VPYTHON

zum Verstehen der Grundstruktur der Newtonschen Mechanik

Hildegard Urban-Woldron, Martin Hopf

Im Oberstufenlehrplan für Physik wird im Sinne der Zukunftsorientierung des Physikunterrichts die Modellbildung als wichtiger didaktischer Grundsatz explizit angeführt: *„Entsprechend der Zukunftsorientierung des Unterrichts sind auch moderne Methoden der Informationsbeschaffung, der Datenerfassung (Messen, Steuern, Regeln) und –verarbeitung sowie der Modellbildung im Unterricht einzusetzen."* Die Modellbildung entspricht nicht nur der Denk- und Arbeitsweise der Physik, sondern fördert auch die Hypothesenbildung und das Denken in Zusammenhängen im Physikunterricht. Im Physikunterricht treten darüber hinaus häufig Differentialgleichungen auf, wo eine geschlossene Lösung des Problems nicht vermittelt werden kann, weil die mathematischen Kenntnisse der Schülerinnen und Schüler auf der entsprechenden Schulstufe nicht ausreichend sind.

Schon seit etwa zwei Jahrzehnten stehen den Schulen Tabellenkalkulationsprogramme, die sich auch sehr gut für die Modellbildung und Visualisierung im Physikunterricht eignen, zur Verfügung und werden von Physiklehrkräften auch in diesem Sinne eingesetzt. Heute sind noch weitere, leicht erlernbare Modellbildungssysteme (wie z.B. EASYSIM, VENSIM) kostenlos verfügbar und ermöglichen Kolleginnen und Kollegen einen schnellen Einstieg in die Programmnutzung. Eine besonders interessante Software ist VPYTHON. VPYTHON ist eine graphische Erweiterung zum Programm PYTHON, die das Erstellen von Animationen und Simulationen in 3D, aber auch die Darstellung der Simulationsergebnisse als Graphen in einem eigenen Fenster ermöglicht. Zur relativ einfachen Bedienung kommt hinzu, dass der Programmeinsatz die Kreativität der Lernenden und vor allem auch das lustbetonte Arbeiten an physikalischen Fragestellungen fördern kann.

Fachdidaktische Überlegungen

In diesem Beitrag werden Beispiele zur Untersuchung der Bewegung von Körpern unter dem Einfluss von Kräften mit Hilfe der objektorientierten Programmsprache VPYTHON für den Einsatz in der Sekundarstufe 2 angeregt. Im Idealfall könnten die programmiertechnischen Grundlagen im Pflichtgegenstand Informatik vermittelt werden, so dass die Lehrperson in Physik den Fokus hauptsächlich auf die systemdynamische Modellbildung legen könnte. Ausgewählt wurden Modelle, in denen die Grundstruktur der Newtonschen Dynamik deutlich hervortritt. Kennt man die auf einen Körper wirkenden Kräfte, kann man daraus seine Bahnkurve berechnen. Dabei lassen sich die Grundgleichungen (1) und (2) der Newtonschen Grundstruktur nicht nur auf Phänomene anwenden, die im engeren Sinn zur Mechanik zählen, sondern gelten z.B. auch für die Berechnung der Bahnen elektrisch geladener Teilchen unter dem Einfluss magnetischer Kraftfelder oder elektrischer Felder.

$$\vec{a} = \ddot{\vec{x}} = \frac{\Sigma \vec{F}_i}{m} \qquad (1)$$

beziehungsweise

$$d\vec{p} = \Sigma \vec{F}_i \cdot dt \qquad (2)$$

Wie Gleichung (1) zeigt, geht es bei der Berechnung von Bahnkurven mathematisch darum, eine Differentialgleichung zweiter Ordnung zu integrieren. Diese Grundbeziehung stellt zusammen mit den Definitionen von Geschwindigkeit und Beschleunigung das Rüstzeug für die Bearbeitung von Bewegungen und Kräften dar. Bei Gleichung (2) wird zum Ausdruck gebracht, dass eine Kraft, die eine bestimmte Zeit auf einen Körper einwirkt, eine Impuls- und damit Bewegungsänderung hervorruft. Die Kraft tritt dabei als Änderungsrate des Impulses auf:

$$\vec{F} = \frac{\overrightarrow{\Delta p}}{\Delta t} \rightarrow \overrightarrow{\Delta p} = \vec{F} \cdot \Delta t \rightarrow \vec{p}_{neu} = \vec{p}_{alt} + \vec{F} \cdot \Delta t$$

Die Beschleunigung eines Körpers ergibt sich aus der Summe der Einzelkräfte:

$$\vec{a} = \frac{\Sigma \vec{F}_i}{m}$$

Die Beschleunigung ist anschaulich gesehen die Intensität der Geschwindigkeitsänderung, d.h. die Änderungsrate der Geschwindigkeit:
Die Geschwindigkeit v ändert sich im Zeitintervall Δt um einen bestimmten Wert $\overrightarrow{\Delta v} = \vec{a} \cdot \Delta t$, der zur vorhandenen Geschwindigkeit addiert wird:

$$\vec{v}_{neu} = \vec{v}_{alt} + \overrightarrow{\Delta v} \rightarrow \vec{v}_{neu} = \vec{v}_{alt} + \vec{a} \cdot \Delta t$$

Die Geschwindigkeit ist die Intensität der Änderung des Ortes. In einem Zeitintervall Δt bewirkt sie eine Ortsänderung von $\overrightarrow{\Delta v} = \vec{a} \cdot \Delta t$, die zur vorhandenen Ortskoordinate addiert wird: $\vec{s}_{neu} = \vec{s}_{alt} + \vec{v} \cdot \Delta t$.

Eine Modellbildungssoftware wie z. B. VPYTHON erlaubt eine stärkere Betonung dieser Kernstruktur der Newtonschen Dynamik und arbeitet automatisch mit Vektoren: $\vec{F} \rightarrow \vec{a} \rightarrow \vec{v} \rightarrow \vec{s}$ beziehungsweise $\vec{F} \rightarrow \vec{p} \rightarrow \vec{v} \rightarrow \vec{s}$.

Dr. Hildegard Urban-Woldron, Univ. Prof. Dr. Martin Hopf / Universität Wien, AECC Physik. E-Mail: hildegard.urban-woldron@univie.ac.at

Einführung in das Arbeiten mit VPYTHON

Die Programme PYTHON und VPYTHON sowie Dokumentationen und Beispiele stehen auf *http://www.vpython.org/* kostenlos zum Download bereit. Videos, die z. B. auf http://*www.youtube.com/vpythonvideos* verfügbar sind, unterstützen die Anwender/innen beim selbstständigen Erlernen des Umgangs mit dem Programm. Unter *http://vpython.org/webdoc/visual/index.html* findet man zusätzliche Detailinformationen zu den einzelnen in VPYTHON verfügbaren Objekten. Auf YouTube stehen zahlreiche Videos bereit, die verschiedenste Anwendungen von VYPTHON zeigen.

VPYTHON ist auf den Systemen Windows, Linux/Unix und Mac OS X lauffähig. VIDLE steht für Visual Integrated Development Environment und stellt das Interface zum Schreiben und Ausführen von VPYTHON Programmen dar.

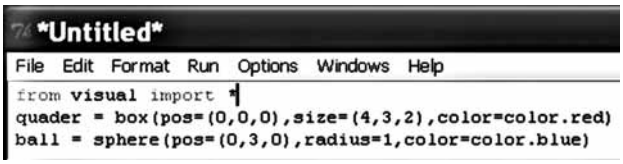
Das Programm ist einfach zu installieren und problemlos auf Netbooks lauffähig. Es wurde im Rahmen einer Intensivwoche im Talentezentrum Schloss Drosendorf mit Schülerinnen und Schülern der 3. und 4. Klassen niederösterreichischer Hauptschulen und AHS erfolgreich eingesetzt.

Objekte in VYPTHON darstellen

Nach dem Starten der VIDLE wird das Programm im sogenannten Code–Fenster eingegeben (vgl. Abb. unten rechts).

Es werden hier zwei Objekte dargestellt:

(1) Ein Quader mit Länge 4, Breite 2 und Höhe 3 in der Farbe rot.

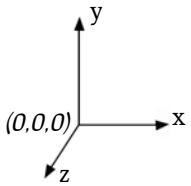


Der Mittelpunkt des Quaders liegt im Ursprung des Koordinatensystems.

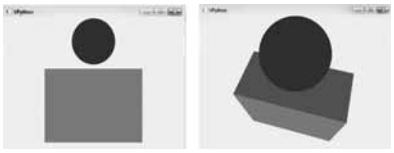
(2) Eine blaue Kugel mit Radius 1. Der Mittelpunkt der Kugel liegt 3 Einheiten über jenem des Quaders.

Durch Betätigen der Taste F5 wird das Programm gestartet. Im graphischen Ausgabefenster erhalten wir die Darstellung der beiden Objekte in 3D.

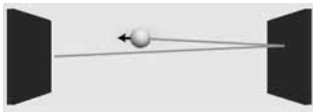
Im Zentrum des Anzeigefensters liegt der Ursprung des Koordinatensystems. Die positive x–Achse verläuft nach rechts, die positive y–Achse nach oben und die positive z–Achse aus dem Bildschirmfenster heraus. Die Einheiten auf den Achsen werden automatisch angepasst (vgl. Abb. rechts).



Bewegt man die Maus mit gedrückter Maustaste, so kann man die räumliche Ansicht beliebig variieren (vgl. Abb.). Hält man beide Maustasten gedrückt, kann man durch Bewegen der Maus zoomen.



Beispiel: Bewegungen eines Balls darstellen

from visual import * Ball = sphere(pos = (-5,0,0), radius = 0.5, color = color.yellow) Wand = box(pos = (6,0,0), size = (0.2,4,4), color = color.blue)	*Objekte definieren*
dt = 0.05 Ball.v = vector(2,0,0)	*Anfangsbedingungen festlegen. Die Geschwindigkeit des Balls wird als Vektorgröße festgelegt.*
while (1==1): rate(100)	*rate(100) stellt sicher, dass die While - Schleife unabhängig von der Rechnergeschwindigkeit nicht öfter als 100 Mal pro Sekunde ausgeführt wird.*
Ball.pos = Ball.pos + + Ball.v * dt	*Die Bewegung des Balls wird in einer einzigen Zeile dargestellt.*
if Ball.x > Wand.x - 0.15: Ball.v.x = - Ball.v.x	*Der Ball wird beim Auftreffen auf die Wand reflektiert. Er geht aber auf der linken Seite aus dem Bildbereich hinaus.*
Erweiterungen: 1. Es soll eine zweite Wand auf der linken Seite hinzugefügt werden, so dass sich der Ball zwischen den beiden Wänden hin und her bewegt.	
2. Der Ball soll eine andere Anfangsgeschwindigkeit haben, z.B. Ball.v = vector(1,0.1,0)	
3. Im nächsten Schritt soll die Geschwindigkeit des Balls visualisiert werden. Folgende Änderungen sind durchzuführen: a) Im Abschnitt Anfangsbedingungen wird die Zeile **bv=arrow (pos (Ball.pos, axis=Ball.v, color=color.red)** eingefügt.	
b) In der While-Schleife werden die Position und die Richtung des Pfeiles mit den beiden Befehlen **bv.pos=Ball.pos** und **bv.axis=Ball.v** aktualisiert	
4. Der Ball soll nun noch eine Spur hinterlassen. Dazu wird a) bei den Anfangsbedingungen **Ball.Spur=curve(color= (0.8,0.8,0.8))** und b) am Ende der Schleife **Ball.Spur.append(pos= Ball.pos)** eingefügt.	
5. Es sollen nun noch eine Boden- und eine Deckfläche eingefügt und die Reflexionen des Balls an diesen beiden zusätzlichen Flächen animiert werden.	

Mögliche Arbeitsaufträge für SchülerInnen

Beispiel 1: Federschwingung

- Studiere das folgende Beispiel und beschreibe, was damit simuliert wird.
- Wodurch unterscheidet sich die Methode der Simulation grundsätzlich vom Beispiel mit dem Ball in der Box?
- Baue schließlich einen Reibungsterm ein.

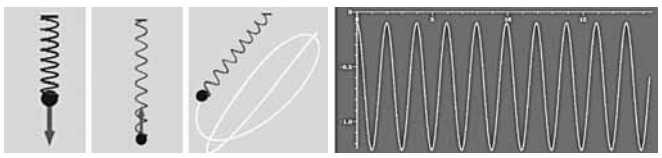
```
from visual import *
scene.range = (1,1,1)
scene.center = (0,.1,0)
scene.width = 800
scene.height = 275
scene.background = (0.8,0.8,0.8)

k = 10
m = 10

x0 = vector(.65,0,0)
v = vector(0,0,0)
a = vector(0,0,0)
F = vector(0,0,0)

Surface=box(size=(2,.02,.5), pos=(0,-.1,0))
wall=box(size=(.04,.5,.3), pos=(-.77,.15,0))
spring=helix(pos=(-.75,0,0),axis=x0,
radius=.08, coils=6, thickness=.01,
color=color.red)
block=box(pos=(0,0,0), size=(.2,.2,.2),
color=color.blue)

block.pos = (0.15,0,0)
x = block.pos
spring.axis = x0 + x
finished = False
dt = .01

while not finished:
rate(100)
F = - k*x
a = F / m
v = v + a*dt
x = x + v*dt +
      + .5*a*dt**2
block.pos = x
spring.axis = x0 + x
```

Die physikalischen Größen werden als Vektoren definiert.

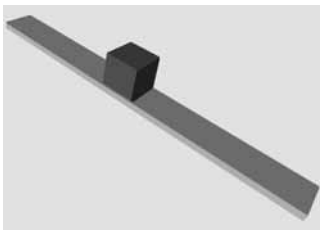
Diese vier Zeilen enthalten die eigentliche „Physik" der Problemstellung

Beispiel 2: Federpendel

- Erstelle eine Simulation für ein Federpendel, das nach allen möglichen Richtungen ausgelenkt werden kann.
- Stelle die resultierende Kraft auf den Pendelkörper sowie die Bahn des Pendelkörpers und die Auslenkung in Abhängigkeit von der Zeit dar.

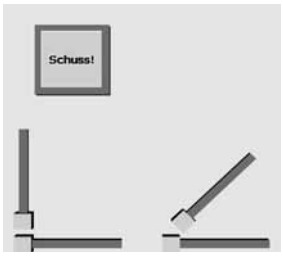


Beispiel 3: Bewegung eines Körpers auf einer schiefen Ebene



- Stelle die Bewegung eines Körpers, der reibungsfrei eine schiefe Ebene hinuntergleitet, dar.
- Welche Kräfte wirken auf den Körper? Visualisiere die entsprechenden Kraftvektoren.
- Erstelle ein v-t-Diagramm.
- Wie hängt die Endgeschwindigkeit des Körpers von der Länge und vom Neigungswinkel des Brettes ab?

Beispiel 4: Basketballwurf



Studiere das Programm und adaptiere es in der Weise, dass du damit Wurfbewegungen simulieren und sowohl Geschwindigkeitsvektoren als auch Zeit-Weg- sowie Zeit-Geschwindigkeitsdiagramme anzeigen kannst.
Visualisiere die Bahn und die Geschwindigkeit des Balls.

```
from visual.controls import*
def setx(obj):
   ball.x=obj.value
   ball.x0=ball.x
def sety(obj):
   ball.y=obj.value
   ball.y0=ball.y
def setangle(obj):
   ball.angle=obj.value
def setv0(obj):
   ball.speed=obj.value
def shoot():
ball.vel=vector(ball.speed*cos(ball.angle),
    ball.speed*sin(ball.angle),0)
   v=ball.vel
   a=vector(0,-9.8,0)
   dt=0.1
while ball.y > -5.0:
      rate(10)
       v.y=v.y - 9.8*dt
       ball.pos=ball.pos+v*dt+
       +0.5*a*dt**2
       ball.x=ball.x0
       ball.y=ball.y0

w=350
display(x=w,y=0,width=3*w,h
    eight=w,range=10)
c=controls(x=0,y=0,width=w,
    height=w,range=60)
ball=sphere(radius=0.2,
    color=(1,1,0), angle=0.0,
    x0=0,y0=0)

ball.speed=10.0
ball.vel=vector(ball.speed,0,0)
ring(pos=(8,2,0),axis=(0,1,0),r
    adius=0.5,thickness=0.1,
color=color.red)
```

```
s1=slider(pos=(-50,-50), width=7, length= 0, axis=(1,0,0), min=-10,
    max=10,  action=lambda: setx(s1))
s2=slider(pos=(-50,-40),  width=7, length=40, axis=(0,1,0), min=-5,
    max=5, action=lambda: sety(s2))
s3=slider(pos=(15,-40),width=7, length=40, axis=(0.7,0.7,0),min=0,
    max=pi/2.0,
     action=lambda: setangle(s3))
s4=slider(pos=(10,-50), width=7, length=40, axis=(1,0,0),
    min=0,max=20,
      action = lambda: setv0(s4))

b1 = button(pos=(-30,30),height = 30, width = 30, text = „Schuss", action
    = lambda: shoot())

s1.value = 0
setx(s1)
s2.value = 0
sety(s2)
s3.value = 0
setangle(s3)
s4.value = 0
setv0(s4)

while 1:
   rate(100)
   c.interact()
```