**Sprawozdanie**

Z przedmiotu pt. „Matematyczne Metody w Grafice Komputerowej”

z laboratorium

Piotr Baranowski 253463

Grupa SAPJEE

Informatyka – 2 stopień - zaoczne - sem.2 rok 2024

# Zad 3 – Modelowanie rzeczywistych obiektów i zjawisk

Zadanie. Zamodelować ruch sprężyny oraz kuli połączonych według zamieszczonego poniżej rysunku z uwzględnieniem praw fizyki i ich rzeczywistego wyglądu

Dodatkowe założenia:

* Kula wykonana z drewna/szkła
* Sprężyna wykonana z drutu stalowego
* W początkowej chwili sprężyna jest naciągnięta
* Zakładamy zerową prędkość początkowa kuli.
* Model w pełni 3-D, zastosować tekstury do modelowania powierzchni.
* Powierzchnia sprężyny zamodelowana na podstawie wzoru:

Obraz zawierający Czcionka, tekst, pismo odręczne, kaligrafia

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, wizytówka, diagram, design

Opis wygenerowany automatycznie

Rysunek 1 Zamodelowana kula ze sprężyną

* Uwzględnić górne i dolne wykończenie sprężyny (sfera- cylinder-sfera-cylinder).

Kod programu:

import math

import numpy as np

import pygame

from OpenGL.GL import \*

from OpenGL.GLU import \*

from OpenGL.GLUT import \*

from PIL import Image

from pygame.locals import \*

vertex\_points = (

(50, -2, -50),

(50, 2, -50),

(-50, 2, -50),

(-50, -2, -50),

(50, -2, 50),

(50, 2, 50),

(-50, -2, 50),

(-50, 2, 50)

)

surface\_faces = (

(0, 1, 2, 3),

(3, 2, 7, 6),

(6, 7, 5, 4),

(4, 5, 1, 0),

(1, 5, 7, 2),

(4, 0, 3, 6)

)

def texture\_init(image\_file):

img = Image.open(image\_file)

img\_data = np.array(list(img.getdata()), dtype=np.uint8)

texture\_id = glGenTextures(1)

glBindTexture(GL\_TEXTURE\_2D, texture\_id)

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST)

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST)

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, img.size[0], img.size[1], 0, GL\_RGB, GL\_UNSIGNED\_BYTE, img\_data)

return texture\_id

def create\_wall():

glBindTexture(GL\_TEXTURE\_2D, texture\_dict["wall"])

glEnable(GL\_TEXTURE\_2D)

for surface in surface\_faces:

glBegin(GL\_QUADS)

index = 0

for vertex in surface:

if index == 0:

x\_coord = -1.0

y\_coord = -1.0

elif index == 1:

x\_coord = 1.0

y\_coord = -1.0

elif index == 2:

x\_coord = 0.0

y\_coord = 0.0

else:

x\_coord = -1.0

y\_coord = 0.0

glTexCoord2f(x\_coord, y\_coord)

glVertex3fv(vertex\_points[vertex])

index += 1

glEnd()

def draw\_cylinder(radius, height, slices, angle, x\_axis, y\_axis, z\_axis, tx, ty, tz):

glBindTexture(GL\_TEXTURE\_2D, texture\_dict["metal"])

glEnable(GL\_TEXTURE\_2D)

glPushMatrix()

glTranslatef(tx, ty, tz)

glRotatef(angle, x\_axis, y\_axis, z\_axis)

# sides

glBegin(GL\_QUAD\_STRIP)

for i in range(slices + 1):

angle = 2 \* math.pi \* i / slices

x = radius \* math.cos(angle)

y = radius \* math.sin(angle)

glTexCoord2f(i / slices, 0)

glVertex3f(x, y, -height / 2)

glTexCoord2f(i / slices, 1)

glVertex3f(x, y, height / 2)

glEnd()

# top

glBegin(GL\_TRIANGLE\_FAN)

glTexCoord2f(0.5, 0.5)

glVertex3f(0, 0, height / 2)

for i in range(slices + 1):

angle = 2 \* math.pi \* i / slices

x = radius \* math.cos(angle)

y = radius \* math.sin(angle)

glTexCoord2f((math.cos(angle) + 1) / 2, (math.sin(angle) + 1) / 2)

glVertex3f(x, y, height / 2)

glEnd()

# bottom

glBegin(GL\_TRIANGLE\_FAN)

glTexCoord2f(0.5, 0.5)

glVertex3f(0, 0, -height / 2)

for i in range(slices + 1):

angle = 2 \* math.pi \* i / slices

x = radius \* math.cos(angle)

y = radius \* math.sin(angle)

glTexCoord2f((math.cos(angle) + 1) / 2, (math.sin(angle) + 1) / 2)

glVertex3f(x, y, -height / 2)

glEnd()

glPopMatrix()

def generate\_spring\_points(scale\_factor):

t = np.linspace(0, 8 \* np.pi, 100)

u = np.linspace(0, 2 \* np.pi, 100)

t, u = np.meshgrid(t, u)

x = np.cos(t) \* (3 + np.cos(u))

y = np.sin(t) \* (3 + np.cos(u))

z = scale\_factor \* (0.6 \* t + np.sin(u))

return x, y, z

def draw\_spring(X, Y, Z):

glPushMatrix()

glRotatef(90, 1, 0, 0)

glTranslatef(0, 0, 6)

glEnable(GL\_TEXTURE\_2D)

glBindTexture(GL\_TEXTURE\_2D, texture\_dict["metal"])

glBegin(GL\_TRIANGLES)

for i in range(X.shape[0] - 1):

for j in range(X.shape[1] - 1):

# Vertex 1

glTexCoord2f(i / X.shape[0], j / X.shape[1])

glVertex3f(X[i, j], Y[i, j], Z[i, j])

# Vertex 2

glTexCoord2f((i + 1) / X.shape[0], j / X.shape[1])

glVertex3f(X[i + 1, j], Y[i + 1, j], Z[i + 1, j])

# Vertex 3

glTexCoord2f(i / X.shape[0], (j + 1) / X.shape[1])

glVertex3f(X[i, j + 1], Y[i, j + 1], Z[i, j + 1])

# Vertex 2

glTexCoord2f((i + 1) / X.shape[0], j / X.shape[1])

glVertex3f(X[i + 1, j], Y[i + 1, j], Z[i + 1, j])

# Vertex 3

glTexCoord2f((i + 1) / X.shape[0], (j + 1) / X.shape[1])

glVertex3f(X[i + 1, j + 1], Y[i + 1, j + 1], Z[i + 1, j + 1])

# Vertex 4

glTexCoord2f(i / X.shape[0], (j + 1) / X.shape[1])

glVertex3f(X[i, j + 1], Y[i, j + 1], Z[i, j + 1])

glEnd()

glDisable(GL\_TEXTURE\_2D)

glPopMatrix()

def draw\_sphere(radius, slices, stacks, tx, ty, tz):

glBindTexture(GL\_TEXTURE\_2D, texture\_dict["wood"])

glEnable(GL\_TEXTURE\_2D)

quadric = gluNewQuadric()

gluQuadricTexture(quadric, GL\_TRUE)

glPushMatrix()

glTranslatef(tx, ty, tz)

gluSphere(quadric, radius, slices, stacks)

glPopMatrix()

gluDeleteQuadric(quadric)

def render\_scene():

glClearColor(0, 0, 0, 1)

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glPushMatrix()

create\_wall()

draw\_cylinder(1.0, 5.0, 20, 90, 1, 0, 0, 0, -4, 0)

draw\_cylinder(1.0, 2.5, 20, 90, 0, 1, 0, 1.5, -6, 0)

X, Y, Z = generate\_spring\_points(scale\_factor)

draw\_spring(X, Y, Z)

end\_z = Z[-1, -1]

draw\_cylinder(1.0, 2.5, 20, 90, 0, 1, 0, 1.5, -end\_z-6, 0)

draw\_cylinder(1.0, 5.0, 20, 90, 1, 0, 0, 0, -end\_z - 8, 0)

draw\_sphere(5.0, 20, 20, 0, -end\_z - 8 - 5, 0)

glPopMatrix()

pygame.init()

display = (800, 600)

pygame.display.set\_mode(display, DOUBLEBUF | OPENGL)

gluPerspective(45, (display[0] / display[1]), 1, 200.0)

glTranslatef(0.0, 30.0, -100)

glDrawBuffer(GL\_BACK)

glEnable(GL\_DEPTH\_TEST)

glDepthFunc(GL\_LESS)

glEnable(GL\_LIGHT0)

glMatrixMode(GL\_MODELVIEW)

glLightfv(GL\_LIGHT0, GL\_AMBIENT, (0.5, 0.5, 0.5, 1.0))

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, (1.0, 1.0, 1.0, 1.0))

glLightfv(GL\_LIGHT0, GL\_POSITION, (5.0, 5.0, 5.0, 1.0))

texture\_dict = {"metal": texture\_init("metal\_texture.jpg"),

"wood": texture\_init("wood\_texture.jpg"),

"wall": texture\_init("wall\_texture.jpg")} # textures

# Physical parameters

spring\_spread = 40.0

spring\_length = 0.6 \* 8 \* math.pi

velocity = 0.0

spring\_constant = 0.1

gravity = 9.8

mass = 0.1

scale\_factor = 1.0

mouse\_rotate = False

mouse\_translate = False

last\_mouse\_pos = (0, 0)

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT or event.type == pygame.KEYDOWN and event.key == pygame.K\_q:

pygame.quit()

sys.exit(0)

elif event.type == pygame.MOUSEBUTTONDOWN:

if event.button == 1:

mouse\_rotate = True

pygame.mouse.get\_rel()

elif event.button == 3:

mouse\_translate = True

pygame.mouse.get\_rel()

elif event.type == pygame.MOUSEBUTTONUP:

if event.button == 1:

mouse\_rotate = False

elif event.button == 3:

mouse\_translate = False

elif event.type == pygame.MOUSEMOTION:

if mouse\_rotate:

dx, dy = pygame.mouse.get\_rel()

glRotatef(dy, 1, 0, 0)

glRotatef(dx, 0, 1, 0)

if mouse\_translate:

dx, dy = pygame.mouse.get\_rel()

glTranslatef(dx \* 0.01, -dy \* 0.01, 0)

delta\_X = spring\_spread - spring\_length

velocity += -spring\_constant \* delta\_X + gravity \* mass

spring\_spread += velocity

scale\_factor = spring\_spread / (0.6 \* 16 \* math.pi)

render\_scene()

pygame.display.flip()

pygame.time.wait(10)

Po uruchomieniu programu wyświetlany zostaje model sprężyny z kulą:

Obraz zawierający zrzut ekranu, sufit, sztuka

Opis wygenerowany automatycznie

Rysunek 2 Wyświetlany model po uruchomieniu programu

Dodatkowo program posiada możliwość obrotu kamery co pozwala zaobserwować trójwymiarowość modelu:

Obraz zawierający księżyc

Opis wygenerowany automatycznie

Rysunek 3 Widok modelu z innej perspektywy