Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

22. April 2023

Inhaltsverzeichnis

1 Zusammenfassung

2 Aufbau des Experiments

Für den Aufbau des Experimentes sind zwei Würfel mit den Dimensionen von 1.5m Seitenlänge und dem Gewicht von 2 Kilogramm gegeben. Wie in der Abbildung 1 zu entnehmen, ist linke Würfel Julia und der Rechte Romeo benannt. Daneben existiert eine Feder die horizontal an einer Wand befestigt ist. Bei dem gesammten Experimentes wird der Reibungswiederstand ignoriert. Ablauf des Experimentes:

1. Romeo wird mit einer konstanten Kraft (grüner Pfeil in Abbildung 1) auf 2m/s nach rechts beschleunigt.

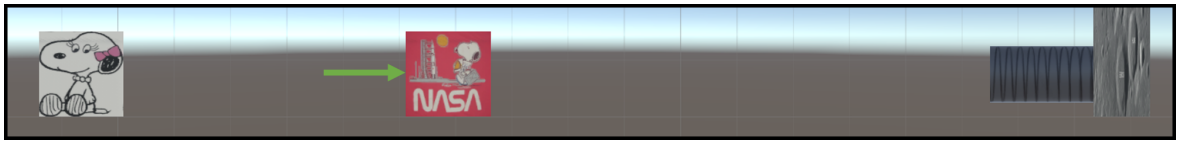

Abbildung 1: Beschleunigung des Würfels

2. Romeo trifft nun auf die Feder. Dabei soll die Federkonstante (gelber Pfeil in Abbildung 2) so gewählt werden, dass Romeo elastisch zurückprallt ohne die Wand zu berühren.

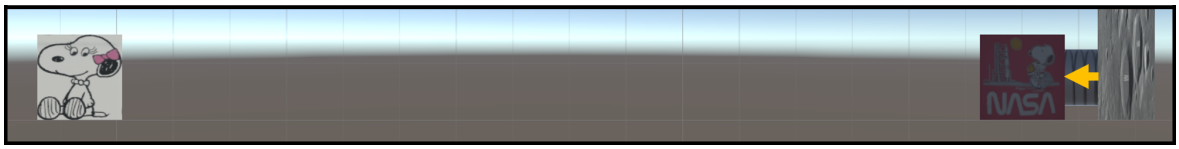

Abbildung 2: Elastischer Zusammenstoss mit der Feder

3. Nach dem abgefederten Stoss gleitet Romeo zurück in die Richtung aus der er gekommen ist und stösst inelastisch mit Julia zusammen. Über einen FixedJoint haften die Beiden nun zusammen und gleiten mit der aufgeteilten Energie (blaue Pfeile in Abbildung 3) weiter nacht links.

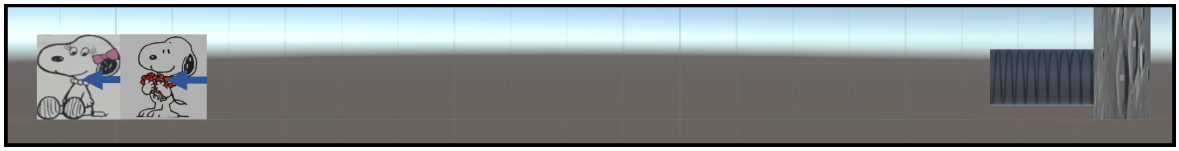

Abbildung 3: Inelastischer zusammenstoss mit dem anderen Würfel

3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalische Vorgänge des Versuches beschrieben. Die gegebenen Massen sind:

- Gewicht(m) = 2kg
- Velocity(v) = 2m/s
- Würfelseite = 1.5m

3.1 Teil 2: Würfel bewegt sich und stösst

Es geschehen drei Vorgänge, die Beschleunigung durch die konstante Kraft, einen elastischen Stoss und einen inelastischen Stoss. Ein Würfel,namens Romeo, wird durch die konstante Kraft beschleunigt, bis maximal eine Geschwindigkeit von 2m/s erreicht wird. Romeo trifft auf eine Feder zu, die an eine Wand befestigt ist. Dabei geschieht ein elastischer Stoss und der Würfel gleitet wieder zurück und stösst dabei einen zweiten Würfel, Julia, diesmal passiert der Stoss inelastisch. Sämtliche Vorgänge erfolgen ohne Reibungskräfte.

3.1.1 Konstante Kraft

Um die konstante Kraft zu berechnen nehmen wir die gewünschte Geschwindigkeit und berechnen damit die Beschleunigung, weil die Kraft sowohl von der Masse wie auch der Beschleunigung abhängt und gegeben ist durch die Formel[1]

$$F = m * a$$

Um dieses Anfangwertproblems zu lösen leiten wir die Geschwindigkeit ab [1]:

$$\dot{v} = a$$
$$2m * s^{-1} \rightarrow -2m * s^{-2} \rightarrow a = [\tfrac{2m}{s^2}]$$

Die Zeit, die gebraucht wird um den Würfel zu beschleunigen, wird durch folgende Formel beschrieben [1]:

$$v = a * t \rightarrow t = \tfrac{v}{a} \rightarrow \tfrac{2m/s}{2m/s^2} = 1s$$

Somit können wir nun die Kraft ausrechnen:

$$F = 2kg * \tfrac{2m}{s^2} => \tfrac{4kg*m}{s^2} = 4N$$

4N werden deshalb als konstante Kraft angewendet, damit auch die gewünschte Geschwindigkeit erreicht wird, danach wird keine Kraft mehr hinzugefügt und Romeo gleitet auf die Feder zu.

3.1.2 Elastischer Stoss

Beim elastischen Stoss ist die kinetische Energie vom Stosspartner vor und nach der Kollision gleich [1]. Gemäss Auftrag wird die Federlänge und Federkonstante so dimensioniert, dass der Würfel nicht auf die Wand trifft. Die kinetische Energie des Würfels wird mit folgender Formel berechnet [1]:

$$E_{kin_{Romeo}} = \frac{1}{2} * m * v^2$$

Setzt man die Massen von diesem Projekt ein bekommt man:

$$\frac{1}{2} * 2kg * (\frac{2m}{s})^2 = 4J$$

Während des Stosses wird die kinetische Energie auf die Feder übetragen. Die Feder speichert diese Energie in Form von potentieller Energie, da sie zusammenngedrückt wird. Sobald sie Romeo zurück stösst, wird diese Energie in eine kinetische zurückgewandelt.

Um die Federkonstante zu berechnen, nehmen wir die Tatsache der Energieerhaltung zu Nutze und setzen die ausgerechnete kinetische Energie gleich mit der potentiellen Energie der Feder.
Die Formel für die pontentielle Energie des Feders lautet[1]:

$$E_{pot_{Feder}} = \frac{1}{2} * k * x^2$$

Die Gleichsetzung der Energien, sieht folgendermassen aus[1]:

$$E_{kin_{Romeo}} = E_{pot_{Feder}}$$

$$\frac{1}{2} * m * v^2 = \frac{1}{2} * k * x^2$$

Diese Gleichung stellen wir um und lösen nach der Federkonstante k auf:

$$k = \frac{m*v^2}{x^2}$$

Mit den eingesetzen Massen und die gewählte maximale Auslenkung erhalten wir:

$$\frac{2kg*(2m/s)^2}{1.3m^2} = 4.73N/m$$

Jetzt wo wir die Federkonstante und Länge haben, können wir einen langsamen Stoss gewährleisten.

3.1.3 Inelastischer Stoss

Beim vollständigen inelastischen Stoss, werden beide Stosspartner nach der Kollision verrbunden sein und dieselbe Geschwindigkeit haben[1]. Die Formeln, die wir für diesen Vorgang brauchen sind, die der Impulse der beiden Körper:

$$Impuls_{Romeo} = m_{Romeo} * v_{Romeo}$$

$$Impuls_{Julia} = m_{Julia} * v_{Julia}$$

Bei diesem Vorgang wird ein Teil des Impulses von Romeo auf Julia übertragen. Der Gesamtimpuls bleibt erhalten vor und nach dem Stoss und wird durch den Impulserhaltungsatz beschrieben[1]:

$$m_{Romeo} * v_{Romeo} + m_{Julia} * v_{Julia} = (m_{Romeo} + m_{Julia}) * v_{Ende}$$

Die Endgeschwindigkeit ist die Geschwindigkeit, die beide Körper gemeinsam haben nach dem Stoss. Die Relation zwischen der kinetischen Energie und des Impulses, können wir folgendermassen herleiten[1]:

$$E_{kin} = \frac{1}{2} * m * v^2 = \frac{(mv)^2}{2m} = \frac{p^2}{2m}$$

Wenden wir dies nach dem Stoss an, sehen wir, dass die kinetische geringer wird:

$$E_{kin_{Ende}} = \frac{p^2}{2(m_{Romeo}+m_{Julia})}$$

4 Beschreibung der Implentierung inklusive Screenshots aus Unity

5 Rückblick und Lehren aus dem Versuch

6 Resultate mit grafischer Darstellung

Während dem Durchlauf des Experimentes werden diverse Daten zu allen im Experiment vorkommenden Objekten gesammelt. Diese Daten umfassen Geschwindigkeit, Kinetische, sowie Potentielle Energie. Dabei wird zwischen dem Eleastischen sowie Inelastischen Zusammentoss unterschieden.

6.1 Elastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem Elastischen Zusammenstoss in den Abbildungen Abbildung 4 bis Abbildung 7 aufgegliedert.

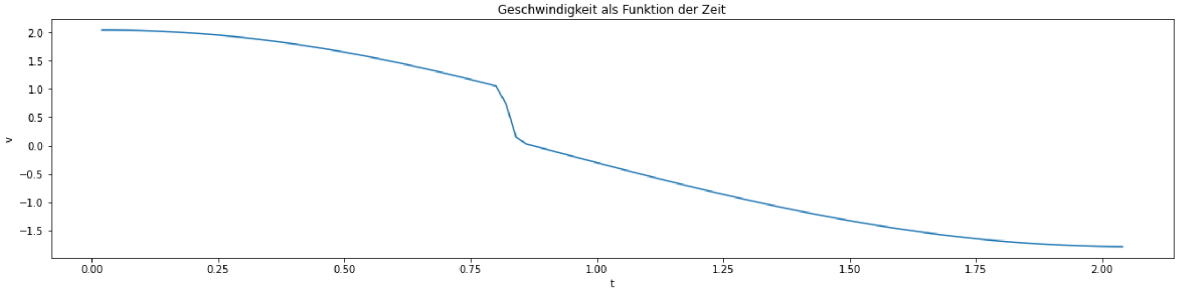

Abbildung 4: Geschwindigkeit als Funktion der Zeit

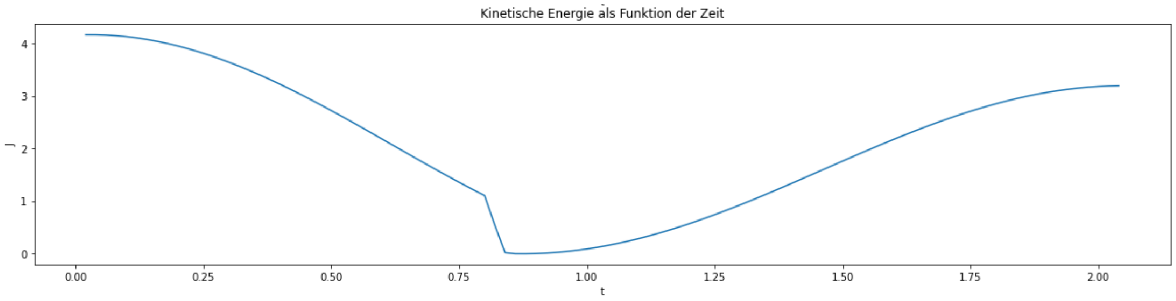

Abbildung 5: Kinetische Energie als Funktion der Zeit

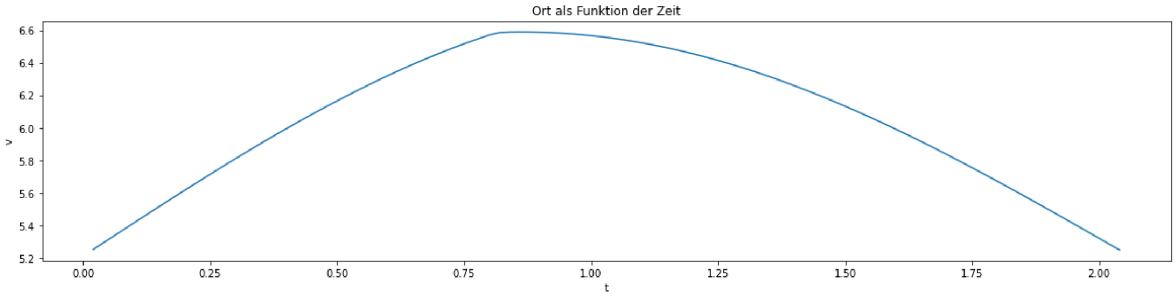

Abbildung 6: Ort als Funktion der Zeit

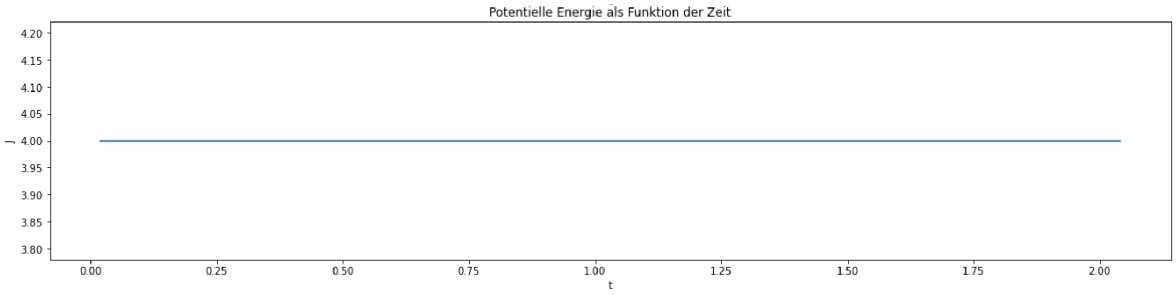

Abbildung 7: Potentielle Energie als Funktion der Zeit

6.2 Inelastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem Inelastischen Zusammenstoss in den Abbildungen Abbildung 8 bis Abbildung 11 aufgegliedert.

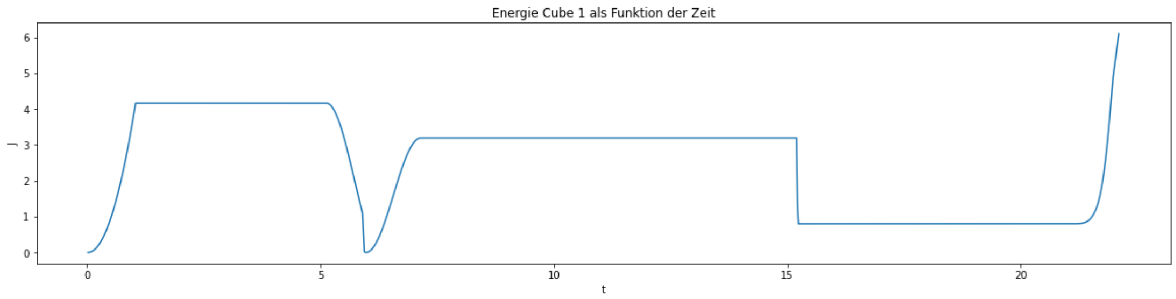

Abbildung 8: Energie Cube 1 als Funktion der Zeit

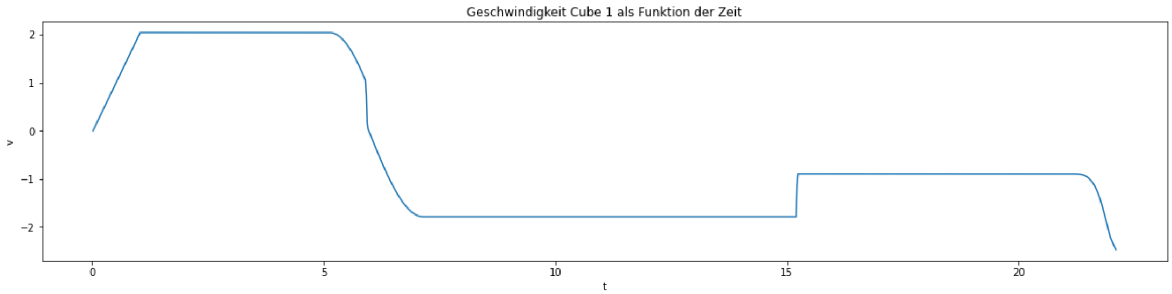

Abbildung 9: Geschwindigkeit Cube 1 als Funktion der Zeit

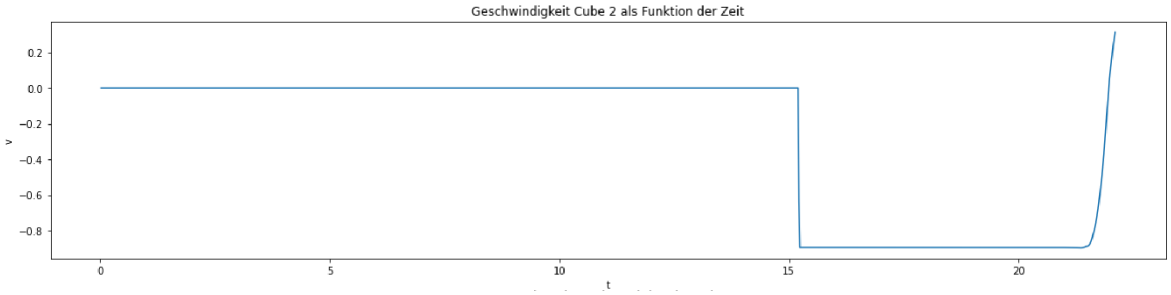

Abbildung 10: Geschwindigkeit Cube 2 als Funktion der Zeit

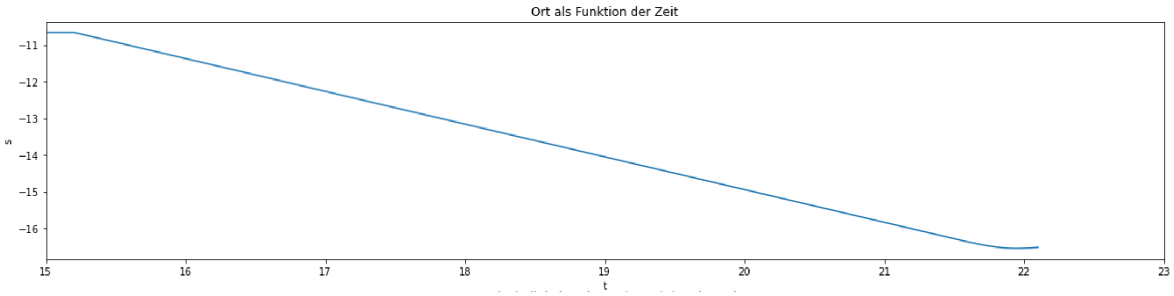

Abbildung 11: Ort als Funktion der Zeit

7 Code

7.1 Code für das Experiment

Nachfolgend ist der für das Experiment benötigte Code abgebildet. Dieser Umfasst alle Physikalischen berechnungen, sowie die Bewegungen.

```
using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;


public class CubeController : MonoBehaviour
{
    public Rigidbody cubeRomeo;
    public Rigidbody cubeJulia;
    public GameObject spring;

    private float currentTimeStep; // s
    private float cubeJuliaTimeStep;

    private List<List<float>> timeSeriesElasticCollision;
    private List<List<float>> timeSeriessInelasticCollision;

    private string filePath;
    private byte[] fileData;
    float springPotentialEnergy = 0f;
    float cubeRomeoKinetic = 0f;
    float cubeRomeoImpulse = 0f;
    float cubeJuliaImpulse = 0f;
    float GesamtImpluls = 0f;
    float ImpulsCheck = 0f;
    float forceOnJulia = 0f;
    float velocityEnd = 0f;
    float cubeKineticEnd = 0f;
    float constantForce = 4f;
    double startime = 0;
    double accelarationTime = 1.0;
    float springConstant = 0f;
    float springMaxDeviation = 0f;
    float springContraction = 1.3f;

    // Start is called before the first frame update
    void Start()
    {
        startime = Time.fixedTimeAsDouble;

        timeSeriesElasticCollision = new List<List<float>>();
        timeSeriessInelasticCollision = new List<List<float>>();

        //Maximale Auslenkung gerechnet anhand der linken seite des Feders
        springMaxDeviation = spring.transform.position.x - spring.transform.localScale.y/2;
        // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
            (Math.Pow(springContraction, 2.0)));

        //Maximale Auslenkung gerechnet anhand der linken seite des Feders
        springMaxDeviation = spring.transform.position.x - spring.transform.localScale.y/2;
         // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
            (Math.Pow(springContraction, 2.0)));
```

```
}

// Update is called once per frame
void Update()
{
}
// FixedUpdate can be called multiple times per frame
void FixedUpdate()
{
    double currentTime = Time.fixedTimeAsDouble-startime;

    if (accelarationTime >= currentTime)
    {
        constantForce = 4f;
        cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
    }

    // 1/2*m*v^2
    cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
        Math.Pow(cubeRomeo.velocity.x, 2.0)));


    float collisionPosition = cubeRomeo.transform.position.x +
        cubeRomeo.transform.localScale.x / 2;

    if (collisionPosition >= springMaxDeviation)
    {
        float springForceX = (collisionPosition - springMaxDeviation) * -springConstant;
        springPotentialEnergy =(float)(0.5 * springConstant * Math.Pow(collisionPosition -
            springMaxDeviation, 2.0));
        cubeRomeo.AddForce(new Vector3(springForceX, 0f, 0f));
        ChangeCubeTexture();
        currentTimeStep += Time.deltaTime;
        timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
            cubeRomeo.position.x, cubeRomeo.velocity.x, springPotentialEnergy,
            cubeRomeoKinetic, springForceX });
    }

    // 1/2*m*v^2
    cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
        Math.Pow(cubeRomeo.velocity.x, 2.0)));
    cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
    cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
    GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
    velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
        cubeJulia.mass);
    ImpulsCheck = (cubeRomeo.mass + cubeJulia.mass )* velocityEnd;
    cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
        Math.Pow(velocityEnd, 2.0)));
    forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);

    cubeJuliaTimeStep += Time.deltaTime;
    timeSeriessInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
        cubeRomeo.position.x, cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse,
        cubeRomeoKinetic, cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass,
        cubeJuliaImpulse, velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls,
        ImpulsCheck });
}
void OnApplicationQuit()
{
    WriteElasticTimeSeriesToCsv();
```

```
    WriteInelasticTimeSeriesToCsv();
}
void WriteElasticTimeSeriesToCsv()
{
    using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
    {
        streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
            cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic, springForceX");

        foreach (List<float> timeStep in timeSeriesElasticCollision)
        {
            streamWriter.WriteLine(string.Join(",", timeStep));
            streamWriter.Flush();
        }
    }

}

void WriteInelasticTimeSeriesToCsv()
{
    using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
    {
        streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
            cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse, cubeRomeoKinetic,
            cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse,
            velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck }");

        foreach (List<float> timeStep in timeSeriessInelasticCollision)
        {
            streamWriter.WriteLine(string.Join(",", timeStep));
            streamWriter.Flush();
        }
    }
}

void ChangeCubeTexture()
{
    // the path of the image
    filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
    // 1.read the bytes array
    fileData = File.ReadAllBytes(filePath);
    // 2.create a texture named tex
    Texture2D tex = new Texture2D(2, 2);
    // 3.load inside tx the bytes and use the correct image size
    tex.LoadImage(fileData);
    // 4.apply tex to material.mainTexture
    GetComponent<Renderer>().material.mainTexture = tex;
}

void OnCollisionEnter(Collision collision)
{
    if (collision.rigidbody != cubeJulia)
    {
        return;
    }
    if (collision.rigidbody == cubeJulia)
    {
        FixedJoint joint = gameObject.AddComponent<FixedJoint>();
        ContactPoint[] contacts = new ContactPoint[collision.contactCount];
        collision.GetContacts(contacts);
        ContactPoint contact = contacts[0];
        joint.anchor = transform.InverseTransformPoint(contact.point);
```

```
            joint.connectedBody =
                collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();


            // Stops objects from continuing to collide and creating more joints
            joint.enableCollision = false;
        }
    }
}
```

7.2 Code für die Datenaufbereitung des Elastischen Stosses

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Elastischen Stosses der Grafiken benötigt wurde.

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("../UnityProj/TimeSeries/time_seriesElastic.csv")

plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["currentTimeStep"], df[" cubeRomeo.position.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Ort als Funktion der Zeit")



plt.subplot(4,1,2)
plt.plot(df["currentTimeStep"], df[" cubeRomeo.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Geschwindigkeit als Funktion der Zeit")



plt.subplot(4,1,3)
plt.plot(df["currentTimeStep"], df[" cubeRomeoKinetic"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Kinetische Energie als Funktion der Zeit")


plt.subplot(4,1,4)
plt.plot(df["currentTimeStep"], df[" springPotentialEnergy"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Potentielle Energie als Funktion der Zeit")

plt.show()
```

7.3 Code für die Datenaufbereitung des Inelastischen Stosses

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Inelastischen Stosses der Grafiken benötigt wurde.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("../UnityProj/TimeSeries/time_seriesInelastic.csv")

plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.position.x"])
plt.ylabel("s")
plt.xlabel("t")
plt.xlim(15,23)
plt.title("Ort als Funktion der Zeit")


plt.subplot(4,1,2)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Geschwindigkeit Cube 1 als Funktion der Zeit")


plt.subplot(4,1,3)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Geschwindigkeit Cube 2 als Funktion der Zeit")


plt.subplot(4,1,4)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoKinetic"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Energie Cube 1 als Funktion der Zeit")

plt.show()

plt.plot(df["cubeJuliaTimeStep"], df[" cubeJuliaImpulse"])
plt.ylabel("Ns")
plt.xlabel("t")
plt.title("Impuls Cube 2 als Funktion der Zeit")


plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoImpulse"])
plt.ylabel("Ns")
plt.xlabel("t")
plt.title("Impuls Cube 1 als Funktion der Zeit")

plt.subplot(4,1,2)
plt.plot(df["cubeJuliaTimeStep"], df[" velocityEnd"])
plt.ylabel("s")
plt.xlabel("t")
plt.title("Endgeschwindigkeit als Funktion der Zeit")


plt.subplot(4,1,3)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeKineticEnd"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Endkinetik als Funktion der Zeit")
```

```
plt.subplot(4,1,4)
plt.plot(df["cubeJuliaTimeStep"], df[" forceOnJulia "])
plt.ylabel("N")
plt.xlabel("t")
plt.title("Kraft auf Cube 2 als Funktion der Zeit")
plt.show()
```

A Anhang

test

Abbildungsverzeichnis

Literatur

[1] Tipler, Paul A. u. a. *Physik Für Studierende Der Naturwissenschaften Und Technik*. Springer Spektrum. ISBN: 978-3-662-58280-0.