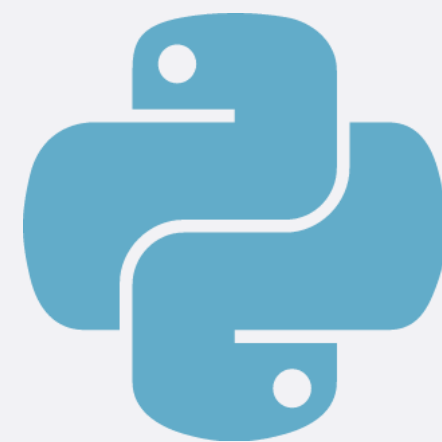


PYCONAMAZÔNIA 2017

「Aprendendo conceitos de Física com Python: uma introdução ao VPython!」



Tópicos Abordados

01 **Introdução**
Quem sou eu, de onde venho, o que faço, etc.

02 **Integração de conteúdos**
Conceitos de multidisciplinaridade e interdisciplinaridade para desenvolver projetos integradores.

03 **O que é o VPython?**
O que é o VPython, quem criou, para que isso serve?

04 **Primitivas**
Exemplos de formas básicas que podem ser criadas com o VPython.

05 **Exemplos**
Exemplos de simulações criadas utilizando o VPython.

06 **Conclusões**
Conclusão final.

Introdução...

Quem sou eu?



Paulo Giovanni de Faria Zeferino

- Natural de Campos do Jordão, SP.
- Graduado em Computação Científica (UNITAU), especializado em Jogos Digitais, Banco de Dados, Docência e Gestão Pública e mestre em Computação Aplicada (INPE).
- Professor no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo – Campos do Jordão.

Áreas de Interesse

- Programação (Python!!!)
- Banco de dados
- Mineração de dados
- Computação gráfica
- Desenvolvimento de games
- Educação
- Ensino e aprendizagem
- Aikido

Integração de conteúdos

Desenvolvendo projetos interdisciplinares

A importância da interdisciplinaridade na Educação

Com o desenvolvimento da tecnologia, inúmeras mudanças ocorreram no comportamento da sociedade. Essas mudanças também se refletem no âmbito educacional. Torna-se cada vez mais difícil despertar nos alunos, os quais vivem numa sociedade amplamente tecnológica e em constante transformação, o interesse por aulas cuja metodologia baseia-se apenas em exposição oral e têm como único recurso o quadro e o giz. Contudo, em geral, os professores não estão preparados para trabalhar nesta nova realidade.

*Rodrigo Donizete Terradas, 2011.
A Importância da Interdisciplinaridade na Educação Matemática.
Revista da Faculdade de Educação, ano IX, n. 16.*

Integração de conteúdos

Desenvolvendo projetos interdisciplinares

Multidisciplinaridade

Ocorre quando há mais de uma área de conhecimento em um determinado projeto, mas cada uma mantém seus métodos e teorias particulares. Serve para resolver problemas imediatos e não possui foco na articulação e nos ganhos colaborativos.

Interdisciplinaridade

Ocorre quando mais de uma disciplina se une em um projeto comum, com um planejamento que as relacione. Durante o processo, estas áreas trocam conhecimentos e enriquecem ainda mais as possibilidades. Esta visão dá significado à experiência escolar.



Projetos Integradores



O que é um projeto integrador?

Atividade acadêmica desenvolvida com o objetivo de integrar conhecimentos de todas as disciplinas que compõem um determinado período, onde os alunos desenvolvem trabalhos práticos utilizando conceitos e fundamentos vistos em sala de aula.



Quais suas vantagens?

Integração de conteúdos vistos nas disciplinas, desenvolvimento de produtos e fixação da aprendizagem.



Quais suas desvantagens?

Requer planejamento do docente e do aluno. "... Contudo, em geral, os professores não estão preparados para trabalhar nesta nova realidade." - Rodrigo Donizete Terradas.

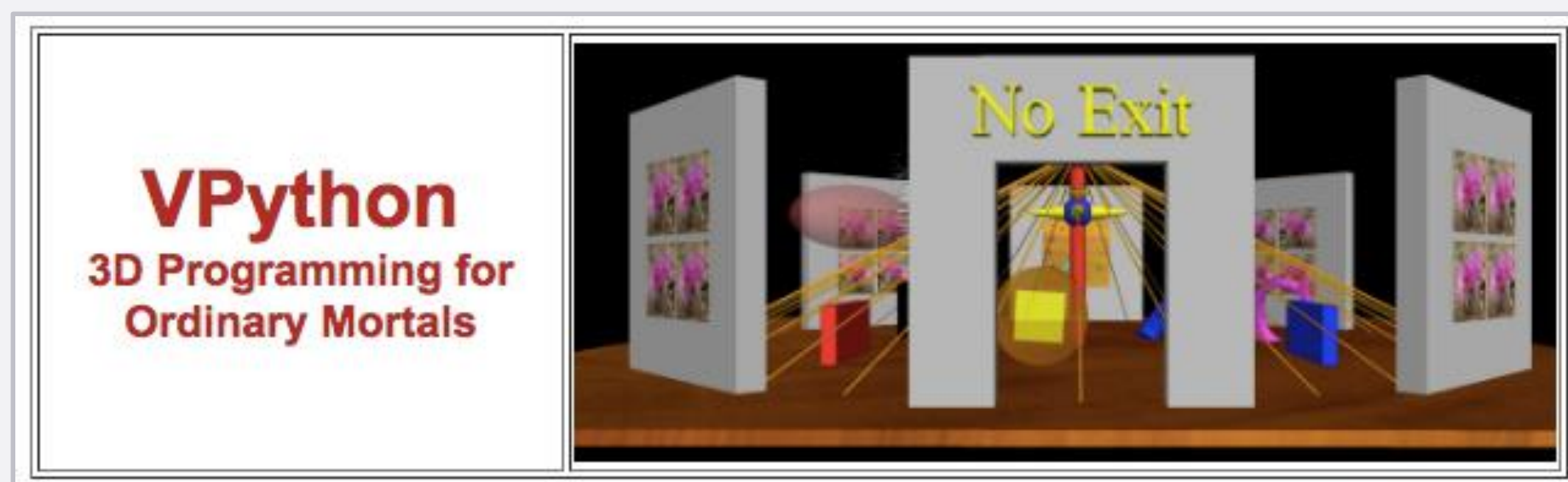


“Tá... mas dá pra fazer alguma coisa utilizando alguns dos conceitos da Física junto com a linguagem Python?”

Dúvida enviada por Albert Einstein
Físico alemão

VPython

O que é isso?



É uma linguagem de programação de fácil aprendizado e adequada à criação de modelos 3D interativos de sistemas físicos.

www.vpython.org





O que é o VPython?

O VPython é um módulo de gráficos 3D que atua em conjunto com a linguagem de programação Python.



Quais são seus recursos?

O VPython possui recursos que permitem criar uma variedade de objetos 3D que podem ser manipulados e animados.



Para que ele serve?

O VPython pode ser utilizado na criação de animações da vida real, tanto para diversão quanto como para uso educacional. É especialmente útil na criação de exemplos que envolvem leis simples da Física.

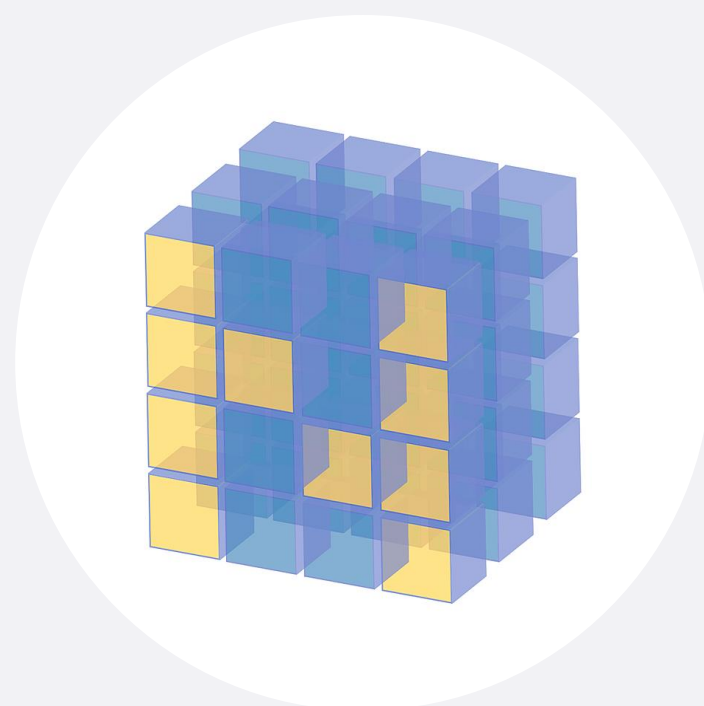
VPython

O que é isso?



O que é?

O VPython é a linguagem de programação Python em conjunto com um módulo para a criação de objetos 3D, desenvolvido no ano 2000 pelo aluno **David Scherer**.



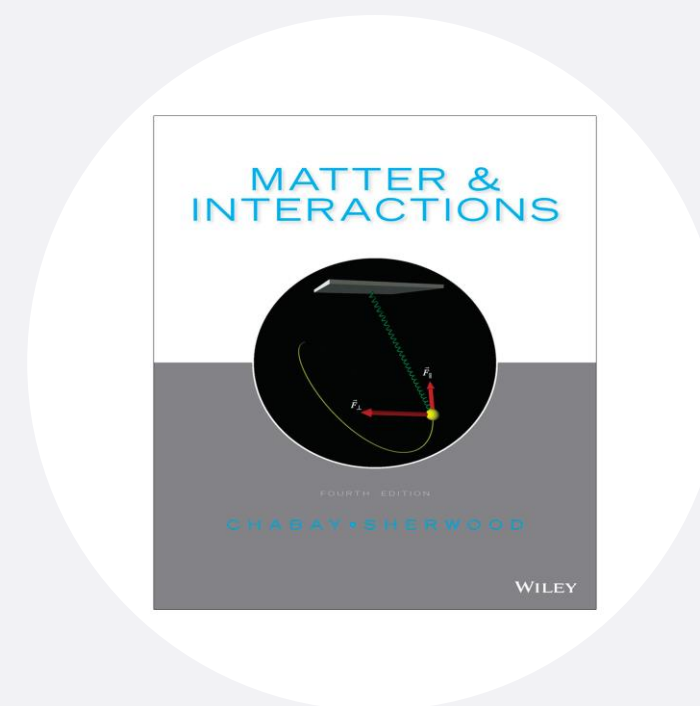
Por que Vpython?

O nome **VPython** é a combinação do módulo 3D, chamado Visual, em conjunto com a linguagem de programação Python.



Colaboradores?

A implementação original foi realizada por uma equipe da Universidade Carnegie Mellon liderada por David Scherer, com o auxílio de David Andersen, Ruth Chabay, Ari Heitner, Ian Peters e **Bruce Sherwood**.



Uso na Educação?

Estudantes de cursos introdutórios de Física têm utilizado o VPython para fazer a modelagem em computador, sem se preocupar com a parte da visualização gráfica (**Matter and Interactions**).

- Onde faço o download?

O VPython pode ser obtido através do site www.vpython.org. Existem opções disponíveis para Windows, Linux e Mac. O VPython 7 pode ser instalado utilizando um comando `[pip]` ou `[conda]`.

- Qual versão devo utilizar?

A versão denominada **Classic VPython**, correspondente ao VPython 5.74, utiliza uma versão do IDLE, denominada **VIDLE**, para a criação de seus programas. O VIDLE utiliza o Python 2.7. Entretanto, os desenvolvedores não fornecem mais suporte para essa versão. Versões mais novas, como o **VPython 6** e o **VPython 7** utilizam o **Jupyter** e permitem a escrita de código utilizando o Python 3. Esse código é compatível com o **GlowScript**.

VPython

Começando a utilizar

VPython – www.vpython.org

Baseado em **Python 2** ou **Python 3**. ✓

Ambiente mais maduro. ✓

Permite a utilização de módulos. ✓

Executa no computador. ✓

Não executa em um navegador web (VPython 5). ✓

Novas versões executam no navegador e possuem a sintaxe do GlowScript. ✓

GlowScript – www.glowscript.org

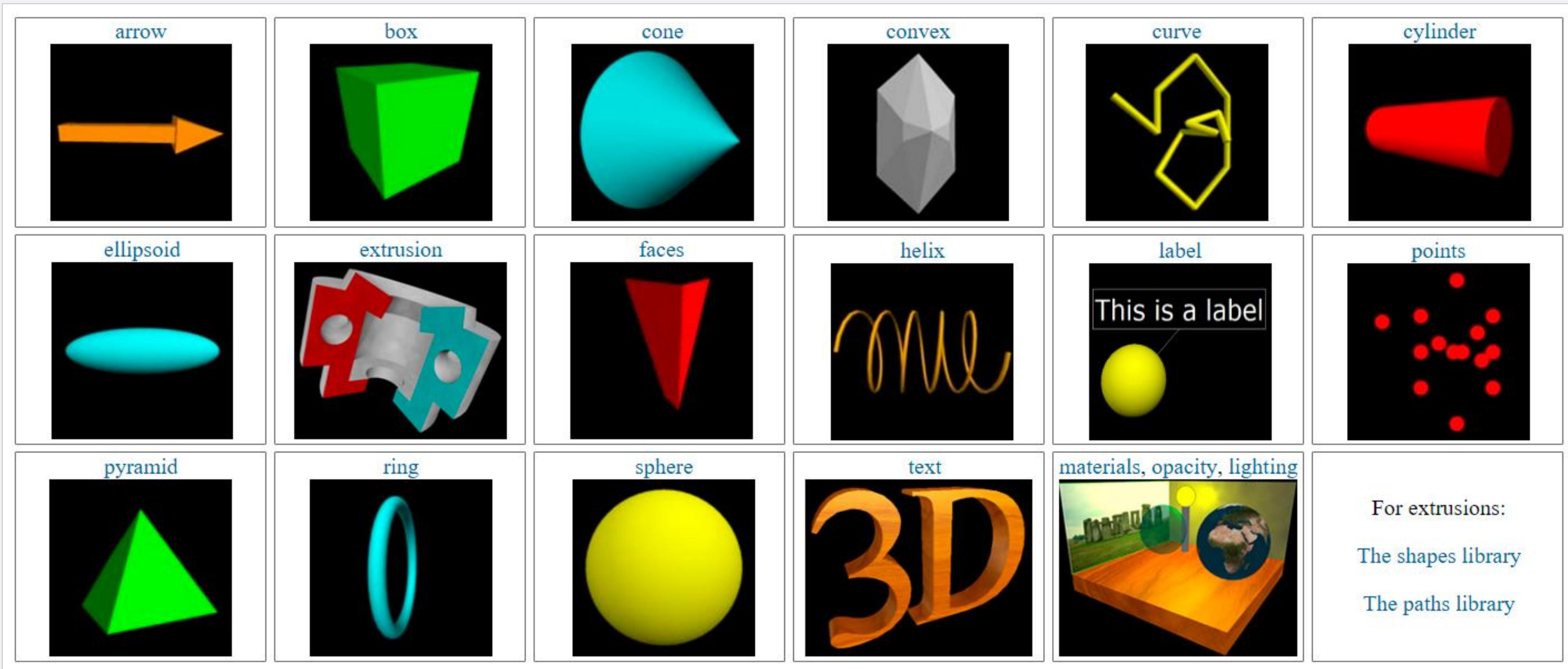
✓ Baseado em **Javascript** e **WebGL**.

✓ Ambiente mais atual.

✓ Não permite a utilização de módulos desenvolvidos pelo usuário.

✓ Necessita de um navegador web.

VPython Primitivas



VPython

Primitivas

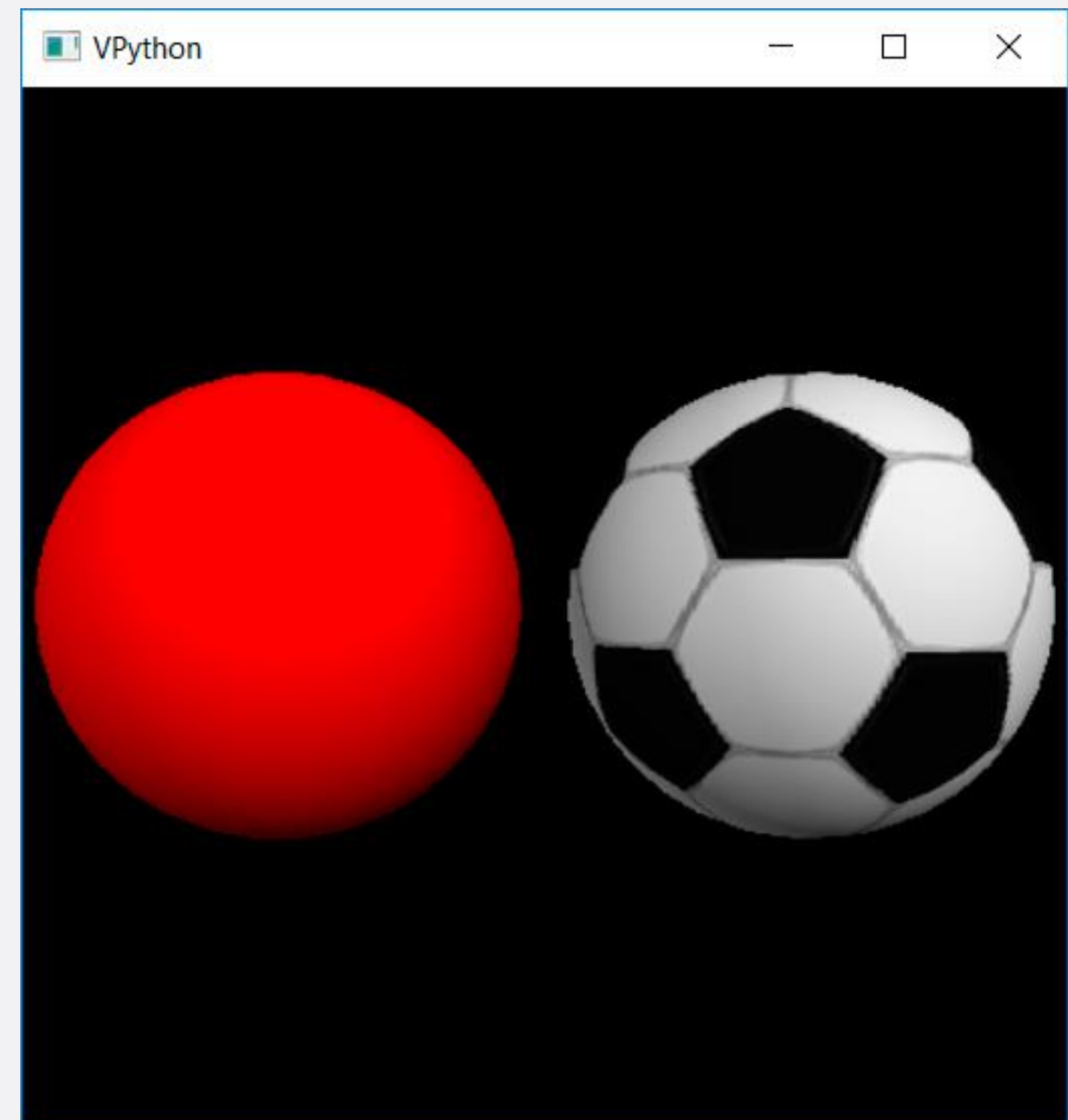
```
from visual import *

# Cria uma esfera vermelha
sphere(pos = vector(-1.1, 0, 0),
       color = color.red)

# Cria uma esfera utilizando uma textura
tex = materials.texture(data =
    materials.loadTGA('textura_bola.tga'),
    mapping = 'spherical',
    interpolate = False)

sphere(pos = vector(1.1, 0, 0),
       material = tex)

# Exibe a simulação
while True:
    rate(1)
```




```
# -*- coding: cp1252 -*-  
from visual import *  
  
def cria_plano(largura, profundidade, altura, cor):  
    """Cria um plano utilizando um objeto do tipo box."""  
  
    # Cria o plano  
    plano = box(pos = (0, -1, 0),  
                length = profundidade,  
                height = altura,  
                width = largura,  
                color = cor)  
  
    # Retorna o objeto  
    return plano
```

VPython Primitivas

```
def cria_bola(raio, massa, pressao):  
    """Cria uma bola de futebol utilizando uma esfera."""  
  
    # Textura da bola  
    tex = materials.texture(data =  
                             materials.loadTGA('textura_bola.tga'),  
                             mapping = 'spherical', interpolate = False)  
  
    # Cria a bola  
    bola = sphere(pos = vector(0, 0, 0), material = tex)  
  
    # Define os atributos da bola  
    bola.massa = massa  
    bola.circunferencia = 2.0 * raio  
    bola.pressao = pressao  
  
    # Retorna o objeto  
    return bola
```

```
# Criação da cena
```

```
# Dimensões e cor de fundo da janela da aplicação
```

```
largura_janela = 640
```

```
altura_janela = 480
```

```
cor_janela = (0.069, 0.343, 1.000)
```

```
# Define a cena
```

```
scene = display(title = 'Aprendendo Física com VPython',  
                x = 0,  
                y = 0,  
                autoscale = True,  
                width = largura_janela,  
                height = altura_janela,  
                center = (0, 0, 0),  
                background = cor_janela)
```


VPython Primitivas

```
# Define as dimensões e cor do piso
largura_piso = 10
profundidade_piso = 10
altura_piso = 0.01
cor_piso = (0.9, 0.9, 0.9)

# Cria o piso
piso = cria_plano(largura_piso,
                  profundidade_piso,
                  altura_piso,
                  cor_piso)

# Define alguns atributos para a bola
raio_bola = 0.7
massa_bola = 0.45
pressao_bola = 1.1
```

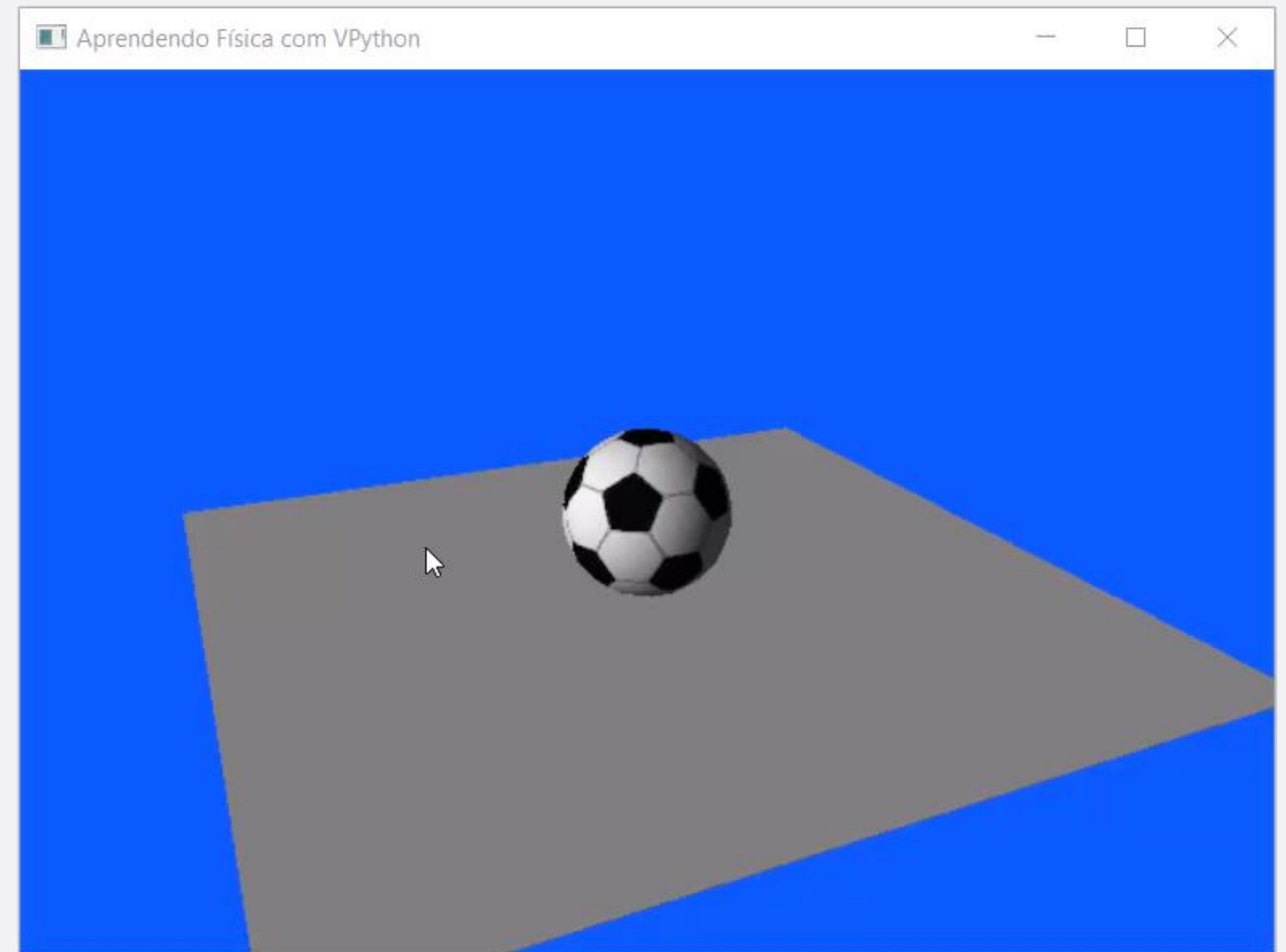
VPython

Primitivas

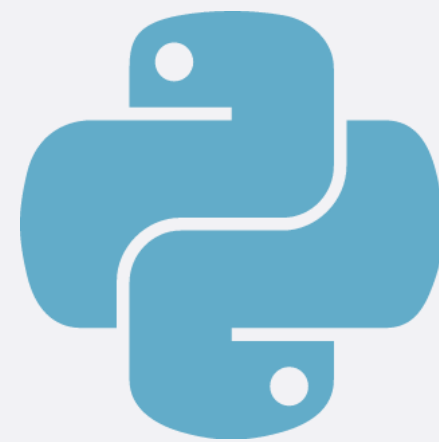
```
# Cria a bola
bola = cria_bola(raio_bola,
                 massa_bola,
                 pressao_bola)

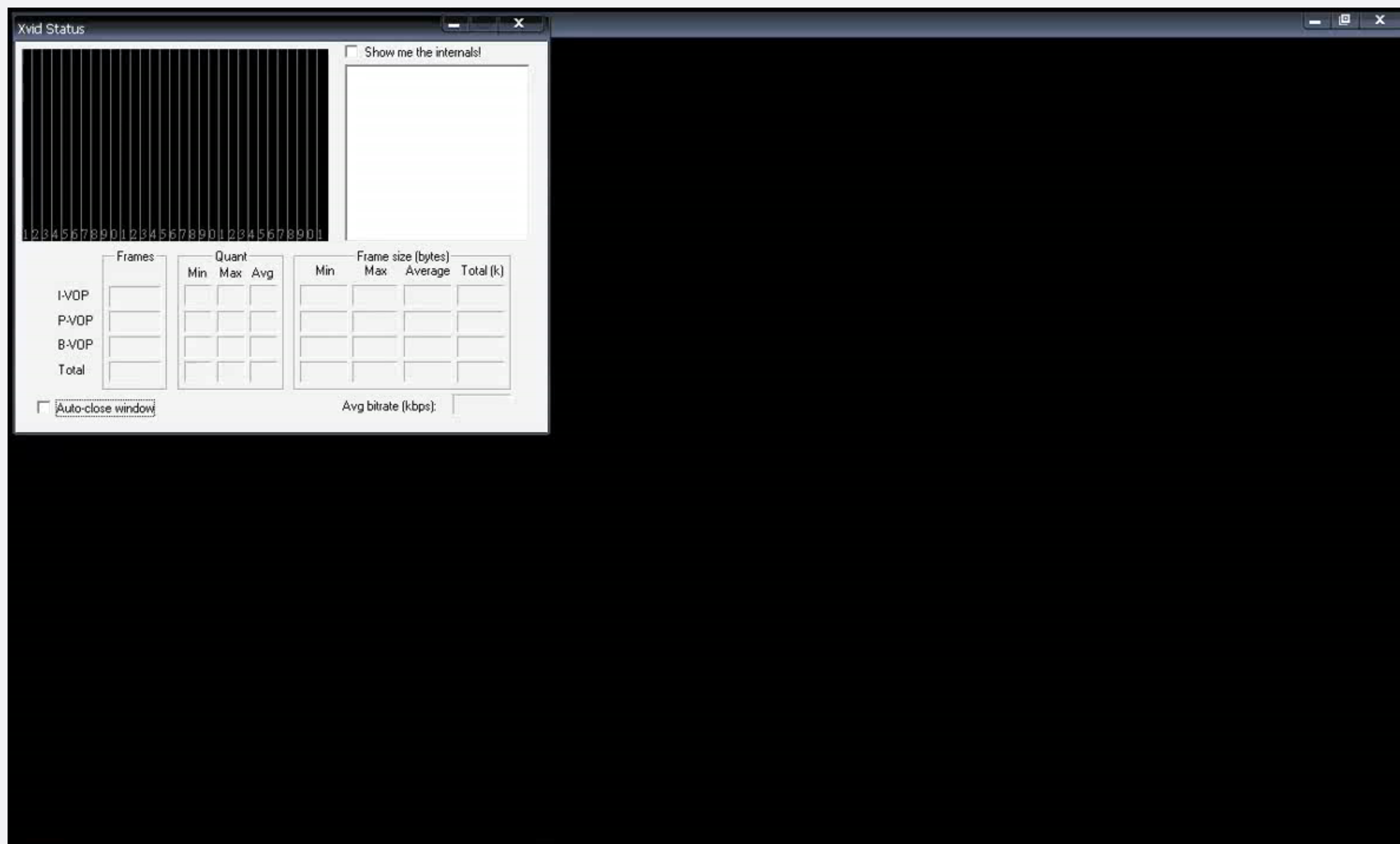
# Exibe a simulação
while True:

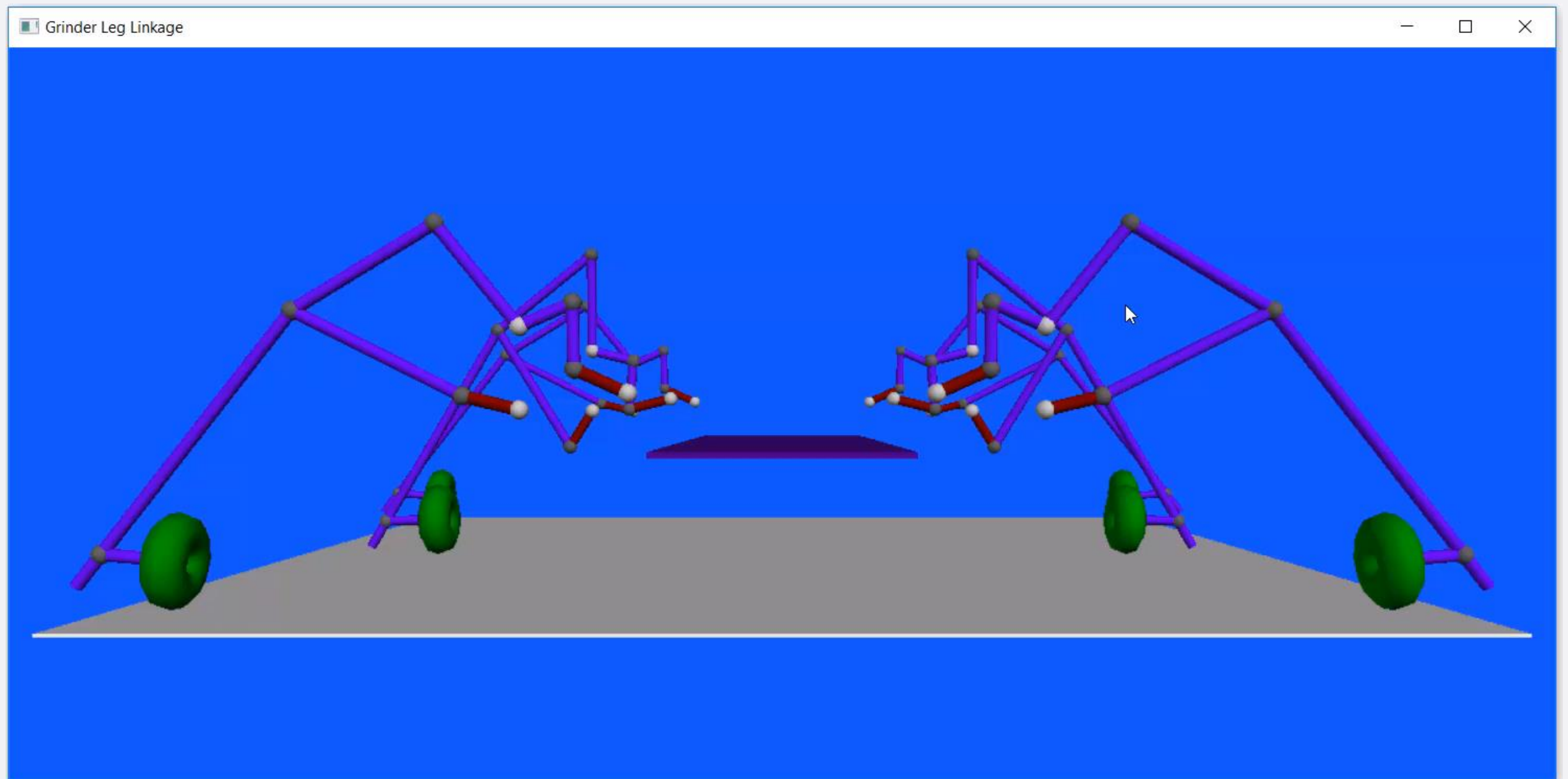
    # Taxa de animação
    rate(1)
```



Exemplos

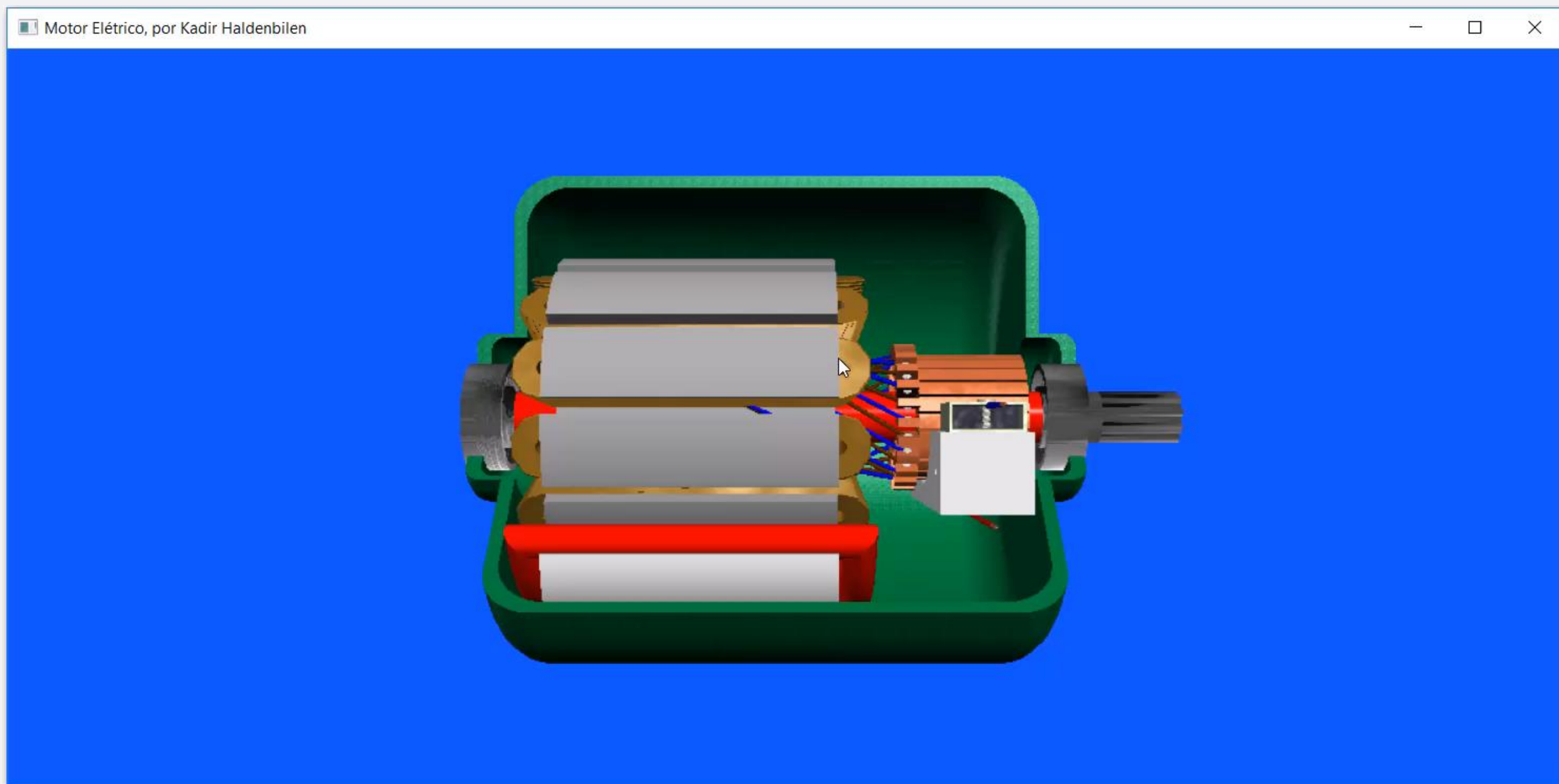






```
jakub@jakub-Satellite-A300:~$ recordmydesktop --full-shots
Initial recording window is set to:
X:0 Y:0 Width:1280 Height:800
Adjusted recording window is set to:
X:0 Y:0 Width:1280 Height:800
Your window manager appears to be KWin

Initializing...
Buffer size adjusted to 4096 from 4096 frames.
Opened PCM device default
Recording on device default is set to:
1 channels at 22050Hz
Capturing!
█
```

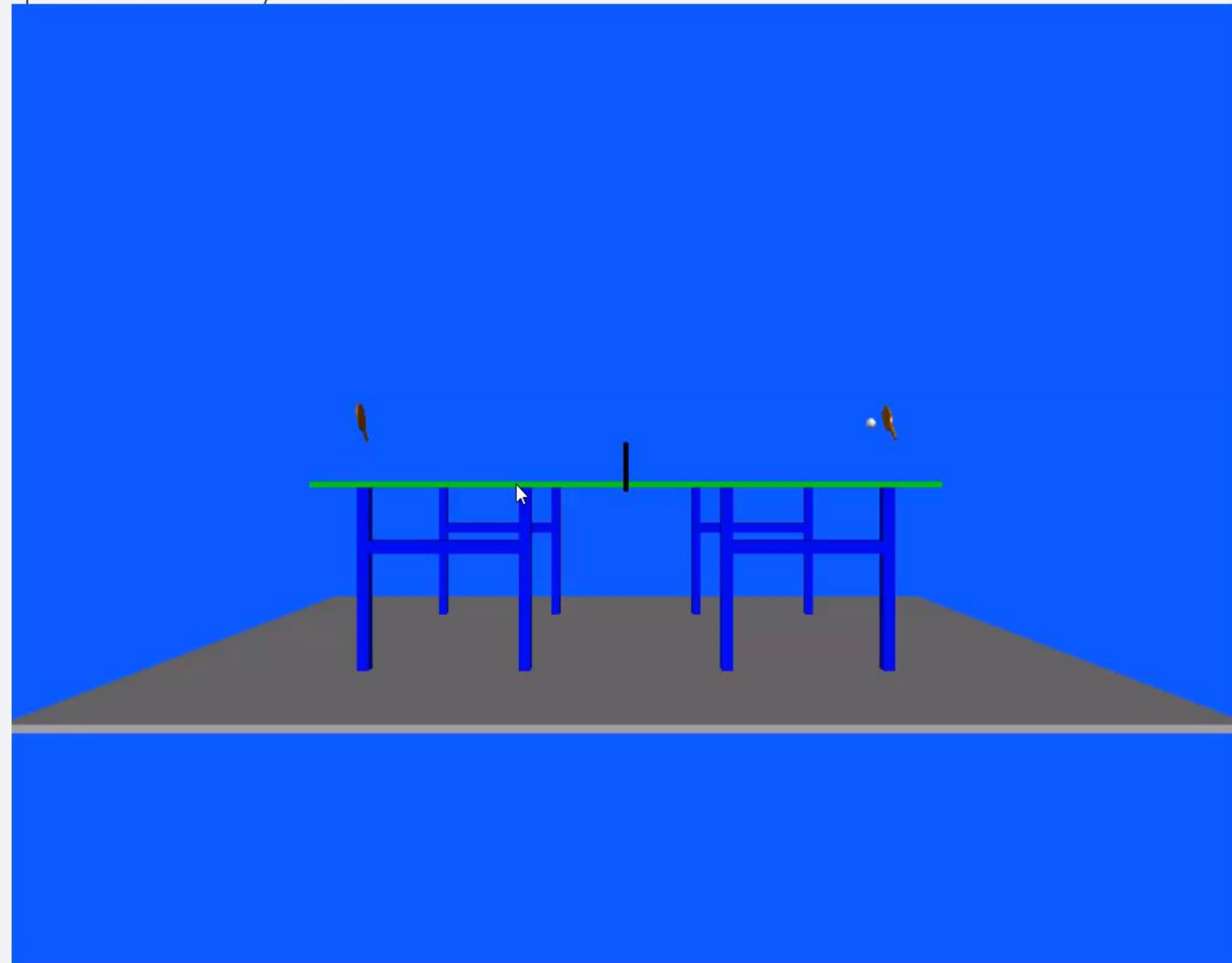



```
# Cabo da raquete
cabo = box(pos = vector(0, -1, 0),
           length = 2,
           height = 1,
           width = 0.1,
           color = vector(0.72, 0.42, 0))

# Raquete
raquete = cylinder(pos = vector(0, 0, 0),
                  size = vector(.1, 1.5, 10),
                  axis = vector(0, 0, 1),
                  color = vector(0.72, 0.42, 0))

# Une os objetos e cria a raquete direita
raquete_direita = compound([cabo, raquete])
raquete_direita.axis = vector(0, 0, 1)
raquete_direita.pos = vector(14, 3, 0)
raquete_direita.rotate(angle = 3.14/4,
                       axis = vector(-1, 0, 0))
```

Aprendendo Física com VPython



```
# Incremento de tempo
dt = 0.01

# Força da gravidade
g = 9.8

# Exibe a simulação
while True:

    # Taxa de animação
    rate(100)

    # Forças que atuam na bola
    # F = ma (segunda lei de Newton)
    forca_bola = vector(0,
                        -bola.massa * g,
                        0)

    aceleracao = forca_bola / bola.massa
```

```
# Atualiza a velocidade da bola
# v = v0 + at
bola.velocidade += aceleracao * dt

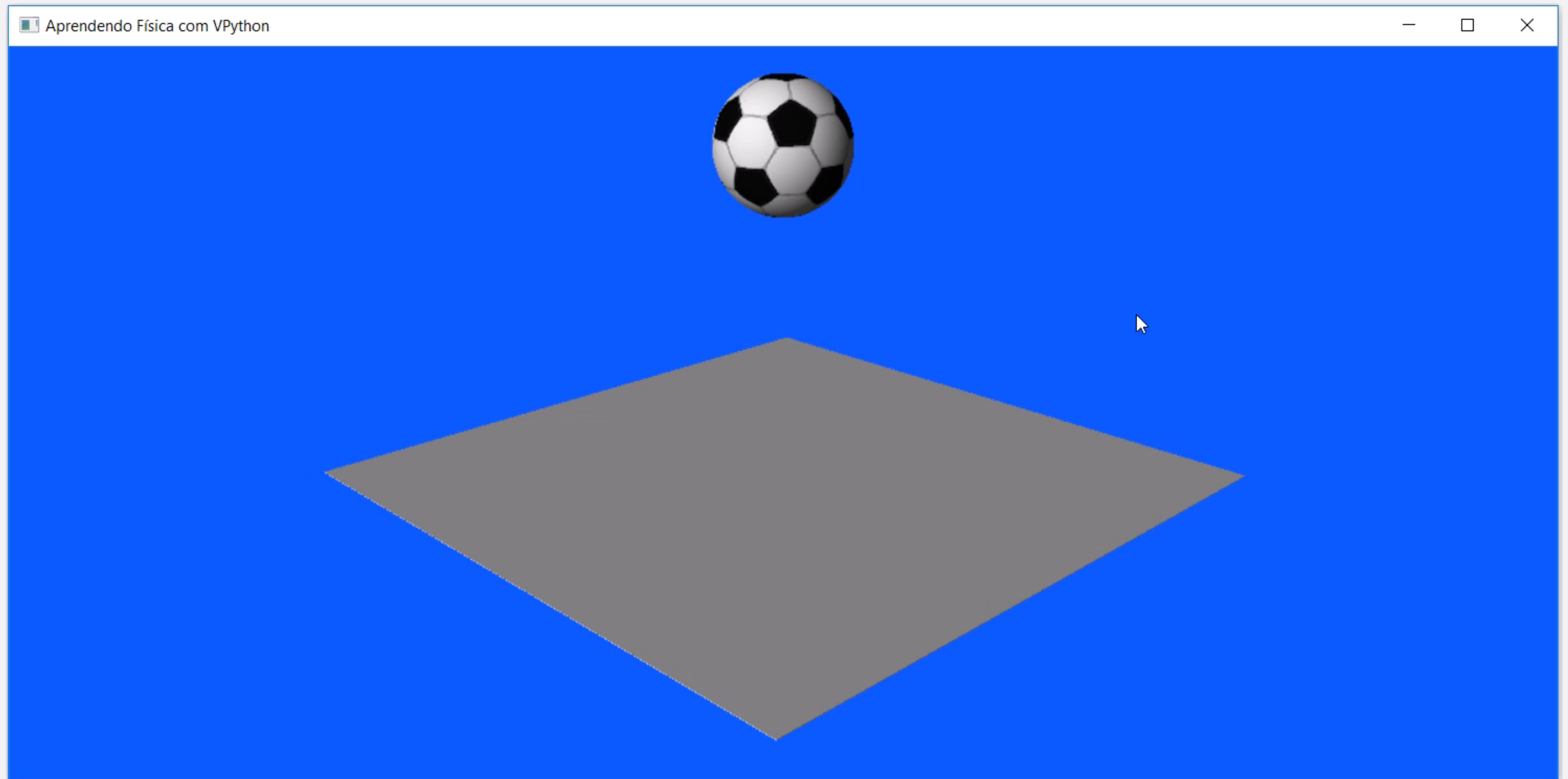
# Atualiza a posição da bola
# s = s0 + vt - 0.5at^2
bola.pos += bola.velocidade * dt -
0.5 * aceleracao * dt**2

# Adiciona uma rotação na bola
bola.rotate(angle = -0.02/4,
            axis = vector(0, 0, 1),
            origin = bola.pos)

# Verifica se a bola colidiu com o
# piso
if (bola.pos.y - bola.raio - 0.3) <=
piso.pos.y:
    bola.velocidade.y = -
bola.velocidade.y
```

VPython

Exemplos – Bola pulando



```
# Escala
escala = 10.0

# Cria o Sol
tex = materials.texture(data =
    materials.loadTGA('textura_sol.tga'),
    mapping = 'spherical',
    interpolate = False)

Sol = sphere(color = vector(1, 1, 0),
    pos = vector(0, 0, 0),
    radius = 109.0 / escala,
    material = tex,
    shininess = 10)

# Período de rotação do Sol (equatorial)
periodoSol = 157.08

# Período sideral da Terra = 365.256
omegaTerra = 2 * 3.14159 / 365.256
periodoTerra = 6.266
```

```
while True:

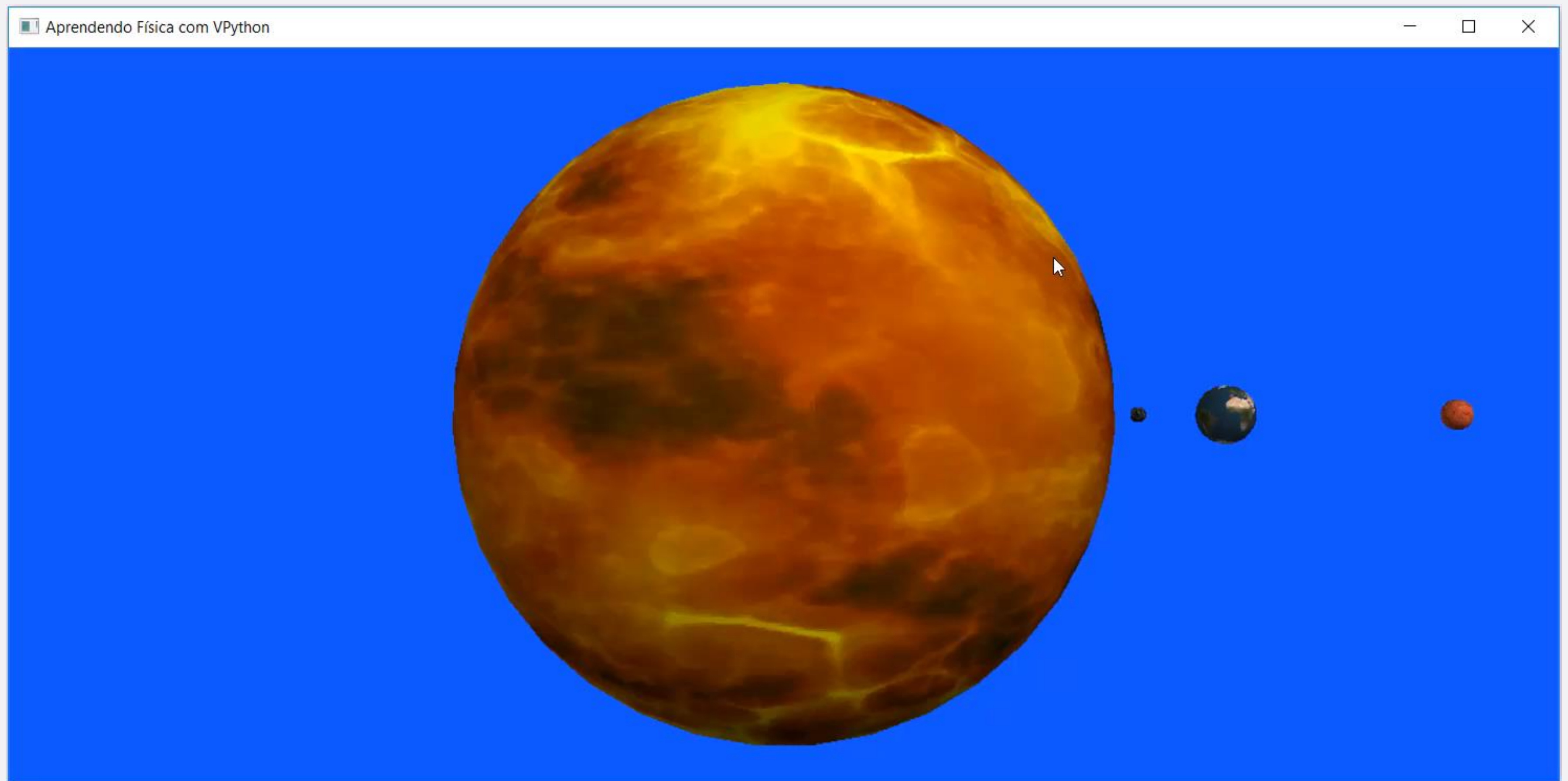
    rate(30)

    # Revolução da Terra
    Terra.rotate(angle = omegaTerra,
        axis = vector(0, 1, 0),
        origin = vector(0, 0, 0))

    # Rotação da Terra
    Terra.rotate(angle = periodoTerra,
        axis = vector(0, 1, 0),
        origin = Terra.pos)

    # Atualiza o ângulo de revolução da Lua
    anguloLua += omegaLua

    # Atualiza a posição e revolução da Lua
    Lua.pos = Terra.pos + vector(2, 0, 0)
    Lua.rotate(angle = anguloLua,
        axis = vector(0, 1, 0),
        origin = Terra.pos)
```

Exemplos – Sistema massa-mola

- Lei de Hooke

$$F = -kx$$

- 2ª Lei de Newton

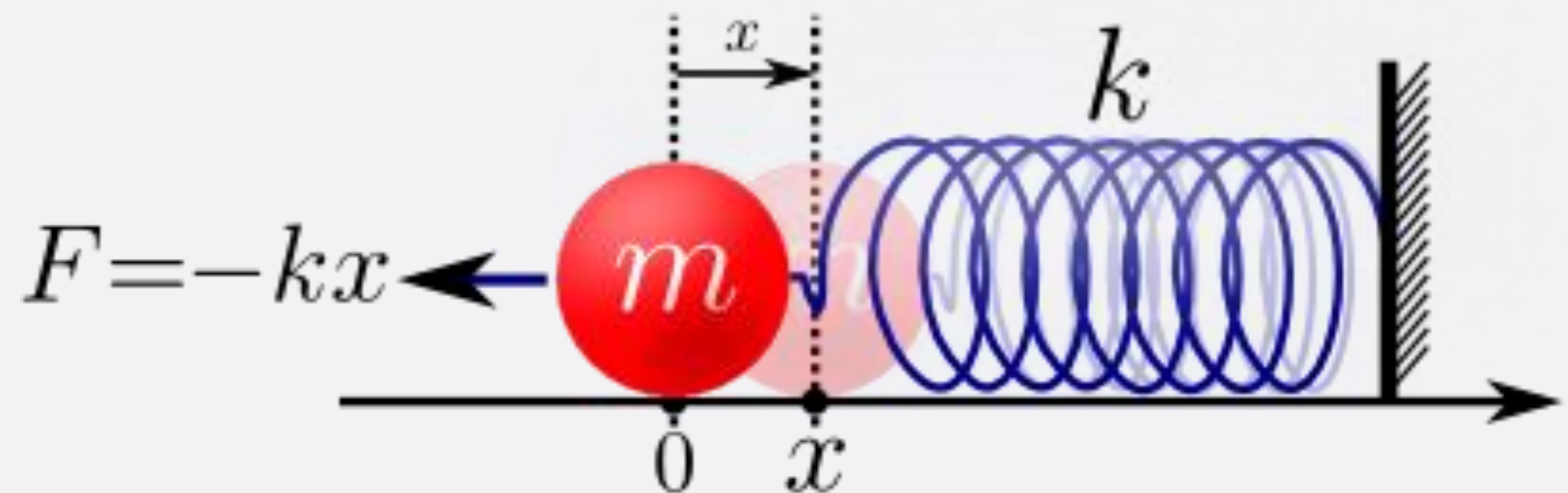
$$F = ma$$

- Calculando a aceleração

$$ma = -kx$$

$$a = -\frac{kx}{m} \rightarrow a = -\frac{k}{m} x$$

A **Lei de Hooke** estabelece que a força elástica F de uma mola é proporcional à sua distensão x , na direção oposta a esta. Se uma massa m está presa a uma das extremidades de uma mola horizontal e pode se mover livremente, e a outra extremidade da mola está fixa, então $F(t) = -kx(t)$, onde k é a constante elástica da mola. (Fonte: <https://goo.gl/mRMCDW>)



```
# Parâmetros iniciais
x_inicial = 3.0
x_deslocamento = 0.5
v_inicial = 0.0
m = 25
k = 15
t = 0.0
dt = 0.01

# Simulação
while True:

    rate(100)

    # Calcula o deslocamento
    delta_x = (x_inicial - x_deslocamento)

    # Calcula a aceleração
    a = -(k / m) * delta_x

    # Calcula a velocidade de deslocamento
    v_final = v_inicial + a * dt
```

```
# Calcula a posição
x_final = x_inicial + v_final * dt

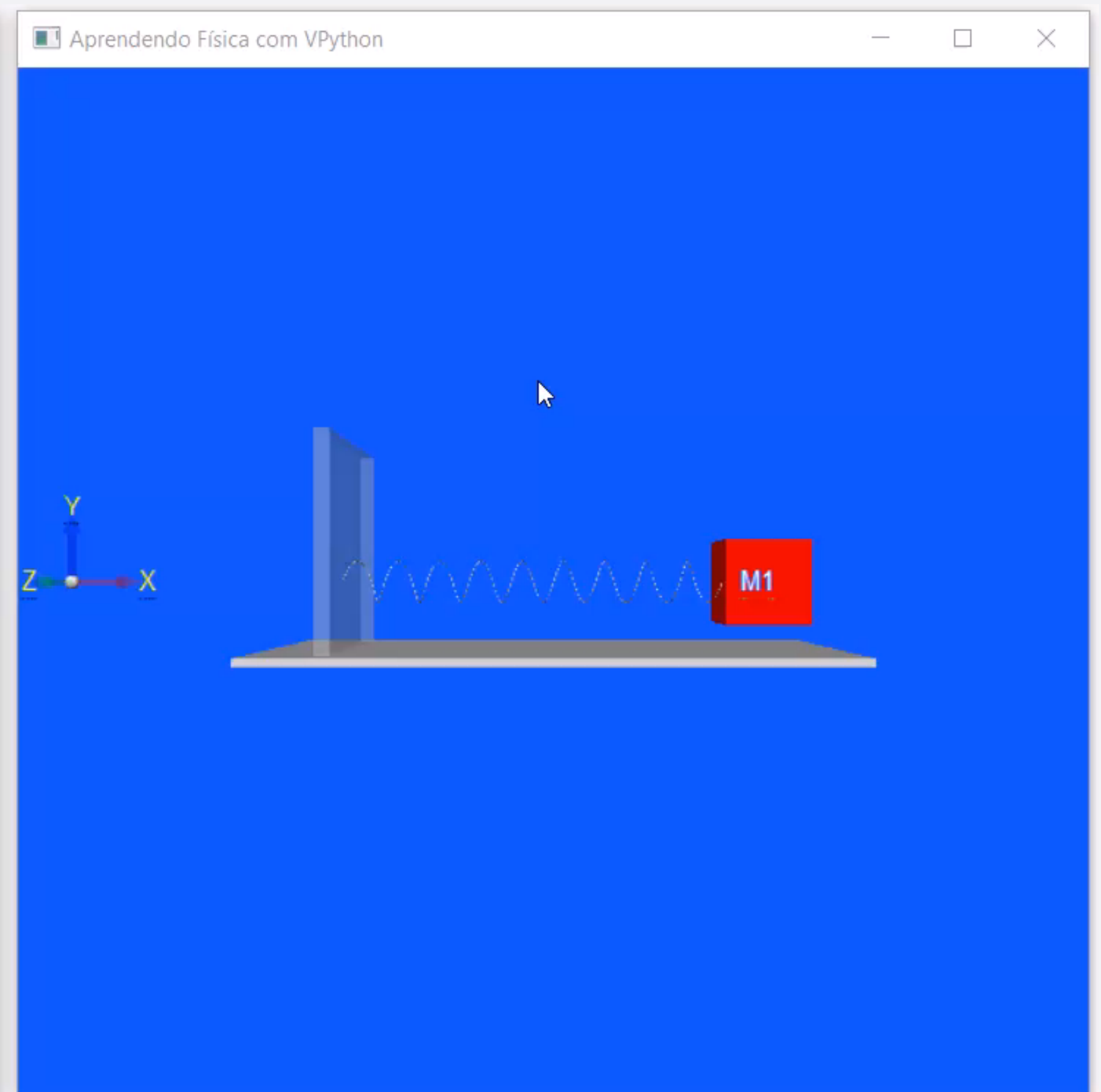
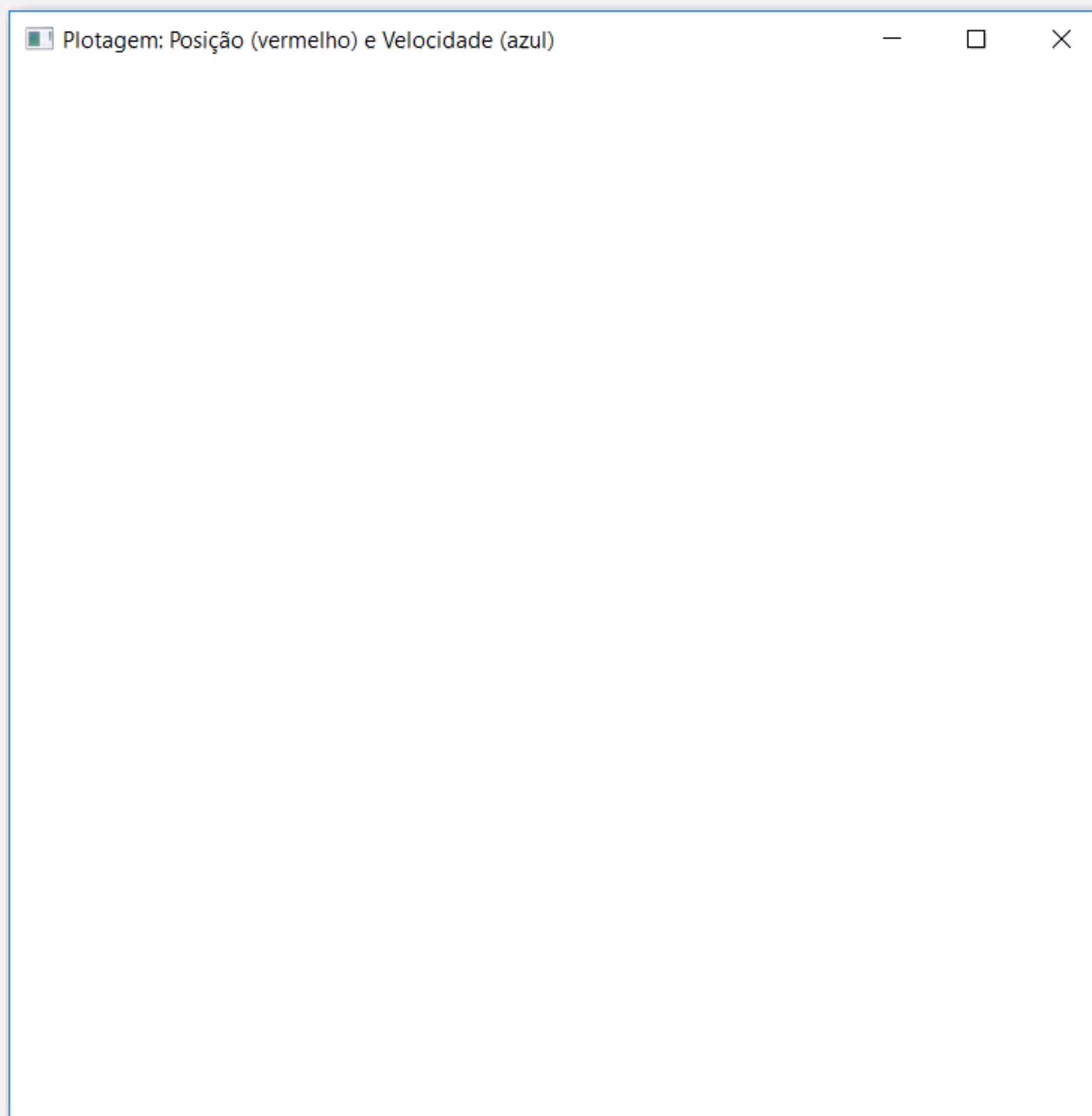
# Movimenta o cubo e a sua legenda
cubo.pos = (x_final, 0, 0)
cubo.label.pos = cubo.pos

# Atualiza o comprimento da mola
mola.length = cubo.pos.x - parede.pos.x

# Plota a posição e a velocidade
posicao.plot(pos = (t, x_final))
velocidade.plot(pos = (t, v_final))

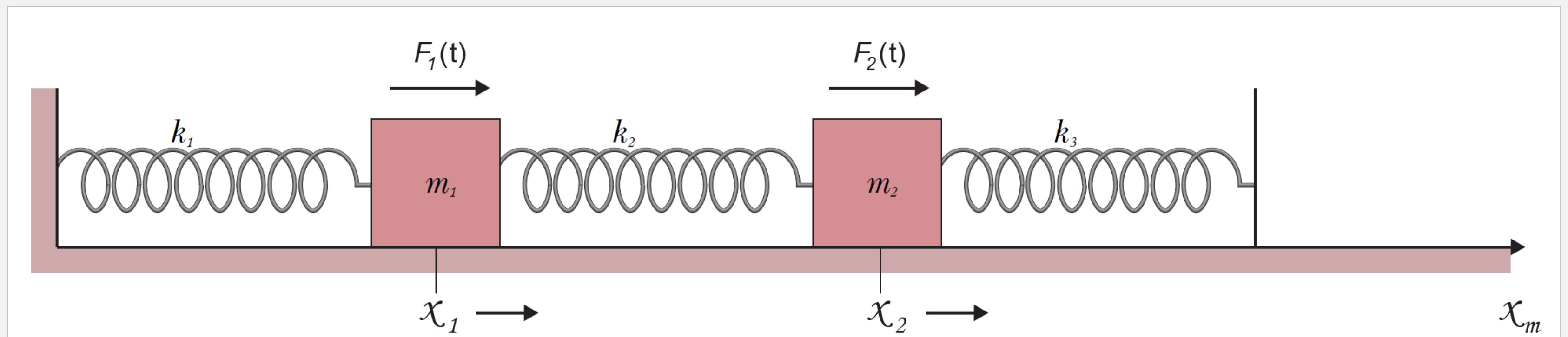
# Atualiza a posição e velocidade
x_inicial = x_final
v_inicial = v_final

# Atualiza o tempo
t = t + dt
```



- Oscilador harmônico acoplado

Existem casos onde o sistema oscilante é constituído por diversas partículas de massas m_1, m_2, \dots, m_n , acopladas por molas de constantes de forças k_1, k_2, \dots, k_n . Nessa situação, temos o que é denominado de oscilador harmônico acoplado:



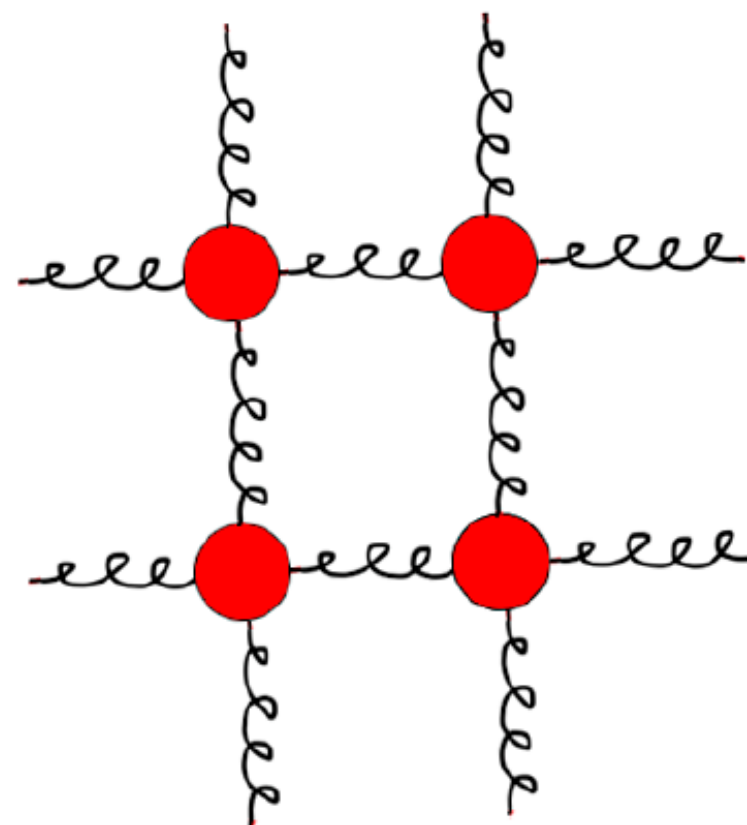
Exemplo de oscilador harmônico acoplado. Fonte: Adaptado de Walker et al (2014) e Oliveira (2008).

- Sistema de osciladores

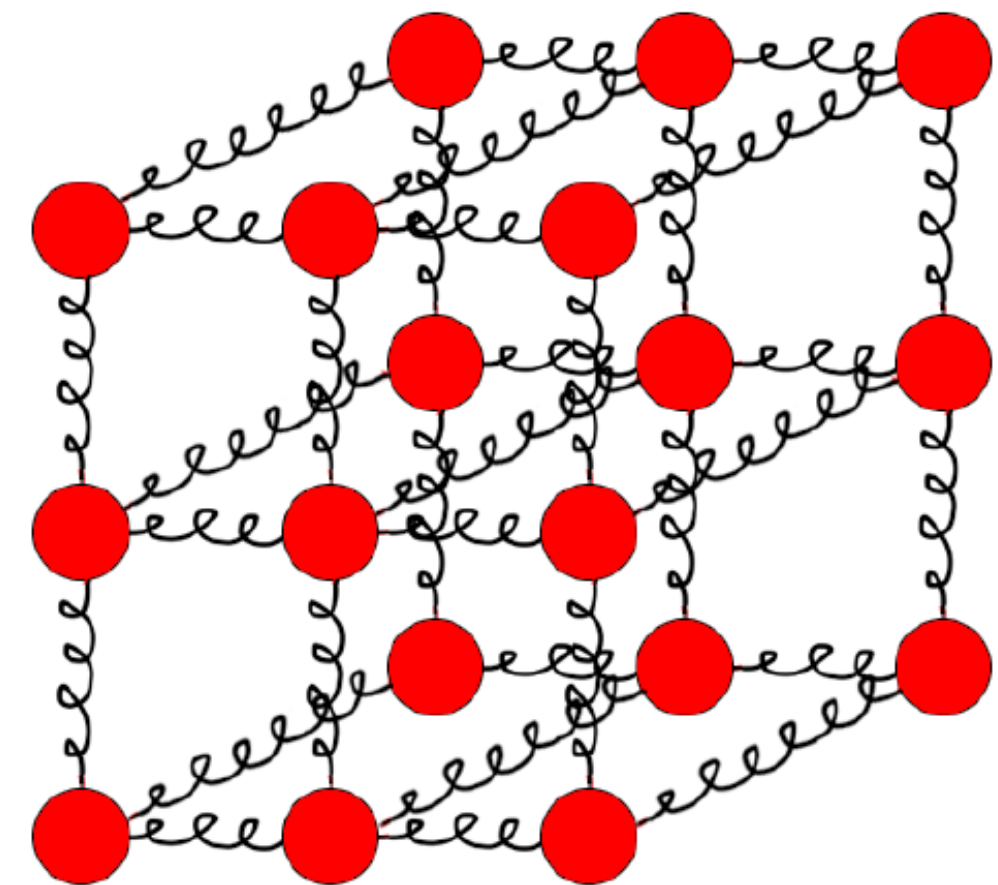
O sistema de osciladores acoplados do 3DBMO pode ser composto por elementos ligados entre si, configurados de acordo com a dimensão onde se deseja distribuir uma quantidade de elementos.



(a) Configuração em 1D



(b) Configuração em 2D

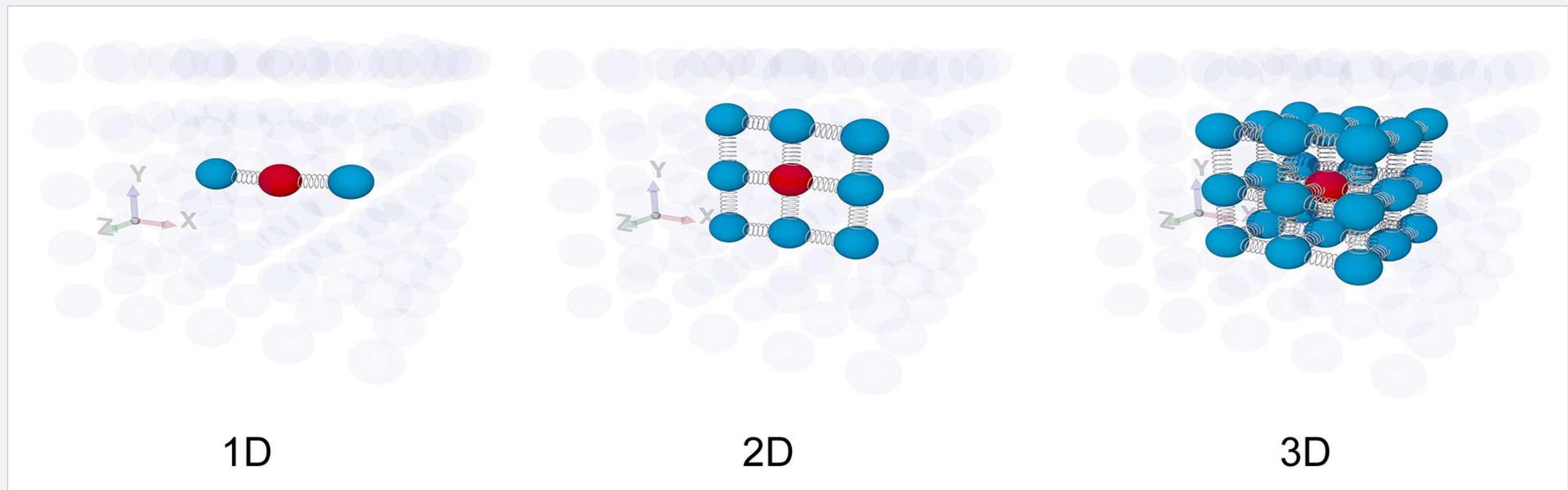


(c) Configuração em 3D

Exemplo de possíveis configurações para um sistema de osciladores acoplados.

- Implementação

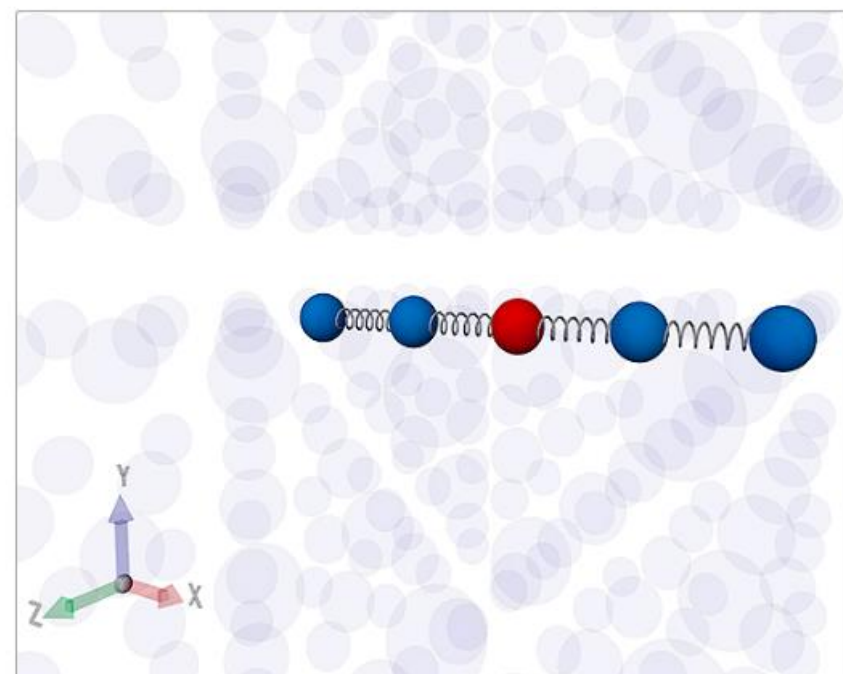
A implementação do 3DBMO utilizou o **wxPython**, o **VPython** e o módulo **KineticsKit**, desenvolvido por Markus Gritsch, em conjunto com algumas adaptações desenvolvidas para a criação das estruturas de osciladores e execução das simulações.



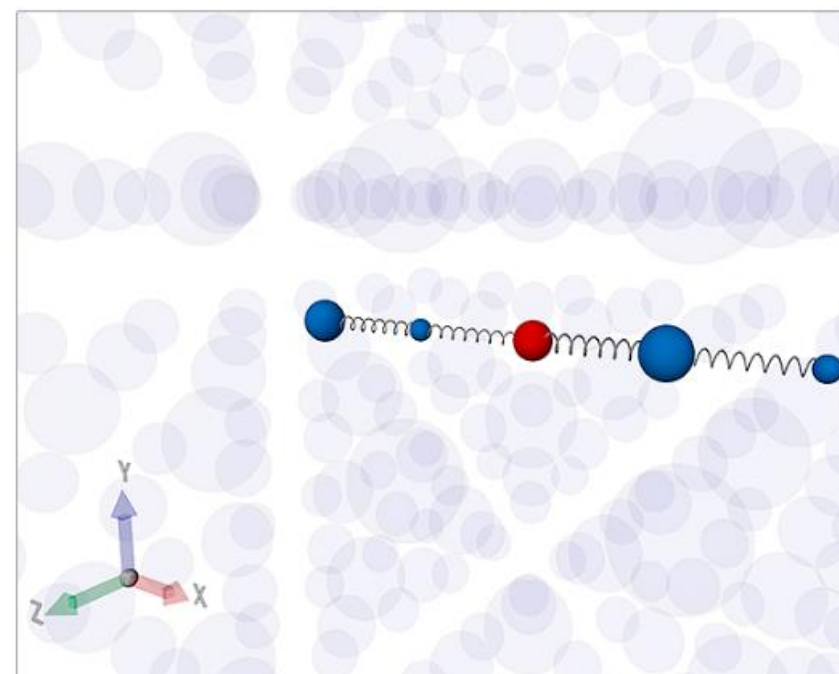
Exemplo de estruturas geradas pelo sistema 3DBMO.

3DBMO – 3D Big Mechanical Oscillator (<https://goo.gl/Vhi7NF>)

1D

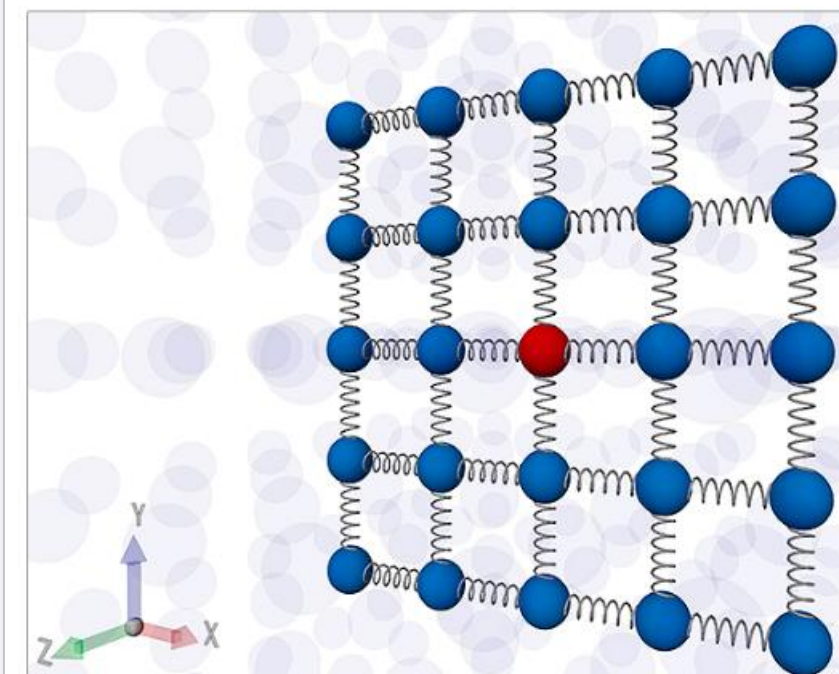


Homogênea

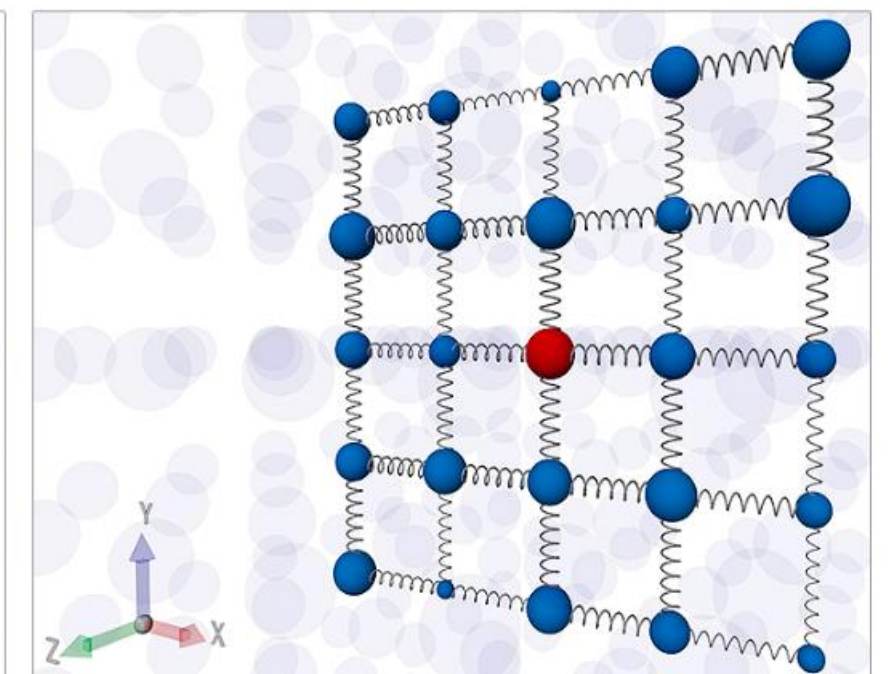


Heterogênea

2D

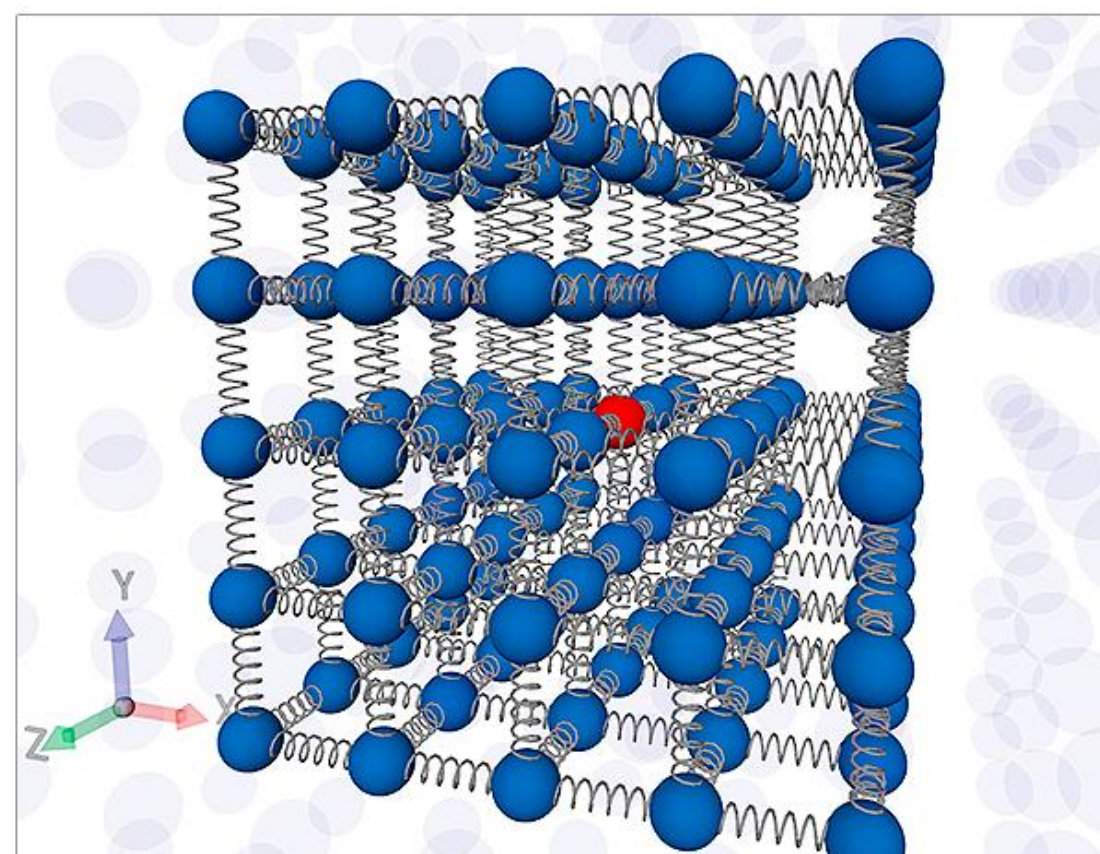


Homogênea

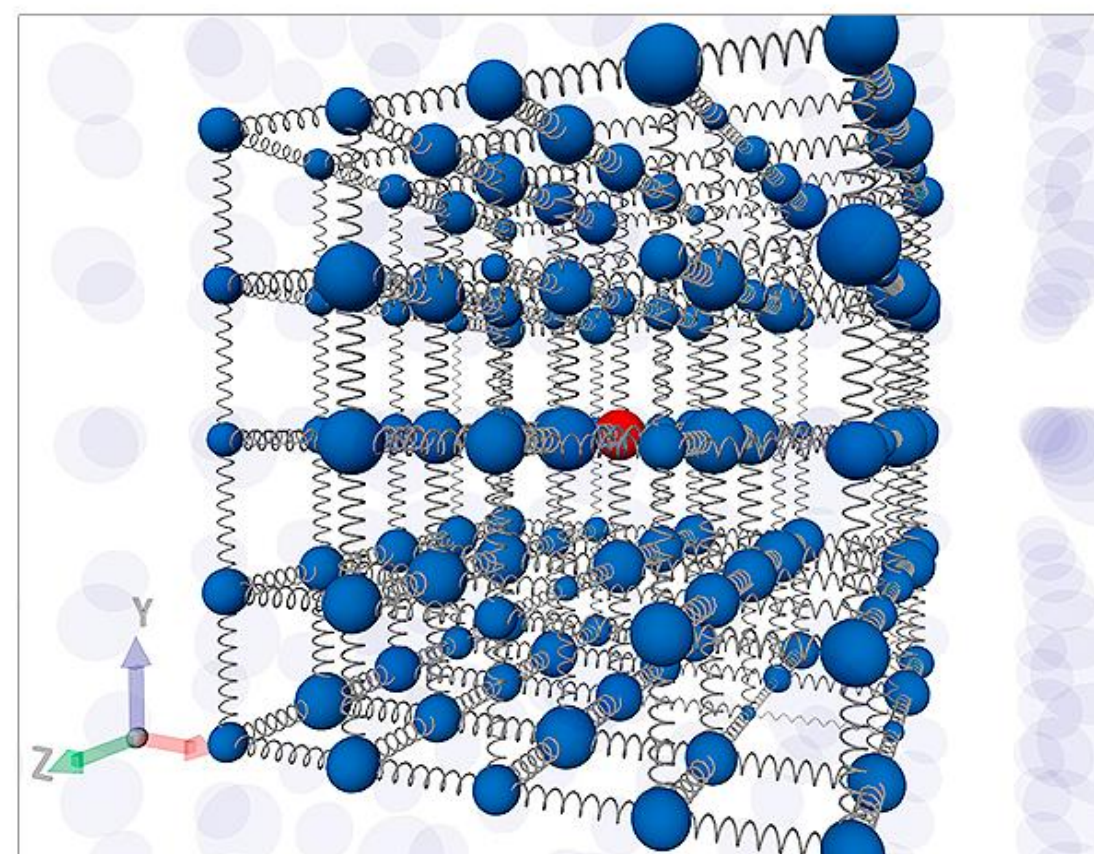


Heterogênea

3D

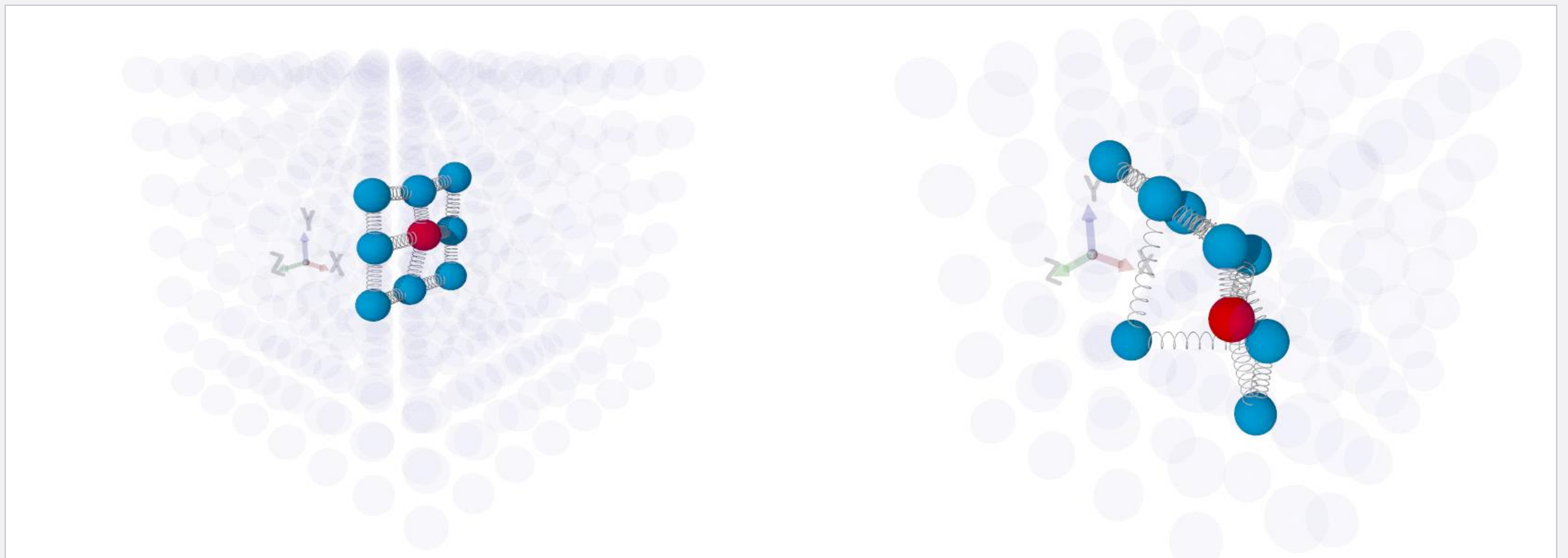


Homogênea



Heterogênea

- Animação dos snapshots gerados pelo POV-Ray.

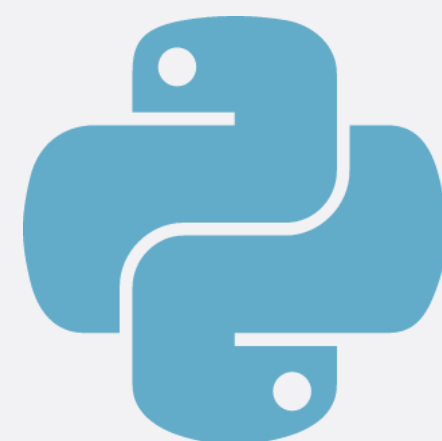


Exemplos de animação gerada através dos snapshots gerados pelo POV-Ray.

Python Shell
Python 2.7.9 (default, Dec 10 2014, 12:28:03) [MSC v.1500 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART
>>>
3DBMO: 3D Big Mechanical Oscillator
Setting system variables...

3DBMO - Configuration Manager
File Help
System Configuration
File Parameters
Filename to save images: Exemplo2 Amount of time steps: 65536 System dimension: 1D 2D 3D Simulation axes: X Y Z
Structure Configuration
Global Structure Parameters
Size: 5 Mass color: blue Mass opacity: 0.03
Local Structure Parameters
Size: 3 Mass color: real blue Mass opacity: 1.00 Show label
Mechanical Constraints
Distance between masses: 1 Mass selected for analysis: m14 Selected initial amplitude: X 0.50 Y 0.50 Z 0.50 Show plotting
Minimum mass value: 0.10 Maximum mass value: 1.00 Global mass value: 1.00 Mass type: Global Random Lock all other masses
Minimum spring constant: 0.00 Maximum spring constant: 1.00 Global spring constant: 1.00 Spring constant type: Global Random
Minimum spring damping: 0.00 Maximum spring damping: 1.00 Global spring damping: 0.00 Spring damping type: Global Random
System viscosity: 0.00 Selected mass color: red
Start Simulation Cancel

「Conclusões finais」



Obrigado!

Paulo Giovani

pg_faria@yahoo.com.br

<https://goo.gl/nbiPZo>