Eduardo Costa de Oliveira
PPGMMC
eduardo.oliveira@sou.unijui.edu.br

1 Proposição

Entregar um artigo em pdf de até 2 páginas mostrando a simulação, e o script da simulação.

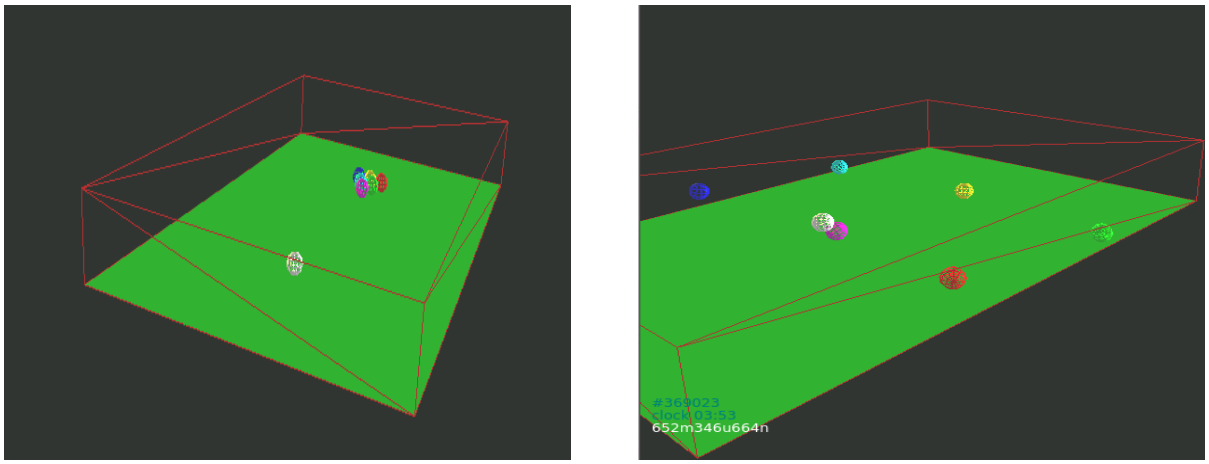
1.1 Resolução

O presente script em Yade simula o comportamento da fase inicial de um jogo de bilhar.

A esfera isolada recebe um impulso, na horizontal no sentido do eixo y, *função impulso*, colide-se com a primeira esfera do agrupamento em forma de triângulo realizando um choque elástico que transfere energia para as demais esferas até cessar a energia do sistema.

Através da função *addPlotData* foram coletados os contadores de posição e velocidade no eixo z das esferas 1 e 2, adicionalmente o módulo *energy* foi invocado para obter os contadores de energia total no sistema, com base nestas informações, é gravado um arquivo *dados.txt* e é gerado um gráfico, empregando o *pacote plot*, que é atualizado em tempo real e exibido durante a execução da simulação.

Para fins de reprodutibilidade, os algoritmos implementados estão disponıveis no repositório `https://github.com/OliveiraEdu/yade`.



(a) posição inicial (b) esferas em movimento

Figura 1: Simulação de mesa de bilhar

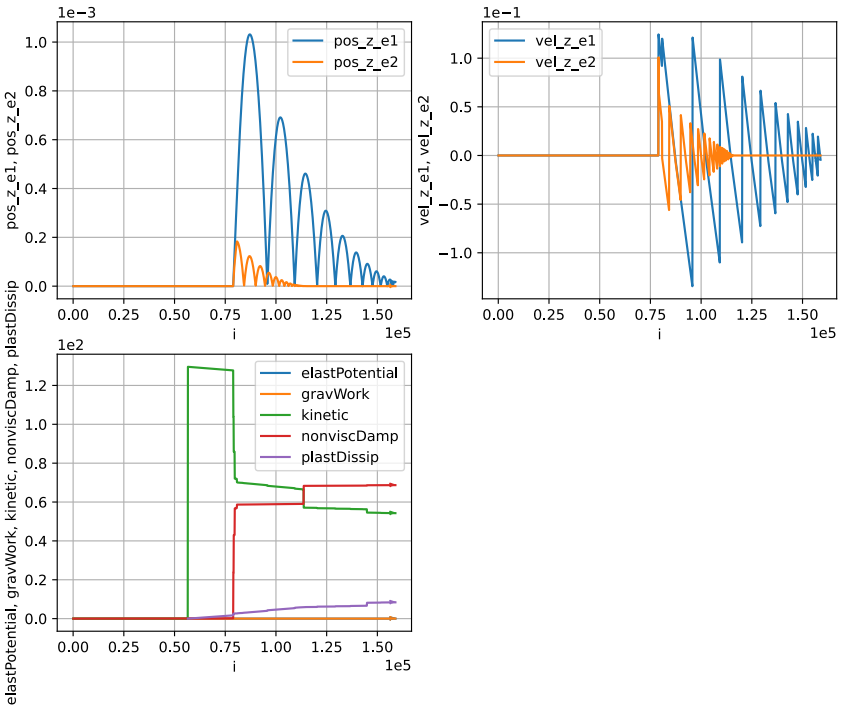


Figura 2: Gráficos gerados em tempo de simulação.

```
from yade import export
from yade import plot

#cria material f sico
mat_madeira1=CohFrictMat(density=550,young=1.1e11,poisson=0.25,frictionAngle
    =0.4712,label="mat1")

#cria esfera isolada
esfera1=sphere(center=(0,-1.25,0.05),color=(1,1,1),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)

#cria esferas em triangulo
esfera2=sphere(center=(0,0.03,0.05),color=(1,0,1),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)
esfera3=sphere(center=(-0.05,0.13,0.05),color=(0,1,1),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)
esfera4=sphere(center=(0.05,0.13,0.05),color=(0,1,0),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)
esfera5=sphere(center=(0,0.23,0.05),color=(1,1,0),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)
esfera6=sphere(center=(-0.10,0.23,0.05),color=(0,0,1),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)
esfera7=sphere(center=(0.10,0.23,0.05),color=(1,0,0),radius=0.05,material=
    mat_madeira1,fixed=False,wire=True)

#integra as esferas na simula  o
O.bodies.append(esfera1)
O.bodies.append(esfera2)
O.bodies.append(esfera3)
O.bodies.append(esfera4)
O.bodies.append(esfera5)
O.bodies.append(esfera6)
O.bodies.append(esfera7)

#define quatro pontos formando um quadrado no plano z=0
p1=[-1,1,0]
```

```
p2=[-1,-2,0]
p3=[1,-2,0]
p4=[1,1,0]

#define quatro pontos para forma  o das paredes
p1b=[-1,1,0.4]
p2b=[-1,-2,0.4]
p3b=[1,-2,0.4]
p4b=[1,1,0.4]

#Cria faces do chao
fchao=[]
f=facet([p1,p2,p4],wire=False,material=mat_madeira1,color=(0,1,0), fixed=True)
fchao.append(f)
f=facet([p2,p3,p4],wire=False,material=mat_madeira1,color=(0,1,0), fixed=True)
fchao.append(f)
f=facet([p1,p4,p4b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True)
fchao.append(f)
f=facet([p1,p4b,p1b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True
    )
fchao.append(f)
f=facet([p2,p1,p2b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True)
fchao.append(f)
f=facet([p2b,p1,p1b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True
    )
fchao.append(f)
f=facet([p2,p3,p2b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True)
fchao.append(f)
f=facet([p2b,p3,p3b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True
    )
fchao.append(f)
f=facet([p3,p4,p3b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True)
fchao.append(f)
f=facet([p3b,p4,p4b],wire=True,material=mat_madeira1,color=(1,0,0), fixed=True
    )
fchao.append(f)

#Adiciona as faces na simula  o
O.bodies.append(fchao)

#O.engines contem as os motores da simulacao
O.engines=[
#reinicia as forcas
ForceResetter(),
#motor que analisa os contatos
InsertionSortCollider([Bo1_Sphere_Aabb(),Bo1_Facet_Aabb()]),
# iteracao ptinciais
InteractionLoop(
#modelo de colisao esfera-esfera
[Ig2_Sphere_Sphere_ScGeom(), Ig2_Facet_Sphere_ScGeom()],
#material fisico
[Ip2_FrictMat_FrictMat_FrictPhys()],
#modelo de contato inear
[Law2_ScGeom_FrictPhys_CundallStrack()]
),
#integracao das forcas
#damping eh o fator de amortecimento
NewtonIntegrator(gravity=(0,0,-9.81),damping=0.1),
PyRunner(command='impulso()',virtPeriod=0.1),
PyRunner(command='addPlotData()',iterPeriod=100)
]
#calcula o tamanho do passo de tempo
O.dt=(0.5*PWaveTimeStep())

# salva o estado da simulacao para reiniciar de necessario
O.saveTmp()

```

```
#Habilita o contador das m tricas de energia
O.trackEnergy=True

#Gera impulso para a esfera isolada
def impulso():
    e = O.bodies[0]
    e.state.vel[1]=30.0
    O.engines[4].dead=True
#Define dados para plots
def addPlotData():
    plot.addData(i=O.iter,pos_z_e1=esfera1.state.pos[2]-esfera1.state.refPos
        [2],pos_z_e2=esfera2.state.pos[2]-esfera2.state.refPos[2],vel_z_e1=
        esfera1.state.vel[2], vel_z_e2=esfera2.state.vel[2],**O.energy)

#N mero de itera  es para iniciar os plots
O.run(1000,True)

#Salva arquivo com os dados de addPlotData
plot.saveDataTxt('dados.txt')

#plota os dados de posi  o , velocidade e energias
plot.plots={'i':('pos_z_e1','pos_z_e2'),
            'i ':('vel_z_e1','vel_z_e2'),
            'i  ':(O.energy.keys)
            }

plot.plot()
```

Listing 1: Código da simulação