Hedgehog – a 3D graphics library in Python

Veit Berger, Hans-Martin Hilbig

T³ DEUTSCHLAND

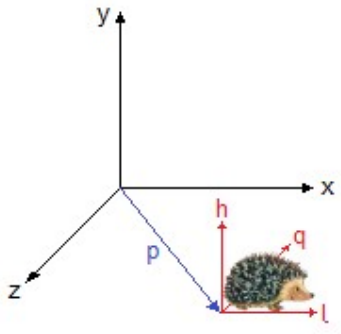
Teachers Teaching with Technology™

Overview

Two-dimensional libraries like turtlegraphics are popular tools to make students familiar with basic coding structures in Computer Science lessons. Aimed to train students in spatial notion skills, teachers of the Pädagogischen Hochschule Ludwigsburg created a 3D-graphics coding library in Logo as early as in 1987[1]. This library has become popular under the name 'Raumigel' which translates to 'spatial hedgehog', to illustrate the difference to the 2-dimensional 'turtle' graphics library. Hedgehog1 is based on Raumigel1, with the only difference in adapting names of the raumigel1 methods to English language [4].
Since the object-oriented version of hedgehog had been successfully ported to Pascal and Delphi [2], a Python implementation recently has been created to be used in the MicroPython environment of TI's most recent release of TI-Nspire-CXII Graphing Calculators. Besides a brief explanation of the library's features and methods, this article is focused on practical applications useful in today's Computer Science lessons as well as other MINT/STEM classrooms.

Attributes and Methods of hedgehog

Like known from turtle graphics in the 2-dimensional x-y plane, hedgehog can freely move in its 3-dimensional space, leaving a trace with its on-board pen. This trace will be shown as a parallel projection in cavalier perspective on the 2-dimensional display canvas of the Nspire-CXII. Hedgehog's movement in the static xyz-system is defined in vector graphics, based on the three unity vectors l (longitudinal), q (transverse) and h (vertical) (see picture). Besides rotations of the hedgehog around the three spatial axes, all movements happen along the longitudinal axis only, with hedgehog facing forward or backward.



The basic movement methods of hedgehog are listed here. All movements scale in pixels, all angles in degrees:

Method	Description
`fwd(pixels) / bwd(pixels)`	Move forward / backward
`rt(degrees) / lt(degrees)`	Rotate to the right, to the left (rotate around h-axis)
`ifwd(degrees) / ibwd(degrees)`	Incline forward/backward (rotate around q-axis)
`irt(degrees) / ilt(degrees)`	Incline right/left (rotate around l-axis)

Miscellaneous methods:

Method	Description
`clear()` `clearscreen()`	Clear Graphics Display Clear Graphics, including hedgehog position
`set_rgb(r,g,b)`	Set color r, g, b = 0 ... 255
`set_text(x,y,text)`	Show text at x,y pixel coordinates
`pen_down() / pen_up()`	Draw while moving / pause drawing
`set_cur()`	Set hedgehog cursor as small triangle
`show_cur() / hide_cur()`	Show hedgehog cursor / hide hedgehog cursor

A useful feature of hedgehog is the ability to rotate objects in 3D space.
Methods to rotate graphical object in 3D space:

Method	Description
`x_rotation(degrees)`	Rotate object around the x-axis in 3D space
`y_rotation(degrees)`	Rotate object around the y-axis in 3D space
`z_rotation(degrees)`	Rotate object around the z-axis in 3D space

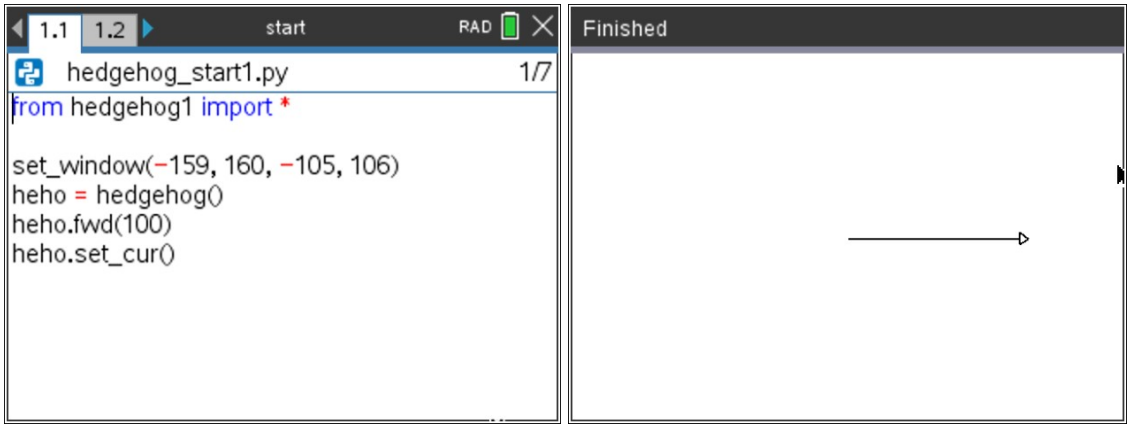
First steps using hedgehog

Hedgehog is coded as an independent library. There is no need to include the Hedgehog code in each application document. For the Nspire software to recognize Hedgehog as a library, copy Hedgehog1.tns in your local PyLib folder of both, the Nspire Handheld and the Nspire Desktop software:

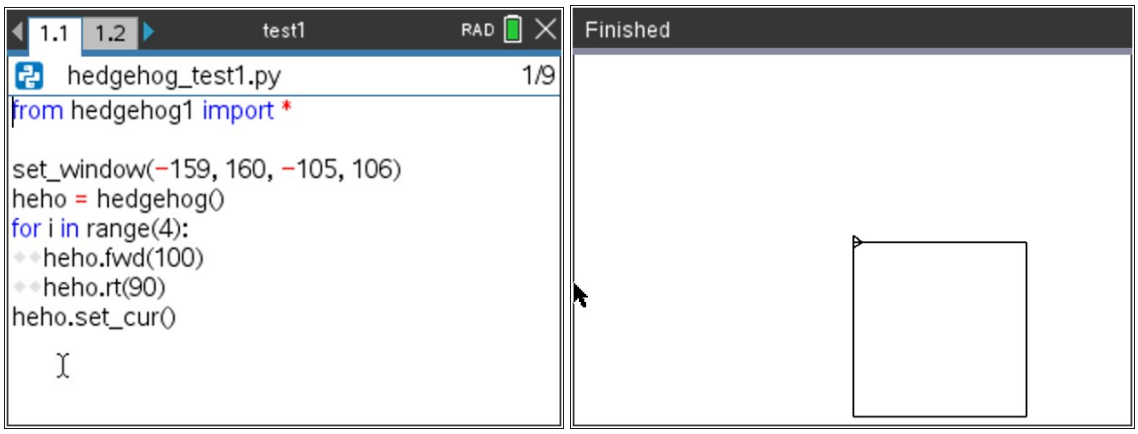
Handheld: My Documents\PyLib
Windows: C:\Users\YourUserName\Documents\TI-Nspire CX\PyLib

As a second step, press <refresh libraries> under the <Tools> menu of the Nspire Desktop or press the <doc> button on your handheld, followed by selecting <6 Refresh Libraries> from the menu, to include Hedgehog1 in the catalog of Nspire Python libraries.

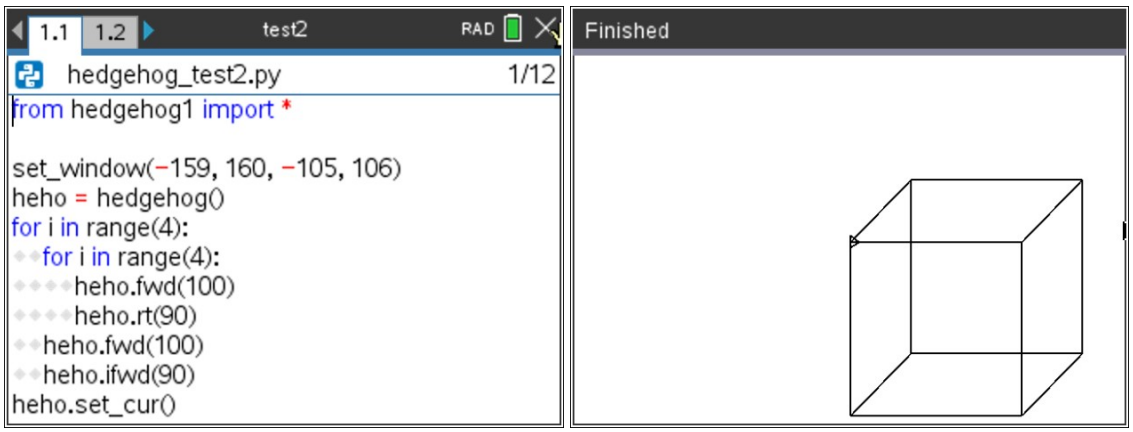
The following screenshot shows the instantiation of the Hedgehog Object, followed by a simple horizontal movement a hundred pixels long. It is a good practice to show the Hedgehog cursor at the end of the movement, to aid orientation for the human eye.



To draw a square in the xy-plane, we take the single line example from above screenshot, add a 90 degrees turn at the end of the line and repeat this four times in a loop, like shown here:



Adding another loop of four inclinations in the z-plane after a square is drawn, another three perspective squares are added, resulting in a 3-dimensional cube. The following compact code shows how easy it is to draw 3-dimensional objects using Hedgehog:



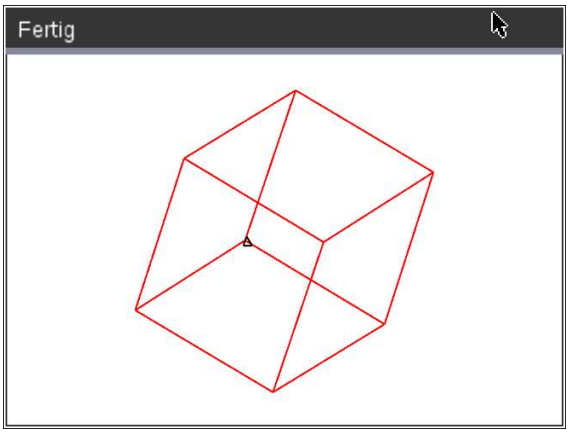
Let us center the cube on the screen and implement two powerful rotation methods. Just like in a 3D CAD program, the cube can be rotated around the x- and y-axis by pressing the <up><down><left><right> cursor buttons of the Handheld or the PC:

```
from hedgehog1 import *
from ti_system import get_key

def draw(obj, dist):
  for i in range(4):
    for j in range(4):
      obj.fwd(dist)
      obj.lt(90)
    obj.fwd(dist)
    obj.ifwd(90)
  obj.set_cur()

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

Procedure to draw a cube

Center cube in the x-/y-/z-system

Use cursor keys to rotate the cube around its axes.



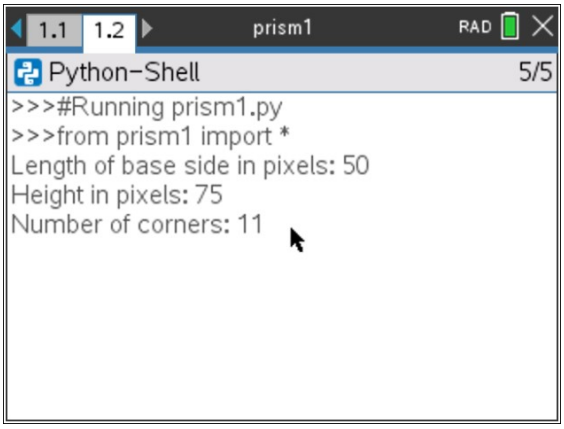
Advanced projects using Hedgehog

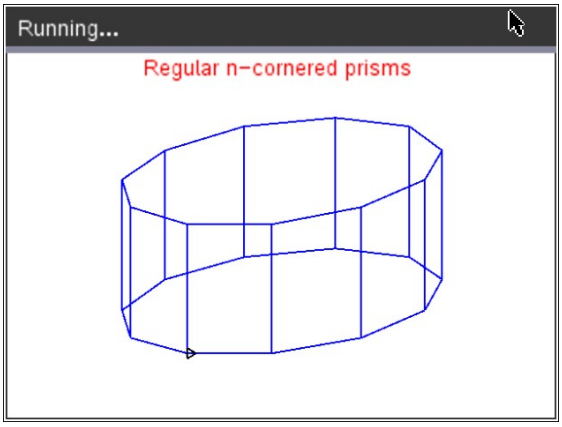
Here are some examples to inspire projects for students at different grades and skillsets.

1. Visualization of regular n-cornered prisms

Migrating from a cube object to a regular n-cornered prism is easy. Just the intermediate part of forward inclination **ifwd(angle)** needs to be adjusted to the number of corners of the prism, as shown in this code example:

```
from hedgehog1 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.fwd(l)
      obj.lt(90)
      obj.fwd(h)
      obj.lt(90)
    obj.fwd(l)
    obj.ifwd(angle)
  obj.set_cur()

length = int(input("Length of base side in pixels: "))
height = int(input("Height in pixels: "))
n = int(input("Number of corners: "))
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)
heho = hedgehog()
heho.set_rgb(0, 0, 255)
heho.set_txt(-80, 90, "Regular n-cornered prisms")
heho.set_xyz(0, -height / 2, 0)
heho.pen_up()
heho.ifwd(in_angle)
heho.bwd(radius)
heho.ibwd(in_angle)
heho.pen_down()
key = ""
use_buffer()
while key != "esc":
  draw(heho, length, height, n)
  paint_buffer()
  key = get_key(1)
  if key == "left":
    heho.y_rotation(-rot_angle)
    heho.clear()
  if key == "right":
    heho.y_rotation(rot_angle)
    heho.clear()
  if key == "up":
    heho.x_rotation(-rot_angle)
    heho.clear()
  if key == "down":
    heho.x_rotation(rot_angle)
    heho.clear()
```

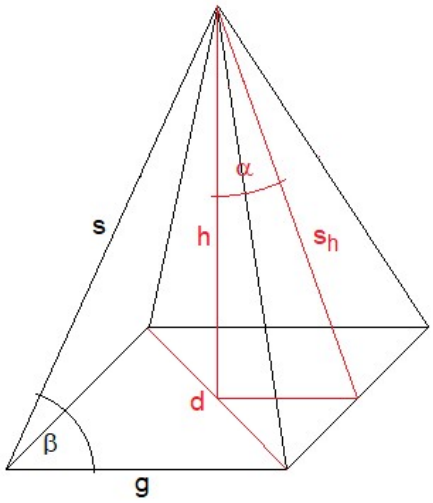




2. Scaled visualization of square pyramids

A square pyramid is given by any length of its base and any height. It should be displayed true to scale on the screen in such a way that it is not clipped by the screen boundaries even when rotating around the spatial axes. Both scaled enlargements and reductions should be possible.

Preliminary considerations:
First, let us consider which angles are suitable for the movement of the hedgehog. In addition to calculating the length of the pyramid's side *s* and the height of its side s_h, we can calculate angles α and β as follows:



Where:

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

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

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

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

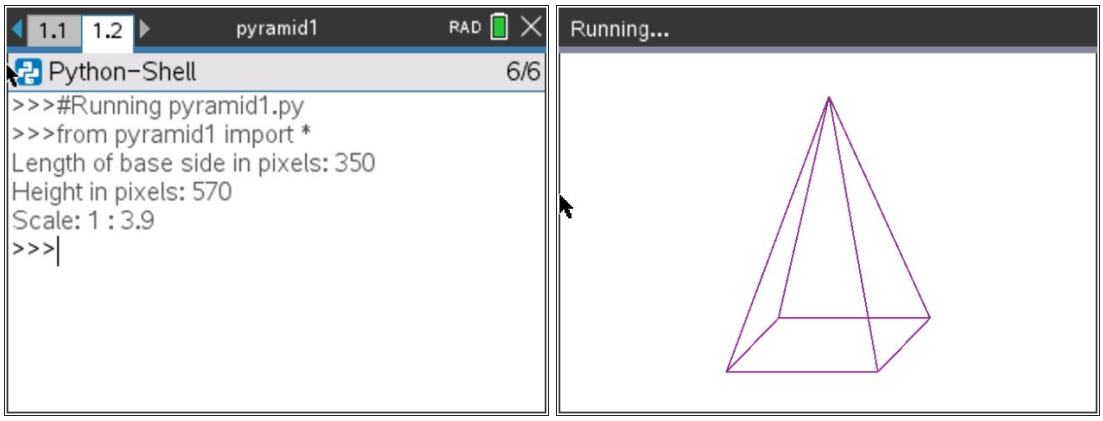
$$\beta = \arctan\left(\frac{2 \cdot s_h}{g}\right)$$

For true-to-scale representation, we use the shorter side of the 318 x 212 pixel display as a reference. Let's assume min = 200 (pixels) for the sake of simplicity.
The largest dimensions of the pyramid are either the diagonal max = *d* of the pyramid's base or the pyramid's side max = *s*.
After deciding which of the two lengths is the larger, the scale can be calculated as $k = 0{,}8 \cdot \frac{\min}{\max}$. Factor 0.8 is used as a safety margin to avoid using the display up to its boundaries.

Scaled visualization of square pyramids code example:

```
from hedgehog1 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.fwd(g)
    obj.ifwd(90)
  for i in range(4):
    obj.ilt(al)
    obj.lt(be)
    obj.fwd(s)
    obj.rt(2 * be)
    obj.fwd(s)
    obj.lt(be)
    obj.irt(al)
    obj.ifwd(90)
  obj.set_cur()

min = 200
length = float(input("Length of base side in pixels: "))
height = float(input("Height in pixels: "))
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("Scale: 1 : %1.1f" % k_rez)
else:
  print("Scale: %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
heho= hedgehog()
heho.hide_cur()
heho.set_rgb(150, 50, 150)
heho.set_xyz(-g / 2, -y / 2, g / 2)
key = ""
use_buffer()
while key != "esc":
  draw(heho, g, h)
  paint_buffer()
  key = get_key(1)
  if key == "left":
    heho.y_rotation(-angle)
    heho.clear()
  if key == "right":
    heho.y_rotation(angle)
    heho.clear()
  if key == "up":
    heho.x_rotation(-angle)
```

```
    heho.clear()
  if key == "down":
    heho.x_rotation(angle)
    heho.clear()
```



3. Spatial control of objects

There are several ways to animate the graphical object on the display by using the x/y/z_rotation() methods of the hedgehog class. Either you code a constant rotation around one or more of the three orthogonal axes by periodically incrementing the three spatial angles at a given frequency. Or you let the user control the rotation by scanning the computer's keyboard for any of the cursor keys (left, right, up, down) pressed. From science fiction movies we remember scenes where computer displays have been controlled by gestures of the users. This can be accomplished by using an accelerometer sensor connected to the TI-Innovator Hub as the HMI (Human-Machine-Interface).

Preliminary considerations:

Accelerometer sensors are small micro-electro-mechanical systems (MEMS) integrated in a small package. Accelerometers are widely used in today's electronic systems, ranging from inertial shock / crash sensing in Automotive Airbag systems, stabilizing quadcopter drones up to counting steps or physical movement monitoring by fitness watches or smartphones. For prototyping and experimental use, accelerometers are available mounted on a printed circuit board (PCB) and can directly be connected to Microcontroller boards like the TI-Innovator Hub.
While the applications mentioned above are all based on analyzing the accelerometer's dynamic signal waveform, there is also a permanent static signal component which can be used to determine the spatial orientation of the accelerometer relative to the gravitational force G. For additional details and a copy of the accelerometer code library, please refer to a separate publication on T3 Europe [3].

Code example:

`from hedgehog1 import *` `from ti_system import get_key`	

`def draw(obj, dist):`
`  for i in range(4):`
`  for j in range(4):`
`  obj.fwd(dist)`
`  obj.lt(90)`
`  obj.fwd(dist)`
`  obj.ifwd(90)`
`  obj.set_cur()`
	Procedure to draw a cube.

`try:`
`  from ADXL335_09c import *`
`  myadxl = adxlc()`
`  device = True`
`except:`
`  device = False`
	Looking for presence of the accelerometer driver library and checking if TI-Hub is connected. If successful, set HMI flag
	If failed, switch to keyboard as the human machine interface (HMI)

`set_window(-159, 160, -105, 106)`
`dist = 100`
`angle = 5`
`heho = hedgehog()`
`heho.set_rgb(255, 0, 0)`
`heho.set_xyz(-dist/2,-dist/2,dist/2)`
`key = ""`
`use_buffer()`
`while key != "esc":`
`  draw(heho, dist)`
`  paint_buffer()`	

`  if device:`
`  key = myadxl.get_dirxy(0,0.2)`
`  if key == "left":`
`  heho.y_rotation(-angle)`
`  heho.clear()`
`  if key == "right":`
`  heho.y_rotation(angle)`
`  heho.clear()`
`  if key == "up":`
`  heho.x_rotation(-angle)`
`  heho.clear()`
`  if key == "down":`
`  heho.x_rotation(angle)`
`  heho.clear()`
	If HMI flag is set, redirect accelerometer data to control rotation of the cube. Otherwise, use keyboard data.

`  key = get_key(0)` `if device: print(ver())`	Check for <esc> key pressed to terminate program For convenience, print the current version of the accelerometer driver

Summary:

The Hedgehog library can be implemented as is and used extensively on the TI-nspire CX II with an amazing performance. It is likely there will be a growing desire for extending the library while users are practicing their skills. For example, visual distinction between visible and hidden edges of the object would be just as desirable as variable background colors or supporting view of the object in a central projection. Here the authors speculate on the initiative of interested users to expand the hedgehog object class. Suggestions and ideas can be found under [1], if necessary.

Sources:

[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; Der Raumigel unter Python, T^3 Deutschland Materialdatenbank, Nov 2020