Гетманова Елена Евгеньевна

VISUAL PYTHON – ЯЗЫК ДЛЯ МОДЕЛИРОВАНИЯ ФИЗИЧЕСКИХ ЯВЛЕНИЙ

От редакции. Язык Visual Python является одним из популярных языков моделирования. Его интерпретатор распространяется свободно, а формат входит в стандарт, рекомендуемый для создания образовательных продуктов.

Умение работать с компьютерной графикой и создавать физические модели необходимо каждому специалисту. Даже если он не будет впоследствии заниматься инженерными и физическими задачами, возможно, ему придется создавать экранные заставки, компьютерные игры или web-страницы. Поэтому заинтересованность студентов в обучении физике с помощью компьютерных программ высока. Общеизвестно, что, чем выше заинтересованность обучаемого в предлагаемом материале, чем интенсивней его эмоциональный настрой, тем лучше проходит процесс усвоения учебного материала.

Моделировать физические процессы путем программирования 3D-графики достаточно сложно. Появление языка Python позволило преодолеть эту сложность.

Python – интерпретируемый, объектно-ориентированный, высокоуровневый язык программирования, изобретенный Гвидо ван Россумом. Синтаксис языка Python достаточно понятный. Интерпретатор Python и большая стандартная библиотека распространяются свободно.

Python имеет подключаемый графический модуль, который называется «Visual». С помощью модуля VPython (*Visual Python*) создается 3D-графика.

Во многих университетах мира студенты используют VPython для моделирования физических процессов. Поскольку синтаксис языка осваивается быстро, а трехмерные модели строятся при подключении графического модуля, то основное внимание уделяется изучению физического явления.

ОПИСАНИЕ ЯЗЫКА VISUAL PYTHON

Visual Python – это название 3D-графического модуля, в котором используется язык программирования Python. Интерактивную среду, которая используется в Visual Python, называют «IDLE». В Visual Python объекты представлены в 3D-графике.

Начало координат (0, 0, 0) расположено в центре экрана. Ось *x* направлена вправо, ось *y* – вверх, ось *z* направлена на вас (рисунок 1).

Листинг 1 показывает, как создать на экране трехмерные объекты (рисунок 2). Для запуска программы нажать *F5* или использовать *Run menu.*

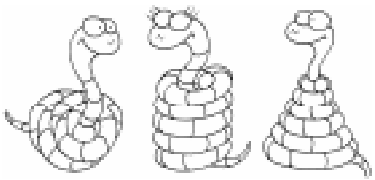
Листинг 1.

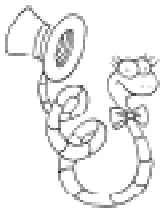
```
from visual import *
scene1=display(title='OBJECTS',width=300,height=300,center=(0,0,0),
        background=(1,1,1))
redbox=box(pos=vector(4,2,3), size=(8.,4.,6.),color=color.red)
greenball=sphere(pos=vector(4,7,3), radius=2, color=color.green)
bluering=ring(pos=vector(4,-7,3),axis=(1,1,0),radius=7, color=color.blue)
```

Перемещение мыши при нажатой левой кнопке вызывает вращение экрана.

В результате выполнения программы на экране появляется сцена (рисунок 2) с именем OBJECTS, размеры которой в пикселях составляют 300, центр сцены расположен в точке с координатами (0,0,0), фон сцены – белый. Сцена включает красный параллелепипед, зеленую сферу и синий тор. Более детально атрибуты, присвоенные объектам, будут рассмотрены ниже.

ТИПЫ СОЗДАВАЕМЫХ ОБЪЕКТОВ





ОБЪЕКТЫ, ИМЕНА И АТРИБУТЫ

Все создаваемые графические объекты (сферы, цилиндры, кривые и т. д.) отображаются на экране. Каждый объект должен иметь имя, если далее имеются ссылки на этот объект. Например, **`redbox`**, **`greenball`**, **`bluering`** в приведенном выше примере. Каждый объект должен иметь атрибуты: положение (например, **`pos=vector(4,2,3)`**), размеры (**`radius=2`**) и цвет (**`color=color.green`**). При изменении атрибутов происходит автоматическое изменение положения, цвета, размеров, соответственно.

Программа на листинге 2 создает кубы, которые имеют различный цвет и центры которых расположены в различных точках (рисунок 3).

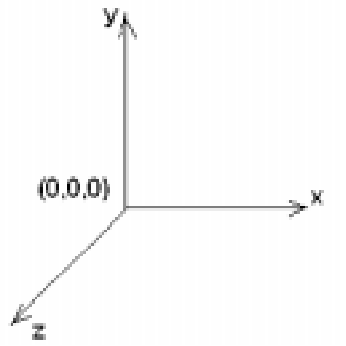


Рисунок 1.

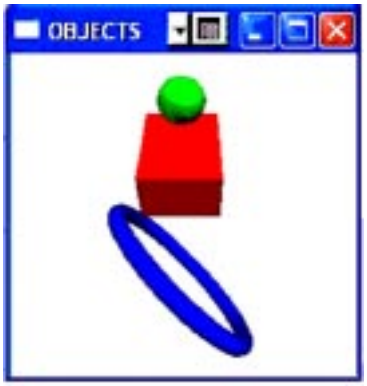


Рисунок 2.

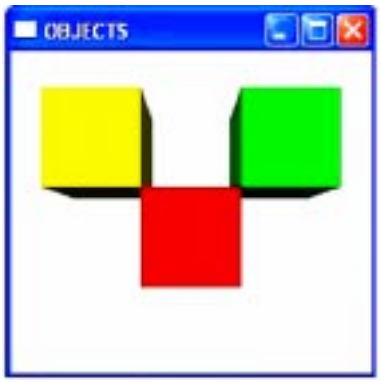
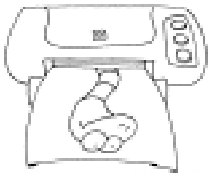


Рисунок 3.

Листинг 2.

```
from visual import *
scene1=display(title='OBJECTS',width=300,height=300,center=(0,0,0),
        background=(1,1,1))
redbox=box(pos=vector(0,0,0), size=(4.,4.,4.),color=color.red)
redbox1=box(pos=vector(4,4,0), size=(4.,4.,4.),color=color.green)
redbox2=box(pos=vector(-4,4,0), size=(4.,4.,4.),color=color.yellow)
```

В дополнение к имеющимся атрибутам можно добавить новые атрибуты, характеризующие физические свойства тела. Например, `body.mass = 20`. В качестве атрибутов могут использоваться такие трехмерные величины, как импульс тела или скорость тела. Например, можно указать `body.momentum =vector(10,5,7)`, то есть задать импульс тела, проекции которого на оси координат равны 10, 5 и 7 кг*м/с.



ПЕЧАТЬ РЕЗУЛЬТАТОВ

Чтобы напечатать число, вектор, список и т.д. используют команду **print**. Например,

```
print ball.momentum
```

Чтобы напечатать текст, его заключают в кавычки:

```
print "Box moves with speed ", v, "m/s."
```

ПРОГРАММИРОВАНИЕ НА VISUAL PYTHON

Первая строка должна подключать графический модуль

```
from visual import *
```

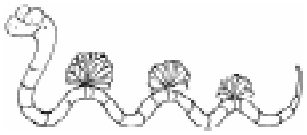
Комментарии в программе начинаются со значка «#». Например,

```
# components of velocity
```

ПЕРЕМЕННЫЕ

Возможны следующие типы переменных

```
a = 3       # целое число
b = -2.     # число с плавающей точкой
c = vector(0.4, 3e3, -1e1) # вектор
Earth = sphere(pos=(0,0,0), radius=6.4e6)
          # визуализируемый объект
bodies = [ship, Earth, star]
          # список объектов
```



ДЕЛЕНИЕ

Результат деления целых чисел округляется до ближайшего целого числа. Например,

```
a = 2/3
print a # a равно 0
```

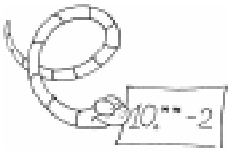
Чтобы избежать этого, следует поместить десятичную точку после каждого числа:

```
b = 2./3.
print b # b равно 0.6666667
```

Чтобы избежать ошибок при делении, в начало программы помещают строку

```
from __future__ import division
```

При задании положения объекта выражение `object.pos=(1,2,3)` эквивалентно `object.pos=(1.,2.,3.)`.

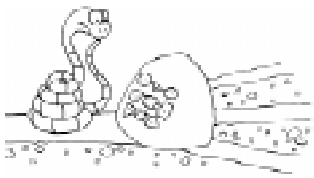


ЭКСПОНЕНТА

Экспоненциальные функции записываются следующим образом

```
x**2 # Выражение x^2 неправильно
10**-2 дает ошибку; следует писать 10.**-2
```

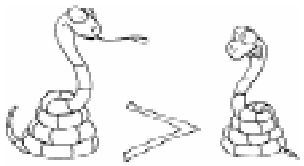
ЛОГИЧЕСКИЕ ОПЕРАЦИИ



Логические операторы **if**, **elif** (**"else if"**), **else**:

```
if a == b: # см. таблицу логических
           # выражений ниже
   c = 3.5 # значение присваивается,
           # если условие справедливо
elif a < b:
   c = 0.  # c будет равно нулю при
           # условии, если a < b
else:
    c = -23.2
```

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ



`==`	равно
`!=`	не равно (также <>)
`<`	меньше чем
`>`	больше чем
`<=`	меньше или равно
`>=`	больше или равно
`or`	логическое ИЛИ
`and`	логическое И
`in`	член последовательности
`not in`	не член последовательности

СПИСКИ

Возможно создание списка некоторой последовательности объектов. Он заключается в квадратные скобки. Например,

```
moons = [Io, Europa, Ganymede, Callisto]
```

Функция **arange** (сокращенное от «arrayrange») создает последовательность чисел

```
angles = arange (0., 2.*pi, pi/100.)
# последовательность от 0. до
# 2.*pi-(pi/100.) с шагом (pi/100.)
```

Функция **numbers** создает последовательность целых чисел

```
numbers = arange(10) # целые числа
```

Листинг программы

```
from visual import *
numbers = arange(10)
print numbers
```

при выполнении дает

[0 1 2 3 4 5 6 7 8 9]

ЦИКЛЫ

Простейшим циклом является цикл **while**. Цикл выполняется до тех пор, пока справедливо логическое выражение

Листинг программы

```
from visual import *
x=10
vx=3
dt=4
while x < 43:
    print x
    x = x + vx*dt
print x
```

Результат выполнения

10
22
34

Для написания бесконечного цикла используется логическое выражение, которое справедливо всегда

```
while 1==1:
    ball.pos = ball.pos + (ball.momentum/
                           ball.mass)*dt
```

Бесконечный цикл прерывается при выборе *Stop Program* из *Run menu*.

Можно создавать цикл для членов последовательности.

Можно начать цикл повторно или прервать цикл командами:

```
if a == b: continue
           # возвращение к началу цикла
if a > b: break # выход из цикла
```

СОЗДАНИЕ ОБЪЕКТОВ

СОЗДАНИЕ ЦИЛИНДРА

С помощью выражения

```
rod = cylinder(pos=(0,2,1), axis=(5,0,0),
               radius=1)
```

создается цилиндр (рисунок 4). Центр одного из оснований цилиндра расположен в точке $x = 0$, $y = 2$, $z = 1$. Его ось параллельна оси x, высота цилиндра равна 5. Таким образом, его второе основание находится в точке (5, 2, 1) как показано на рисунке 4.

Можно изменять положение цилиндра после его создания. Цилиндр сразу перемещается в новое положение.

```
rod.pos = (15,11,9)
          # изменилось положение (x,y,z)
rod.x = 15 # изменилось только pos.x
```

По умолчанию цвет цилиндра белый. После создания цвет цилиндра можно изменить:

```
rod.color = (0,0,1) # цилиндр синий
```

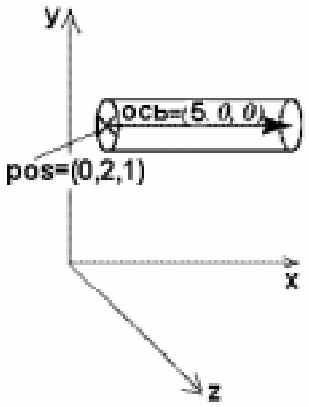


Рисунок 4.

Можно установить величину компонентов цвета. Например;

```
rod.red = 0.4
rod.green = 0.7
rod.blue = 0.8
```

При создании цилиндра можно использовать и другие атрибуты, которые записываются в произвольном порядке. Ниже приводится полный список атрибутов, большинство из которых применимо и к другим объектам:

pos – положение центра одного из концов цилиндра; три цифры, заключенные в скобки, например (3,4,5);

axis – очки по оси от основания до другого конца цилиндра;

x,y,z – покоординатное положение основания цилиндра **pos.x, pos.y, pos.z**;

radius – радиус цилиндра;

length – длина цилиндра по оси; в общем случае ось определяет длину цилиндра по этой оси; если длина указана, то длина переопределяется;

color – цвет объекта в системе RGB;

red, green, blue – компоненты цветов можно задавать отдельно;

up – устанавливает положение верха цилиндра.

СТРЕЛКА

Объект *стрелка* – это прямой параллелепипед с заострением в виде стрелки на одном из концов. Следующая команда создает стрелку, параллельную оси *x* (рисунок 5):

```
pointer = arrow(pos=(0,2,1),
            axis=(5,0,0), shaftwidth=1)
```

Объект имеет атрибуты **pos**, **x,y,z**, **axis**, **length**, **color**, **red**, **green**, **blue** и **up**, которые имеют те же значения, что и у цилиндра. Атрибут **up** имеет существенное значение для стрелки, поскольку основание и верхняя часть стрелки различны.

Дополнительные атрибуты стрелки **shaftwidth** (ширина стрелки), **headwidth** (ширина заострения), **headlength** (длина заострения).

По умолчанию ширина не расширяющейся части стрелки составляет одну десятую длины стрелки, ширина заострения равна удвоенной ширине стрелки, длина заострения равна утроенной ширине стрелки.

ОБЪЕКТ СФЕРА

Следующая команда создает сферу (рисунок 6):

```
ball = sphere(pos=(1,2,1), radius=0.5)
```

Центр масс сферы расположен в точке (1,2,1) , радиус сферы 0.5, цвет назначен по умолчанию.

Атрибуты **pos**, **x,y,z**, **axis**, **length**, **color**, **red**, **green**, **blue** и **up** имеют такое же значение, как и для цилиндра. Амплитуда оси указывает только ориентацию.

Атрибут **pos** соответствует центру сферы.

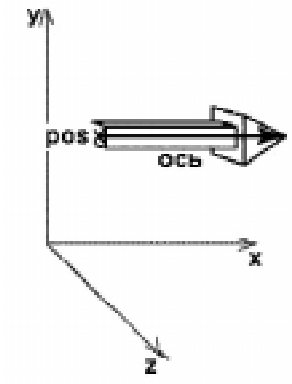


Рисунок 5.

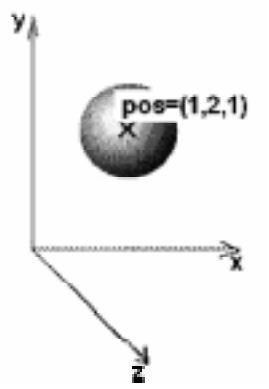


Рисунок 6.

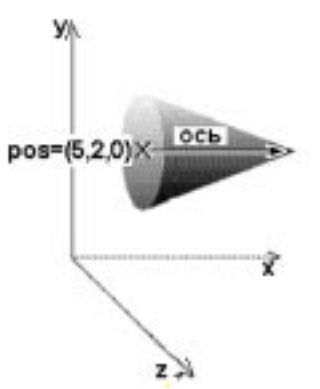


Рисунок 7.

ОБЪЕКТ КОНУС

Следующая команда создает конус, ось которого параллельна *x* (рисунок 7):

```
cone(pos=(5,2,0), axis=(12,0,0),
     radius=1)
```

Атрибуты **pos**, **x,y,z**, **axis**, **length**, **color**, **red**, **green**, **blue** и **up** имеют такое же значение, как и для цилиндра. Дополнительный атрибут **radius** определяет радиус основания конуса.

СОЗДАНИЕ КОЛЬЦА

Кольцо является круговым объектом с заданным внешним радиусом и толщиной. Центр кольца задается в атрибуте **pos**.

Следующая команда создает кольцо (рисунок 8):

```
ring(pos=(1,1,1), axis=(0,1,0),
              radius=0.5, thickness=0.1)
```

Атрибуты **pos**, **x,y,z**, **axis**, **length**, **color**, **red**, **green**, **blue** и **up** имеют такое же значение, как и для цилиндра.

Атрибуты осей влияют только на ориентацию кольца, величина атрибута значения не имеет.

Дополнительные атрибуты кольца:
radius – внешний радиус кольца,
thickness – толщина кольца (по умолчанию толщина кольца определяется как 1/10 радиуса).

Атрибут **pos** определяет центр кольца.

ОБЪЕКТ ПАРАЛЛЕЛЕПИПЕД

Следующая команда показывает, как создать параллелепипед (рисунок 9):

```
mybox = box(pos=(x0,y0,z0),
            length=L, height=H,
             width=W)
```

Атрибут **pos** определяет положение центра масс параллелепипеда. Можно назначать положение центра параллелепипеда отдельно: **mybox.x**, **mybox.y**, **mybox.z**. Следующий набор атрибутов определяет длину (*L*), высоту (*H*) и ширину (*W*) параллелепипеда вдоль осей *x*, *y*, *z*.

Если параллелепипед не ориентирован вдоль координатных осей, то атрибут **axis=(a,b,c)** показывает ориентацию сторон параллелепипеда относительно осей координат (рисунок 10):

```
mybox = box(pos=(x0,y0,z0), axis=(a,b,c),
            length=L,height=H, width=W)
```

По умолчанию направление **up** совпадает с осью *y*. Длина y параллелепипеда, ориентированного вдоль осей, располагается по оси *x*, высота – по оси *y*, ширина – по оси *z*.

Можно вращать параллелепипед вокруг его собственных осей, изменяя направление **up**:

```
mybox = box(pos=(x0,y0,z0), axis=(a,b,c),
length=L, height=H, width=W, up=(q,r,s))
```

Эта команда показывает, что ребро, определяющее ширину параллелепипеда, лежит в плоскости перпендикулярной вектору (*q*, *r*, *s*), а вектор, перпендикулярный ему и

Рисунок 8.

Рисунок 9.

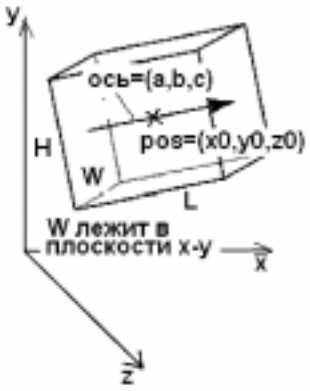


Рисунок 10.

вектору (a, b, c), определяет высоту параллелепипеда.

Атрибуты **pos**, **x,y,z**, **axis**, **length**, **color**, **red**, **green**, **blue** и **up** имеют такое же значение, как и для цилиндра.

Дополнительные атрибуты для параллелепипеда:

height – указывает высоту в направлении оси y при параллельной ориентации,

width – указывает ширину в направлении оси z при параллельной ориентации,

size(length, height, width) – устанавливает длину, высоту и ширину параллелепипеда.

ОБЪЕКТ КРИВАЯ

Кривая определяется как ломанная, проходящая через точки, и если точки расположены достаточно близко друг к другу, то получается достаточно гладкая кривая. Кривая также позволяет строить графики функций.

Некоторые атрибуты, такие как **pos** и **color** могут быть различны для каждой точки кривой. Эти атрибуты сохраняются как числовые массивы.

Можно задать кривую как явный перечень координат точек, заключенных в скобки. Например, массив точек, расположенных на плоскости, может быть задан так:

```
square = curve(pos=[(0,0),(0,1),(1,1),
                    (1,0),(0,0)])
```

Кривые могут иметь толщину, которая задается атрибутом **radius**. Например,

```
curve(pos=[(0,0,0), (1,0,0), (2,1,0)],
      radius=0.05)
```

По умолчанию радиус равен 0. В приведенном случае рисуется очень тонкая кривая.

На листинге 4 представлена программа, создающая спиральную линию.

Результат выполнения показан на рисунке 11.

Для построения графика можно задавать атрибуты **x,y,z**, как массивы. Например,

```
curve( x=arange(100), y=arange(100)**0.5,
                      color=color.red)
```

Пример построения графика параметрической функции:

```
t = arange(0, 10, 0.1)
curve( x = sin(t), y = 1.0/(1+t),
       z = t**0.5,
      red = cos(t), green = 0,
                blue = 0.5*(1-cos(t)) )
```

Кривые могут создаваться с помощью функции **append()**:

```
helix = curve( color = color.cyan )
for t in arange(0, 2*pi, 0.1):
   helix.append( pos=(t,sin(t),cos(t)) )
```

Можно использовать кривые для демонстрации траектории, по которой движется объект.

Например, если объект *ball* представляет собой движущуюся сферу, то показ траектории осуществляется следующей командной строкой:

```
trail = curve()
ball = sphere()
...# В каждый момент времени
   # положение мяча обновляется:
trail.append(pos=ball.pos)
```

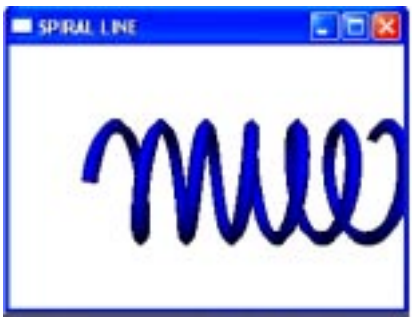


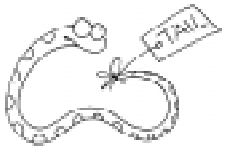
Рисунок 11.

Листинг 4

```
from visual import *
scene1=display(title='SPIRAL LINE',width=300,height=300,center=(7,0,0),
        background=(1,1,1))
c = curve( x = arange(0,20,0.1), color=color.blue,radius=0.5 ) # Draw a helix
c.y = 4*sin( 2.0*c.x )
c.z = 4*cos( 2.0*c.x )
```

ЦЕЛЕВОЙ ОБЪЕКТ

Целевой объект содержит список точек для атрибута **pos**, подобно объекту кривая. Целевыми могут быть точки, расположенные на выпуклой поверхности объекта. Точки, расположенные на вогнутой части поверхности, не рассматриваются. Если все точки расположены на плоскости, объект является плоской поверхностью.

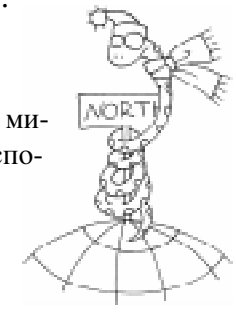


ОБЪЕКТ МЕТКА (LABEL)

Объект метка позволяет помещать текст в рамке и присоединять с помощью линии к определенной точке. На рисунке 12 сфера представляет собой Землю (Earth), которая имеет привязанную к ней метку, несущую текст «Earth» в рамке. Рамка присоединена к сфере с помощью линии, которая заканчивается на поверхности сферы:

```
earthlabel = label(pos=earth.pos,
text='Earth', xoffset=20, yoffset=12,
space=earth.radius, height=10, border=6)
```

Необычность характеристик объекта метки состоит в том, что некоторые атрибуты указаны через пиксели экрана вместо обычных пространственных координат. Например, высота текста дается в пикселях с учетом того, чтобы текст оставался удобочитаемым, даже когда сфера удаляется. Другие пиксель-атрибуты включены в **xoffset**, **yoffset** и **border**.



АТРИБУТЫ МЕТКИ

pos; x,y,z – точки в мировом пространстве, где располагается надпись;

xoffset, yoffset – компоненты *x* и *y* в пикселях (рисунок 12);

text – текст, который отображается;

height – высота шрифта в пикселях;

color, red, green, blue – цвет текста;

opacity – непрозрачность фона, заключенного в рамку (по умолчанию 0.66, 0 устанавливает полную прозрачность, 1 – непрозрачность);

border – расстояние в пикселях от текста до края рамки;

box – отображение рамки (если **box=1**, то рамка отображается, если **box=0**, то рамка не отображается);

line – отображение линии от рамки до объекта (если **line=1**, то линия от рамки до объекта отображается, если **line=0**, то линия не отображается);

linecolor – цвет линии и рамки;

space – радиус сферы в мировом пространстве, окружающей точку, в которой расположен центр объекта (внутрь сферы линия, проведенная от рамки, не попадает);

font – название шрифта, например «helvetica».

СОЗДАНИЕ СЛОЖНЫХ ОБЪЕКТОВ С ПОМОЩЬЮ ФРЕЙМОВ

Можно группировать объекты вместе, создавая сложный объект, который будет двигаться и вращаться как единый объект.

Создать фрейм и связанные с ним объекты можно следующим образом:

```
f = frame()
cylinder(frame=f,  pos=(0,0,0),
radius=0.1, length=1, color=color.cyan)
sphere(frame=f, pos=(1,0,0), radius=0.2,
color=color.red)
f.axis = (0,1,0)
f.pos = (-1,0,0)
```

По умолчанию **frame()** имеет положение (0,0,0) и ось в направлении *x* (1,0,0).

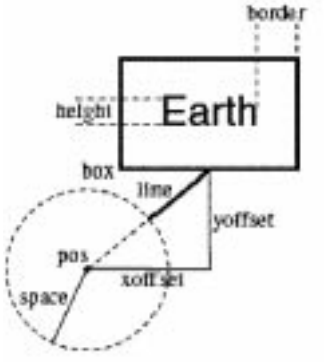


Рисунок 12.

Цилиндр и сфера созданы внутри фрейма. Когда некоторые из атрибутов фрейма изменяются (**pos.x, x, y, z, axis, up**), весь объект меняет ориентацию и положение.

Другим атрибутом фрейма является **objects**, который содержит перечень объектов, входящих во фрейм.

Можно сделать все объекты в фрейме временно невидимыми, используя следующий оператор (имя фрейма **f**):

```
for obj in f.objects:
        obj.visible = 0
```

Можно использовать подобный метод, чтобы назначить всем объектам в фрейме одинаковый цвет.

ДОПОЛНИТЕЛЬНЫЕ АТРИБУТЫ

Следующие атрибуты также применяются ко всем объектам, создаваемым в VPython:

visible – видимость объекта (если **ball visible=0**, объект невидим, если **ball visible=1**, то объект видимый);

frame – помещает объект в определенный фрейм, например, **ball=sphere(frame=f1)**;

display – при запуске VPython создается смотровое окно, которое называется сценой, можно выбрать и поместить объект в различные сцены, например:

```
scene2 = display( title = "Act IV, Scene 2" )
rod = cylinder( display = scene2 )
```

class – имя класса объекта, например, **ball_class_=sphere** (обязательны подчеркивания перед и после слова **class**).

member – перечень атрибутов объекта, например, **ball_member_** имеет атрибуты **['axis','blue','color', ...]** (обязательны подчеркивания перед и после слова **member**).

ОБЪЕКТЫ, СОЗДАВАЕМЫЕ ПО УМОЛЧАНИЮ

По умолчанию создаются объекты со следующими атрибутами:

cylinder() – эквивалентно
cylinder(pos=(0,0,0), axis=(1,0,0), radius=1);

arrow() – эквивалентно
arrow(pos=(0,0,0), axis=(1,0,0), radius=1);

cone() – эквивалентно
cone(pos=(0,0,0), axis=(1,0,0), radius=1);

sphere() – эквивалентно
sphere(pos=(0,0,0), radius=1);

ring() – эквивалентно
ring(pos=(0,0,0), axis=(1,0,0), radius=1);

box() – эквивалентно
box(pos=(0,0,0), length=1, height=1, width=1);

curve() – устанавливает «пустую» кривую, все точки которой могут быть присоединены;

convex() – устанавливает «пустой» объект, все точки которого могут быть присоединены;

frame() – устанавливает фрейм с атрибутами **pos=(0,0,0) and axis=(1,0,0)**.

ВРАЩЕНИЕ ОБЪЕКТА

Цилиндр, стрелку, конус, сферу, кольцо и куб можно повернуть на некоторый угол относительно оси, проходящей через основание объекта:

```
object.rotate(angle=pi/4.,
          axis=axis, origin=pos)
```

Функция вращения преобразует соответствующий объект. Под преобразованием понимается поворот на угол (выраженный в радианах), выполненный против часовой стрелки относительно линии, проходящей через точку основания и ось объекта. По умолчанию вращение происходит вокруг собственной точки основания и оси.

ОПРЕДЕЛЕНИЕ ЦВЕТОВ

В RGB цветовой системе цвет определяется как соединение красного, зеленого и синего. В RGB цветовой системе белый цвет соответствует цвету с максимальной интенсивностью красного, зеленого и синего (1,1,1). Черный соответствует минимальной величине (0,0,0). Самый

яркий красный соответствует значению (1,0,0).

Некоторые примеры RGB цветовых сочетаний, используемых в VPython:

(1,0,0) красный цвет (1,1,0) желтый цвет
(0,1,0) зеленый цвет (0,1,1) голубой
(0,0,1) синий цвет (1,0,1) малиновый
(1,1,1) белый цвет (0,0,0) черный цвет

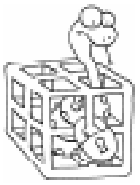
Можно создавать собственные цвета, например,

(0.5, 0.5, 0.5) темно-серый
(1,0.7,0.2) медный цвет

Подключаемый модуль *colorsliders.py* позволяет использовать RGB ползунки для регулировки цветов и HSV ползунки для регулировки оттенка, насыщенности и яркости. С помощью системы HSV (оттенок, насыщенность и яркость) также можно описать цвет. В VPython имеется возможность преобразовать цвет из составляющих RGB в HSV:

```
c = (1,1,0)
c2 = color.rgb_to_hsv(c)
          # convert RGB to HSV
print hsv  # (0.16667, 1, 1)
c3 = color.hsv_to_rgb(c2)
          # convert back to RGB
print c3   # (1, 1, 0)
```

Другой пример: `sphere(radius=2, color=hsv_to_rgb( (0.5,1,0.8) ).`

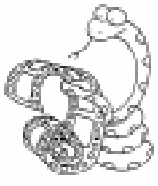


УДАЛЕНИЕ ОБЪЕКТОВ

Чтобы удалить объект нужно сделать его невидимым: `ball.visible=0`

Если позже использовать имя `ball` для создания нового объекта, программа будет использовать память, выделенную для предыдущего объекта, для создания нового объекта.

ЧАСТОТА СМЕНЫ КАДРОВ



Частота смены кадров – это скорость, с которой кадры создаются и отображаются на экране монитора. Если установить `rate (50)` внутри циклического процесса, то выполнится 50 кадров анимации в секунду.

ОБЪЕКТ ВЕКТОР

Объект вектор не визуализируется, но имеет широкое применение в трехмерной графике. Его свойства такие же, как приняты в научных и инженерных исследованиях. Компоненты вектора могут представляться числовым массивом

`vector(x,y,z)`

Компоненты вектора представляют собой числа с плавающей точкой. Векторы можно складывать, вычитать и умножать. Например,

```
v1 = vector(1,2,3)
v2 = vector(10,20,30)
print v1+v2      # результат (11 22 33)
print 2*v1       # результат (2 4 6)
```

Можно рассматривать отдельно компоненты вектора:

```
v2.x =10, v2.y =20, v2.z =30
```

Модуль вектора и квадрат модуля вектора вычисляются следующим образом:

```
mag( vector )
     # вычисляет длину (модуль) вектора
mag(vector(1,1,1)) # равен sqrt(3)
mag2(vector(1,1,1))
     # равен 3, квадрат модуля вектора
```

Можно присвоить модулю вектора и квадрату модуля вектора новые значения:

```
v2.mag = 5 # присваивает модулю
           # вектора v2 значение 5
v2.mag2 = 2.7   # присваивает квадрату
     # модуля вектора v2 значение 2.7
```

Можно нормировать вектор и задать модуль, равный 1:

```
norm( vector ) # нормировка; длина равна 1
norm(vector(1,1,1))
          equals vector(1,1,1)/sqrt(3)
```

Выражение `cross( vector1, vector2 )` создает векторное произведение двух векторов. Длина результирующего вектора равна произведению длин исходных векторов на синус угла между ними.

Листинг 4.

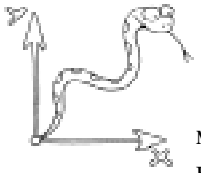
```
from visual.graph import *
funct1 = gcurve(color=color.cyan)           # задаем кривую
for x in arange(0., 8.1, 0.1):              # задаем переменную x от 0 до 8
 funct1.plot(pos=(x,5.*cos(2.*x)*exp(-0.2*x)))  # строим график
```

Выражение **dot(vector1, vector2)** создает скалярное произведение, которое определяется как произведение длин векторов на косинус угла между ними.

Выражение

```
v2 = rotate(v1, angle=theta, axis=(1,1,1))
```

поворачивает вектор вокруг оси.



ПОСТРОЕНИЕ ГРАФИКОВ

Для построения графиков с маркерами и метками (графики и гистограммы) следует подключить модуль *from visual.graph import * .*

График строится следующим образом (см. листинг 4).

Модуль *visual.graph* позволяет просматривать все объекты и строить графики. График полностью располагается в просмотровом окне.

Кроме графика, представленного сплошной кривой (**gcurve**), возможны графики в виде отдельных точек (**gdots**), вертикальных столбцов (**gvbars**), горизонтальных столбцов (**ghbars**) и графики, показанные в виде гистограмм. При создании графика можно определить его цвет. При создании графиков типа **gvbars**, **ghbars** и **ghistogram** атрибут **delta** указывает ширину полоски (по умолчанию **delta=1** для **gvbars** и **ghbars**) или ширину столбика для **ghistogram**.

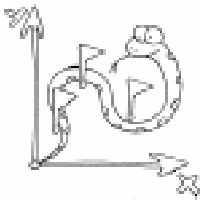
Можно построить несколько графиков в одном и том же просмотровом окне (см. листинг 5).

При построении графика можно изменить первоначальный цвет:

```
mydots.plot(pos=(x1,y1),
                    color=color.green)
```

При создании **gcurve**, **gdots**, **gvbars** или **ghbars** объектов можно указать перечень точек, по которым будет строиться график:

```
points = [(1,2), (3,4), (-5,2), (-5,-3)]
data = gdots(pos=points,
                    color=color.blue)
```



ПОЛНЫЙ ПЕРЕЧЕНЬ АТРИБУТОВ GDISPLAY

Окно, в котором просматривается график, определяется опцией *gdisplay*. Перед созданием графика можно установить размер, положение и название графика, определить оси *x* и *y*, установить максимальные значения для каждой координаты.

В примере на листинге 6 начало просмотрового окна графика расположено в точке (0,0), размер окна 600×150 пикселей, название графика *'N vs. t'*. По горизонтальной оси графика отложена переменная *t*, а по вертикальной – переменная *N*.

Значения по горизонтальной оси будут меняться от –20 до +50, а по вертикальной шкале – от –2000 до 5000 (**xmin** и **ymin** могут быть отрицательными величинами, а **xmax** и **ymax** должны быть положительными).

Передний план (белый по умолчанию) установлен черным, фон (черным по умолчанию) установлен белым. Если указать **gdisplay()**, тогда по умолчанию **x=0**, **y=0**, **width=800**, **height=400**, название отсутствует, установлен режим автомасштабирования.

Листинг 5.

```
funct1 = gcurve(color=color.cyan)
funct2 = gvbars(delta=0.05, color=color.blue)
for x in arange(0., 8.1, 0.1):
    funct1.plot(pos=(x,5.*cos(2.*x)*exp(-0.2*x)))     # curve
    funct2.plot(pos=(x,4.*cos(0.5*x)*exp(-0.1*x)))    # vbars
```

Листинг 6.

```
graph1 = gdisplay(x=0, y=0, width=600, height=150,
          title='N vs. t', xtitle='t', ytitle='N',
          xmax=50., xmin=-20., ymax=5E3, ymin=-2E3,
          foreground=color.black, background=color.white)
```

Каждая функция **gdisplay** имеет атрибуты **display**. Это дает возможность изменять основные характеристики просмотрового окна.

Например,

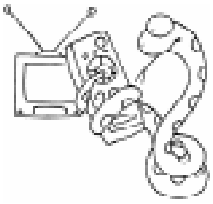
```
graph1.display.visible = 0
  # делает просмотровое окно невидимым
```

Можно создавать несколько просмотровых окон, создавая другие функции **gdisplay**.

По умолчанию каждый последующий графический объект создается выше предыдущего.

Можно указать, в каком окне расположен график, принадлежащий объекту:

```
energy = gdots(gdisplay=graph1,
                        color=color.blue)
```



СРЕДСТВА КОНТРОЛЯ: КНОПКИ, ПОЛЗУНКИ, ПЕРЕКЛЮЧАТЕЛИ И МЕНЮ

Чтобы контролировать программу, можно создавать кнопки, ползунки, переключатели и меню.

Для получения возможности создания необходимо подключить модуль *from visual.controls import **

Импортирование из *visual controls* позволяет просматривать все графические объекты и подключает модуль контроля (*controls module*). Для использования объектов контроля необходимо создать специальное окно контроля и добавить объекты контроля, указав, какие изменения будут происходить под действием объектов контроля.

Например, нажатие кнопки будет изменять цвет визуализируемого предмета.

В програме на листинге 7 текст, написанный на кнопке, изменяется при нажатии кнопки.

ОКНО КОНТРОЛЯ

controls() создает окно контроля со специальными атрибутами и возвращает их. Например, командная строка:

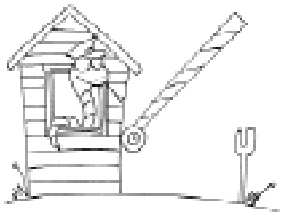
```
c = controls(title='Controlling the Scene',
  x=0, y=400, width=300, height=300,
                                range=50)
```

создает окно размером 300 × 300 пикселей, расположенное в точке (0,400) относительно верхнего левого угла экрана, с названием '**Controlling the Scene**' в титульной строке с диапазоном 50 (координаты окна изменяются от –50 до +50 по осям *x* и *y*)

Листинг 7.

```
from visual.controls import *

def change():   # Создает функцию контроля при нажатии кнопки
    if b.text == 'Click me':
        b.text = 'Try again'
    else:
        b.text = 'Click me'
c = controls()  # Создаем окно контроля
# Создаем кнопку в окне контроля
b = button( pos=(0,0), width=60, height=60,
              text='Click me', action=lambda: change() )
while 1:
    c.interact() # Проверяем событие и выполняем соответствующие действия
```



Атрибуты окна контроля:

x,y – положение окна на экране (пиксели от верхнего левого угла);

width, height – ширина и высота окна в пикселях;

title – текст в титульной строке;

range – протяженность области просмотра от центра вдоль каждой оси (по умолчанию составляет 100, центр окна контроля всегда расположен в точке (0,0)).

ОБЪЕКТЫ, С ПОМОЩЬЮ КОТОРЫХ ОСУЩЕСТВЛЯЕТСЯ КОНТРОЛЬ

После создания окна контроля можно создать следующие объекты контроля, которые появятся в окне:

button – кнопка (контроль осуществляется щелчком на кнопке);

slider – ползунок (контроль осуществляется при перетаскивании ползунка);

toggle – переключатель (контроль осуществляется щелчком по переключателю);

menu – выпадающее меню (содержит опции, которые осуществляют контроль);

Объекты контроля имеют следующие атрибуты:

pos – положение объекта контроля (центр кнопки или переключателя, одного из концов ползунка, верхнего левого угла меню);

color – серый цвет по умолчанию;

width – ширина кнопки, переключателя или меню;

height – высота кнопки, переключателя или меню;

axis – ось ползунка, проведенная из положения pos (основания);

length – высота ползунка (по оси);

min, max – минимальная и максимальная величина, которая устанавливается при перемещении ползунка;

value – значение переключателя (0 или 1) или ползунка (зависит от значений min и max); если значение установлено, то изменения объекта будут происходить при указанных величинах;

text – текст, который показан на кнопке или названии меню;

text() – текст, который устанавливается внизу переключателя (показывается на переключателе при значении 0);

text1 – текст, который устанавливается вверху переключателя (показывается на переключателе при значении 1);

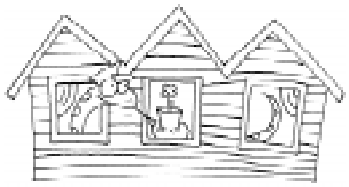
action – устанавливается выполнение при манипулировании объектами контроля;

items – устанавливается только для меню (перечень пунктов меню, с помощью которых осуществляется контроль).

Ниже показано добавление в меню пункта, названного **m1**:

```
m1.items.append( ('Red', lambda:
                 cubecolor(color.red)) )
```

КОНТРОЛИРОВАНИЕ ОДНОГО ИЛИ БОЛЕЕ ПРОСМОТРОВЫХ ОКОН

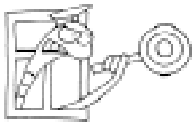


Обычно существует одно просмотровое окно, которое называется сценой (**scene**).

Опция **display()** создает просмотровое окно с определенными атрибутами.

Командная строка, приведенная ниже, создает смотровое окно размером 600 × 200, указывает название **'Graph of position'** в титульной строке, размещает его в точке (5,0,0) и устанавливает малиновый цвет фона:

```
scene2 = display(title='Graph of position',
width=600, height=200,
center=(5,0,0), background=(0,1,1))
```



АТРИБУТЫ ОБЩЕГО НАЗНАЧЕНИЯ

select () – устанавливает смотровое окно, все объекты отображаются в этом смотровом окне, то есть **scene select()**.

foreground – устанавливает цвет, который будет использоваться по умолчанию для создания новых объектов в цвете. Первоначально по умолчанию используется белый цвет. Например, **scene.foreground=(1,0,0)**).

background) – устанавливает цвет, который используется, чтобы заполнить фон окна. По умолчанию – черный.

ambient – величина ненаправленного освещения (цвета «подсветки»). По умолчанию устанавливается значение 0.2.

lights – список векторов, показывающих направления от начала координат до источников света. Величина вектора определяет интенсивность источника света, например, **scene.lights = [vector (1,0,0)]** со значением **scene.ambient = 0** будет освещать сцену с правой стороны, свет подсветки будет равен нулю слева. По умолчанию есть два источника света в списке. Атрибуты источников света и подсветки должны использоваться с осторожностью, так как если общая интенсивность превысит значение 1 в сцене, то результаты будут непредсказуемы.

cursor visible – при установке **scene.cursor.visible = 0** курсор мыши становится невидимым. Это бывает необходимо при перемещении объекта с помощью мыши. Для восстановления видимости курсора следует установить **scene.cursor.visible = 1**.

objects – перечень всех видимых объектов в сцене; объекты, которые невидимы, в перечень не входят Например, приведенная в листинге 8 командная строка делает всех кубы в сцене видимыми и красными.

УПРАВЛЕНИЕ ОКНОМ ПРОСМОТРА

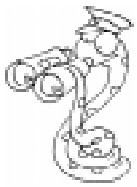
x,y – положение просмотрового окна на экране (пиксели от верхнего левого угла).

width, height – ширина и высота просмотровой области в пикселях. Например, **scene.height = 200**.

title – текст, появляющийся в названии просмотрового окна. Например, **scene.title = 'Planetary Orbit'**.

visible – командная строка **scene2.visible = 1** позволяет визуализировать все объекты сцены. Установка **scene2.visible = 0** скрывает показ объектов.

exit If sceneb.exit=0 – просмотровое окно не закрывается при нажатии кнопки. По умолчанию **sceneb.exit=1** (просмотровое окно закрывается при нажатии кнопки).



УПРАВЛЕНИЕ ПРОСМОТРОМ

center – место, на которое направлена камера. Если положение центра меняется, камера направлена на место нового центра. По умолчанию положение центра (0,0,0).

autocenter – центр сцены непрерывно обновляется и располагается в просмотровом окне при движении внутри сцены.

forward – вектор, указывающий в том же направлении, что и камера (то есть от текущего положения камеры, заданного в командной строке **scene.mouse.camera**, по направлению к центру сцены **scene.center**). При вращении камеры направление вектора изменяется автоматически. При изменении вектора положение камеры меняется. Камера по-прежнему направлена в центр. По умолчанию (0,0,–1).

fov – поле просмотра камеры, выраженное в радианах. Определяет максимальное горизонтальное и вертикальное поля просмотра. По умолчанию **pi/3.0** радиан (60 градусов).

range – область сцены, которая просматривается от центра вдоль каждой из осей. По умолчанию (10,10,10).

scale – масштаб, который накладывается на область просмотра. По умолчанию (0.1, 0.1, 0.1).

Листинг 8.

```
for obj in scene2.objects:
    if obj.__class__ == box # can say either box or 'box'
       obj.color = color.red
```

uniform=0 – для каждой оси устанавливаются различные единицы и масштаб. Масштаб каждой оси выбирается в зависимости от размеров просмотрового окна.

uniform=1 – каждая ось имеет тот же самый масштаб. Масштабирование осей происходит одинаково.

up – вектор, указывающий верх мирового пространства. Этот вектор всегда будет проектироваться на вертикальную линию на экране. По умолчанию ось *Y* – вектор, показывающий верх мирового пространства.

autoscale=0 – автоматическое изменение масштаба не выполняется (следует устанавливать масштабирование явно).

autoscale=1 – производится автоматическое изменение масштаба (устанавливается по умолчанию).

userzoom=0 – пользователь не может увеличивать сцену.

userzoom=1 – пользователь может увеличивать сцену (по умолчанию).

userspin=0 – пользователь не может производить вращение сцены.

userspin=1 – пользователь может производить вращение.

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Модуль, позволяющий вычислять математические функции, доступен всегда. Для вычисления функция комплексного переменного следует использовать модуль *cmath*. Возможно вычисление следующих функций:

acos(x) – возвращает значение arccos(*x*).

asin(x) – возвращает значение arcsin(*x*).

atan(x) – возвращает значение arctg(*x*).

atan2(y, x) – возвращает значение arctg(*y*/*x*).

ceil(x) – возвращает целое, большее или равное *x*.

cos(x) – возвращает значение cos(*x*).

cosh(x) – возвращает значение cosh(*x*).

exp(x) – возвращает значение e***x*.

fabs(x) – возвращает значение абсолютной величины числа с плавающей точкой *x*.

floor(x) – возвращает целое, меньшее или равное *x*.

fmod(x, y) – возвращает значение fmod(*x*, *y*), как определено в языке C.

frexp(x) – возвращает значение мантиссы и экспоненты *x* как пары (*m*, *e*); *m* является числом с плавающей точкой, *e* является целым, так что *x* == *m* * 2****e*. Если *x* равен нулю, то возвращаемое значение равно (0.0, 0), иначе 0.5 <= abs(*m*) < 1.

hypot(x,y) – возвращает значение sqrt(*x***x* + *y***y*).

ldexp(x, i) – возвращает значение *x* * (2****i*).

log(x) – возвращает натуральный логарифм *x*.

log10(x) – возвращает десятичный логарифм *x*.

modf(x) – возвращает дробную и целую часть *x*. Оба результата имеют знак *x*. Целая часть возвращается как плавающая величина.

pow(x, y) – возвращает значение *x****y*.

sin(x) – возвращает значение sin(*x*).

sinh(x) – возвращает значение sinh(*x*).

sqrt(x) – возвращает значение $\sqrt{x}$.

tan(x) – возвращает значение tg(*x*).

tanh(x) – возвращает значение tanh(*x*).

Определены также две математические константы:

pi – математическая константа π;

e – математическая константа *e*.

Литература

1. *Сузи Р.А.* Python. СПб: БВХ-Петербург, 2002.

2. *Гетманова Е.Е.* Моделирование физических процессов в VPython.-Финарт, Харьков, 2004.

Гетманова Елена Евгеньевна кандидат физико-математических наук, доцент кафедры физики Харьковского национального университета радиоэлектроники.