Bogenmaß (rad) entspricht 0 Grad 0π , 180 Grad π , 360 Grad 2π . Das Winkel-Zeit-Gesetz ist eine Funktion $\vec{\omega}(t)$ die jedem Zeitpunkt einen Winkel zuordnet.

3.4.1.2 Winkelgeschwindigkeit

Die Winkelgeschwindigkeit Ω $\vec{\omega}$ wird in der Einheit $rad \cdot s^{-1}$ angegeben. Er zeigt auf wie schnell sich ein Winkel mit der Zeit ändert. Der Betrag ist die Ableitung des Winkels nach der Zeit, somit gilt $\omega = \dot{\varphi}$. In der euklidischen Ebene kann man sich auf

die Betrachtung des Betrags beschränken da der Vektor immer senkrecht auf der Ebene stehen würde.

3.4.1.3 Winkelbeschleunigung

Die Winkelbeschleunigung α wird in der Einheit $\text{rad} \cdot \text{s}^{-2}$ angegeben. Sie ist die zeitliche Änderung der Winkelgeschwindigkeit. Es gilt $\alpha = \dot{\omega}$.

3.4.2 Trägheitsmoment

Das Trägheitsmoment hat das Formelzeichen I und die Einheit $\text{kg} \cdot \text{m}^2$. Diese Einheit ist, äquivalent zur trägen Masse für die translatorische Bewegung, entscheidend für die Rotationsbewegung. Das Trägheitsmoment ist abhängig von der Massenverteilung zur Drehachse. Für die zweidimensionale Darstellung ist es daher die Massenverteilung in Abhängigkeit zum Masseschwerpunkt.

Für eine Punktmasse, die sich um einen Rotationspunkt dreht, vereinfacht sich das Integral, was im allgemeinen Fall für die Berechnung betrachtet werden muss, zu:

$$I = m \cdot a^2$$

3.4.3 Drehmoment

Das Drehmoment, oder Kraftmoment, ist für Rotationsbewegung dasselbe wie die Kraft für translatorische Bewegungen. Die Einheit ist das Newtonmeter und das Formelzeichen ein \vec{M} . Es bezeichnet also die Drehwirkung eines Kräftesystems auf einen Körper und kann dessen Drehung beschleunigen oder bremsen.

Ist der Ortsvektor \vec{r} der Angriffspunkt einer Kraft im Bezugssystem des Drehmoments dann gilt $\vec{M} = \vec{r} \times \vec{F}$. Daher steht der Vektor senkrecht auf der durch \vec{r} und \vec{F} aufgespannten Ebene und entspricht somit der Richtung der Drehachse. Findet die Physik wie in dieser Simulation nur in der euklidischen Ebene statt dann ist die aufgespannte Ebene zwangsläufig die xy-Ebene. Eine Drehachse existiert ebenfalls nicht da Rotationen um einen Punkt stattfinden.

3.4.4 Drehimpuls

Der Drehimpuls \vec{L} auch Drall genannt hat die Einheit Joulesekunden $\text{kg} \cdot \text{m}^2 \cdot \text{s}^{-1} = \text{Js}$. Er berechnet sich durch $\vec{L} = \vec{r} \times \vec{p}$ und bezieht sich auf den Bezugspunkt der

Drehbewegung. Der Drehimpuls eines Systems ist die Summe der Drehimpulse seiner Komponenten $\vec{L} = \sum_{k=1}^n m_k \vec{x}_k \times \dot{\vec{x}}_k$.

Bei einem starren Körper mit Masseschwerpunkt \vec{s} und den Komponenten bzw. Massepunkten $\vec{p} = \vec{x} - \vec{s}$ kann dies auch als $\vec{L} = \sum_{k=1}^n m_k \vec{x}_k \times (\dot{\vec{s}} + \vec{\omega} \times \vec{p}_k)$ wobei $\dot{\vec{s}}$ die Schwerpunktgeschwindigkeit des Körpers ist. Im Allgemeinen besteht der Körper natürlich aus unendlich vielen Massepunkten und es ergibt sich ein Integral.

3.4.5 Drehstoß

Der Drehstoß $\Delta \vec{L}$ entsteht durch ein Drehmoment und dessen Einwirkungsdauer auf einen Körper. Die Einheit des Drehstoßes ist $N \cdot m \cdot s$. Ist das Drehmoment im Zeitintervall konstant dann gilt $\Delta \vec{L} = \vec{M} \cdot \Delta t$. Im Allgemeinen Fall muss der Drehstoß über Integration bestimmt werden.

3.4.6 Rotationsbewegungsenergie

Die Rotationsbewegungsenergie berechnet sich in der euklidischen Ebene durch $E_r = \frac{1}{2} \cdot I \cdot \omega^2$. Die Einheit ist das Joule $J = kg \cdot m^2 \cdot s^{-2} = Nm$.

3.4.7 Drallsatz

Der Drallsatz ist ein Gesetz, welches besagt, dass zur Änderung des Drehimpulses eines Körpers ein Drehmoment an ihm wirken muss. Es gilt $\dot{\vec{L}} = \vec{M}$ und bei konstantem I gilt $I \cdot \dot{\vec{\omega}} = \vec{M}$.

3.5 Starre Körper

Der starre Körper ist eine Modellvorstellung in der klassischen Mechanik welche nicht vollständig der „realen Welt“ entspricht. Wie das Adjektiv „starr“ andeutet ist ein starrer Körper nicht verformbar. Jedes beliebige Paar von Punkten innerhalb des Körpers hat zu jedem beliebigen Zeitpunkt den gleichen Abstand zueinander und somit sind alle Formen von Deformation ausgeschlossen. Durch diese Eigenschaft sind starre Körper ideal geeignet, um eine einfache Mechanik zu simulieren in welcher Körper unter dem Einfluss von Kräften keine Verformungen aufweisen. Der starre Körper kann nur eine Translatorische Bewegung oder eine Rotationsbewegung um eine Achse bzw. einen Drehpunkt ausführen.

3.6 Stoß

Ein Stoß ist das Zusammentreffen von zwei oder mehr Körpern, die für eine kurze Zeit Kräfte aufeinander auswirken die ihren Bewegungszustand ändern. In einem Inertialsystem gelten für Stöße der Impulserhaltungssatz und auch die Energieerhaltung. Allerdings beschränkt sich die Energie nicht nur auf mechanische Formen.

3.6.1 Ideal elastischer Stoß

Beim ideal elastischen Stoß bleibt die gesamte Energie in Bewegungsenergie erhalten. Keine Energie geht in Deformation, Wärme und Ähnliches über.

3.6.2 Ideal unelastischer Stoß

Beim ideal unelastischen Stoß wird der maximal mögliche Anteil kinetischer Energie in innere Energie umgewandelt. Daraus folgt, dass zwei Körper nach dem Stoß mit gleicher Geschwindigkeit weitergleiten und somit quasi aneinanderhaften.

3.6.3 Realer Stoß

Beim realen Stoß vermischen sich elastischer und unelastischer Stoß. Dieser Fall ist der häufigste und wird durch die Stoßzahl k beschrieben. Dabei gilt $k = 0$ entspricht dem ideal unelastischen Stoß und $k = 1$ dem ideal elastischen Stoß.

3.6.4 Exzentrische Stoß

Der exzentrische Stoß behandelt allgemeine Stoßvorgänge zweier starrer Körper in der euklidischen Ebene. Daraus folgt, dass der Stoß in unendlich kurzer Zeit stattfindet und dabei andere Kräfte, Lageänderungen oder Deformationen keine Rolle spielen. Bekannt sein müssen Geschwindigkeit, Winkelgeschwindigkeit, Masse, Trägheitsmoment und Position der beiden Körper und berechnet werden sollen die neuen Geschwindigkeiten und Winkelgeschwindigkeiten nach dem Stoß. Außerdem muss die Stoßzahl (Restitutionskoeffizient) des Stoßes bekannt sein.

3.6.4.1 Stoßkoordinatensystem

Für die Berechnung ist es wichtig, die Größen Geschwindigkeit und Ort in das Stoßkoordinatensystem zu transformieren. Der Stoßpunkt ist dabei der Ursprung des Stoßkoordinatensystems. Der Stoßpunkt kann Beispielsweise eine Ecke sein, die auf eine Kante trifft oder der Berührungspunkt zweier Kreise. Außerdem besitzt jeder Stoß auch eine Berührungsebene bzw. Berührungslinie. Die Berührungslinie kann z.B. die Kante sein, auf die eine Ecke stößt, oder die Tangente an der Stelle wo zwei Kreise

aufeinanderstoßen. Die y-Achse des Stoßkoordinatensystems liegt auf dieser Berührungslinie und somit entspricht die x-Achse der Stoßnormale. Somit existiert für jeden Stoß ein Stoßkoordinatensystem.

3.6.4.2 Berechnung

Zur Berechnung werden zuerst die Geschwindigkeiten und Orte der Körper in das Stoßkoordinatensystem überführt. Der kürzeste Abstand vom Ort zur Stoßnormalen (x-Achse) wird als Zwischenvariable c benannt. Gegeben sind also v_{1x} v_{2x} v_{1y} v_{2y} C_1 C_2 ω_1 ω_2 m_1 m_2 I_1 I_2 . Zuerst berechnet man den Kraftstoß:

$$\hat{F}_x = (1 + k) \cdot \frac{v_{1x} - v_{2x} - C_1 \omega_1 + C_2 \omega_2}{\frac{1}{m_1} + \frac{1}{m_2} + \frac{C_1^2}{I_1} + \frac{C_2^2}{I_2}}$$

und daraufhin lassen sich die neuen Geschwindigkeiten berechnen über:

$$v'_{1x} = v_{1x} - \frac{\hat{F}_x}{m_1}, \quad v'_{1y} = v_{1y}$$

$$v'_{2x} = v_{2x} + \frac{\hat{F}_x}{m_2}, \quad v'_{2y} = v_{2y}$$

Wie man sieht ist die y-Komponente gleichbleibend. Die neuen Winkelgeschwindigkeiten sind dann:

$$\omega'_1 = \omega_1 + \frac{c_1 \hat{F}_x}{I_1}$$

$$\omega'_2 = \omega_2 + \frac{c_2 \hat{F}_x}{I_2}$$

letztendliche müssen die Geschwindigkeiten zurück ins Inertialsystem transformiert werden. Damit ist der exzentrische Stoß vollzogen.

(Quelle: Wandinger, Johannes: Exzentrischer Stoß.

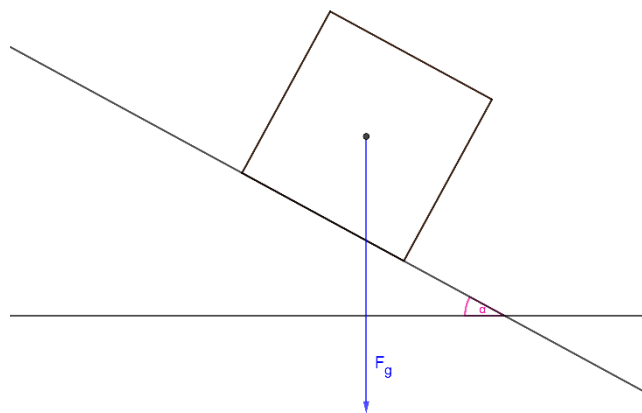
Abgerufen von http://wandinger.userweb.mwn.de/LA_Dynamik_2/v4.pdf am 13.02.2021)

3.7 Kräfte

3.7.1 Gewichtskraft

Die Gewichtskraft wirkt ständig auf jeden Körper und wirkt auf der Erde in die Richtung des Erdmittelpunkts. Definiert ist die aus der Schwerebeschleunigung g (im Fall der Erde, wird mit dem Näherungswert $g = 9,81 \frac{m}{s^2}$ gerechnet) und der Masse m .

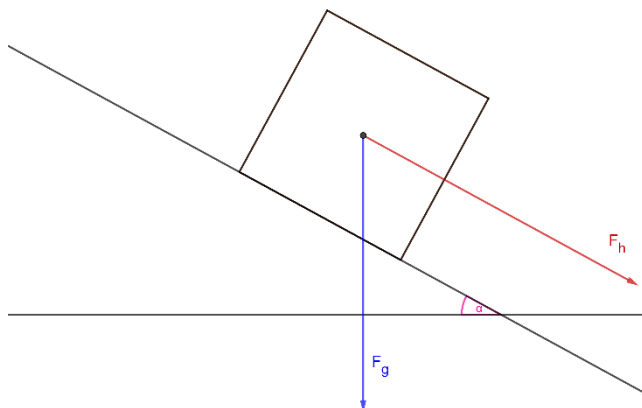
$$F_g = m \cdot g$$



3.7.2 Hangabtriebskraft

Die Hangabtriebskraft \vec{F}_h ist eine auf den Körper wirkende Kraft, welche parallel zur schiefen Ebene hangabwärts gerichtet ist. Diese ist definiert durch die Gewichtskraft F_g und den Winkel der schiefen Ebene zur Horizontalen α wie folgt:

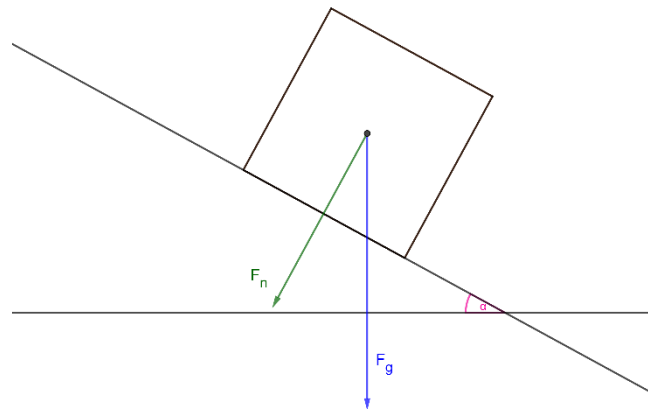
$$\vec{F}_h = \vec{F}_g \cdot \sin \alpha$$



3.7.3 Normalkraftkomponente

Die Normalkraftkomponente steht immer senkrecht zur Schiefen Ebene nach unten und beschreibt, mit welcher Kraft der Körper auf die schiefe Ebene gedrückt wird. Sie besteht aus der Gewichtskraft \vec{F}_g und dem Winkel der schiefen Ebene zur Horizontalen α .

$$\vec{F}_N = \vec{F}_g \cdot \cos \alpha$$



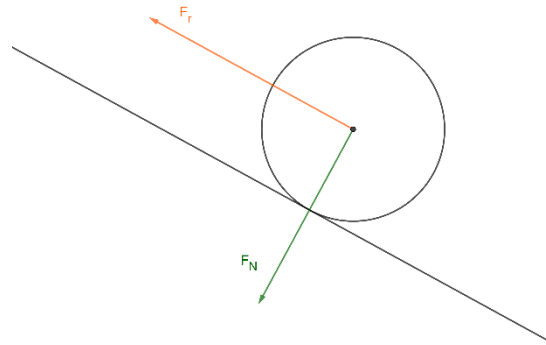
Liegt der Körper auf einer horizontalen Ebene so gilt:

$$\vec{F}_N = \vec{F}_g$$

3.7.4 Rollreibungskraft

Die Rollreibung \vec{F}_r wirkt auf einen Körper, wenn eines seiner runden Kreisteile auf einer schiefen Ebene rollt. Sie wirkt immer entgegengesetzte der Geschwindigkeit \vec{v} . Da diese proportional zur Normalkraft ist gilt:

$$\vec{F}_r = |\vec{F}_N| \cdot c_r$$



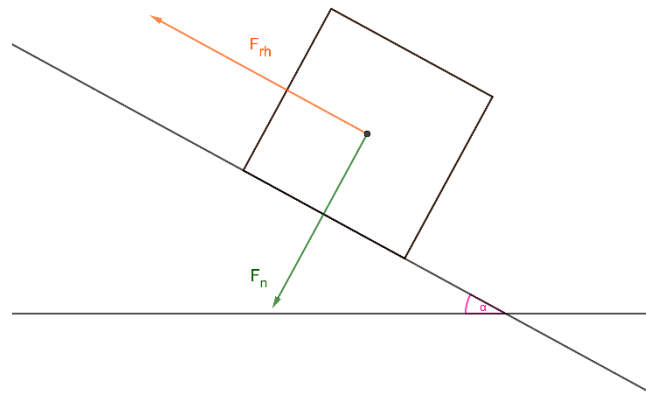
c_r ist der Rollwiderstandskoeffizient. Dieser hängt von der Materialkombination von Körper und schiefer Ebene ab.

3.7.5 Haftreibung

Durch die Haftreibung \vec{F}_{rh} wird das Rutschen eines Körpers auf einer Oberfläche verhindert. Sie tritt auf, wenn sich zwei Körper relativ zueinander nicht bewegen. Diese gilt, solange $|\vec{F}_{rh}| \geq |\vec{F}_h|$ ist. Gilt dies nicht, so greift die Gleitreibung.

Definiert ist sie die Haftreibungskonstante μ_h und der Normalkraftkomponente \vec{F}_n .

$$\vec{F}_{rh} = \mu_h \cdot \vec{F}_n$$

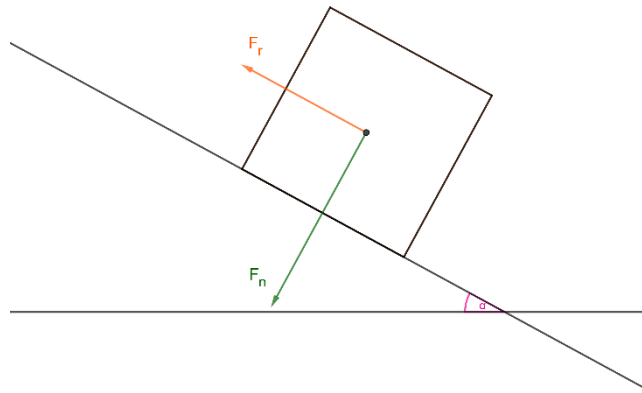


μ_h ist abhängig von der Materialpaarung von den beiden Körpern.

3.7.6 Gleitreibung

Bewegt sich zwei Körper relativ zueinander, so entsteht zwischen ihnen Gleitreibung \vec{F}_r . Diese ist definiert durch die Materialpaarung abhängigen Gleitreibungskoeffizienten μ und der Normalkraftkomponenten \vec{F}_n .

$$\vec{F}_r = \mu \cdot \vec{F}_n$$



Dies gilt, solange $|\vec{F}_r| < |\vec{F}_h|$. Andernfalls gilt die Haftreibung.

3.7.7 Resultierende Kraft

Die resultierende Kraft \vec{F}_{res} gibt die Differenz der in Bewegungsrichtung wirkenden Kräfte und der entgegengesetzten Kräfte an. Sie zeigt an, mit welcher Kraft ein Körper schlussendlich bewegt

4 Physikalische Simulation starrer Körper

4.1 Das physikalische System

Das physikalische System ist im Physolator der Ausgangspunkt jeder Simulation. Dort wird die Simulation initialisiert und es ist auch die Wurzel des Komponenten-Baums. In diesem Fall erweitert die Simulation starrer Körper daher die Klasse „physikalisches System“ des Physolators. Dies ermöglicht es Methoden, die alle physikalischen Systeme haben, zu überschreiben. Das wird später nochmal aufgegriffen. Außerdem erlaubt es einem die Grafikkomponenten zu initialisieren und das User-Interfaces des Physolators zu gestalten.

4.2 Ein starrer Körper

Ein einzelner starrer Körper in der Simulation ist eine Instanz der Klasse starrer Körper. Die Klasse starrer Körper beinhaltet die Attribute und Methoden, die zur Simulation der starren Körper benötigt werden. Der starre Körper muss im Komponenten-Baum der Simulation sein. Dafür setzt man eine Referenz als Attribut in das physikalische System.

4.2.1 Attribute starrer Körper

Zur eindeutigen Bezeichnung der Körper bekommt ein starrer Körper eine eindeutige ID eine UIN.

Der starre Körper verfügt für die physikalische Simulation über Masse, Ort, Geschwindigkeit, Beschleunigung, Trägheitsmoment, Winkel, Winkelgeschwindigkeit und Winkelbeschleunigung. Über Annotations wird dem Physolator bekannt gegeben, wie die Ableitungsverhältnisse zwischen den physikalischen Größen sind damit dieser zur Laufzeit entsprechende Werte errechnen kann. Die Ableitungsverhältnisse sind dabei $\vec{a}(t) = \dot{\vec{v}}(t) = \ddot{\vec{r}}(t)$ und $\alpha(t) = \dot{\omega}(t) = \ddot{\varphi}(t)$ wie bereits im Grundlagen-Teil ausgeführt.

4.2.1.1 Form

Außerdem besitzt jeder Körper eine Form deren Massemittelpunkt durch den Ort definiert ist. Diese Form bezieht sich dabei auf das Koordinatensystem, das durch Ort und Winkel gegeben ist. Die Form wird über eine eigene abstrakte Form-Klasse dargestellt, von welcher spezielle Formen abgeleitet werden können. Im Rahmen der Entwicklung des Projektes wurden Kreisformen und Polygonformen implementiert. Jede

abgeleitete Klasse muss die Methoden zur Bestimmung des Trägheitsmomentes und die Methode zum Zeichnen der Form anbieten.

Zur Vereinfachung wurde die Festlegung getroffen, dass die Masse in einer Form gleichverteilt ist. Das erleichtert spätere Berechnungen.

4.2.1.1.1 Kreis

Der Schwerpunkt eines Kreises liegt immer in der Mitte, wenn die Masse, wie hier, gleichverteilt ist. Der Kreis wird definiert über seinen Radius.

Die Methode zum Zeichnen, zeichnet einen Kreis mit entsprechendem Radius, und außerdem eine Linie, durch welche die Drehbewegung des Kreises veranschaulicht wird. Die Methode zur Bestimmung des Trägheitsmomentes bestimmt nach der Formel $I = m \cdot r^2$ das Trägheitsmoment des Kreises.

4.2.1.1.2 Polygon

Das Polygon wird definiert über eine Liste an geordneten Punkten, die zusammen eine geschlossene Form ohne Löcher bilden. Dabei wird der letzte Punkt mit dem ersten verbunden.

Die Methode zum Zeichnen zeichnet dann das Polygon, indem sie alle Punkte der Reihe nach durch Linien verbindet. Die Methode zur Bestimmung des Trägheitsmomentes ist in diesem Fall komplizierter als beim Kreis. Zur Bestimmung muss integriert werden. Dafür wird zuerst der Massemittelpunkt bestimmt. Dieser entspricht bei gleichverteilter Masse dem geometrischen Schwerpunkt der Form. Der geometrische Schwerpunkt einer Menge an Punkten ist definiert als die Summe aller Punkte geteilt durch die Anzahl an Punkten. Somit lässt sich der geometrische Schwerpunkt numerisch annähern, indem man die Form gerastert abtastet und in eine Liste von Punkten überführt, die in der Form liegen. Die Abtastweite ist dabei in x und y Richtung gleich klein gewählt. Je kleiner man die Abtastweite wählt, desto präziser die Näherung. Letztendlich bestimmt man dann einfach den geometrischen Schwerpunkt dieser Punkte.

Zur Bestimmung des Trägheitsmomentes muss zuerst die Masse der einzelnen Abtastpunkte bestimmt werden. Diese entspricht der Masse des Körpers geteilt durch die Anzahl an Abtastpunkten. Wie in den Grundlagen erklärt gilt $I = m \cdot a^2$ für einen einzelnen Punkt. Somit lässt sich numerisch integrieren indem für alle Punkte $I = m \cdot r^2$

(mit m als der Abtastpunktmasse, und r als dem Abstand eines der Abtastpunkte zum geometrischen Schwerpunkt) bestimmt wird. Die Summe aller dieser Trägheitsmomente für jeden Abtastpunkt entspricht dann ungefähr dem Trägheitsmoment des gesamten Körpers. Zuletzt werden die formgebenden Punkte noch so verschoben, dass der geometrische Schwerpunkt zugleich dem Ursprung des lokalen Koordinatensystems der Form entspricht.

4.2.1.1.3 Weitere Formen

Weitere Formen sind alle vom Polygon abgeleitet und umfassen Dreiecke, Vierecke, perfekte Pentagone und Hexagone mit entsprechenden Konstruktoren zur einfachen Erstellung dieser Formen. Dabei wurden einfache geometrische Gesetze genutzt.

4.2.1.2 Status

Die Variable `state` eines Körpers zeigt seinen Aktuellen Status an.

4.2.1.2.1 FLYING – Fliegen

Ist der Status auf FLYING gesetzt, so bewegt sich der Körper durch die Luft, ggf. mit Gravitation.

4.2.1.2.2 STOPPED – Stoppen

Der Status STOPPED zeigt an, dass sich der Körper nicht mehr bewegt. Er steht also still.

Dabei werden translatorische Bewegung und Beschleunigung sowie Rotationsgeschwindigkeit und -beschleunigung auf 0-gesetzt.

4.2.1.2.3 ROLLING – Rollen

Das Rollen wird durch den Status ROLLING signalisiert.

4.2.1.2.4 SLIDING – Trockene Reibung

Erfährt ein Körper trockene Reibung so wird dies durch den Status SLIDUNG dargestellt. Körper in diesem Status erfahren keine Rotationsgeschwindigkeit und -beschleunigung.

4.2.2 Berechnung der Energie

Ein starrer Körper hat eine kinetische Energie, die sich aus seiner Rotationsenergie und Translationsenergie zusammenfügt. Es gilt $E = E_r + E_t = \frac{1}{2} \cdot I \cdot \omega^2 + \frac{1}{2} \cdot m \cdot v^2$.

Die Energie wird daher bei jedem Aufruf der überschriebenen Physolator-Methode `f`

neu berechnet. Die Methode `f` wird vom Physolator automatisch aufgerufen, wenn ein neuer physikalischen Zustand berechnet wird.

4.3 Mehrere starre Körper

Die Simulation eines Systems mit mehreren starren Körpern stellt Herausforderungen, die über die eines einzelnen starren Körpers weit hinausgehen. Zuerst wurde dazu im physikalischen System ein Array an starren Körpern angelegt. Dieses ersetzt die einfache Referenz auf einen einzelnen starren Körper. Dies ermöglicht es eine beinahe beliebige Zahl an starren Körpern zu simulieren. Doch wie interagieren diese miteinander? Effekte wie Stöße, Rollen und Reibungskräfte erschweren eine einfache Implementierung.

4.3.1 Stöße

Den Stoß zwischen zwei Körpern physikalisch korrekt durchzuführen ist eine komplexe Angelegenheit. Zuerst wurde dazu im physikalischen System die Methode `g` überschrieben. Die Methode erwartet das Auslösen eines Events, woraufhin sie dieses durch Intervallschachtelung in der Zeit genau annähert. Dafür müssen alle Kombinationen aus starren Körpern stets auf Stöße geprüft werden, und im Stoßfall ein Event ausgelöst werden. Im Folgenden Beispiel sieht man wie das Array der starren Körper in der Methode `g` so durchsucht wird, dass jede Kombination genau einmal geprüft wird und kein Körper einen Stoß mit sich selbst prüft.

```
for (int i = 0; i < rigidBodies.length - 1; i = i + 1) {  
    for (int j = rigidBodies.length - 1; j > i; j = j - 1) {  
        rigidBodies[i].collisionWithRigidBodyCheck(aed,  
            rigidBodies[j], t, rigidBodies);  
    }  
}
```

Beispielsweise für ein Array der Länge 4 kämen folgende Index Kombinationen vor:

0-3 0-2 0-1

1-3 1-2

2-3

Alle weiteren Kombinationen wären überflüssige Doppelungen

4.3.1.1 Stoßerkennung

Wie im oberen Beispielcode schon ersichtlich hat jeder starre Körper eine Methode zu der Überprüfung auf einen Stoß mit einem anderen Körper. Diese Methode überprüft dann, ob Körper A in Körper B eindringt und erzeugt im Falle des Eindringens eine ausführbare „Runnable“ Klasse. Daraufhin wird ein Event ausgelöst und die „Runnable“ Klasse mit Informationen zur Stoßauflösung wird an das Physolator System übergeben:

```
if (this.in(r2)) {  
    Runnable handler = new RigidBodyCollisionHandler(impactpoint(r2));  
    aed.reportEvent(handler, "collision of rigidbodies: ",  
        this.toString(), r2.toString());  
}
```

Wie also erkennt die Methode „in“ ob zwei starre Körper ineinander gedrungen sind? Dafür wurde die Überlegung getroffen, dass zwei starre Körper genau dann Stoßen, wenn sich die Formen schneiden. Das heißt, wenn sich Kanten mit Kanten oder Kreise mit Kreisen oder Kreise mit Kanten schneiden. Dafür werden zuerst die Werte aus den Formen entnommen und aus dem lokalen Form-Koordinatensystem ins Koordinatensystem des physikalischen Systems überführt (Transformation mit Rotationswinkel und Translationspunkt des Körpers).

Im Fall Polygon trifft auf Polygon muss für jede Kombination von Kanten überprüft werden, ob diese sich schneiden. Die Implementierung kann dann so aussehen:

```
for (Line2D.Double a : edges1)  
    for (Line2D.Double b : edges2)  
        if (a.intersectsLine(b))  
            return true;  
return false;
```

Im Fall Kreis trifft auf Kreis reicht es zu prüfen, ob die Distanz zwischen den Kreismittelpunkten größer ist als die Summe der Radien.

```
double distance = VectorMath.sub(this.r, r2.r).abs();  
if ((circleShape_r1.radius + circleShape_r2.radius) >= distance)  
    return true;  
else  
    return true;
```

Und im Fall, dass ein Kreis auf ein Polygon trifft wird überprüft, ob der Abstand zwischen den Kanten des Polygons und dem Kreis kleiner ist als der Radius des Kreises. In diesem Fall sind sie ineinander eingedrungen. Im Folgenden Codebeispiel steht die Methode „ptSegDist“ für den Abstand zwischen der Kante und dem Punkt.

```
double r = circleShape_r1.radius;

for (Line2D.Double edge : edges)
    if (edge.ptSegDist(this.r.x, this.r.y) <= r)
        return true;
return false;
```

4.3.1.2 Stoßkoordinatensystem

Hat man den Stoß erkannt, will man das Stoßkoordinatensystem definieren. Man muss sich klar machen, dass zu diesem Zeitpunkt, durch die Intervallschachtelung der g-Methode des Physolators, die Körper nichtmehr ineinander dringen, sondern dieser Moment genau zeitlich bevorsteht. Das Stoßkoordinatensystem hat als Ursprung den Stoßpunkt bzw. die Stoßecke, und die Stoßkante bzw. Stoßebene gibt die Ausrichtung der y-Achse an. Damit ist auch die Ausrichtung der orthogonalen x-Achse bekannt. Es gilt also diese beiden Werte genau zu bestimmen.

Im „Polygon trifft auf Polygon“ Fall wird ein Algorithmus verwendet, der für jede Kombination von Ecken und Kanten beider Körper den Abstand bestimmt. Wenn dieser Abstand kleiner ist als der bisher kleinste Abstand, so werden Ecke und Kante als Stoßecke und Stoßkante gesetzt. Der kleinste Abstand wird mit dem größtmöglichen Wert initialisiert, so dass jeder Abstand kleiner sein muss. Am Ende der Schleifendurchläufe bleiben daher die echte Stoßebene und der Stoßpunkt übrig. Es werden zwei Doppel-For-schleifen benötigt, da sowohl die Ecken des ersten Körpers mit den Kanten des zweiten Körpers als auch umgekehrt, getestet werden muss. Am Ende wird aus der Stoßkante noch ein einzelner Richtungsvektor erzeugt, da für das Stoßkoordinatensystem die Ausrichtung der Y-Achse allein von der Ausrichtung der Stoßkante abhängt und nicht von deren genauer Position.

```

Point2D.Double impactpoint = vertices1[0];
Line2D.Double impactededge = edges2[0];
double smallestDistance = Double.MAX_VALUE;
for (int i = 0; i < edges1.length; i++) {
    for (int j = 0; j < vertices2.length; j++) {
        double distance = edges1[i].ptSegDist(vertices2[j]);
        if (distance < smallestDistance) {
            smallestDistance = distance;
            impactpoint = vertices2[j];
            impactededge = edges1[i];
        }
    }
}

for (int i = 0; i < edges2.length; i++) {
    for (int j = 0; j < vertices1.length; j++) {
        double distance = edges2[i].ptSegDist(vertices1[j]);
        if (distance < smallestDistance) {
            smallestDistance = distance;
            impactpoint = vertices1[j];
            impactededge = edges2[i];
        }
    }
}

Vector2D p = new Vector2D(impactpoint.x, impactpoint.y);
Vector2D e = new Vector2D(impactededge.x2 - impactededge.x1,
    impactededge.y2 - impactededge.y1);

```

Im Fall „Kreis trifft auf Kreis“ ist die Bestimmung deutlich einfacher. Die Stoßkante steht senkrecht zum Verbindungsvektor der Kreise und der Stoßpunkt entspricht dem Punkt, der auf dem Kreisrand liegt und auf dem Verbindungsvektor der Kreise.

```

Vector2D r1_r2 = VectorMath.sub(r2.r, this.r);
Vector2D impactEdge = VectorMath.perpendicular(this.r, r1_r2);
impactEdge.normalize();
Vector2D impactPoint = VectorMath.add(this.r,
    (VectorMath.mult(circleShape_r1.radius,
        VectorMath.normalize(r1_r2))));

```

Der Fall „Kreis trifft auf Polygon“ ist wieder etwas komplizierter. Hier muss man unterscheiden zwischen einem Kreis, der auf eine Kante stößt, und einem Kreis, der auf eine Ecke stößt. In ersterem Fall ist die Kante die Stoßkante und der nächste

Kreis zum Stoßpunkt und in letzterem Fall ist die Ecke der Stoßpunkt und die Tangente am Kreis die Stoßebene.

```
double smallestDistanceToLine = Double.MAX_VALUE;
Line2D.Double impactEdge = edges[0];
for (Line2D.Double edge : edges) {
    if (edge.ptSegDist(this.r.x, this.r.y) <= smallestDistanceToLine) {
        smallestDistanceToLine = edge.ptSegDist(this.r.x, this.r.y);
        impactEdge = edge;
    }
}
for (Point2D.Double vertex : vertices) {
    if (vertex.distance(this.r.x, this.r.y) <= smallestDistanceToLine) {
        // Case: hit vertex
        Vector2D impactPoint = new Vector2D(vertex.x, vertex.y);
        Vector2D impactPoint_circle = VectorMath.sub(this.r,
            impactPoint);
        Vector2D trueImpactEdge = VectorMath.perpendicular(impactPoint,
            impactPoint_circle);
        return new Impactpoint(impactPoint, trueImpactEdge, r2, this);
    }
}
// Case: hit edge
Vector2D trueImpactEdge = new Vector2D(impactEdge.x2 - impactEdge.x1,
    impactEdge.y2 - impactEdge.y1);
Vector2D impactPoint = VectorMath.footOfPerpendicular(this.r,
    new Vector2D(impactEdge.x1, impactEdge.y1), trueImpactEdge);
return new Impactpoint(impactPoint, trueImpactEdge, this, r2);
```

4.3.1.3 Stoßauflösung

Die Stoßauflösung erfolgt unter Anwendung der Formeln von Herr Wandinger die im Grundlagenteil bereits vorgestellt wurden. Bei Aufruf der „run“ Methode der „Runnable“ Klasse zur Auflösung von Stößen werden zuerst alle Geschwindigkeiten und Positionen der zwei Körper in das Stoßkoordinatensystem transformiert. Danach werden aus den neuen Positionen die Werte a_1 und a_2 entnommen, die deren Abstand zur Y-Achse entsprechen, und somit ihr negativer Y-Wert sind. Dann wird der Kraftstoß berechnet und damit die neuen Geschwindigkeiten und Winkelgeschwindigkeiten. Zuletzt werden die Geschwindigkeiten zurück ins Inertialsystem transformiert. Danach ist der Stoß gelöst. Der k -Wert ist in dieser Klasse gespeichert.

```

Vector2D r1mr = rotateVector2D(r1m, rot);
Vector2D r2mr = rotateVector2D(r2m, rot);

Vector2D v1r = rotateVector2D(rb_p.v, rot);
Vector2D v2r = rotateVector2D(rb_e.v, rot);

double a1 = -r1mr.y;
double a2 = -r2mr.y;

double Fx = impulseFx(v1r.x, v2r.x, a1, a2);

Vector2D V1r = new Vector2D(v1r.x - (Fx / rb_p.m), v1r.y);
Vector2D V2r = new Vector2D(v2r.x + (Fx / rb_e.m), v2r.y);

double Omega1 = rb_p.omega + ((a1 * Fx) / rb_p.I);
double Omega2 = rb_e.omega - ((a2 * Fx) / rb_e.I);

Vector2D V1r_ = rotateVector2D(V1r, -rot);
Vector2D V2r_ = rotateVector2D(V2r, -rot);

```

```

private double impulseFx(double v1x, double v2x, double a1, double a2) {
    double a1omega1 = a1 * rb_p.omega;
    double a2omega2 = a2 * rb_e.omega;
    double zaehler = v1x - v2x - a1omega1 + a2omega2;
    double nenner = (1 / rb_p.m) + (1 / rb_e.m) + (a1 * a1 / rb_p.I) +
                    (a2 * a2 / rb_e.I);
    double Fx = (1 + k) * (zaehler / nenner);
    return Fx;
}

```

4.3.1.4 Probleme beim Stoß

Während und nach der Implementierung von Stößen sind mehrere Probleme aufgefallen. Spezialfälle wie Ecke auf Ecke und Kante auf Kante Stöße führen zu unerwarteten, nicht physikalischen Effekten. Stößt z.B. eine Kante auf eine Kante mit keinem oder einem nur sehr geringen Winkel dann kann es zu einem extremen Aufschaukeln kommen wo die Kanten ständig aneinanderstoßen. Insbesondere wenn die Stoßzahl k klein ist und das System bei jedem Stoß an Energie verliert oder die Körper sich unter dem Einfluss von Kräften wie z.B. der Gewichtskraft befinden. Fallen z.B. starre Körper auf einen unbeweglichen Körper und verlieren dabei ständig an Energie so wird der Abstand zwischen den Stößen logischerweise immer kleiner. Dabei kommt es zu einer Überlastsituation die den Physolator oder den Computer, der ihn ausführt, zu stark beanspruchen. Man kann die Körper in so einem Fall in einen anderen Zustand überführen wie z.B. Rollen, Gleitreibung oder Stillstand. Außerdem besteht die Gefahr von Mehrfachstößen. Mehrfachstöße sind Fälle in denen zeitgleich mehr als zwei Körper aneinander stoßen. Solche Fälle treten häufiger auf als man das zuerst annehmen

würde. Ein typischer Fall wäre z.B. ein Trichter auf den starren Körper fallen. Dabei kommt es fast unweigerlich zu solchen Mehrfachstößen, die diese Simulation noch nicht lösen kann.

4.3.2 Rollen

Rollen ist eine Kombination aus translatorischer und Rotationsbewegung, welche auf runde Körper gleichzeitig angewendet werden. Dies betrifft bei uns Kreise.

4.3.2.1 Zustandsübergang Fliegen zu Rollen

Ein runder Körper wechselt seinen Zustand von fliegen zu rollen, wenn die vertikale Geschwindigkeit nach einem Stoß gleich 0 ist.

Da der Stoß in einem Stoßkoordinatensystem berechnet wird, betrachten wir hierfür stattdessen die darin berechnete horizontale Geschwindigkeit. Diese entspricht der vertikalen Geschwindigkeit im Inertialsystem. Da dieser Wert nie exakt 0 sein wird, haben wir uns für einen Schwellwert von 0,1 entschieden. Wenn die Neuberechnete horizontale Geschwindigkeit v'_x kleiner als $0,1 \frac{m}{s}$ ist, wechselt der runde Körper in den Zustand Rollen. Dadurch erfährt er neue physikalische Zusammenhänge, welche sich auf das Inertialsystem beziehen.

Zunächst wird unabhängig von der Art der Ebene der Körper um 0,1 mm nach oben gesetzt. Somit ist gewährleistet, dass der Körper keine weiteren Stöße mit der Ebene eingeht.

Tritt dieser Fall ein, so unterscheiden wir, ob der Stoß mit einer horizontalen oder einer schiefen Ebene durchgeführt wurde.

4.3.2.1.1 Horizontale Ebene

Findet dieser Zustandswechsel auf einer horizontalen Ebene statt, so wird der horizontale Anteil der Rollreibung \vec{F}_{r_x} bestimmt. Der vertikale Anteil davon \vec{F}_{r_y} , sowie von der Geschwindigkeit \vec{v}_x und Beschleunigung \vec{a}_x ist 0, da sich bei einem rollenden Körper auf einer horizontalen Ebene die Geschwindigkeit nur in horizontaler Richtung ändert.

Nun wenden wir das zweite Newtonsche Axiom $\vec{F} = m \cdot \vec{a}$ auf die soeben berechnete x-Komponente der Rollreibung an und formen diese nach a_x um. Somit haben wir die

letzte benötigte Komponente für die translatorische Bewegung des rollenden Körpers bestimmt. Geschwindigkeit \vec{v} und Position \vec{r} bestimmt der Physolator durch die Ableitungsbeziehungen zwischen \vec{r} , \vec{v} und \vec{a} selbstständig.

Der Code hierfür:

```
rb.Fr.x = -9.81 * rb.m * friction * (-signum(rb.v.x));

rb.v.y = 0;

rb.a.set(rb.Fr.x / rb.m, 0);
```

Um nun die Rotationsbewegung korrekt bestimmen zu können, genügt es uns, die Winkelbeschleunigung α zu berechnen. Hierzu nutzen wir den physikalischen Zusammenhang aus der Winkelbeschleunigung α , der Beschleunigung a und dem Kreisradius r .

$$\alpha = \frac{|\vec{a}|}{r}$$

Die Rotationsgeschwindigkeit ω sowie der Drehwinkel φ werden aus den Ableitungsbeziehungen zu α bestimmt.

4.3.2.1.2 Stoppen

Ein Körper kann in seiner Rollbewegung zum Stoppen kommen. Dies passiert, wenn er auf einer Horizontalen die Geschwindigkeit 0 erlangt. In diesem Fall erlangt er den Status STOPPED.

```
if (direction == BodyDirection.LEFT && v.x > 0 ||
    direction == BodyDirection.RIGHT && v.x < 0) {
    state = BodyState.STOPPED;
}
```

4.3.2.1.3 Schiefe Ebene

Findet der Zustandswechsel von Fliegen zu Rollen stattdessen auf einer schiefen Ebene statt, so betrachten wir die Differenz auf Hangabtriebskraft \vec{F}_h und der Rollreibung \vec{F}_r , welche die resultierende Kraft \vec{F}_{res} ergibt.

$$\vec{F}_{res} = \vec{F}_h - \vec{F}_r$$

Diese zeigt an, mit welcher Kraft der Kreis die schiefe Ebene nach unten rollt.

4.3.2.1.4 Bestimmung der Hangabtriebskraft

Um diese durchzuführen wird zunächst die Hangabtriebskraft nach 3.7.2 bestimmt. Zuvor ist eine Fallunterscheidung notwendig. Je nach Neigungsrichtung der der schiefen Ebene, muss al erstes die Gewichtskraft entweder im oder gegen den Uhrzeigersinn gedreht werden. Dieser Vektor ist nun parallel zur Schiefen Ebene abwärtsgerichtet. `angle` beschreibt den Winkel der schiefen Ebne zur horizontalen.

```
if (collisionEdge.x > 0 && collisionEdge.y > 0)
    rb.Fh.set(rotateVector2D(rb.Fg, toRadians(-90) + angle));
else if (collisionEdge.x > 0 && collisionEdge.y < 0)
    rb.Fh.set(rotateVector2D(rb.Fg, toRadians(-90) - angle));
```

Im Folgenden wird dieser Vektor normiert und der Variable `FhN` zugewiesen.

```
Vector2D FhN = VectorMath.normalize(rb.Fh);
```

Nun wird mit Hilfe der Länge des Vektors `FgA` und Definition der Hangabtriebskraft die Länge deren Vektors `FgAsin` bestimmt.

```
double FgA = VectorMath.abs(rb.Fg);
double FgAsin = FgA * sin(angle);
```

Diese wird nun mit Hinzunahme des normierten Vektors der Hangabtriebskraft multipliziert. Daraus erhält man die Hangabtriebskraft des rollenden Körpers.

```
rb.Fh.set(VectorMath.mult(FgAsin, FhN));
```

4.3.2.1.5 Bestimmung der Normalkraftkomponente

Im folgenden Schritt wird die in 3.7.3 definierte Normalkraftkomponente bestimmt. Hierzu wird zunächst der orthogonale Vektor zur schiefen Ebene bestimmt. Dazu wird der Vektor der Gewichtskraft um den zuvor bestimmten Winkel `-angle` gedreht. Das Ergebnis wird bereits der Variable der Gewichtskraft zugewiesen und sogleich normiert.

```
rb.Fn.set(rotateVector2D(rb.Fg, -angle));
Vector2D FnN = VectorMath.normalize(rb.Fn);
```

Nun wird, der in 4.3.2.1.5 bestimmte Betrag der Gewichtskraft F_{gA} erneut verwendet, um die Länge des Vektors der Normalkraftkomponente. Hierzu wird die Definition aus 3.7.3 verwendet.

```
double FgAcos = FgA * cos(angle);  
rb.Fn.set(VectorMath.mult(FgAcos, FnN));
```

4.3.2.1.6 Bestimmung der Rollreibung

Für die Bestimmung der Rollreibung wird nun die die Länge F_{nA} , also der Betrag, der Normalkraftkomponente ermittelt. Nun müssen die die x- und y-Komponente (F_{rx} und F_{ry}) separat bestimmt werden. Dies geschieht durch, die in der Kräftezerlegung erkennbaren, trigonometrischen Funktionen. Da die Reibung zu Beginn des Status ROLLING in die entgegengesetzte Richtung der Hangabtriebskraft zeigt, werde die soeben ermittelten Komponenten mit Hilfe der `-signum()`-Funktion und der entsprechenden Komponente der Hangabtriebskraft multipliziert.

F_{rx} und F_{ry} können nun dem Vektor F_r des Körpers zugewiesen werden.

```
double FnA = VectorMath.abs(rb.Fn);  
double Frx = FnA * cos(angle) * friction * (-  
    signum(rb.Fh.x));  
double Fry = FnA * sin(angle) * friction * (-  
    signum(rb.Fh.y));  
rb.Fr.set(Frx, Fry);
```

4.3.2.1.7 Bestimmung der resultierenden Kraft

Die resultierende Kraft wird nach der Definition in 3.7.7 errechnet.

Die als Kräfte in Bewegungsrichtung gilt ausschließlich die Hangabtriebskraft.

In entgegengesetzte Richtung wirkt stattdessen nur die Rollreibung.

```
rb.Fres.set(VectorMath.sub(rb.Fh, rb.Fr));
```

4.3.2.1.8 Bestimmung der Rotationsbewegung

Um die Rotationsbewegung zu bestimmen, muss ausschließlich die Winkelbeschleunigung α errechnet werden. Hierfür wird der Betrag der der Beschleunigung durch den Radius des Kreises geteilt.

```
rb.alpha = rb.a.abs() / rb.shape.getRadius();
```

Rotationsbeschleunigung und Drehwinkel werden durch die Ableitungsbeziehungen selbstständig bestimmt.

4.3.2.1.9 Bestimmung der translatorischen Bewegung

Zum Schluss wird nun die Translation bestimmt. Dafür wird das zweite Newtonsche Axiom verwendet und nach der Beschleunigung umgeformt.

$$a = \frac{F}{m}$$

Für die Kraft F wird die resultierende Kraft F_{res} eingesetzt. m ist die Masse des rollenden Körpers. Dies muss für diese Rechnung für die x- und y-Komponente separat durchgeführt werden. Somit erhält man den Code:

```
rb.a.x = rb.Fh.x / rb.m;  
rb.a.y = rb.Fh.y / rb.m;
```

Beschleunigung und Position werden wiederum durch die Ableitungsbeziehungen selbstständig ermittelt.

4.3.2.2 Richtungswechsel

Rollt der Körper eine schiefe Ebene nach oben, so wird er durch die Hangabtriebskraft ausgebremst und ändert seine Bewegungsrichtung, wenn die x-Komponente der der Geschwindigkeit v ihr Vorzeichen ändert.

Dieses Verhalten wurde innerhalb der $f()$ -Methode des `RigidBodies` implementiert.

Hierbei wechseln die x-Komponente der Hangabtriebs und die x-Komponente der Rollreibung ihre Vorzeichen.

4.3.3 Trockene Reibung

Grundsätzlich gilt, dass nur Körper mit Kanten trockene Reibung erfahren können. Im Zustand der trockenen Reibung gewinnt ein Körper keine Rotation. Höchstens kann er eine Translatorische Bewegung parallel zum Untergrund erfahren.

Analog zur Rollbewegung wirken auf den Körper mit trockener Reibung ebenso die Gewichtskraft, Normalkraft und Hangabtriebskraft. Statt der Rollreibung wirkt hingegen die in 3.7.5 beschriebene Haftreibung oder die in 3.7.6 definierte Gleitreibung.

4.3.3.1 Zustandsbestimmung Fliegen zu trockener Reibung

Ein Körper wechselt seinen Zustand von fliegen zu trockener Reibung kann sich nur bilden, wenn eine seiner Kanten parallel auf eine weitere, eines anderen Körpers, trifft. Zusätzlich muss der resultierende Geschwindigkeitsanteil nach dem Stoß, senkrecht relativ zur Ebene 0 sein.

Da der Stoß in einem Stoßkoordinatensystem berechnet wird, betrachten wir hierfür wieder horizontale Geschwindigkeit nach dem Stoß. Da dieser Wert nie exakt 0 sein wird, haben wir uns für einen Schwellwert von 0,15 entschlossen.

Im nächsten Schritt wird überprüft, ob eine Kante des Körpers parallel zur schiefen Ebene ist. Hierzu werden die beiden benachbarten Eckpunkte (B, C) der Stoßecke (A), welche gleichzeitig der Stoßpunkt ist, des Körpers ermittelt.

Im nächsten Schritt werden die Vektoren zwischen Stoßpunkt und der jeweiligen benachbarten Ecke ermittelt:

$$\overrightarrow{ab} = \vec{b} - \vec{a}$$

$$\overrightarrow{ac} = \vec{c} - \vec{a}$$

Nun werden die Winkel $\angle \overrightarrow{ab}\vec{s} = \alpha_1$ und $\angle \overrightarrow{ac}\vec{s} = \alpha_2$ zwischen den soeben berechneten Vektoren und dem Richtungsvektor \vec{s} des Untergrunds ermittelt. Sie werden im Gradmaß angegeben.

Ferner werden diese beiden Winkel überprüft, ob einer dieser gleich 0° oder 180° ist. Bei der Parallelität spielt es nämlich keine Rolle, ob der Winkel 0° oder 180° beträgt. In beiden Fällen sind die Vektoren parallel zueinander.

Da die Körper bei diesem Vergleich optisch bereits parallel sind, mathematisch aber noch nicht, musste die Variable $\Delta = 0,5$ eingeführt werden. Sie bewirkt, dass die Winkel α_1 und α_2 einen Spielraum von $\pm 0,5^\circ$ besitzen dürfen, damit diese auch mathematisch als parallel angesehen werden. Somit ergeben sich hierfür die Intervalle

$$[-\Delta, \Delta]$$

und

$$[180 - \Delta, 180 + \Delta].$$

```

if (angleEdgeToPre >= -delta && angleEdgeToPre <= delta
    || angleEdgeToPre >= 180 - delta && angleEdgeToPre <=
180 + delta) {
    rb_p.slidingEdge = (edgeToPre);
    return true;
} else if (angleEdgeToNext >= -delta && angleEdgeToNext <=
delta || angleEdgeToNext >= 180 - delta && angleEdgeToNext
<= 180 + delta) {
    rb_p.slidingEdge = (edgeToNext);
    return true;
}

```

Zunächst wird unabhängig von der Art der Ebene der Körper um $0,1\text{ mm}$ nach oben gesetzt. Somit ist gewährleistet, dass der Körper keine weiteren Stöße ohne externe Krafteinwirkung mit der Ebene eingeht.

Im selben Zug bekommt der Körper den Zustand `SLIDING` zugewiesen, welcher bewirkt, dass die Rotationsbeschleunigung α und -geschwindigkeit ω 0-gesetzt werden. Somit behält der Körper, während er trockene Reibung erfährt, die Parallelität zu seinem Untergrund.

Ebenso wird analysiert, ob zwischen den beiden Körpern der Haftreibungskoeffizient μ_h oder der Gleitreibungskoeffizient μ wirkt. Um diese Unterscheidung vornehmen zu können wir ein Schwellwert der für den Betrag der Geschwindigkeit $|v|$ gesetzt. Dieser beträgt $0,01 \frac{m}{s}$.

Ist der Betrag der Geschwindigkeit $|v|$ größer als der Schwellwert, so wird der Gleitreibungskoeffizient zur Berechnung der hier wirkenden Gleitreibung verwendet. Andernfalls wird durch den Haftreibungskoeffizient die Haftreibung berechnet.

```

if (rb.v.abs() <= 0.01)
    friction = rb.mu_h;
else
    friction = rb.mu;

```

4.3.3.1.1 Horizontale Ebene

Die Berechnung der nötigen Parameter für eine erfolgreiche physikalische Simulation erfolgen für die trockene Reibung analog zur Rollbewegung. Diese wird in 4.3.2.1.1 erläutert.

4.3.3.1.2 Schiefe Ebene

Auch hier verläuft die Bestimmung analog zur Rollbewegung. Diese wird in 4.3.2.1.3 beschrieben.

Ist nun der Betrag der Hangabtriebskraft größer als der der Haftreibung, so bewegt sich der Körper mit wirkender Gleitreibung parallel zur schiefen Ebene herab.

Andernfalls bleibt der Körper still und es wirkt Haftreibung.

4.3.3.2 Richtungswechsel

Erfährt ein Körper trockene Reibung und bewegt sich dabei eine schiefe Ebene nach ob, so kommt dieser zu dem Punkt, dass der Betrag dessen Geschwindigkeit gleich 0 ist.

Wenn die Hangabtriebskraft an diesem Zeitpunkt größer, als die Haftreibung ist, so bewegt sich der Körper auf der Schiefen Ebene nach unten und erfährt Gleitreibung.

Ist die Hangabtriebskraft aber kleiner als die Haftreibung, so bleibt der Körper an diesem Punkt feststehen.

4.3.4 Von der Ebene Fallen

Besitzt ein Körper den Status ROLLING oder SLIDING so kann es passieren, dass dieser sich dabei über das Ende einer Ebene hinausbewegt. Ist der Körperschwerpunkt dabei außerhalb der Ebene, ändert er seinen Zustand in FLYING. Gleichzeitig wird die horizontale Beschleunigung auf 0 und die vertikale Beschleunigung auf die Gewichtskraft gesetzt.

```
rb.state = BodyState.FLYING;  
rb.a.set(0, rb.g);
```

4.3.4.1 Kreis

Ist der Körper ein Kreis so wird seine vertikale Geschwindigkeit um $0,1 \frac{m}{s}$ erhöht. Somit wird ein Trippeln an der Kante verhindert.

4.3.4.2 Polygon

Ist der Körper hingegen ein Polygon, so wird zunächst ein Wert aus der Geschwindigkeit und der Kantenlänge des Körpers ermittelt. Dieser Wert wird im Anschluss der vertikalen Geschwindigkeit zugewiesen, um ebenso ein Trippeln zu verhindern.

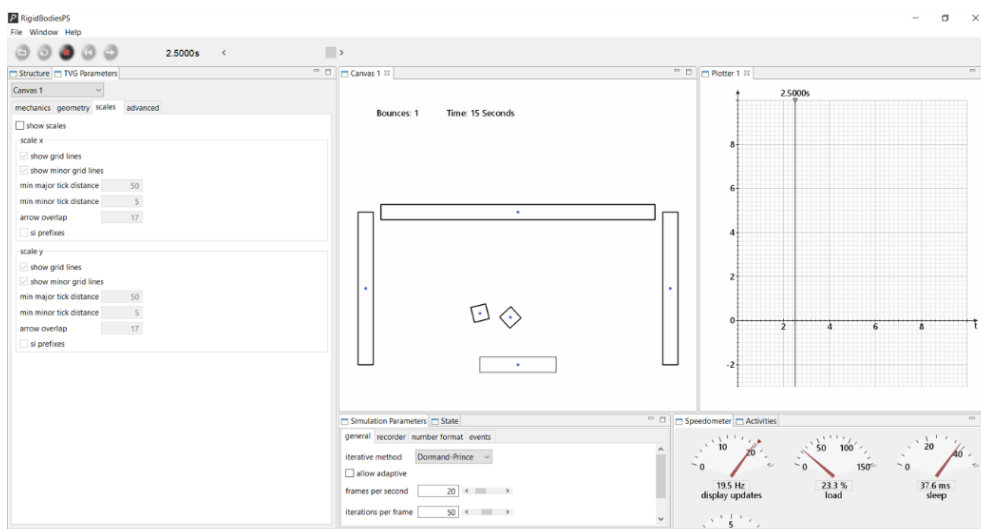
Außerdem wird das Kippen eines Polygons an einer Kante simuliert, indem dieses, je nachfallender Richtung der schiefen Ebene, entweder eine positive oder negative Rotationsgeschwindigkeit zugewiesen wird.

```
double edgeLength = rb.slidingEdge.abs();
double v = rb.v.abs();
if(edgeLength >= v) {
    rb.v.y = edgeLength/v/10;
}else {
    rb.v.y = v/edgeLength/10;
}
if (rb.r.x > impactEdge.x1 && rb.r.x > impactEdge.x2) {
    rb.omega = -1;
} else {
    rb.omega = 1;
}
```

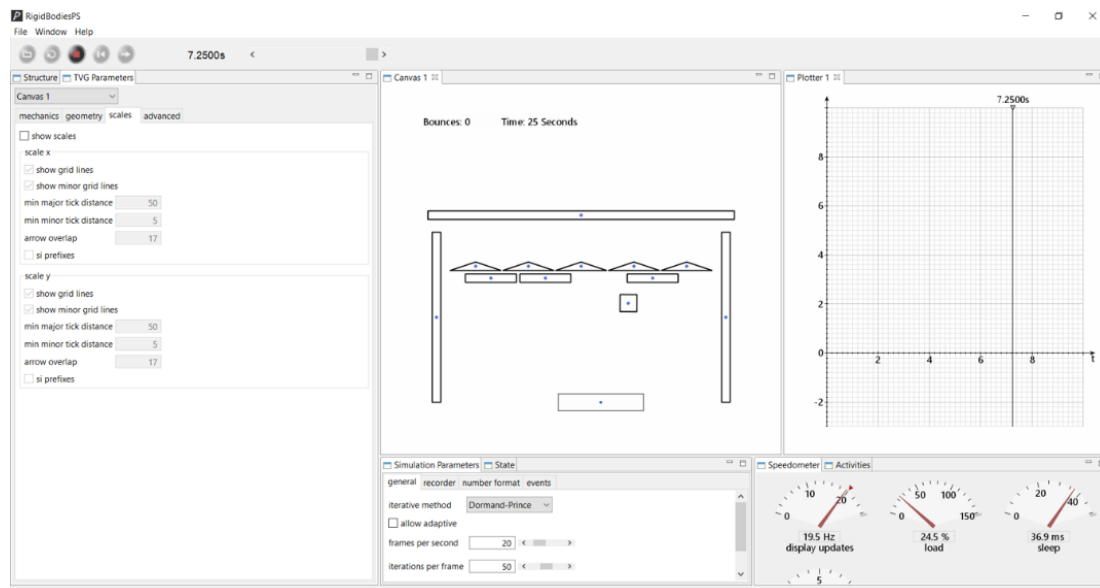
5 Spiele

Um die Funktionen der Simulation besser zur Schau stellen zu können, haben wir einige interaktive Spiele auf deren Basis entwickelt. Zu diesen zählen unter anderem Pong, Brick Breaker und eine Variante von Angry Birds.

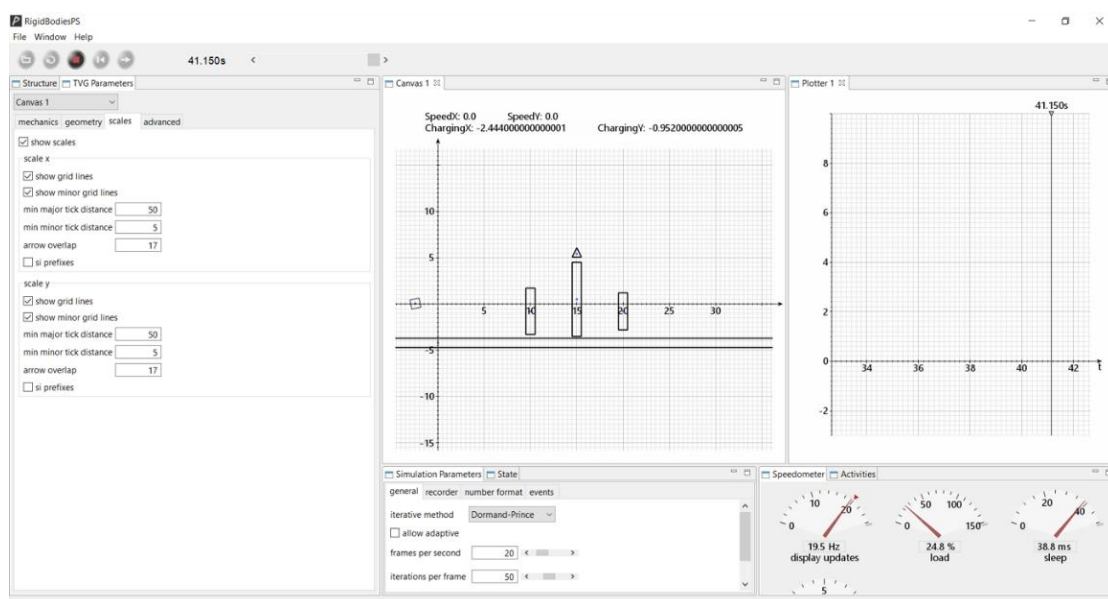
Bei unserer Version von Pong bekommt ein Ball (oder auch 2 Bälle) zunächst einen Startimpuls, anschließend werden durch das Abprallen an Wänden, einer beweglichen Plattform oder auch durch Kollision mit einem anderen Ball neue Vektoren gebildet. Ziel dieses Spiels ist, dass der Spieler den Ball mithilfe der beweglichen Plattform daran hindert, unter das Spielfeld zu fallen.



Ähnlich verhält es sich bei Brick Breaker, nur das hier zusätzlich einige verschiedenförmige Objekte platziert wurden, in deren Richtung der Spieler den Ball durch das Abprallen lenken soll. Auch hier endet das Spiel, sobald der Ball neben der beweglichen Plattform aus dem Spielfeld hinausfällt, allerdings ist es hier auch möglich, das Spiel zu beenden, indem alle der platzierten Objekte durch Berührung mit dem Ball verschwunden sind.



Bei Angry Birds hingegen kann der Spieler ein kleines Viereck mithilfe einiger Knöpfe für eine gewisse Distanz sowohl nach links als auch nach unten bewegen. Hierbei ist zu beachten, dass aus der Auslenkung an diese neue Position ein Vektor berechnet wird, welcher beeinflusst, in welche Richtung bzw. wie schnell das Viereck nach einem Druck auf den G Knopf geschossen wird. Ziel ist es hier, eine Kollision zwischen einem kleinen Dreieck und dem beweglichen Viereck herbeizuführen



Für diese Spiele werden zunächst einige Parameter wie zum Beispiel Schwerkraft, oder Reibung festgelegt, welche die Simulation der verschiedenen Objekte innerhalb des Spieles beeinflussen. Zusätzlich können verschiedene Körper, beispielsweise Quadrate oder Polygone in der Simulationsszene platziert werden. Diese besitzen verschiedene manipulierbare Parameter wie Größe, Anfangsgeschwindigkeit, Rotationen, Position sowie ihre Länge und Breite. Ein Beispiel eines solchen Körpers wäre:

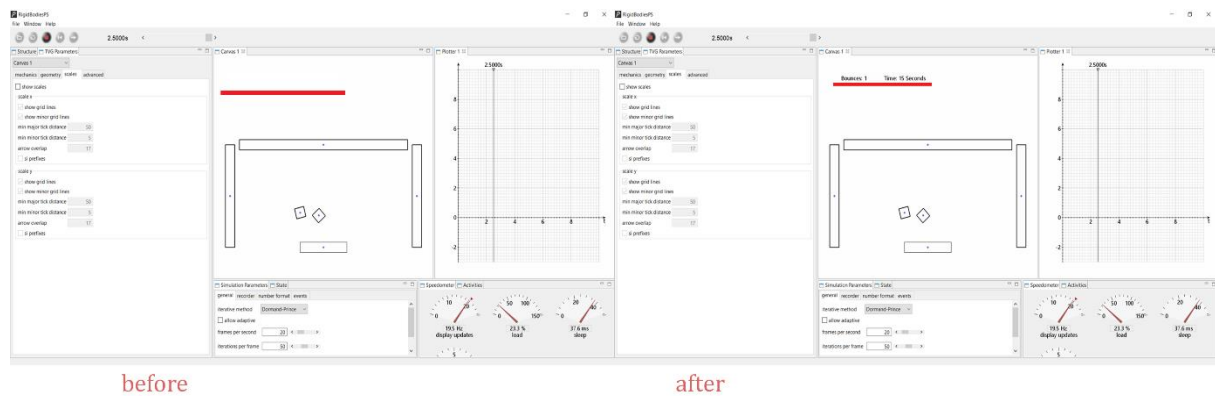
```
"rigidBodies.add(new RigidBody(Double.MAX_VALUE, new Vector2D(0, -2), new Vector2D(0, 0), new Vector2D(0, 0), Double.MAX_VALUE, 0, 0, 0, false, new Rectangle(10.5, 1)));".
```

Durch diesen Befehl wird ein rechteckiger Rigid-Body mit einer Länge von 10.5 und einer Höhe von 1 erstellt, welcher an der Position 0,-2 erscheint

Die eigentliche Spiellogik wird in einer separaten Datei festgelegt. Hierbei ist zu beachten, dass nur vorgegebene Parameter der Objekte modifiziert werden können. Eine Funktion, die dafür sorgt, dass ein RigidBody sich beispielsweise nach Links bewegt könnte folgendermaßen aussehen:

```
"public void handleAfterKeyEventAction(long millis) {  
  
    pongPS.position = startPosition - millis * 0.002;  
  
    pongPS.rigidBodies[0].r.set(pongPS.rigidBodies[0].r.x- millis * 0.002, pongPS.rigidBodies[0].r.y);"  
  
}
```

Bei Bedarf ist es möglich, bereits festgelegte Physolatorregeln oder Funktionen zu überschreiben. Wenn zum Beispiel eine Score-Anzeige eingefügt werden soll, muss die jeweils hierfür zuständige Methode (in diesem Fall die paint-Methode) so verändert bzw. erweitert werden, dass diese das neue Objekt erzeugen und darstellen kann. Dies kann mit einem @Override bewerkstelligt werden:



5.1 Pong

Pong ist ein weltbekannter Spieleklassiker aus dem Jahr 1972. Im Original treten man gegen einen Computergegner im Eins-gegen-Eins gegeneinander an. Jeder Teilnehmer bewegt eine rechteckige Plattform von oben nach unten. Mit Hilfe dieser, muss man ein Projektil abwehren, welches über das Spielfeld von links nach rechts fliegt (und zurück), um das eigene Tor zu verteidigen. Schafft man es nicht, das Projektil zum Gegner zurückzulenken, gelangt es in die Punktezone an der eigenen Plattform vorbei und der Gegner erhält einen Punkt. Wer mehr Punkte sammelt gewinnt das Match sobald die Zeit abgelaufen ist.

In unserer Version spielt man alleine im Singleplayer. Zusätzlich ist das Spielfeld so aufgebaut, dass man das Projektil von unten nach oben gegen eine Decke spielt. Die Plattform wird dementsprechend von links nach rechts gesteuert. Für die Bewegung des Projektils ist natürlich unsere Physik-Engine verantwortlich. Die bedeutet, dass Auf- und Abprallwinkel sowie Gravitation möglichst realistisch auf das Polygon einwirken, und so für einen einzigartigen Spielablauf sorgen. Zusätzlich besteht die Möglichkeit mit mehreren Projektilen gleichzeitig zu spielen, was die Schwierigkeit erhöht.

6 Organisation des Projektes durch Dimitrios Stüber

6.1 Meine Aufgaben im 2. Teil des Projektes

Wie im letzten Ausblick geplant, so wurden mir die Organisatorischen Aufgaben zugeteilt. Um welche ich mich auch gekümmert habe.

6.1.1 Zu diesen Aufgaben zählten:

- Kommunikation über Termine und Fristen innerhalb der Gruppe
- Kommunikation mit den Veranstaltern des Tags der Medien
- Erstellung und Bearbeitung der einzelnen Elemente auf der TdM Homepage für unser Projekt.
- Bearbeitung der Gruppen Fotos und Erstellung der Collage
- Planung, Erstellung und Bearbeitung des Projekt Logos
- Zusammenstellung und Gestaltung der TdM Präsentation

6.1.2 Weitere Aufgaben für meinen eigenen Teil des Projektes:

- Planung, Recherche, Skript, Stock Bilder, Grafiken, Videos und Audio Beschaffung
- Erstellung, Schnitt und Bearbeitung der Videos
- Planung, Coding und Bearbeitung der Homepage für die Videos

Die Planung und die Kommunikation rund um den Tag der Medien habe ich übernommen. Videos die eigentlich für den Tag der Medien geplant waren, um die Aufmerksamkeit der Besucher auf unser Projekt lenken sollten waren geplant. Da aber aufgrund des immer noch durch die Corona Maßnahmen geregelten Einschränkungen und Regeln, gab es auch in diesem Jahr keine Präsensts Veranstaltungen. Weshalb der Tag der Medien auch in einem Alphaview Raum abgehalten und auf die Fünfzehn Minütige Präsentation begrenzt wurde. Die Videos habe ich stattdessen umgeändert und in einer kleinen Homepage mit ergänzenden Texten zusammengefasst.

6.2 Logo für TdM

Die Arbeit an dem Logo das uns als Gruppe am Tag der Medien repräsentiert hat mehrere Stunden in Kauf genommen, war aber an sich das, was mir am meisten Spaß gemacht hat. Dadurch, dass ich sehr große Kreative Freiheit hatte konnte ich viel ausprobieren und an Ecken und Enden Dinge verbessern und anpassen. Das Endergebnis wurde auch von allen aus der Gruppe direkt ohne Kritik angenommen.

6.3 Videos/ Homepage

Meine Arbeit an den Videos bzw. der Homepage war der größte Teil meiner Tätigkeit.

Erst sollten es Videos werden, die auf der Internetplattform Simpleshow.com erstellt wurden. Nach wenigen Stunden auf dieser Plattform war für mich jedoch klar, dass diese dem Qualitätsanspruch, den ich habe, nicht gerecht wird.

Der größte Kritikpunkt war, dass die Plattform simpleshow.com in der kostenlosen Benutzung ein sehr begrenztes Angebot an Bildern, Animationen, Audiooptionen oder Sprechern hat.

Der Text wird in ein Video umgewandelt, in dem die Grafik einer Hand einzelne Bilder, die zum Text passen sollen mit einer Animation auf den Bildschirm schieben.

Die Optionen für die angezeigten Bilder waren bis auf wenige Ausnahmen nur Pfeile oder Bilder die zwar ein Wort aus meinem Text als genommen haben, dann aber den gesamten Kontext und das Verständnis des Videos geändert haben.

z.B.

Aus Sicherheits- und Kostengründen ist es für fast alle konkreten Problemkreise notwendig, sie aus der Realität zu lösen und abstrakt zu behandeln;

Bei dem o.g. Textausschnitt wollte ich ein Bild einfügen, das etwas mit Kosten, Problemen oder Realität zu tun hatte in dem Fall ein Geldschein oder einen Geldsack für die Kosten.

Stattdessen wurden mir bei den gewählten Worten nur Pfeile angezeigt und bei der Automatischen Erkennung hat mir die Plattform das Bild einer Person in einem Krankenhausbett für das Wort „*behandeln*“ angezeigt.

Die Option dazu wäre eine Premium Mitgliedschaft, die aber einerseits sehr teuer und zum anderen, als Privatperson ohne eine Firma oder ähnliches nicht erwerblich war, abzuschließen.

Aufgrund der erschwerten Arbeitsweise durch diese Methode, habe ich mich entschieden die Videos mit Adobe Premiere Pro selbst zu erstellen. Dafür war eine sehr langwierige Recherche von Film und Bildmaterial notwendig.

Wobei wieder ein Problem entstand. Da dieses Projekt und somit auch diese Videos Wissenschaftlich erarbeitet werden sollen, konnte ich keine Bilder und Videos die rechtlich geschützt sind benutzen. Wegen der Ästhetik und Anschaulichkeit wollte ich auch kein Material, dass mit Wasserzeichen über den gesamten Bildschirm verziert ist, benutzen.

Über verschiedene Internetplattformen, wie pexels.com und andere, bei denen die Nutzer selbst das Material erstellen und dieses, solange es nicht Kommerziell verwendet wird zur freien Verfügung stellen, habe ich mir mein Film und Bildmaterial beschafft.

Das Erstellen und Bearbeiten der Videos stellte keine allzu große Herausforderung dar. Durch die schon animierten Stock Videos musste ich nur einige Grafiken erstellen und animieren. Den Text habe ich über eine Text to Speech Webseite auslesen lassen.

Die Musik ist auch Rechtfrei und da die Videos alle zusammen ein Oberthema haben, wurde auch nur ein Stück für alle verwendet

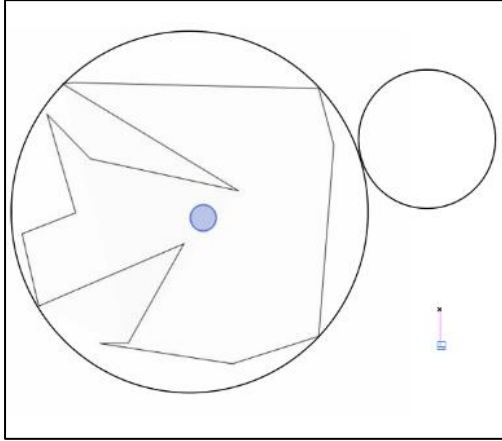
7 Fazit

Letztendlich konnten nicht alle Ziele des Projektes zufriedenstellend erreicht werden. Es hat sich herausgestellt, dass gerade die Implementierung des exzentrischen Stoßes einen Entwickler vor viele Herausforderungen stellt. Dies beginnt schon damit, dass es schwer ist festzustellen, ob das Ergebnis eines Stoßes von starren Körpern auch korrekt ist. Zwar gibt es viele Beispiele im Internet zu Stößen zwischen Kreisen allerdings erstaunlich wenige bei denen echte starre Körper zum Einsatz kommen die auch ihre Rotationsbewegung ändern. Außerdem können schlecht gewählte Trägheitsmomente einen Stoß sehr unrealistisch aussehen lassen. Vor der Implementierung der automatischen Bestimmung des Trägheitsmomentes viel es daher sehr schwer die Ergebnisse eines Stoßes einzuschätzen. Man hat sich daher auf ein überlegtes „Gefühl“ verlassen oder anhand der Formeln nachrechnen müssen. Dabei entstand zuerst der fälschliche Verdacht, dass möglicherweise die zugrundeliegenden Formeln falsch sein könnten.

Die Implementierung von Zuständen, die über das Fliegen und Ruhen hinausgehen ist noch nicht zufriedenstellend. Zwar gibt es eine Rollbewegung und auch das Gleiten zweier Flächen aufeinander, doch fehlt hier die Wechselwirkung, dass ein aufliegender Körper auch eine Gewichtskraft auf den darunterliegenden Körper ausübt. Außerdem existieren viele Spezialfälle, die geklärt werden müssten wie Mehrfachstöße. Es gibt aber auch Zustände in denen ein Körper auf einer Ecke, entlang einer Kante rutschen sollte oder z.B. über mehrere Ecken mit einer Kante verbunden ist und auf dieser „rutscht“. Außerdem gibt es kritische Zustandsübergänge, wie z.B., wenn ein Körper eine Kante herabrutscht und dann auf eine andere Kante stößt. Dabei könnte es wieder zu der Situation kommen, dass ein Körper über zwei Ecken „rutscht“ und das sogar auf zwei Körpern gleichzeitig. Man sieht also schon an wenigen Beispielen, wie problematisch eine Fehlerfreie Implementierung ist. Leider ist es nicht gelungen alle diese und weitere Situationen zu lösen.

Auch in der Performance Frage sind einige Vorhaben noch nicht umgesetzt. So war geplant, die Polygone mit einem umschließenden Kreis zu beschreiben, sodass unnötig Lange Stoßprüfungen vermieden werden, solange keine Möglichkeit besteht, dass

sich die Körper wirklich treffen. Der Test ob zwei Kreise ineinander gedrungen sind geht, wie beschrieben, deutlich einfacher und schneller.



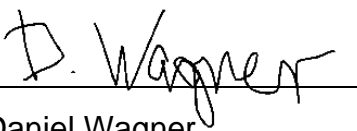
8 Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

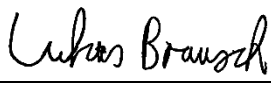
Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

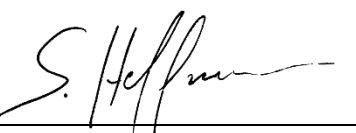
Balingen, den 13.02.2021


Daniel Wagner


Furtwangen, 13.02.2021


Lukas Brausch


Furtwangen, den 13.02.2021


Sebastian Hoffmann

Michelbach an der Bilz, den 13.02.2021


Manuel Maringolo

Furtwangen, den 13.02.2021


Dimitrios Stüber