Graphing in VPython

This document describes how to use VPython to generate 2-dimensional graphs and plots. If your program is to generate a graph or plot, the graphing module needs to be imported:

```
from visual.graph import *
```

The graphing module allows you to plot connected curves (gcurve), disconnected dots (gdots), and to draw vertical bars (gvbars). You can show multiple curves and plots in one graphing window; if you want your program to generate more than one plotting window, you can do so by giving each window a name. When you import `visual.graph`, you do not need to import `visual`.

The following examples illustrate the most common graphing features used in Matter&Interaction related programs. For a more complete description of the graph module see [http://www.vpython.org/webdoc/visual/graph.html](http://www.vpython.org/webdoc/visual/graph.html).

**Example 1**
The first example draws two curves in a window (the default window which does need to be given a name). Load the program `VPplot1.py`.

```
# Plotting in VPython: VPplot1.py
# Two curves in one window
# Written by: Your Instructor

from visual.graph import *

funct1 = gcurve(color=color.cyan)
funct2 = gcurve(color=color.red)

# defines functions funct1 and funct12 will plotted as gcurves
for x in range(80):
    # with range(80), x takes on the values 0, 1, 2, ... 79
    funct1.plot(pos=(x, 6*cos(2.*x)))
    # adds one point to the gcurve of funct1
```

---

1 This work is supported by the National Science Foundation under Grant No. CCF-0722210. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
funct2.plot(pos=(x, 0.3*x-10))
    # adds one point to the gcurve of funct2

Run program VPplot1.py. Next:
- add rate(20) to the body of the for-loop and run the program. Slowing down the plotting allows you to see in what order points are added to the curves
- replace range(80) by arange(0., 80.1, 0.1) and run the program (both expressions are still followed by a colon); arange is a version of range available when using VPython. It allows one to iterate over lists with a non-integer step size. To see what arange does, use the interpreter window, type from visual.graph import * and then type arange(0., 80.1, 0.1) to see the list it generates.

Example 2
Program VPplot2.py names and creates two drawing windows. Load the program and read through the comments in it.

# Plotting in VPython: VPplot2.py
# Plotting in two windows
# Written by: Your Instructor

from visual.graph import *

# defines and names window myWindow1
# xtitle and ytitle specify what text to place on x- and y-coordinates
myWindow1 = gdisplay(xtitle="Time", ytitle="Position")

# defines f1 as a curve to be displayed in myWindow1
f1 = gcurve(gdisplay=myWindow1, color = color.cyan)

# defines and names window myWindow2
myWindow2 = gdisplay(xtitle="Time", ytitle="Cost")

# anything defined right after a window is defined will be placed in
# this window; hence gdisplay=myWindow2 is not actually necessary
# but make sure to not move the code! It is always safer to use the name of the window when
# defining a graphing object
# f2 and f3 will be displayed in myWindow2
f2 = gcurve(color = color.red)
f3 = gcurve(color = color.green)

# generate the points for the three functions using arange
for x in arange(0., 25.1, 0.1):
    f1.plot(pos = (x, 10.*cos(2.*x)*exp(-0.2*x)))
f2.plot(pos= (x, 3*x**2 + 12*x - 44))
f3.plot(pos= (x, 1.3**x))

Run program VPplot2.py. You will need to move one window to see the second one.

Example 3
Program VPplot3.py generates two windows: one drawing disconnected dots (using gdots) and the other showing a function (using gcurve) and vertical bars (using gvbars).

# Graphing in VPython: VPplot3.py
# Plotting in two windows: gdots, gcurve, gvbars
# Written by: Your Instructor
from visual.graph import *

# define two plotting windows
myWindow1 = gdisplay(xtitle="Time", ytitle="y-value of dots")
myWindow2 = gdisplay(xtitle="Time", ytitle="y-value of bars")

# define my_dots as a orange dots displayed in myWindow1
my_dots = gdots(gdisplay=myWindow1, color = color.orange)

# define my_fun as a red curve displayed in myWindow2
my_fun = gcurve(gdisplay=myWindow2, color = color.red)

# define my_bars as green vertical bars displayed in myWindow2
my_bars = gvbars(gdisplay=myWindow2, color = color.green)

# generate dots for my_dots
for x in range(25):
    my_dots.plot(pos = (x,3*x**2))

# generate the points for my_fun
for x in arange(0., 25.1, 0.5):
    my_fun.plot(pos = (x,3*x**2))
    my_bars.plot(pos = (x,3*x**2))

Example 4
The fourth program combines many of the features you have already seen. In addition, it shows a
different way of generating points: two lists are generated, one containing the x-values and the other the
y-values. Once the lists have been created, they are used to plot (in VPplot4.py we draw vertical bars
after the curve has been drawn).

# Graphing in VPython: VPplot4.py
# Plotting in two windows: gdots, gcurve, gvbars
# Written by: Your Instructor
from visual.graph import *

t = 0
deltat = 1e-3
time_interval = 0.1
time_interval_int = int(time_interval/deltat)

# define two windows graphpos and graphv
# the x and y coordinates in gdisplay are set so the windows are not drawn on
# top of each other and in the right side of the screen
graphpos = gdisplay(x=600, y=0, xtitle="Time", ytitle="Position")
carposx = gcurve(gdisplay=graphpos, color=color.red)

graphv = gdisplay(x=600, y=500, xtitle="Time", ytitle="Velocity")
carv = gcurve(gdisplay=graphv, color = color.blue)
velbar = gvbars(gdisplay=graphv, delta=time_interval, color=color.green)
# define a road and a car; will be drawn in the default VPython window
road = box(pos=(5,-0.5,0), length=12, height=1, width=2, color=(.5,.5,.5))
car = box(pos=(0,.5,0), length=2, height=1, width=1, color=color.red)
car.v = vector(0,0,0)

# define an initially empty list tlist; will store time values
# tlist = []

tlist = []

# define an initially empty list vlist; will store velocity values
# vlist = []
vlist = []

while t <= 10:
    rate(1000)
    car.v = vector(sin(t) * sqrt(t), 0, 0)
    car.pos = car.pos + car.v*deltat

    # add new values to the two curves
    carposx.plot(pos=(t,car.pos.x))
carv.plot(pos=(t,car.v.x))

tlist.append(t)  # adds an entry to the time list
vlist.append(car.v.x)  # adds an entry to the velocity list

    # update t
    t = t + deltat

for i in range(0, len(tlist), time_interval_int):
    # draw the bars in window graph after both curves are drawn
    # the values for drawing the bars are stored in the two lists
    # generated by the while loop
    velbar.plot(pos=(tlist[i],vlist[i]))