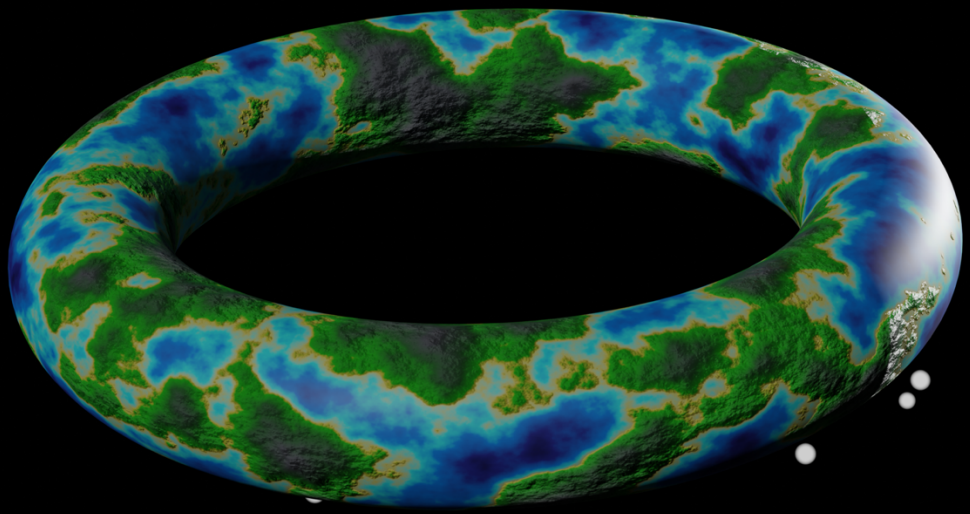
Was, wenn die Welt ein Torus wäre ?



Teilnehmer	Donald Kobbelt, 12 Jahre
Schule	Kaiser-Karls-Gymnasium Augustinerbach 2 – 7 52062 Aachen
Projektbetreuer	Andreas Kral / Leif Kobbelt

Thema des Projekts	Ich möchte physikalische Phänomene auf einem Torus-förmigen Planeten untersuchen und dazu Simulationen mit *Blender* berechnen.
Fachgebiet	Physik
Wettbewerbssparte	Schüler experimentieren
Bundesland	Nordrhein-Westfalen
Wettbewerbsjahr	2022

Was, wenn die Welt ein Torus wäre ?

Donald Kobbelt

Kurzfassung:

Alle bekannten Planeten sind ungefähr kugelförmig. Sie drehen sich um ihre Sonne und ihre Schwerkraft hält Gegenstände und Wasser auf dem Boden fest. Wenn es einen Planeten gäbe, der die Form von einem Torus hat, wäre vieles anders. Das möchte ich in meinem Projekt untersuchen. Dazu werde ich physikalische Phänomene mit der Software Blender simulieren, z.B. wohin fließt Wasser, wie stabil sind Gebäude oder wann und wo scheint die Sonne. Ich möchte herausfinden, ob man auf einem Torus-Planeten überhaupt leben könnte.

Inhaltsverzeichnis:

Was, wenn die Welt ein Torus wäre?

Donald Kobbelt

(1) Einleitung

Vor ein paar Monaten habe ich im Internet ein Video gesehen, wo es um eine Donut-Erde ging. Das Video war aber eher ein Quatsch-Video und nicht wirklich realistisch. Also habe ich mir überlegt, wie ich herausfinden kann, ob es wirklich so einen Planeten geben könnte und wie es sich anfühlen würde, wenn man darauf lebt. Die Lebensbedingungen auf einem Planeten hängen natürlich von sehr vielen Sachen ab, z.B. ob der Planet in der „habitablen Zone“ um seine Sonne kreist. Das hat Einfluss auf die Temperatur und ob es flüssiges Wasser geben kann. In diesem Projekt möchte ich aber erstmal nur untersuchen, ob ein Torus-Planet physikalisch überhaupt existieren könnte, also ob er nicht zusammenfallen würde oder auseinanderreißen.

Ein Physiker würde jetzt vielleicht versuchen das Problem **theoretisch** zu lösen, aber dafür fehlt mir noch das Mathe-Wissen. Man könnte auch versuchen **Experimente** zu machen, aber das ist bei Planeten natürlich nicht möglich. Also habe ich versucht, das Problem mit **Simulationen** zu erforschen, denn dafür kann ich die 3D Software *Blender* benutzen, mit der ich mich schon ziemlich gut auskenne. *Blender* kann die Bewegung von starren Körpern und Flüssigkeiten simulieren, ohne dass ich die komplizierten Formeln selbst ausrechnen muss. Weil ich die Parameter der Simulation immer wieder ändern kann und so ausprobieren, welche Kombination von Kräften gut funktioniert, ist eine Simulation auch wie eine Art von Experiment, aber ohne, dass ich dafür einen echten Planeten benötige.

Simulationen in *Blender* sind **dynamisch**, d.h. sie berechnen, wie sich die Körper oder das Wasser über die Zeit bewegen. Dafür werden viele Einzelbilder („Frames“) berechnet und dann als Video abgespielt. Meine Videos zu diesem Projekt habe ich auf meiner Homepage unter: https://donaldinho-k.github.io/torus.html zum angucken hochgeladen.

Es gibt aber auch viel einfachere Simulationen, z.B. wenn ich ausrechnen möchte, wie groß die Gravitationskraft auf einem Torus ist oder die Zentrifugalkraft, wenn er sich dreht. Für diese Simulationen muss ich dann nur ein paar Kräfte ausrechnen und diese addieren. Dazu habe ich mir ein Python-Programm geschrieben, mit dem ich diese Berechnungen auch ohne *Blender* durchführen kann.

Diese Fragen möchte ich untersuchen:

(1) Welche physikalischen Gesetze sind für einen Torus-Planeten wichtig?
(2) Kann ein Torus-Planet überhaupt existieren?
(3) Wie würden sich das Sonnenlicht, Wasser oder Gebäude auf diesem Planeten verhalten?

(2) große Zahlen

In diesem Projekt muss ich manchmal mit ziemlich großen Zahlen rechnen, z.B. das Gewicht eines Planeten in kg. Diese Werte kennt man nur so ungefähr, d.h. sie werden gerundet. Das ist für die Simulation kein Problem, denn bei dem riesigen Gewicht der Erde kommt es nicht auf ein paar Kilogramm oder Tonnen an. Um nicht

mit den vielen Nullen durcheinander zu kommen, benutze ich eine Schreibweise von Python (die Programmiersprache, die ich für dieses Projekt benutzt habe). Dort gibt man eine Zahl immer so an, dass das Komma nach der ersten Ziffer steht. Dann kommt ein **„e“** und dann schreibt man um wie viele Stellen das Komma verschoben werden muss, um die richtige Zahl zu erhalten. **„+“** bedeutet dabei, dass das Komma nach rechts verschoben wird und **„-“**, dass es nach links verschoben wird.

Die Zahl 123456789 schreibt man also als 1,23456789e+8 und 25 g entspricht 0,000025 t also 2,5e-5 t. Das Gewicht der Erde ist laut Wikipedia 5,974e+24 kg also 5.974.000.000.000.000.000.000.000 kg wobei diese Zahl auf 4 Stellen gerundet ist. In Python sieht das z.B. so aus (man schreibt einen Punkt statt einem Komma):

```
pi          = 3.14159265
Masse       = 5.974e+24           # Masse der Erde          [ kg ]
Volumen     = 1.083e+21           # Volumen der Erde        [ m^3 ]
Gravitation = 6.674e-11           # Gravitationskonstante   [ m^3 / kg / s^2 ]
```

Wenn man zwei solche Zahlen multiplizieren will, muss man die Komma-Zahlen multiplizieren und die Zahlen nach dem „e“ addieren, also 1,234e+3 * 5,678e+5 = 7,006652e+8 also 7,007e+8, wenn man das Ergebnis auf 4 Stellen rundet. Python macht das alles automatisch.

(3) Vektoren

Die Kräfte, die ich in diesem Projekt berechnen möchte, werden durch sogenannte Vektoren beschrieben. Ein Kraft-Vektor hat eine **Richtung** und eine **Stärke**, die seiner **Länge** entspricht. Meistens wird ein Vektor über Koordinaten definiert und man kann mit diesen Koordinaten rechnen. Bei der Addition von zwei Vektoren addiert man einfach jeweils die x-, y- und z-Koordinaten. Den Vektor zwischen zwei Punkten erhält man, indem man von den Koordinaten des Endpunktes die Koordinaten des Anfangspunktes abzieht. Die Länge eines Vektors berechnet man indem man alle drei Koordinaten jeweils quadriert, dann aufsummiert und am Ende die Wurzel aus der Summe zieht (*Satz des Pythagoras*). Wenn man die Länge eines Vektors verändern will, ohne seine Richtung zu verändern, dann kann man alle drei Koordinaten mit derselben Zahl multiplizieren oder dividieren, denn die Richtung eines Vektors wird durch das Verhältnis der Koordinaten zueinander bestimmt und nicht durch die einzelnen Werte, d.h. die Vektoren (1 I 2 I 3) und (2 I 4 I 6) haben die gleiche Richtung, aber eine unterschiedliche Länge.

Beispiel: Der Vektor vom Punkt (3 I 7 I 4) zum Punkt (5 I 8 I 6) berechnet sich aus (5 I 8 I 6) - (3 I 7 I 4) = (2 I 1 I 2). Die Länge dieses Vektors ist die Wurzel aus $2^2 + 1^2 + 2^2 = 4 + 1 + 4 = 9$. Die Wurzel aus 9 ist 3, also hat der Vektor die Länge 3.

Wenn ich die drei Vektor-Koordinaten durch die **Länge** des Vektors dividiere, erhalte ich die **Richtung** des Vektors. In meinem Beispiel also (2 I 1 I 2) / 3 = (2/3 I 1/3 I 2/3). Dieser neue Vektor hat immer die Länge 1, denn $(2/3)^2 + (1/3)^2 + (2/3)^2 = 4/9 + 1/9 + 4/9 = 1$.

Ich kann also bei einem Kraft-Vektor aus den Koordinaten seine Länge und seine Richtung berechnen:

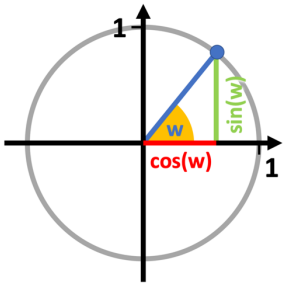
(x-Koordinate I y-Koordinate I z-Koordinate) = Länge * (x-Richtung I y-Richtung I z-Richtung)

wobei die Länge gleich sqrt(x-Koordinate ^{2} + y-Koordinate ^{2} + z-Koordinate ^{2}) ist und es gilt immer sqrt(x-Richtung ^{2} + y-Richtung ^{2} + z-Richtung ^{2}) = 1. Als Python-Programm sieht das dann so aus:

```
import numpy as np                          # Importiere die Wurzel-Funktion: np.sqrt()

vx = bx - ax                                # Vektor von A (Anfangspunkt) nach B (Endpunkt)
vy = by - ay
vz = bz - az

Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors

rx = vx / Laenge                            # Richtung des Vektors
ry = vy / Laenge
rz = vz / Laenge
```

(4) Kreise

Ein Torus besteht aus verschieden großen Kreisen, die in etwa den Längen- und Breitenkreisen auf der Erde entsprechen. Um die Koordinaten von Punkten auf dem Torus berechnen zu können, muss ich also erstmal Punkte auf Kreisen berechnen können. Hierzu verwendet man die Funktionen **Sinus** (sin) und **Cosinus** (cos). Wenn ich einen Punkt auf einem Kreis mit Radius 1 mit dem Winkel w bestimmen will, dann entspricht die x-Koordinate des Punktes dem Cosinus und die y-Koordinate dem Sinus des Winkels w. Das folgende Bild zeigt, wie das aussieht:



Wenn der Kreis einen anderen Radius als 1 hat, dann muss ich die Koordinaten mit dem Radius multiplizieren, damit sich die Länge des blauen Vektors in dem Bild entsprechend verlängert, aber die Richtung (der Winkel) sich nicht verändert.

In Python berechne ich die Koordinaten von einem Punkt mit Winkel W auf einem Kreis mit Radius R so:

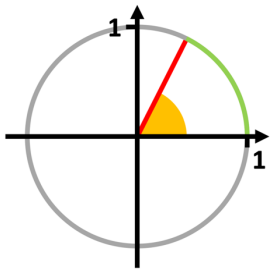
```
import numpy as np                          # Importiere die Funktionen np.sin() und np.cos()

ax = R * np.cos(W)
ay = R * np.sin(W)
```

Später werde ich nicht nur einen einzelnen Punkt auf einem Kreis berechnen, sondern ganz viele, die ich gleichmäßig verteilen möchte. Wenn ich also z.B. 20 Punkte auf einem Kreis berechnen möchte, dann muss ich 360° durch 20 teilen. Das ergibt 18°. Und dann muss ich alle 18° einen Punkt berechnen, also einen Punkt bei 0°, einen bei 18°, dann 36°, dann 54° und so weiter bis 360°. In Python benutze ich dazu eine Zähl-Schleife:

```
n = 20
w = 360 / n

for i in range(n):
    winkel = i * w
```

Im Mathe-Unterricht haben wir Winkel in Grad gemessen, d.h. ein Winkel liegt zwischen 0° und 360°. In Python werden Winkel aber in **Bogenlänge** gemessen, d.h.

anstatt dem orangenen Winkel im folgenden Bild misst man die Länge des grünen Kreisbogens:



Der komplette Kreis hat einen Umfang von 2π, also entspricht der Winkel 360° der Bogenlänge 2π. Ein Halbkreis hat die Länge π was 180° entspricht und ein Viertelkreis hat die Länge $\pi/2$ was 90° entspricht. Wenn ich jetzt wieder 20 Punkte auf dem Kreis berechnen möchte, dann ist der Winkel zwischen zwei Punkten eben 2π / 20. In Python bedeutet das:

```
n  = 20
pi = 3.14159265
w  = 2 * pi / n

for i in range(n):
    winkel = i * w
```

Auf einem Kreis mit Radius R berechne ich für jeden dieser Winkel die Koordinaten von einem Punkt (mit Sinus und Cosinus). Das Python-Programm dazu sieht so aus:

```
import numpy as np                          # Importiere die Funktionen np.sin() und np.cos()

n  = 20
pi = 3.14159265
w  = 2 * pi / n

for i in range(n):
    winkel = i * w
    ax = R * np.cos(winkel)
    ay = R * np.sin(winkel)
```

(5) physikalische Gesetze

Für die Bewegung von Objekten wie bspw. Planeten sind vor allem zwei Arten von Kräften wichtig. Das ist die **Gravitation**, also die Kraft, mit der sich zwei Massen anziehen und die **Zentrifugalkraft**, also die Kraft mit der eine Masse auf einer Kreisbahn gehalten wird. Die Gravitation der Erde nehmen wir als Erdanziehungskraft war, die Zentrifugalkraft merken wir z.B. auf einem Karussell.

In diesem Kapitel erkläre ich, wie man die Gravitation und die Zentrifugalkraft berechnet und wie man mehrere Kräfte kombiniert und deren resultierende Wirkung ausrechnet.

(5.1) Gravitationskraft

Zwei Massen ziehen sich gegenseitig an, wobei die Kraft davon abhängt, wie schwer die beiden Massen **m_1** und **m_2** sind und welchen Abstand **d** sie voneinander haben. Die Formel für die Gravitationskraft ist:

$$F = G \cdot \frac{m_1 \cdot m_2}{d^2}$$

Mit dieser Formel berechne ich also die Länge des Kraft-Vektors. Das **„G“** steht für einen Faktor, mit dem man Längen und Gewichte in eine Kraft umrechnen kann. Es

gilt **G = 6,674e-11** Die Richtung des Kraft-Vektors zeigt vom einen Objekt zum anderen (oder umgekehrt, je nachdem für welches der beiden Objekte die Kraft berechnet wird). Um diese Richtung auch in Koordinaten angeben zu können, setze ich für jedes Objekt einen Punkt ein. Diesen Punkt nennt man **Schwerpunkt** und er liegt genau in der Mitte des Objektes (z.B. der Mittelpunkt einer Kugel). Wenn ich zwei Schwerpunkte A und B und ihre Massen kenne, kann ich den Gravitationsvektor mit dem folgenden Python-Programm berechnen:

```python
import numpy as np                          # Importiere die Wurzel-Funktion: np.sqrt()

gravity = 6.674e-11                         # Gravitationskonstante  [ m^3 / kg / s^2 ]

vx = bx - ax                                # Vektor von A (Anfangspunkt) nach B (Endpunkt)
vy = by - ay
vz = bz - az

Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors

rx = vx / Laenge                            # Richtung des Vektors
ry = vy / Laenge
rz = vz / Laenge

fx = rx * gravity * masse_a * masse_b / (Laenge * Laenge)     # Gravitationsvektor
fy = ry * gravity * masse_a * masse_b / (Laenge * Laenge)
fz = rz * gravity * masse_a * masse_b / (Laenge * Laenge)
```

(5.2) Zentrifugalkraft

Wenn sich ein Objekt auf einer Kreisbahn bewegt, dann fühlt es sich so an, als würde das Objekt nach außen gedrückt. Diese Kraft nennt man Zentrifugalkraft und sie hängt davon ab, wie schnell sich das Objekt bewegt und welchen Radius die Kreisbahn hat. Die Formel für die Zentrifugalkraft ist:

$$F = m \cdot w^2 \cdot r$$

Mit der Formel berechnet man wieder nur die Länge des Kraft-Vektors. Die Richtung des Kraft-Vektors zeigt vom Mittelpunkt der Kreisbahn zu dem Objekt. Man kann die Richtung wieder in Koordinaten berechnen, wenn man von den Koordinaten des Schwerpunktes des Objektes die Koordinaten des Mittelpunktes der Kreisbahn abzieht und diesen Vektor dann durch die Länge teilt.

Die Geschwindigkeit eines Objektes misst man normalerweise in Meter pro Sekunde oder in Kilometer pro Stunde, also indem man eine Strecke misst und dann durch die Zeit teilt, die das Objekt für diese Strecke gebraucht hat. Das funktioniert am besten auf einer geraden Straße. Wenn sich ein Objekt auf einer Kreisbahn bewegt, dann ist es manchmal besser, die Geschwindigkeit in Winkel pro Zeit zu berechnen. D.h. man misst um welchen Winkel sich ein Objekt um den Mittelpunkt gedreht hat und teilt diesen Winkel durch die Zeit. Das nennt man dann Winkelgeschwindigkeit. Wenn ein Objekt T Sekunden benötigt, um eine komplette Umdrehung zu machen (z.B. ein Tag = 86400 sec für eine Drehung um den Mittelpunkt der Erde, ein Jahr = 3,156e+7 sec für eine Drehung um die Sonne), dann ist die Winkelgeschwindigkeit also:

$$w = \frac{360°}{T}$$

In Python verwendet man ja statt dem Winkel die Bogenlänge, d.h. der Winkel 360° wird umgerechnet in 2π und die Winkelgeschwindigkeit ist also:

$$w = \frac{2\pi}{T}$$

Wenn ich also ein Objekt habe, dass sich auf einer Kreisbahn mit dem Radius R bewegt und für eine komplette Runde T sec benötigt, dann berechnet man die Zentrifugalkraft mit dem folgenden Python-Programm:

```
import numpy as np                          # Importiere die Wurzel-Funktion: np.sqrt()

vx = bx - ax                                # Vektor vom Kreismittelpunkt A (Anfangspunkt)
vy = by - ay                                # zum Objekt B (Endpunkt)
vz = bz - az

Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors  =  Radius R der Kreisbahn

rx = vx / Laenge                            # Richtung des Vektors
ry = vy / Laenge
rz = vz / Laenge

pi = 3.14159265
w  = 2 * pi / T                             # Winkelgeschwindigkeit

fx = rx * masse_b * (w * w) * Laenge        # Zentrifugalkraftvektor
fy = ry * masse_b * (w * w) * Laenge
fz = rz * masse_b * (w * w) * Laenge
```

(5.3) Kräftegleichgewicht

Wenn auf ein Objekt verschiedene Kräfte gleichzeitig einwirken, dann kann man die entsprechenden Koordinaten der Kraft-Vektoren addieren und erhält daraus die resultierende Kraft. Wenn also z.B. Kraft-Vektoren (1 I 2 I 3), (4 I 5 I 6) und (7 I 8 I 9) auf ein Objekt einwirken, dann ist die resultierende Kraft (1+4+7 I 2+5+8 I 3+6+9) = (12 I 15 I 18).

Ein besonderer Fall liegt vor, wenn die Addition der Kraft-Vektoren die Summe (0 I 0 I 0) ergibt, denn dann ist die resultierende Kraft gleich Null, d.h. die einzelnen Kräfte heben sich gegenseitig auf. Diese Situation nennt man dann Kräftegleich-gewicht. Als Beispiel kann man die Kräfte (1 I 2 I 3), (-8 I -10 I -12) und (7 I 8 I 9) addieren und erhält (1-8+7 I 2-10+8 I 3-12+9) = (0 I 0 I 0).

Eine Kraft, die auf ein Objekt einwirkt, führt dazu, dass das Objekt sich in die entsprechende Richtung in Bewegung setzt. Wenn mehrere Kräfte wirken, ergibt sich die Bewegung dann aus der resultierenden Kraft (also aus der Summe der Kräfte). Bei einem Kräftegleichgewicht ist die resultierende Kraft gleich Null, d.h. das Objekt ändert seine Bewegung nicht.

Deswegen ist die Berechnung eines Kräftegleichgewichtes für die Simulation des Torus-Planeten so wichtig. Die Gravitationskraft wirkt nach innen und die Zentrifugalkraft wirkt nach außen. Wenn die Gravitation stärker ist als die Zentrifugalkraft, dann kollabiert der Planet. Wenn die Zentrifugalkraft stärker ist als die Gravitation, denn zerreißt der Planet. Nur bei einem Kräftegleichgewicht bleibt der Torus stabil.

(6) Planetenbahnen

Als Beispiel für das Kräftegleichgewicht zwischen Gravitation und Zentrifugalkraft kann man sich die Erde und ihre Umlaufbahn angucken. Das ist zwar nicht ganz genau, denn die Erdumlaufbahn ist ja eigentlich eine Ellipse und keine Kreisbahn. Aber trotzdem kann man dabei die Kräfte und Formeln gut ausprobieren.

Die Sonne hat eine Masse von 1,989e+30 kg und die Erde hat eine Masse von 5,974e+24 kg. Der mittlere Abstand zwischen Sonne und Erde beträgt 1,496e+11 m. Daraus ergibt sich eine Gravitationskraft von:

$$F = G * m_1 * m_2 / d^2 = 6{,}674e\text{-}11 * 5{,}974e+24 * 1{,}989e+30 / (1{,}496e+11)^2$$
$$= \mathbf{3{,}543e+22}$$

die auf die Erde in Richtung zum Mittelpunkt der Sonne hin wirkt. Die Zentrifugalkraft wirkt genau in die gegengesetzte Richtung also von der Sonne weg. Die Winkelgeschwindigkeit der Erde beträgt:

$$w = 2\pi / 1 \text{ Jahr} = 2\pi / 3{,}156e+7 = 1.991e\text{-}7$$

und daher ist die Zentrifugalkraft:

$$F = m * w^2 * r = 5{,}974e+24 * (1{,}991e\text{-}7)^2 * 1{,}496e+11 = \mathbf{3{,}543e+22}$$

Die Gravitationskraft und die Zentrifugalkraft, die auf die Erde wirken sind also genau gleich groß. Da sie aber in entgegengesetzte Richtungen zeigen, heben sie sich gegenseitig auf und die resultierende Kraft ist Null. Die Erde muss also vor 4,5 Milliarden Jahren einmal einen Schubs bekommen haben und bewegt sich seitdem mit der gleichen Geschwindigkeit um die Sonne, denn die Kräfte, die auf sie wirken heben sich gegenseitig auf.

(7) Torus-Planet

Jetzt will ich herausfinden, ob es einen Torus-Planeten geben kann und wie schnell er sich drehen muss, damit er weder kollabiert noch zerreißt. Ich muss also ausrechnen, wie schnell er sich drehen muss, damit die Zentrifugalkraft (nach außen) die Gravitationskraft (nach innen) genau aufhebt. Für den Torus-Planeten sind die Berechnungen natürlich viel komplizierter als für das Erde-Sonne Beispiel wo beide Objekte nur Kugeln sind. Das liegt daran, dass ich beim Torus nicht einfach einen Schwerpunkt in die Mitte setzten kann, denn da hat der Torus ja ein Loch. Stattdessen muss ich viele Schwerpunkte auf einen Kreis setzen und so tun als wäre der Torus aus vielen Kugeln zusammengesetzt. Ganz genau genommen besteht der Torus-Planet ja sogar aus vielen Atomen und jedes Atom erzeugt seine eigene Gravitationskraft. Aber das wäre natürlich viel zu aufwändig zu berechnen und daher setze ich die Schwerpunkte nur auf den mittleren Kreis im Inneren des Torus.

Auch die Zentrifugalkraft ist für den Torus kompliziert auszurechnen, den diese Kraft hängt ja von Radius der Kreisbahn ab und die Bereiche auf der Innenseite des Torus haben einen kleineren Radius als die Bereiche auf der Außenseite und daher wirkt die Zentrifugalkraft dort auch unterschiedlich stark.

Um den Torus-Planeten zu berechnen, möchte ich so vorgehen:

(1) Ich setze einige Schwerpunkte auf den Mittelkreis des Torus. Mit diesen kann ich die Gravitationskraft für jeden anderen Punkt ausrechnen (Bild 1)

(2) Ich berechne die Gravitation für einige Punkte, die ich um den Torus herum auf einem Längengrad (= **„Wurstscheibe“**) platziere (Bild 2)

(3) Wenn ich die Gravitationsvektoren für diese Punkte aufsummiere, erhalte ich die resultierende Gravitationskraft für eine ganze Wurstscheibe (Bild 3).

(4) Ich berechne die Zentrifugalkraft für die Punkte auf der Wurstscheibe um den Torus herum und summiere sie dann auf, um die resultierende Zentrifugalkraft für eine Wurstscheibe zu erhalten.

(5) Wenn die beiden resultierenden Kräfte (Gravitation und Zentrifugal) gleich groß sind, heben sie sich gegenseitig auf und der Torus-Planet bleibt stabil (Bild 4).

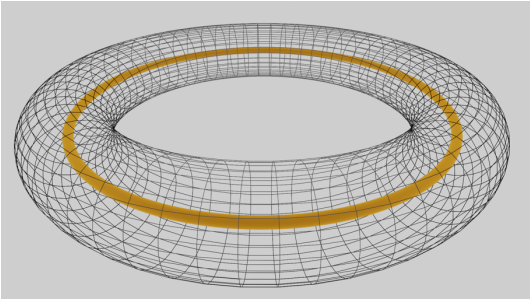


Bild 1: Die Kraftfelder/Schwerpunkte für die Berechnung der Torus-Gravitation werden auf dem Torus-Mittelkreis verteilt.

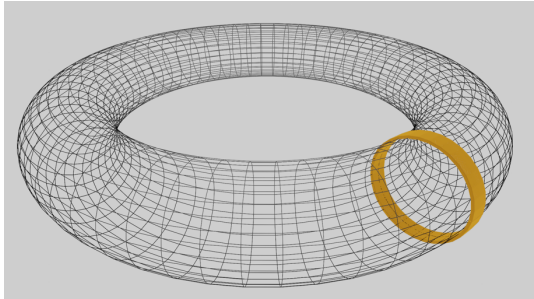


Bild 2: Die Punkte für die Berechnung des Kräftegleichgewichts werden auf einem Torus-Längengrad verteilt

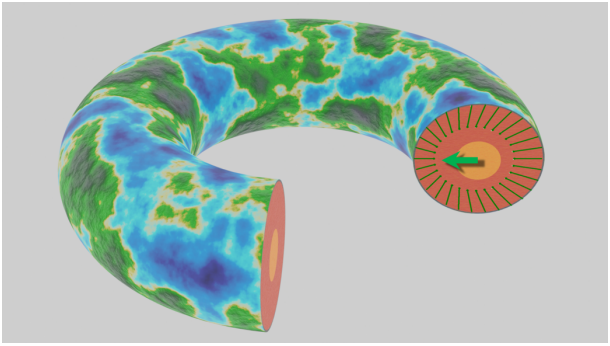


Bild 3: Die resultierende Kraft aus der Summe der Gravitationsvektoren für die Punkte auf einer „Wurstscheibe“ zeigt zum Torus-Zentrum.

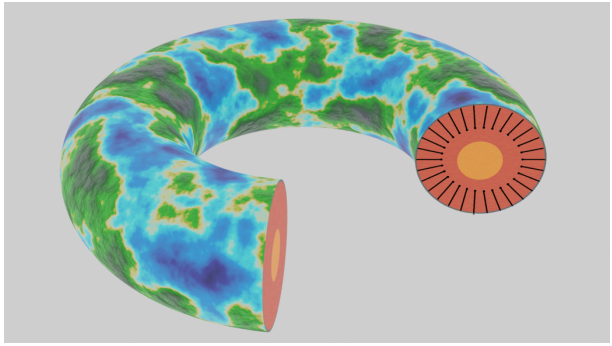
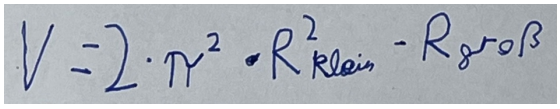


Bild 4: Wenn man die Gravitations-vektoren und die Zentrifugalvektoren addiert, erhält man ein Kräftegleich-Gewicht, so dass der Torus stabil bleibt

(7.1) Größe des Torus-Planeten

Es gibt viele Möglichkeiten, wie ich die Größe und Dicke des Torus-Planeten wählen kann. Mein Planet soll ungefähr so groß sein wie die Erde. Die Erde ist in etwa kugelförmig und hat ein Volumen von 1,083e+21 m^3. Mit ein bisschen Ausprobieren habe ich herausgefunden, dass ich den *großen Radius* des Torus als 11.000 km (also 1,1e+7 m) wählen kann und den *kleinen Radius* als 2.233 km (= 2,233e+6 m).

Das Volumen des Torus kann ich dann berechnen durch:



= 1,974e+1 * $(2,233e+6)^2$ * 1,1e+7
= 1,083e+21

Mein Torus hat also dasselbe Volumen wie die Erde.

(7.2) Torus-Gravitation

Die Gravitationsfelder / Schwerpunkte platziere ich auf dem Mittelkreis des Torus-Planeten (siehe Kapitel 4). Dieser Kreis hat den Radius 11.000 km.

```
import numpy as np                          # Importiere die Funktionen np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
n        = 360

for j in range(n):
    bx = radius_1 * np.cos(2*pi*j/n)
    by = radius_1 * np.sin(2*pi*j/n)
    bz = 0
```

Die Gravitationsvektoren für einen Punkt A berechne ich wie in Kapitel 5.1:

```
import numpy as np                          # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
n        = 360
gravity  = 6.674e-11                        # Gravitationskonstante  [ m^3 / kg / s^2 ]
masse    = 5.974e+24                        # Masse der Erde         [ kg ]
masse_a  = 1.0
masse_b  = masse / n

for j in range(n):
    vx = radius_1 * np.cos(2*pi*j/n) - ax
    vy = radius_1 * np.sin(2*pi*j/n) - ay
    vz = 0                           - az

    Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors

    rx = vx / Laenge                            # Richtung des Vektors
    ry = vy / Laenge
    rz = vz / Laenge

    fx = rx * gravity * masse_a * masse_b / (Laenge * Laenge)
    fy = ry * gravity * masse_a * masse_b / (Laenge * Laenge)
    fz = rz * gravity * masse_a * masse_b / (Laenge * Laenge)
```

wobei ich annehme, dass der Torus-Planet dieselbe Masse hat wie die Erde und dass diese Masse sich gleichmäßig auf alle Gravitationsfelder verteilt (*masse/n*).

Wenn ich die Gravitationsvektoren aufsummiere, dann erhalte ich die resultierende Gravitationskraft, die auf den Punkt A wirkt:

```
import numpy as np                          # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
n        = 360
gravity  = 6.674e-11                        # Gravitationskonstante  [ m^3 / kg / s^2 ]
masse    = 5.974e+24                        # Masse der Erde         [ kg ]
masse_a  = 1.0
masse_b  = masse / n

fx = 0                                      # am Anfang wird die resultierende Kraft auf Null gesetzt
fy = 0
fz = 0

for j in range(n):
    vx = radius_1 * np.cos(2*pi*j/n) - ax
    vy = radius_1 * np.sin(2*pi*j/n) - ay
    vz = 0                           - az

    Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors

    rx = vx / Laenge                            # Richtung des Vektors
    ry = vy / Laenge
    rz = vz / Laenge

    fx += rx * gravity * masse_a * masse_b / (Laenge * Laenge)     # Aufsummieren aller einzelnen Kräfte
    fy += ry * gravity * masse_a * masse_b / (Laenge * Laenge)
    fz += rz * gravity * masse_a * masse_b / (Laenge * Laenge)
```

Jetzt muss ich die Koordinaten der Punkte A auf dem Längengrad / Wurstscheibe des Torus-Planeten berechnen. Diese liegen auf einem Kreis mit Radius 2.233 km, der aber nicht um den Punkt (0 | 0 | 0) geht, sondern um 11.000 km verschoben ist.

```
import numpy as np                              # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
radius_2 = 2.233e+6
m        = 60

for i in range(m):
    ax = radius_2 * np.cos(2*pi*i/m) + radius_1
    ay = 0
    az = radius_2 * np.sin(2*pi*i/m)
```

Als nächstes muss ich für *jeden* dieser Punkte A auf der Wurstscheibe den resultierenden Gravitationsvektor berechnen und diese Vektoren dann *alle* aufsummieren:

```
import numpy as np                              # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
radius_2 = 2.233e+6
m        = 60
n        = 360
gravity  = 6.674e-11                            # Gravitationskonstante  [ m^3 / kg / s^2 ]
masse    = 5.974e+24                            # Masse der Erde         [ kg ]
masse_a  = 1.0
masse_b  = masse / n

fx = 0                                          # am Anfang wird die resultierende Kraft auf Null gesetzt
fy = 0
fz = 0

for i in range(m):
    ax = radius_2 * np.cos(2*pi*i/m) + radius_1
    ay = 0
    az = radius_2 * np.sin(2*pi*i/m)

    for j in range(n):
        vx = radius_1 * np.cos(2*pi*j/n) - ax
        vy = radius_1 * np.sin(2*pi*j/n) - ay
        vz = 0                           - az

        Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)     # Länge des Vektors

        rx = vx / Laenge                            # Richtung des Vektors
        ry = vy / Laenge
        rz = vz / Laenge

        fx += rx * gravity * masse_a * masse_b / (Laenge * Laenge)     # Aufsummieren aller einzelnen Kräfte
        fy += ry * gravity * masse_a * masse_b / (Laenge * Laenge)
        fz += rz * gravity * masse_a * masse_b / (Laenge * Laenge)
```

Die resultierende Gravitationskraft, die dieses Programm für eine Wurstscheibe des Torus berechnet, ist **(fx | fy | fz) = (-69,73 | 0 | 0)**, d.h. der Gravitationsvektor zeigt zum Mittelpunkt des Torus. Die y- und z-Koordinate des Vektors ist Null, da der Torus und die Wurstscheibe beide symmetrisch zur y- und z-Richtung sind und sich daher die Kräfte nach oben und unten und nach vorne und hinten jeweils aufheben und nur eine Kraft nach links übrigbleibt.

(7.3) Torus-Zentrifugalkraft

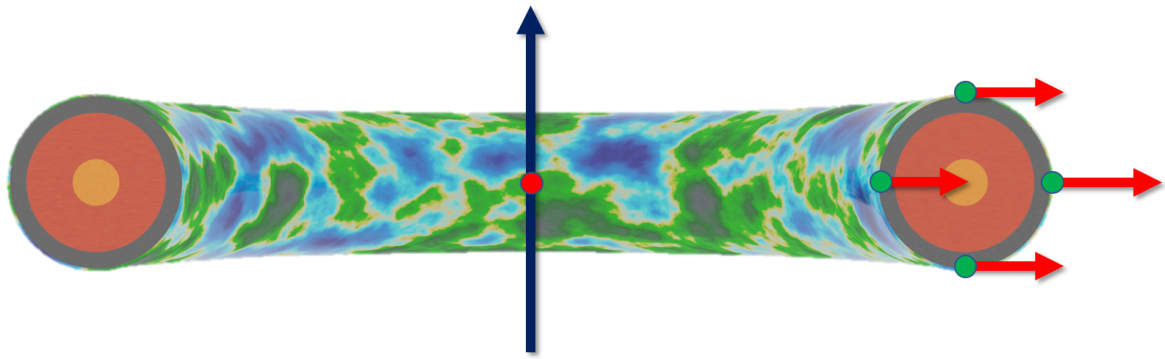
Auch die Zentrifugalkraft muss ich für jeden Punkt auf der Wurstscheibe ausrechnen. Die Koordinaten der Punkte berechnen, geht wie im letzten Kapitel wieder mit:

```
import numpy as np                              # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
radius_2 = 2.233e+6
m        = 60

for i in range(m):
    ax = radius_2 * np.cos(2*pi*i/m) + radius_1
    ay = 0
    az = radius_2 * np.sin(2*pi*i/m)
```

Die Zentrifugalkraft berechne ich dann wie in Kapitel 5.2:

```
import numpy as np                                  # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
radius_2 = 2.233e+6
m        = 60
masse_a  = 1

for i in range(m):
    vx = radius_2 * np.cos(2*pi*i/m) + radius_1     # Vektor von der Rotationsachse zum Punkt A
    vy = 0
    vz = 0

    Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)         # Länge des Vektors  =  Radius der Kreisbahn

    rx = vx / Laenge                                # Richtung des Vektors
    ry = 0
    rz = 0

    w = 2 * pi / T                                  # Winkelgeschwindigkeit

    fx = rx * masse_a * (w * w) * Laenge            # Zentrifugalkraftvektor
    fy = 0
    fz = 0
```

wobei ich diesmal den Vektor von der Rotations***achse*** zum Punkt A berechne und nicht vom Mittelpunkt (0 | 0 | 0), da die z-Koordinate des Punktes A keinen Einfluss auf die Größe und Richtung der Zentrifugalkraft hat, sondern nur der Abstand zur Rotationsachse.



Am Ende muss ich die einzelnen Kraftvektoren noch aufaddieren:

```
import numpy as np                                  # Importiere die Funktionen np.sqrt(), np.sin() und np.cos()

pi       = 3.14159265
radius_1 = 1.100e+7
radius_2 = 2.233e+6
m        = 60
masse_a  = 1

fx = 0
fy = 0
fz = 0

for i in range(m):
    vx = radius_2 * np.cos(2*pi*i/m) + radius_1     # Vektor von der Rotationsachse zum Punkt A
    vy = 0
    vz = 0

    Laenge = np.sqrt(vx*vx + vy*vy + vz*vz)         # Länge des Vektors  =  Radius der Kreisbahn

    rx = vx / Laenge                                # Richtung des Vektors
    ry = 0
    rz = 0

    w = 2 * pi / T                                  # Winkelgeschwindigkeit

    fx += rx * masse_a * (w * w) * Laenge           # Zentrifugalkraftvektor
    fy += 0
    fz += 0
```

Damit kann ich jetzt ausrechnen wie schnell sich der Torus drehen muss, damit die resultierende Zentrifugalkraft die resultierende Gravitation (aus dem letzten Kapitel)

genau aufhebt. Der Wert T misst die Dauer für eine komplette Umdrehung. Mit dem Programm kann ich verschiedene Werte für T ausprobieren. Wenn ich die Werte größer mache, dann dreht sich der Torus langsamer (denn eine komplette Umdrehung dauert dann ja länger) und daher sinkt die Zentrifugalkraft. Wenn ich den Wert von T verkleinere, dann steigt die Zentrifugalkraft.

Für die Suche nach dem richtigen Wert von T bin ich so vorgegangen: Zuerst habe ich den Wert von T solange erhöht bis die Zentrifugalkraft kleiner als die Gravitation ist. Dann habe ich diesen Wert von T halbiert und dann habe ich immer, wenn die Zentrifugalkraft zu groß war, den Wert von T erhöht und wenn die Zentrifugalkraft zu klein war, habe ich den Wert von T verringert. Die Schritte, um die ich T jeweils erhöht oder verringert habe, habe ich auch in jeder Runde halbiert, denn wenn ich einen Wert von T habe, für den die Kraft zu groß ist und einen anderen Wert, für den die Kraft zu klein ist, dann probiere ich am besten als nächstes genau die Mitte zwischen diesen beiden Werten, um den richtigen Wert möglichst schnell zu finden.

In der Tabelle stehen meine Ergebnisse:

Wert von T in sec	**Schrittweite**	**Zentrifugalkraft**	**zu groß zu klein**	**Gravitationskraft**
1		(2,606e+10, 0, 0)	zu groß	(-69,73, 0, 0)
10		2,606e+8	zu groß	-69,73
100		2,606e+6	zu groß	-69,73
1.000		2,606e+4	zu groß	-69,73
10.000		260,6	zu groß	-69,73
100.000	-50.000	2,606	zu klein	-69,73
50.000	-25.000	10,42	zu klein	-69,73
25.000	-12.500	41,69	zu klein	-69,73
12.500	+6.250	166,76	zu groß	-69,73
18.750	+3.125	74,11	zu groß	-69,73
21.875	-1.563	54,45	zu klein	-69,73
20.312	-782	63,15	zu klein	-69,73
19.530	-391	68,31	zu klein	-69,73
19.139	+196	71,13	zu groß	-69,73
19.335	-98	69,70	zu klein	-69,73
19.237	+49	70,41	zu groß	-69,73
19.286	+25	70,05	zu groß	-69,73
19.311	+13	69,87	zu groß	-69,73
19.324	+7	69,77	zu groß	-69,73
19.331		**69,73**		**-69,73**

Ich habe also herausgefunden, dass ein Torus-Planet mit einem großen Radius von 11.000 km und einem kleinen Radius von 2.233 km, der das gleiche Volumen und die gleiche Masse wie die Erde hat, dann stabil ist, wenn er sich in 19.331 sec oder 5h 22m 11s einmal um seine Achse dreht. Dann heben sich nämlich die resultierende Zentrifugalkraft und die resultierende Gravitationskraft, die auf jeden Längengrad (jede Wurstscheibe) wirken, genau auf.

Da ich die Werte in meinen Python-Programmen leicht ändern kann, könnte ich die gleichen Berechnungen auch sehr einfach auf andere Torus-Planeten anwenden, z.B. größere oder kleinere oder dickere oder dünnere.

(8) Simulationen mit *Blender*

Nachdem ich jetzt ausgerechnet habe, dass ein Torus-Planet wirklich existieren kann, möchte ich untersuchen, wie es auf so einem Planeten aussehen würde. Ich benutze dazu die 3D-Software *Blender*, mit der man komplexe 3D Objekte modellieren und rendern kann. *Blender* bietet auch viele Möglichkeiten für die Berechnung von physikalischen Simulationen, so dass man tolle Computergrafik-Animationen herstellen kann. Man kann Lichtquellen und Kraftfelder in einer 3D Szene platzieren und Blender rechnet dann aus, wie die Objekte sich bewegen und wie sie aussehen. Für dieses Projekt habe ich vor allem die Simulation von starren Körpern und die Flüssigkeitssimulation verwendet.

Bis auf die Tageslicht-Simulation ist das Aussehen der Objekte für das Projekt eigentlich gar nicht so wichtig. Aber es macht trotzdem viel mehr Spaß, wenn die Computergraphik-Animationen realistisch aussehen. Ich habe zu den Simulationen ein paar Videos gemacht, die man auf meiner Internet-Seite herunterladen und angucken kann: https://donaldinho-k.github.io/torus.html

(8.1) Entstehung und Stabilität

Auf mehreren Seiten im Internet habe ich gelesen, dass Torus-Planeten zwar theoretisch existieren könnten, dass man sich aber nicht vorstellen kann, wie so ein Planet entstehen sollte. Für meine erste Simulation habe ich daher erstmal ausprobiert, ob ich einen stabilen Torus-Planeten mit *Blender* erzeugen kann.

Dazu habe ich, wie bei meinen eigenen Berechnungen, 360 Kraftfelder auf einem Kreis mit 11.000 km Radius platziert. Man kann bei *Blender* angeben, ob Kraftfelder anziehend oder abstoßend sein sollen und wie schnell die Kraft abnimmt, wenn man sich von dem Kraftfeld entfernt („Falloff“). Bei Gravitationsfeldern ist Falloff = 2, denn wenn man den Abstand zum Gravitationszentrum verdoppelt, nimmt die Kraft um den Faktor 4 ($= 2^2 = 2^{\text{Falloff}}$) ab. In der Mitte des Kreises habe ich ein abstoßendes Kraftfeld platziert, das für die Zentrifugalkraft verantwortlich ist.

Leider kann man bei *Blender* für die Stärke eines Kraftfeldes nur eine Zahl angeben, aber z.B. keine Masse oder Winkelgeschwindigkeit. Deswegen musste ich ausprobieren, bei welchem Verhältnis von anziehenden Gravitationsfeldern und abstoßendem Zentrifugalfeld ein Kräftegleichgewicht entsteht. Mit einer ähnlichen Suche/Tabelle wie im letzten Kapitel ging das aber ganz gut.

Um das Verhalten eines Torus aus flüssiger Lava zu simulieren, habe ich einen Torus aus Wasser mit der entsprechenden Größe in die Szene gesetzt und dann die Flüssigkeitssimulation von *Blender* gestartet. Wenn die Zentrifugalkraft zu klein ist, dann fällt der Torus in sich zusammen, wenn sie zu groß ist, dann spritzt das Wasser auseinander. Bei der richtigen Einstellung ergibt sich aber tatsächlich ein Kräfte-gleichgewicht und der Wasser-(Lava)-Torus bleibt stabil. **[Videos 1a, 1b, 1c]**

(8.2) Tag und Nacht

Durch die Rotation des Torus-Planeten entstehen Tag und Nacht. Wenn, wie bei der Erde, die Rotationsachse gegenüber der Sonne gekippt ist, dann verändern sich Tag und Nacht über die verschiedenen Jahreszeiten. Das Tageslicht auf der Außenseite des Torus verhält sich ganz ähnlich dazu, wie es auch auf der Erde ist. Auf der

Innenseite sieht es aber ganz anders aus, weil die Innenseite oft im Schatten der gegenüberliegenden Torus-Seite liegt. **[Video 2a, 2b, 2c]**

Auf der Erde geht nördlich des Polarkreises die Sonne im Sommer nie unter und im Winter nie auf. Auf der Innenseite des Torus-Planeten hat man zwar im Sommer und Winter ein bisschen Sonnenschein, aber dafür ist es im Frühjahr und Herbst dunkel, weil die Sonne von der anderen Torus-Seite verdeckt wird.

Außerdem sind die Tage auf dem Torus-Planeten im Vergleich zur Erde ziemlich kurz, denn der Torus dreht sich sehr schnell (eine Umdrehung pro 5:22h statt wie bei der Erde 23:56h). Es wäre aber trotzdem denkbar, dass sich Tiere und Pflanzen an diese Situation anpassen können.

(8.3) Wasser auf dem Torus-Planeten

Auf der Erde ist die Gravitation überall gleich stark und wirkt in Richtung des Erdmittelpunktes. Deshalb ist eine Wasseroberfläche bis auf den Wellengang immer flach. Auf dem Torus-Planeten heben sich die Kräfte über eine ganze Wurstscheibe zwar gegenseitig auf, aber die Gravitation ist nicht überall gleich stark und die Richtung der Gravitationskraft zeigt nicht immer zum Mittelkreis des Torus (sie zeigt also nicht immer senkrecht zur Torus-Oberfläche nach unten). Wenn man Wasser auf der Oberfläche des Torus-Planeten fließen lässt, dann bildet sich keine flache Oberfläche, sondern außen und innen türmt sich das Wasser auf und oben und unten bilden sich Dellen. **[Video 3a, 3b, 3c, 3d]**

(8.4) Gebäude auf dem Torus-Planeten

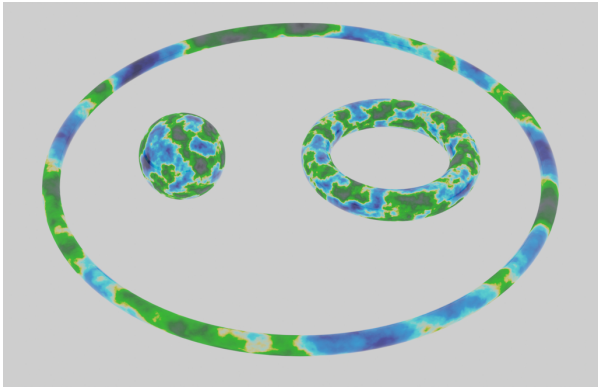
Wenn die Gravitationskraft auf dem Torus-Planeten nicht senkrecht zur Oberfläche nach unten zeigt (nicht in Richtung des Mittelkreises), dann würde es einem so vorkommen, als geht es bergauf oder bergab. Häuser und Türme müssen also so gebaut werden, als ob sie an einem Hang stehen, sonst fallen sie um. In dieser Simulation lasse ich Türme aus Kapla-Klötzen zusammenstürzen, um auszuprobieren, wie diese Unterschiede an verschiedenen Stellen auf dem Torus aussehen. Außen und innen ist alles normal, wie auf der Erde, aber oben und unten fallen die Türme zum Torus-Mittelpunkt hin um. **[Video 4a, 4b, 4c, 4d]**

(9) Fazit

In meinem Projekt habe ich ausgerechnet, dass es einen Torus-Planeten tatsächlich geben kann. Wenn der Torus-Planet sich mit der richtigen Geschwindigkeit um seine Achse dreht, dann heben sich die Gravitationskraft und die Zentrifugalkraft gegenseitig auf. Ich habe dazu ein Python-Programm geschrieben, mit dem man die Gravitationskraft und die Zentrifugalkraft ausrechnen kann und dann habe ich die Umdrehungsdauer so eingestellt, dass ein Kräftegleichgewicht herrscht.

Auf meinem Planeten mit 11.000 km großem Radius dauert ein Tag 5:22h was bestimmt gewöhnungsbedürftig ist, aber noch ok. Auf der Außenseite des Torus geht die Sonne auf und unter wie bei uns auf der Erde. Auf der Innenseite können aber wahrscheinlich keine Pflanzen leben, weil es da fast immer dunkel ist, denn der Torus wirft da meistens einen Schatten auf sich selbst.

Die Summe aller Kräfte, die auf eine Wurstscheibe wirken, heben sich gegenseitig auf. Mit meinem Python-Programm kann ich mir aber auch die einzelnen Kräfte wie in Bild 3 & 4 angucken und ausrechnen, wie stark eigentlich die Erdanziehungskraft wäre. Bei meinem 11.000 km Torus stellt sich heraus, dass die Anziehungskraft zwischen 4.5 auf der Außenseite und 5.3 auf der Oberseite liegt. Das ist ungefähr die Hälfte von der Anziehungskraft auf der Erde. Wir wären auf dem Torus-Planeten also nur halb so schwer wie auf der Erde.



Als nächstes könnte ich die Berechnungen auch für andere Torus-Planeten machen. Wenn ich z.B. den Radius größer mache, dann dreht sich der Torus-Planet langsamer. Bei einem Radius von 36.160 km ist die Umdrehungsdauer (ein Tag) genau wie auf unserer Erde 23:56h. Bei so einem großen Radius wird der Torus aber auch immer dünner, hier 1.232 km, weil ich die Masse und das Volumen wie bei der Erde annehme. Das Bild zeigt das Verhältnis vom 11.000 km und 36.160 km Torus im Vergleich zur Erde.

Leider wurde im Weltall bisher noch kein Torus-Planet entdeckt.

(10) Referenzen

Die Blender-Software kann sich jeder kostenlos unter https://www.blender.org herunterladen und sofort starten. Blender ist eine super tolle Software.

Die Python-Programme habe ich mit Jupyter Notebooks geschrieben. Das Projekt kann man unter https://jupyter.org herunterladen, aber es ist ziemlich kompliziert das auf dem eigenen Computer zu installieren. Da sollte man sich Hilfe holen.

Die Grundlagen zu den großen Zahlen, den Vektoren und dem Sinus und Cosinus hat mir mein Vater erklärt, der mich auch bei dem gesamten Projekt unterstützt hat und mir beim Schreiben des Berichtes geholfen hat.

Die physikalischen Formeln kann man bei Wikipedia (https://www.wikipedia.de) finden:

https://de.wikipedia.org/wiki/Zentrifugalkraft

https://de.wikipedia.org/wiki/Gravitation

https://de.wikipedia.org/wiki/Gravitationskonstante

Es gibt im Internet ziemlich viele Web-Seiten, wo über Torus-Planeten diskutiert wird, z.B.:

https://en.wikipedia.org/wiki/Toroidal_planet

https://www.quora.com/Is-a-torus-shaped-planet-possible

https://www.quora.com/What-is-the-gravitational-field-of-a-torus-shaped-planet