Eduardo Mintzias (eam2285)
Pol Bernat Belenguer (pb2750)

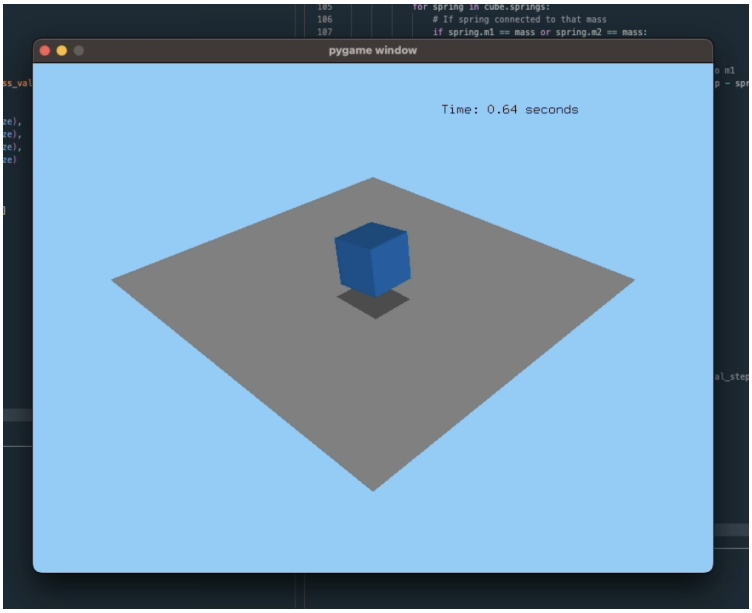
MECS 4510: EVOLUTIONARY COMPUTATION AND DESIGN AUTOMATION

Professor Hod Lipson

Date Submitted: 11/01/2023
Grace Hours Used: 5
Grace Hours Gained: 1
Grace Hours Remaining: 91

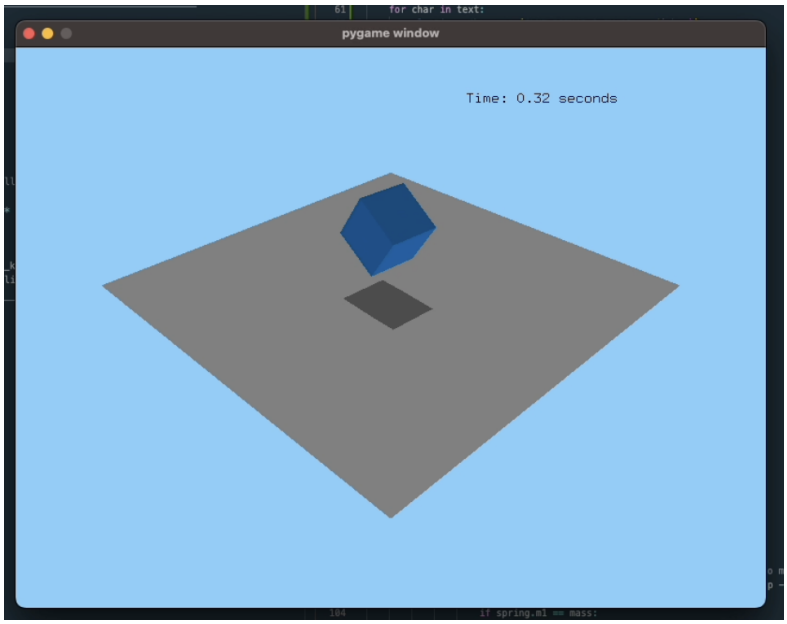
SUMMARY

VIDEO OF BOUNCING CUBE



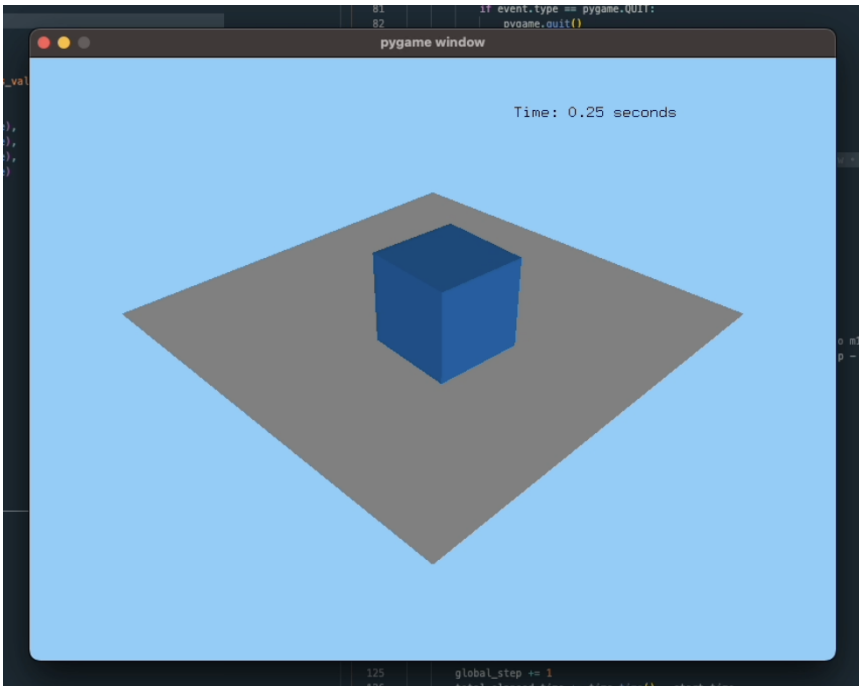
https://youtu.be/bHXIRVgKo48

CUBE LAUNCHED WITH SLIGHT SPIN



https://youtu.be/-UvjM3jd2VA

VIDEO OF BREATHING CUBE



https://youtu.be/Ugb-0O5pegw

METHODS

DESCRIPTION OF ALL SIMULATION PARAMETERS PROVIDED

Global Variables:

Gravity (g): A gravitational acceleration vector [0, 0, −9.81] m/s². It acts in the negative z-direction, simulating the effect of gravity pulling masses downwards.

Time-step (dt): The simulation advances by 0.0001 seconds for every iteration. This dictates the granularity of the simulation's progression through time.

Global Time (T): A counter that keeps track of the total elapsed time since the start of the simulation. Starts at 0 seconds.

Global Step: Counter for the number of iterations the simulation has gone through.

Total Elapsed Time: Keeps track of the total computation time the simulation has consumed. It's used for performance measurement.

Ground Force Constant (kc): A constant value of 100000 that determines the reaction force when a mass collides with the ground. It ensures the mass doesn't penetrate the ground and simulates a very hard surface.

Damping Constant (b): With a value of 0.999, it represents a very mild damping to the motion. Multiplying the velocities by this factor on every iteration simulates effects like air resistance, preventing the simulation from running indefinitely.

Cube Initialization:

Spring Constant (k_value): The cube is initialized with springs having a spring constant of 9000. This determines how stiff the springs are, i.e., how much force they exert when stretched or compressed.
Visual Representation:

Shadow: Represented using a dark gray color (0.3, 0.3, 0.3). The shadow's bounds are computed based on the faces of the cube for z=0.00001 (so that it appears above the ground plane).

Ground: The ground is represented as a gray-colored quad at z=0 (true z=0).

Simulation Loop:

Spring Force Calculation: For each mass in the cube, the spring forces exerted by all springs connected to that mass are computed and summed. The force depends on the current length of the spring, its rest length, and the spring constant.

Gravitational Force: For each mass, a gravitational force is added, determined by multiplying the mass of the object with the gravitational acceleration.

Ground Collision: If any mass's z-coordinate goes below zero (indicating a collision with the ground), a reactive force based on the ground force constant is applied upwards.

Acceleration, Velocity, and Position Updates: Using Newton's second law, the acceleration is computed from the resultant force. The velocity is then updated considering the acceleration and dampening. Finally, the position of each mass is updated based on its velocity.

Rendering Interval: The scene (ground, cube, and time data) is rendered every 20 iterations of the simulation loop.

DESCRIPTION OF ALL BREATHING PARAMETERS PROVIDED

For the ‘breathing’ function of our cube we used a sine wave with amplitude 0.00005 (L0 = 0.1, so it doesn’t seem to have a huge effect but the spring forces caused by the changing L0 showed that this was a good value).

We used the value global_step*0.001 (equivalent to T*10) for the sine wave since, by trial and error, we determined this to be a good frequency for the values to alternate between positive and negative - thus giving it the ‘breathing’ effect.

NUMBER OF SPRING EVALS/SECOND

~5,000 (on my personal laptop pre-parallelization, running python)

BIBLIOGRAPHY

ChatGPT

PyOpenGL - https://pyopengl.sourceforge.net/index.html

Stack Overflow

APPENDIX

```
import timeit
from tqdm import tqdm
import numpy as np
from numpy.linalg import norm
import seaborn as sns
import matplotlib.pyplot as plt
from collections import defaultdict
import sys
import pdb
import math as m
import pandas as pd
import time
import threading
import datetime
import pickle
import os
# pyOpenGL
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
# matplotlib animation
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from matplotlib.animation import FuncAnimation
from matplotlib.patches import Rectangle
```

```
#%%
from Libraries import *

# %%
# Mass class
class Mass:
def __init__(self, m, p, p_0 = np.zeros(3)):
self.m = m
self.p = np.array(p) + p_0
self.v = np.array([0, 0, 0], dtype=np.float64)
self.a = np.array([0, 0, 0], dtype=np.float64)

# Spring class
class Spring:
def __init__(self, m1, m2, k):
self.m1 = m1
self.m2 = m2
self.k = k
self.L0 = np.linalg.norm(m1.p - m2.p)

# Cube class
class Cube:
def __init__(self, cube_size=0.1, p_0= np.dot([0, 0, 1], 0.0), mass_value=0.1,
k_value=10000):
# Initialize vertices
vertices = [
np.dot([0, 0, 0.], cube_size), np.dot([1, 0, 0.], cube_size),
np.dot([1, 1, 0.], cube_size), np.dot([0, 1, 0.], cube_size),
np.dot([0, 0, 1.], cube_size), np.dot([1, 0, 1.], cube_size),
np.dot([1, 1, 1.], cube_size), np.dot([0, 1, 1.], cube_size)
]

# Initialize masses
self.masses = [Mass(mass_value, v, p_0=p_0) for v in vertices]

# Initialize springs
self.springs = []
for i, m1 in enumerate(self.masses):
for j, m2 in enumerate(self.masses[i + 1:]):
self.springs.append(Spring(m1, m2, k_value))

# Initialize faces for PyOpenGL plotting
```

```
self.faces = [
[self.masses[i] for i in [0, 1, 2, 3]],
[self.masses[i] for i in [4, 5, 6, 7]],
[self.masses[i] for i in [0, 4, 7, 3]],
[self.masses[i] for i in [1, 5, 6, 2]],
[self.masses[i] for i in [0, 1, 5, 4]],
[self.masses[i] for i in [3, 2, 6, 7]]
]
```

```
#%%
from Libraries import *
from Datastructures import Cube


# Global Variables
g = np.array([0, 0, -9.81]) # Gravity
dt = 0.0001 # Time-step
T = 0 # Global time variable
global_step = 0
total_elapsed_time = 0
kc = 100000 # Ground force constant
b = 0.999 # Dampening constant

# Initialize cube
cube = Cube(k_value=9000)

def draw_cube_faces(cube):
base_color = (3/255, 148/255, 252/255) # Blue color
glBegin(GL_QUADS)
for i, face in enumerate(cube.faces):
color_factor = 0.4 + (i + 1)*0.05 # '+1' to avoid zero multiplier for the first face
glColor3f(base_color[0]*color_factor, base_color[1]*color_factor,
base_color[2]*color_factor)
for mass in face:
glVertex3fv(mass.p)
glEnd()

def draw_shadow(cube):
glColor3f(0.3, 0.3, 0.3) # Dark gray color for shadow
glBegin(GL_QUADS)
for i, face in enumerate(cube.faces):
```

```
for mass in face:
glVertex3fv([mass.p[0], mass.p[1], 0.00001])
glEnd()

def draw_cube(cube):
# Draw shadow first
draw_shadow(cube)
# Then draw the cube's faces
draw_cube_faces(cube)

def draw_ground():
glColor3f(0.5, 0.5, 0.5)
glBegin(GL_QUADS)
glVertex3f(-0.3, -0.3, 0)
glVertex3f(0.5, -0.3, 0)
glVertex3f(0.5, 0.5, 0)
glVertex3f(-0.3, 0.5, 0)
glEnd()

def render_text(x, y, text):
glMatrixMode(GL_PROJECTION)
glPushMatrix()
glLoadIdentity()
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0)
glMatrixMode(GL_MODELVIEW)
glPushMatrix()
glLoadIdentity()
glColor3f(0, 0, 0)
glRasterPos2f(x, y)
for char in text:
glutBitmapCharacter(GLUT_BITMAP_9_BY_15, ord(char))
glPopMatrix()
glMatrixMode(GL_PROJECTION)
glPopMatrix()
glMatrixMode(GL_MODELVIEW)


def main(cube):
pygame.init()
display = (800,600)
pygame.display.set_mode(display, DOUBLEBUF|OPENGL)
gluPerspective(45, (display[0]/display[1]), 0.1, 100.0)
```

```
gluLookAt(-0.7, -0.7, 0.7, 0, 0, 0, 0, 0, 1)
glClearColor(0.53, 0.81, 0.98, 1)
glDisable(GL_CULL_FACE)
glEnable(GL_DEPTH_TEST)

while True:
for event in pygame.event.get():
if event.type == pygame.QUIT:
pygame.quit()
return
global T, global_step, total_elapsed_time
# Loop over all masses
start_time = time.time()
# Breathing cube ?
for spring in cube.springs:
print(0.005*np.sin(global_step*0.001))
spring.L0 += 0.00005*np.sin(global_step*0.001)

for i, mass in enumerate(cube.masses):
# Initial force is 0
F = np.zeros(3)
# SPRING FORCE
# Loop over all springs
for spring in cube.springs:
# If spring connected to that mass
if spring.m1 == mass or spring.m2 == mass:
# Calculate spring force
L = np.linalg.norm(spring.m1.p - spring.m2.p)
# Update F_spring in the vector direction from m2 to m1
F_spring = spring.k * (L - spring.L0) * (spring.m1.p - spring.m2.p) / L
# Update F with appropriate sign
if spring.m1 == mass:
F -= F_spring
else:
F += F_spring
# GRAVITATIONAL FORCE
# Update F
F += mass.m * g
# GROUND COLLISION FORCE
# Ground collision check and response
if mass.p[2] < 0:
F += np.array([0, 0, -kc * mass.p[2]])
```

```
# UPDATE ACCELERATION
mass.a = F / mass.m
# UPDATE VELOCITY
mass.v += mass.a * dt
mass.v *= b
# UPDATE POSITION
mass.p += mass.v * dt
T += dt
global_step += 1
total_elapsed_time += time.time() - start_time
#print(f"Each update loop takes on avg {total_elapsed_time/global_step:.6f} seconds")
if global_step%20 == 1:
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
draw_ground()
draw_cube(cube)
# Convert the time to a string
text_string = f"Time: {T:.2f} seconds"
# Render the text in the top-right corner using GLUT
render_text(0.6, 0.9, text_string)
pygame.display.flip()
pygame.time.wait(10)

main(cube)
```