Der Raumigel unter Python

Veit Berger, Hans-Martin Hilbig

T³ DEUTSCHLAND

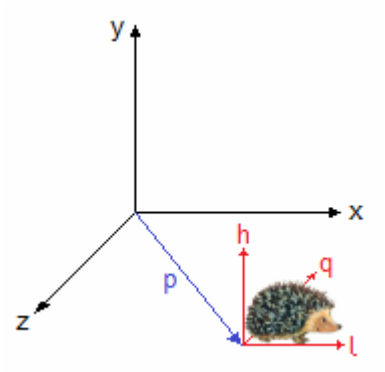
Teachers Teaching with Technology™

Einleitung

Die zweidimensionale Turtle- oder Igelgrafik ist ein häufig genutztes Werkzeug, Schülerinnen und Schülern grundlegende Programmstrukturen im Informatikunterricht zu vermitteln. Um auch die Vorstellung räumlicher Figuren stärker zu schulen, wurde bereits 1987 an der Pädagogischen Hochschule Ludwigsburg ein Raumigel in der Programmiersprache Logo zur Programmierung einfacher koordinatenfreier 3D-Grafiken veröffentlicht [1]. Nachdem sich schon seine objektorientierte Implementierung in den Programmiersprachen Pascal und Delphi bewährt hat [2], soll er nachfolgend mit Anwendungen in der MicroPython-Umgebung der neuen TI-Taschenrechnergeneration vorgestellt werden. Dabei soll der Fokus weniger auf seiner Implementierung als vielmehr auf Vorschlägen für seinen Einsatz im Informatikunterricht und in den anderen MINT-Fächern liegen.

Eigenschaften und Methoden des Raumigels

Wie in einer zweidimensionalen Igelgrafik stellen wir uns vor, dass sich ein kleiner Igel nun im dreidimensionalen Raum bewegt und dabei mit einem Zeichenstift eine Spur hinterlässt. Diese wird als Parallelprojektion (Kavalierperspektive) auf der zweidimensionalen Bildschirmebene abgebildet. Die Bewegung im statischen xyz-System wird vektorgeometrisch mit einem Dreibein beschrieben, das aus den Einheitsvektoren l (Längsachse), q (Querachse) und h (Höhenachse) besteht (vgl. Abbildung). Neben den Drehbewegungen um diese drei Raumachsen werden aber nur Längsbewegungen in l-Richtung zugelassen, die der Blickrichtung des Igels entspricht.



In der Objektklasse Raumigel sind damit die nachfolgenden Methoden zur Bewegung im Raum implementiert. Distanzen werden in Bildpunkten angegeben, Winkel im Gradmaß.

Methode	Beschreibung
`vw(distanz) / rw(distanz)`	vorwärts / rückwärts gehen
`re(winkel) / li(winkel)`	nach rechts bzw. links drehen (Drehung um die h-Achse)
`kvo(winkel) / khi(winkel)`	nach vorn bzw. hinten kippen (Drehung um die q-Achse)
`nre(winkel) / nli(winkel)`	nach rechts bzw. links neigen (Drehung um die l-Achse)

Darüber hinaus stehen weitere nützliche Methoden zur Verfügung. Dazu gehören u. a.:

Methode	Beschreibung
`clear() / clearscreen()(`	Bildschirm löschen bzw. Bildschirm löschen mit Zurücksetzen des Igels im xyz-System
`set_rgb(r,g,b)`	Zeichenfarbe festlegen r, g, b = 0 ... 255
`set_text(x,y,text)`	Text in den Bildschirmkoordinaten (x; y) ausgeben
`pen_down() / pen_up()`	Zeichenstift absenken bzw. anheben
`darstellen()`	Igel als kleines gleichseitiges Dreieck visualisieren
`zeigen() / ausblenden`	Visualisierung des Igels ein- bzw. ausschalten

Ein großer Vorteil der Igelgeometrie besteht darin, gezeichnete Figuren im xyz-System bewegen zu können. Dazu können folgende Methoden genutzt werden:

Methode	Beschreibung
`x_rotation(winkel)`	eine Igelgrafik in einem Winkel um die x - Achse des xyz-Systems drehen
`y_rotation(winkel)`	eine Igelgrafik in einem Winkel um die y - Achse des xyz-Systems drehen
`z_rotation(winkel)`	eine Igelgrafik in einem Winkel um die z - Achse des xyz-Systems drehen

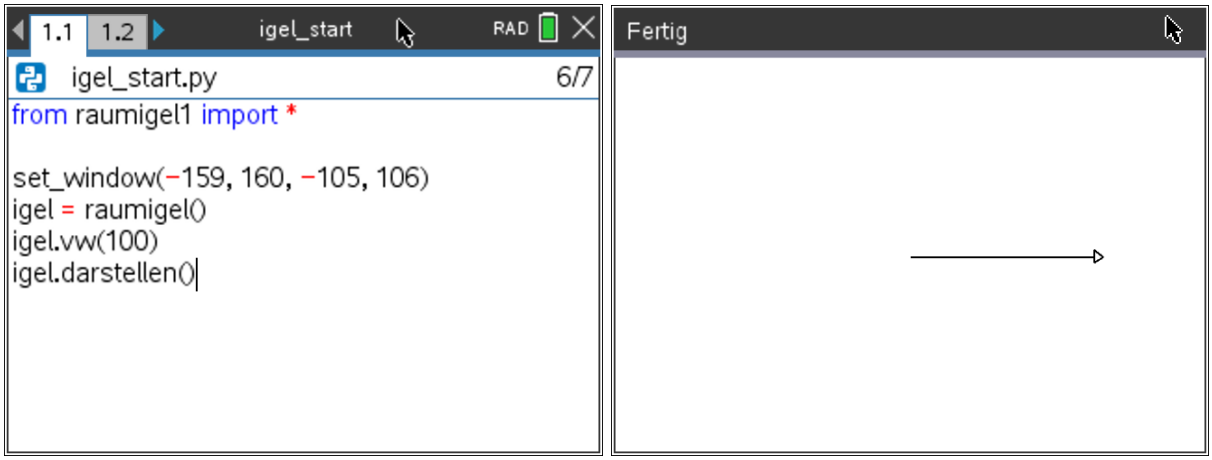
Auch hier werden die Winkel im Gradmaß angegeben.

Erste Schritte mit dem Raumigel

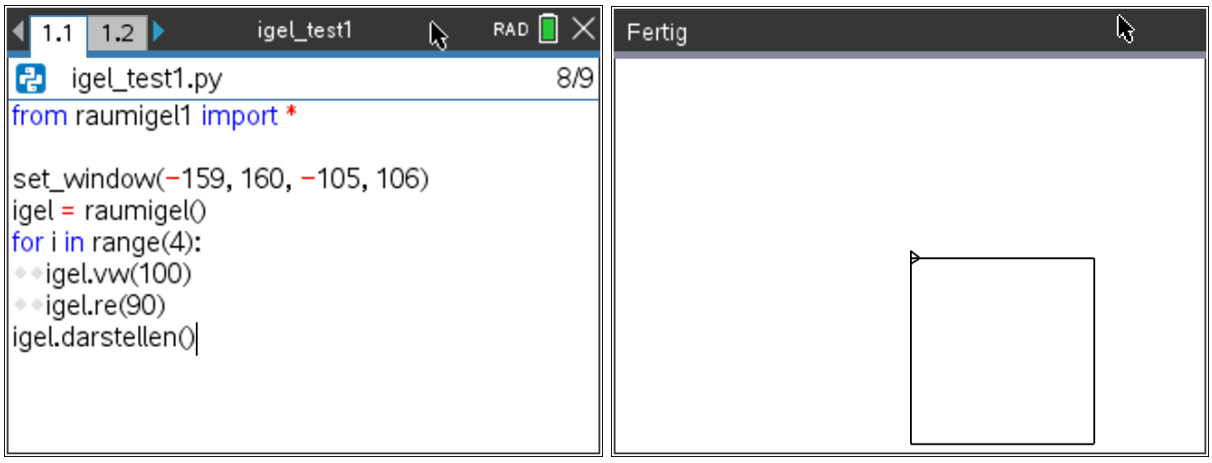
Zunächst muss die Klassenbibliothek **raumigel1.py** in das lokale Python-Bibliotheks-Verzeichnis kopiert und ggf. aktualisiert werden.

Handheld: Eigene Dateien → PyLib
Windows 10: C:\Users\Benutzer\Documents\TI-Nspire CX\PyLib

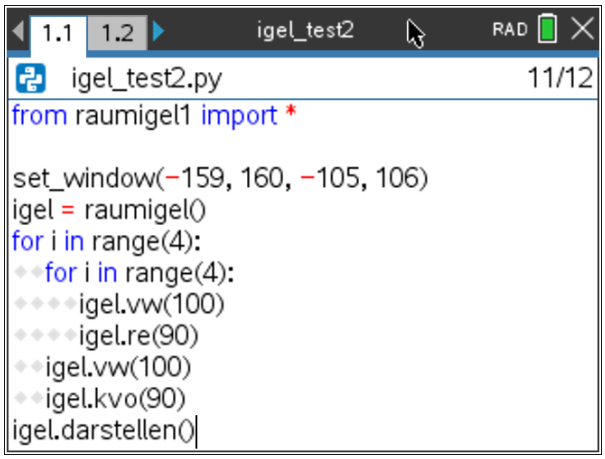
Die nachfolgenden Abbildungen zeigen die Instanziierung und eine einfache Bewegung eines Igel-Objektes. Zu Beginn ist es ratsam, den Igel nach der letzten Bewegung zu visualisieren, um eine Vorstellung von seiner Orientierung im Raum zu haben.

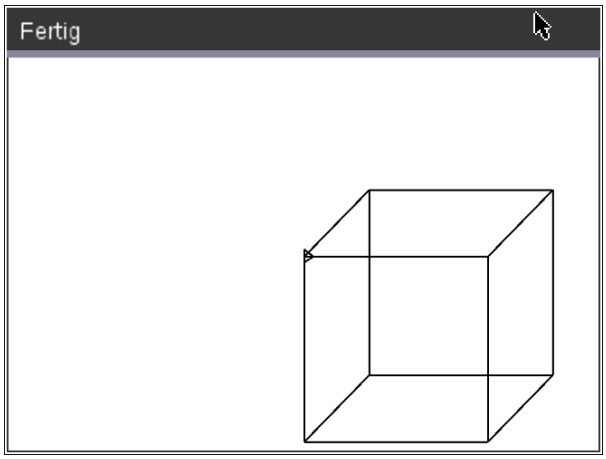


In seiner letzten Position lassen wir den Igel um 90° nach rechts drehen.
Werden die Methoden **vw(100)** und **re(90)** insgesamt 4 mal aufgerufen, entsteht ein Quadrat in der x-y-Ebene des xyz-Systems:



Mit den Zwischenschritten **vw(100)** und **kvo(90)** lässt sich nun aus 4 Quadraten ein Würfel zeichnen:





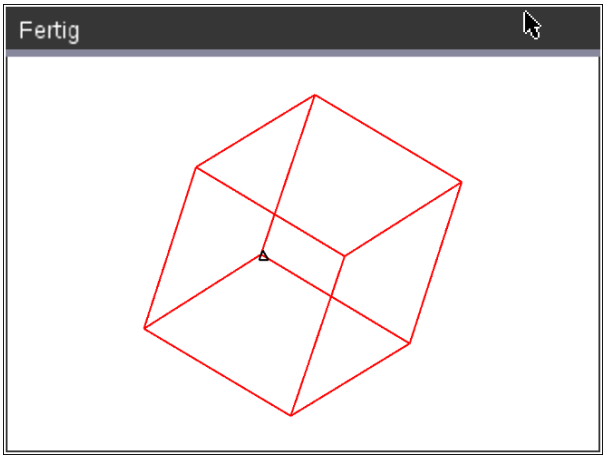
Wir positionieren nun den Würfel zentral im xyz-System und implementieren zwei der leistungsstarken Rotationsmethoden. Wie in einem CAD-Programm kann der Würfel mit den Cursor-Tasten im Raum bewegt werden:

```
from raumigel1 import *
from ti_system import get_key

def draw(obj, dist):
  for i in range(4):
    for j in range(4):
      obj.vw(dist)
      obj.li(90)
    obj.vw(dist)
    obj.kvo(90)
  obj.darstellen()

set_window(-159, 160, -105, 106)
dist = 100
angle = 5
igel = raumigel()
igel.set_rgb(255, 0, 0)
igel.set_xyz(-dist/2,-dist/2,dist/2)
key = ""
use_buffer()
while key != "esc":
  draw(igel, dist)
  paint_buffer()
  key = get_key(1)
  if key == "left":
    igel.y_rotation(-angle)
    igel.clear()
  if key == "right":
    igel.y_rotation(angle)
    igel.clear()
  if key == "up":
    igel.x_rotation(-angle)
    igel.clear()
  if key == "down":
    igel.x_rotation(angle)
    igel.clear()
```

Prozedurdefinition zum Zeichnen des Würfels.

Zentrierung der Würfeldarstellung im xyz-System.

Mit den Cursor-Tasten kann der Würfel um die x- bzw. y-Achse gedreht werden.



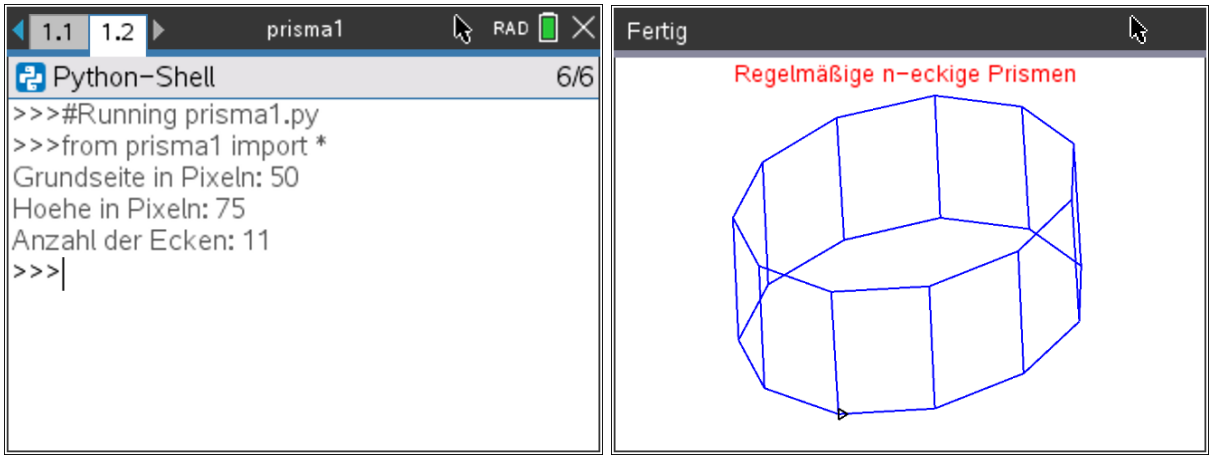
Projekte mit dem Raumigel

Nachfolgend sollen Anregungen für Schülerprojekte mit unterschiedlichem Anforderungsniveau gegeben werden.

1. Darstellung regelmäßiger n-eckiger Prismen

Der Übergang vom Würfel zum regelmäßigen n-eckigen Prisma ist schnell vollzogen. Lediglich der Zwischenschritt **kvo(winkel)** muss an die Eckenzahl angepasst werden.

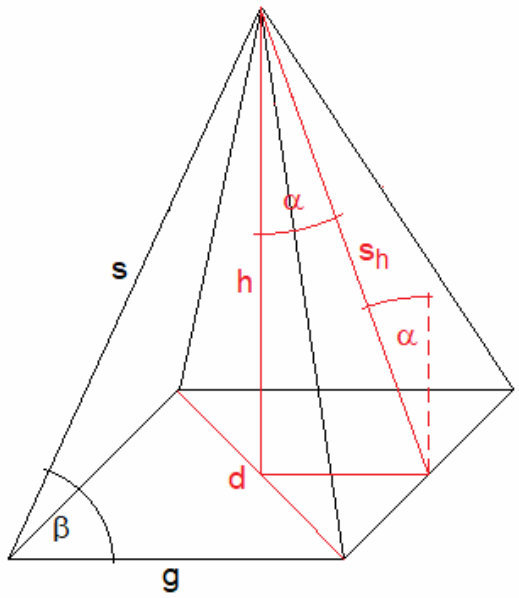
```
from raumigel1 import *
from ti_system import get_key

def draw(obj, l, h, n):
  angle = 360 / n
  for i in range(n):
    for j in range(2):
      obj.vw(l)
      obj.li(90)
      obj.vw(h)
      obj.li(90)
    obj.vw(l)
    obj.kvo(angle)
  obj.darstellen()

length = int(input("Grundseite in Pixeln: "))
height = int(input("Hoehe in Pixeln: "))
n = int(input("Anzahl der Ecken: "))
rot_angle = 5
out_angle = 360 / n
in_angle = (180 - out_angle) / 2
radius = length / 2 / sin(radians(out_angle / 2))

set_window(-159, 160, -105, 106)
igel = raumigel()
igel.set_rgb(0, 0, 255)
igel.set_text(-95, 90, "Regelmäßige n-eckige Prismen")
igel.set_xyz(0, -height / 2, 0)
igel.pen_up()
igel.kvo(in_angle)
igel.rw(radius)
igel.khi(in_angle)
igel.pen_down()
key = ""
use_buffer()
while key != "esc":
  draw(igel, length, height, n)
  paint_buffer()
  key = get_key(1)
  if key == "left":
    igel.y_rotation(-rot_angle)
    igel.clear()
  if key == "right":
    igel.y_rotation(rot_angle)
    igel.clear()
  if key == "up":
    igel.x_rotation(-rot_angle)
    igel.clear()
  if key == "down":
    igel.x_rotation(rot_angle)
    igel.clear()
```



2. Maßstäbliche Darstellung quadratischer Pyramiden

Eine quadratische Pyramide sei durch eine beliebige Länge ihrer Grundseite sowie durch eine beliebige Höhe gegeben. Sie soll derart maßstäblich im Bildschirm dargestellt werden, dass sie auch bei der Rotation um die Raumachsen nicht durch die Bildschirmgrenzen beschnitten wird. Dabei sollen sowohl maßstäbliche Vergrößerungen als auch Verkleinerungen möglich sein.

Vorüberlegungen:
Zunächst überlegen wir uns, welche Winkel sich für die Bewegung des Igels eignen: Zweckmäßigerweise wollen wir neben der Seitenlänge s und der Seitenhöhe s_h die Winkel α und β nutzen.



Es gelten:

$$d = \sqrt{2} \cdot g$$

$$s_h = \sqrt{h^2 + \frac{g^2}{4}}$$

$$s = \sqrt{s_h^{\,2} + \frac{g^2}{4}} \quad \text{bzw.} \quad s = \sqrt{h^2 + \frac{d^2}{4}}$$

$$\alpha = \arctan\left(\frac{g}{2 \cdot h}\right)$$

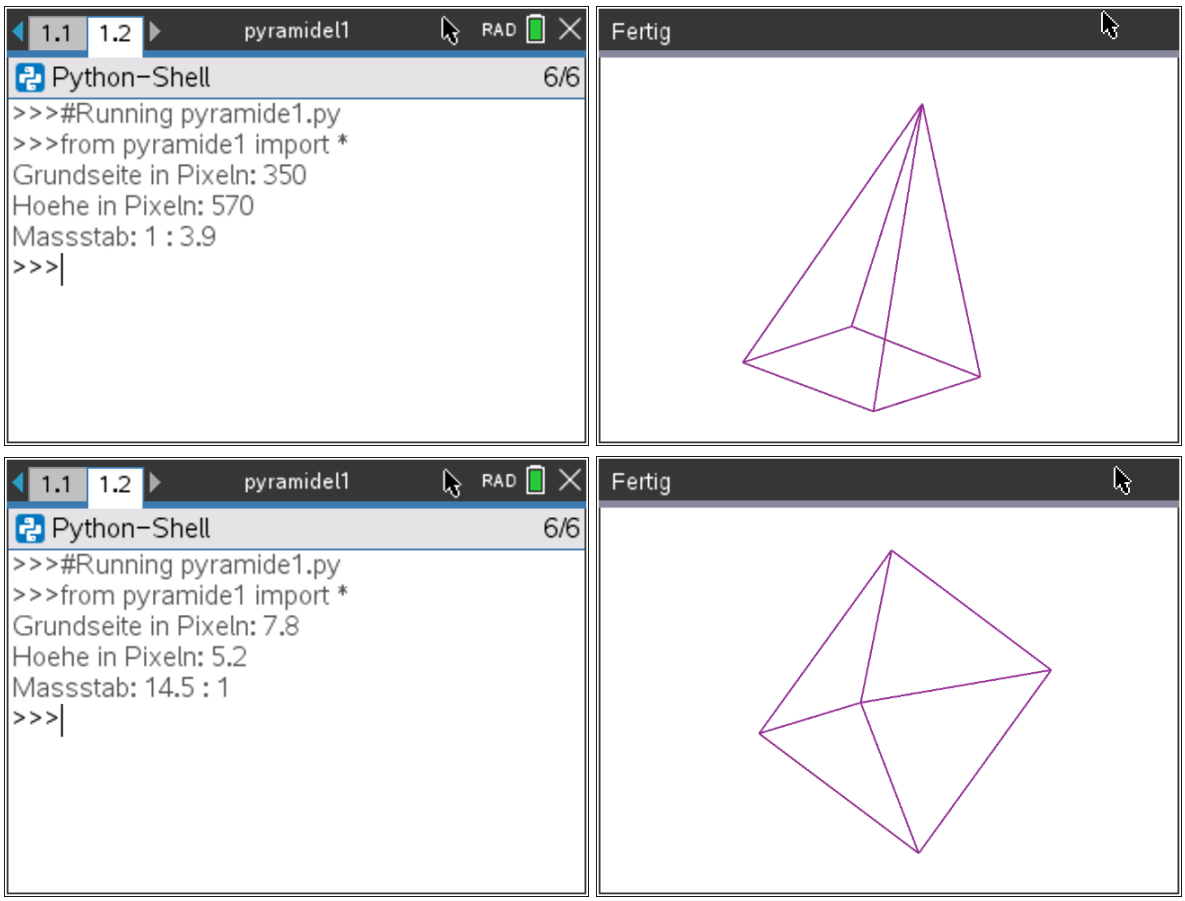
$$\beta = \arctan\left(\frac{2 \cdot s_h}{g}\right)$$

Zur maßstäblichen Darstellung orientieren wir uns an der kleinsten Bildschirmabmessung. Sie beträgt min = 200. Die größten Abmessungen der Pyramide sind max = d oder max = s. Nach der Entscheidung, welche der beiden Längen die größere ist, kann nun der Maßstab mit $k = 0{,}8 \cdot \frac{\min}{\max}$ berechnet werden. Mit dem empirische Faktor 0,8 vermeiden wir, dass der Bildschirm bis zu seinen Grenzen ausgefüllt wird.

Programm:

```
from raumigel1 import *
from ti_system import get_key
from math import sqrt, degrees, atan

def draw(obj, g, h):
  sh = sqrt(h ** 2 + g ** 2 / 4)
  s = sqrt(sh ** 2 + g ** 2 / 4)
  al = degrees(atan(g / h / 2))
  be = degrees(atan(2 * sh / g))
  for i in range(4):
    obj.vw(g)
    obj.kvo(90)
  for i in range(4):
    obj.nli(al)
    obj.li(be)
    obj.vw(s)
    obj.re(2 * be)
    obj.vw(s)
    obj.li(be)
    obj.nre(al)
    obj.kvo(90)
  obj.darstellen()

min = 200
length = float(input("Grundseite in Pixeln: "))
height = float(input("Hoehe in Pixeln: "))
d = sqrt(2) * length
s = sqrt(height ** 2 + d ** 2 / 4)
max = d
if max < s: max = s
k = 0.8 * min / max
if k < 1:
  k_rez = 1 / k
  print("Massstab: 1 : %1.1f" % k_rez)
else:
  print("Massstab: %1.1f : 1" % k)

set_window(-159, 160, -105, 106)
g = k* length
h = k * height
y = h - g / 4 * sin(pi / 4)
angle = 5
igel = raumigel()
igel.ausblenden()
igel.set_rgb(150, 50, 150)
igel.set_xyz(-g / 2, -y / 2, g / 2)
key = ""
use_buffer()
while key != "esc":
  draw(igel, g, h)
  paint_buffer()
  key = get_key(1)
  if key == "left":
    igel.y_rotation(-angle)
    igel.clear()
  if key == "right":
    igel.y_rotation(angle)
    igel.clear()
```

```
if key == "up":
  igel.x_rotation(-angle)
  igel.clear()
if key == "down":
  igel.x_rotation(angle)
  igel.clear()
```



3. Steuerung des Raumigels mit einem Beschleunigungssensor

Bisher haben wir einfache 3D-Objekte auf dem Bildschirm dadurch bewegt, indem wir sie per Tastatur um die x- bzw. y-Raumachse rotieren ließen.
Aus diversen Science-Fiction-Filmen kennen wir Szenarien, bei denen Bildschirmausgaben mit Gesten gesteuert werden. Mit dieser Idee wollen wir Igelgrafiken nun mit einem Beschleunigungssensor (Accelerometer) als Human-Machine Interface (HMI) bewegen.

Vorüberlegungen:

Beschleunigungssensoren sind mikro-elektro-mechanische Systeme (MEMS), die in einem kleinen Gehäuse integriert sind. Sie sind in modernen elektronischen Systemen weit verbreitet und reichen in ihren Anwendungen von Trägheitsschock-/Crash-Sensoren in Auto-Airbagsystemen und stabilisierenden Quadcopter-Drohnen bis hin zu Schrittzählern und physischen Bewegungsüberwachungen in Fitnessuhren oder Smartphones. Für den experimentellen Einsatz sind Beschleunigungssensoren auf kleinen Leiterplatte (PCB) erhältlich und können direkt an Mikrocontroller-Boards wie den TI-Innovator Hub angeschlossen werden.
Während die oben genannten Anwendungen alle auf der Analyse der dynamischen Signalwellenform des Beschleunigungssensors basieren, gibt es auch eine permanente statische Signalkomponente, die zur räumlichen Orientierung relativ zur Gravitationskraft F_G verwendet werden kann. Diese wollen wir zur Bewegung unserer Igelgrafik nutzen.

Zur Steuerung des Raumigels mit dem Beschleunigungssensor ADXL335 steht mit der Bibliothek **ADXL335driver** die sehr leistungsstarke Klasse **adxl()** bereit. Die entsprechende Methode setzt die statischen Beschleunigungsmesswerte in die Zeichenketten "left", "right", "up" und "down" um. Damit können 3D-Igegrafiken alternativ mit dem Beschleunigungssensor oder mit der Tastatur auf dem Bildschirm bewegt werden.
Weiterhin lässt sich der Sensor vor einem ersten Programmstart problemlos kalibrieren. Weitere Einzelheiten sowie die Code-Bibliothek sind in einer separaten Veröffentlichung unter T^3 Europe [3] zu finden.

Programm:

`from raumigel1 import *` `from ti_system import get_key`	

`def draw(obj, dist):`
`  for i in range(4):`
`  for j in range(4):`
`  obj.vw(dist)`
`  obj.li(90)`
`  obj.vw(dist)`
`  obj.kvo(90)`
`  obj.darstellen()`
	Prozedurdefinition zum Zeichnen des Würfels.

`try:`
`  from ADXL335driver import *`
`  myadxl = adxl()`
`  device = True`
`except:`
`  device = False`
	Versuch der Verbindungsaufnahme zum TI-Innovator Hub sowie des Ladens der Bibliothek des Beschleunigungssensors.

`set_window(-159, 160, -105, 106)`
`dist = 100`
`angle = 5`
`igel = raumigel()`
`igel.clearscreen()`
`igel.set_rgb(255, 0, 0)`
`igel.set_xyz(-dist/2,-dist/2,dist/2)`
`key = ""`
`use_buffer()`
`while key != "esc":`
`  draw(igel, dist)`
`  paint_buffer()`
`  if device:`
`  key = myadxl.get_dirxy(0, 0.2)`
`  if key == "left":`
`  igel.clear()`
`  igel.y_rotation(-angle)`
`  if key == "right":`
`  igel.clear()`
`  igel.y_rotation(angle)`
`  if key == "up":`
`  igel.clear()`
`  igel.x_rotation(-angle)`
`  if key == "down":`
`  igel.clear()`
`  igel.x_rotation(angle)`
`  key = get_key(0)`
	Falls der Verbindungsaufbau erfolgreich war, werden die Rückgabewerte "left", "right", "up" oder "down" des ADXL-Objektes genutzt.
	
	Tastaturabfrage zur alternativen Igelsteuerung bzw. zum Programmabbruch.

`if device: print(myadxl.ver())`	Rückgabe der Version der ADXL-Bibliothek.

Zusammenfassung

In einer erstaunlichen Performance lässt sich der Raumigel auf dem TI-nspire CX II implementieren und umfassend anwenden. Es ist klar, dass der Wunsch nach weiteren leistungsstarken Eigenschaften und Methoden bestehen wird. So wäre die Unterscheidung zwischen sichtbaren und verdeckten Körperkanten ebenso wünschenswert wie variable Hintergrundfarben oder die Bildschirmdarstellung in einer Zentralprojektion. Hier spekulieren die Autoren auf die Initiative interessierter Leserinnen und Leser, die Objektklasse des Raumigels ggf. mit weiteren Anregungen, die unter [1] nachgelesen werden können, zu erweitern.

Bildnachweis:

Igeldarstellung S. 2:
https://umbreit.e-bookshelf.de/products/reading-epub/product-id/10980970

Quellen:

[1] Löthe, Wölpert, Wolpert; Raumigel - Einführung, Anwendungen, Implementation; Informatik und Datenverarbeitung in der Schule, Materialien und Berichte Nr. 7, Pädagogische Hochschule Ludwigsburg, 1985
[2] https://www.gymnasium-loebau.de/fachbereiche/naturwissenschaften/informatik
[3] Hilbig, 'Adding Accelerometer Sensor Library to TI-Innovator using Python', T^3 Europe Materials Database, Nov 2020
[4] Berger, Hilbig; Hedgehog – a 3D graphics library in Python, T^3 Europe Materials Database, Nov 2020

T³ DEUTSCHLAND

Teachers Teaching with Technology™