Міністерство освіти і науки України

Національний університет «Львівська політехніка»



**ЗВІТ**

про виконання лабораторної роботи № 5

з курсу:

«Спеціалізовані мови програмування»

**Виконала:**

студент гр. ІТ-31

Полапа МАКСИМ

**Прийняв:**

Сергій ЩЕРБАК

Львів 2023

**Тема:** Розробка ASCII ART генератора для візуалізації 3D-фігур

**Мета роботи:** Cтворення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно - орієнтованого підходу та мови Python

**Хід роботи**

*Завдання 1: Проектування класів*

*Розробіть структуру класів для вашого генератора 3D ASCII-арту. Визначте основні компоненти, атрибути та методи, необхідні для програми.*

*Завдання 2: Введення користувача*

*Створіть методи у межах класу для введення користувача та вказання 3D-фігури, яку вони хочуть намалювати, та її параметрів (наприклад, розмір, кольори).*

*Завдання 3: Представлення фігури*

*Визначте структури даних у межах класу для представлення 3D-фігури. Це може включати використання списків, матриць або інших структур даних для зберігання форми фігури та її властивостей.*

*Завдання 4: Проектування з 3D в 2D*

*Реалізуйте метод, який перетворює 3D-представлення фігури у 2D-представлення, придатне для ASCII-арту.*

*Завдання 5: Відображення ASCII-арту*

*Напишіть метод у межах класу для відображення 2D-представлення 3D-фігури як ASCII-арту. Це може включати відображення кольорів і форми за допомогою символів ASCII.*

*Завдання 6: Інтерфейс, зрозумілий для користувача*

*Створіть зручний для користувача командний рядок або графічний інтерфейс користувача (GUI) за допомогою об'єктно-орієнтованих принципів, щоб дозволити користувачам спілкуватися з програмою.*

*Завдання 7: Маніпуляція фігурою*

*Реалізуйте методи для маніпулювання 3D-фігурою, такі масштабування або зміщення, щоб надавати користувачам контроль над її виглядом.*

*Завдання 8: Варіанти кольорів*

*Дозвольте користувачам вибирати варіанти кольорів для їхніх 3D ASCII-арт-фігур. Реалізуйте методи для призначення кольорів різним частинам фігури.*

*Завдання 9: Збереження та експорт*

*Додайте функціональність для зберігання згенерованого 3D ASCII-арту у текстовий файл*

*Завдання 10: Розширені функції*

*Розгляньте можливість додавання розширених функцій, таких як тінь, освітлення та ефекти перспективи, для підвищення реалізму 3D ASCII-арту.*

import pygame

import math

import colorsys

# Initialize Pygame

pygame.init()

# Define basic colors

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

# Screen dimensions

SCREEN\_WIDTH, SCREEN\_HEIGHT = 800, 600

# Set up Pygame screen

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption('3D Shapes')

clock = pygame.time.Clock()

# Donut settings

A, B = 0, 0  # rotation angles for the donut

center\_x, center\_y = SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2

scale = 200  # Scale factor for the donut size

char\_set = ".,-~:;=!\*#$@"  # Luminance characters

font\_style = pygame.font.SysFont('Arial', 18, bold=True)

hue\_shift = 0  # For color cycling

# Define the render functions

def render\_parallelepiped(angle\_x, angle\_y, angle\_z, vertices, scale, offset\_x, offset\_y):

    projected\_vertices = []

    for v in vertices:

        x, y, z = v

        y, z = y \* math.cos(angle\_x) - z \* math.sin(angle\_x), y \* math.sin(angle\_x) + z \* math.cos(angle\_x)

        x, z = x \* math.cos(angle\_y) - z \* math.sin(angle\_y), x \* math.sin(angle\_y) + z \* math.cos(angle\_y)

        x, y = x \* math.cos(angle\_z) - y \* math.sin(angle\_z), x \* math.sin(angle\_z) + y \* math.cos(angle\_z)

        projected\_x = int(x \* scale) + offset\_x

        projected\_y = int(y \* scale) + offset\_y

        projected\_vertices.append((projected\_x, projected\_y))

    for i in range(4):

        pygame.draw.line(screen, WHITE, projected\_vertices[i], projected\_vertices[(i + 4) % 8], 1)

        pygame.draw.line(screen, WHITE, projected\_vertices[i], projected\_vertices[(i + 1) % 4], 1)

        pygame.draw.line(screen, WHITE, projected\_vertices[i + 4], projected\_vertices[(i + 1) % 4 + 4], 1)

def render\_donut(center\_x, center\_y, scale, A, B, hue\_shift):

    screen.fill(BLACK)

    z\_buffer = [0] \* (SCREEN\_WIDTH \* SCREEN\_HEIGHT)  # Z-buffer for depth calculation

    screen\_buffer = [[' ' for \_ in range(SCREEN\_WIDTH)] for \_ in range(SCREEN\_HEIGHT)]  # Screen buffer

    phi\_spacing = 2

    theta\_spacing = 2

    # Calculate donut points

    for theta in range(0, 628, theta\_spacing):

        for phi in range(0, 628, phi\_spacing):

            c = math.sin(phi)

            d = math.cos(theta)

            e = math.sin(A)

            f = math.sin(theta)

            g = math.cos(A)

            h = d + 2

            D = 1 / (c \* h \* e + f \* g + 5)

            l = math.cos(phi)

            m = math.cos(B)

            n = math.sin(B)

            t = c \* h \* g - f \* e

            x = int(center\_x + scale \* D \* (l \* h \* m - t \* n))  # 3D x coordinate after transformation

            y = int(center\_y + scale \* D \* (l \* h \* n + t \* m))  # 3D y coordinate after transformation

            z = int(D \* scale)

            o = int(x + SCREEN\_WIDTH \* y)  # 1D screen coordinate

            N = int(8 \* ((f \* e - c \* d \* g) \* m - c \* d \* e - f \* g - l \* d \* n))  # Luminance index

            # Update buffers if this point is closer to the viewer than what's already plotted

            if 0 <= x < SCREEN\_WIDTH and 0 <= y < SCREEN\_HEIGHT and z > z\_buffer[o]:

                z\_buffer[o] = z

                screen\_buffer[y][x] = char\_set[N if N > 0 else 0]

    # Draw the screen buffer to the Pygame window

    for y, row in enumerate(screen\_buffer):

        for x, char in enumerate(row):

            if char != ' ':

                hsv = (hue\_shift % 1, 1, 1)  # Hue, saturation, value

                color = pygame.Color(0)

                color.hsva = (hue\_shift % 360, 100, 100)  # Convert HSV to RGB

                text\_surface = font\_style.render(char, True, color)

                screen.blit(text\_surface, (x, y))

    hue\_shift += 1  # Increment hue shift for the next frame to change colors over time

    return hue\_shift

# Get user input

shape\_choice = input("Choose a shape:\n1 - Parallelepiped\n2 - Donut\nEnter number: ")

# Parallelepiped settings

vertices = [

    [1, 1, 1], [1, 1, -1], [1, -1, 1], [1, -1, -1],

    [-1, 1, 1], [-1, 1, -1], [-1, -1, 1], [-1, -1, -1]

]

scale = 100

offset\_x, offset\_y = SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2

angle\_x, angle\_y, angle\_z = 0, 0, 0

# Donut settings

center\_x = SCREEN\_WIDTH // 2

center\_y = SCREEN\_HEIGHT // 2

donut\_scale = 200

char\_set = ".,-~:;=!\*#$@"

font\_style = pygame.font.SysFont('Arial', 18, bold=True)

hue\_shift = 0  # Initialize hue shift for donut color

# Main loop

running = True

while running:

    screen.fill(BLACK)

    if shape\_choice == '1':

        # Render parallelepiped

        angle\_x += 0.01

        angle\_y += 0.01

        angle\_z += 0.01

        render\_parallelepiped(angle\_x, angle\_y, angle\_z, vertices, scale, offset\_x, offset\_y)

    elif shape\_choice == '2':

        # Render donut

        A += 0.07  # Increment the rotation angle A for the donut

        B += 0.03  # Increment the rotation angle B for the donut

        hue\_shift = render\_donut(center\_x, center\_y, donut\_scale, A, B, hue\_shift)

    else:

        print("Invalid choice. Exiting.")

        running = False

    pygame.display.flip()

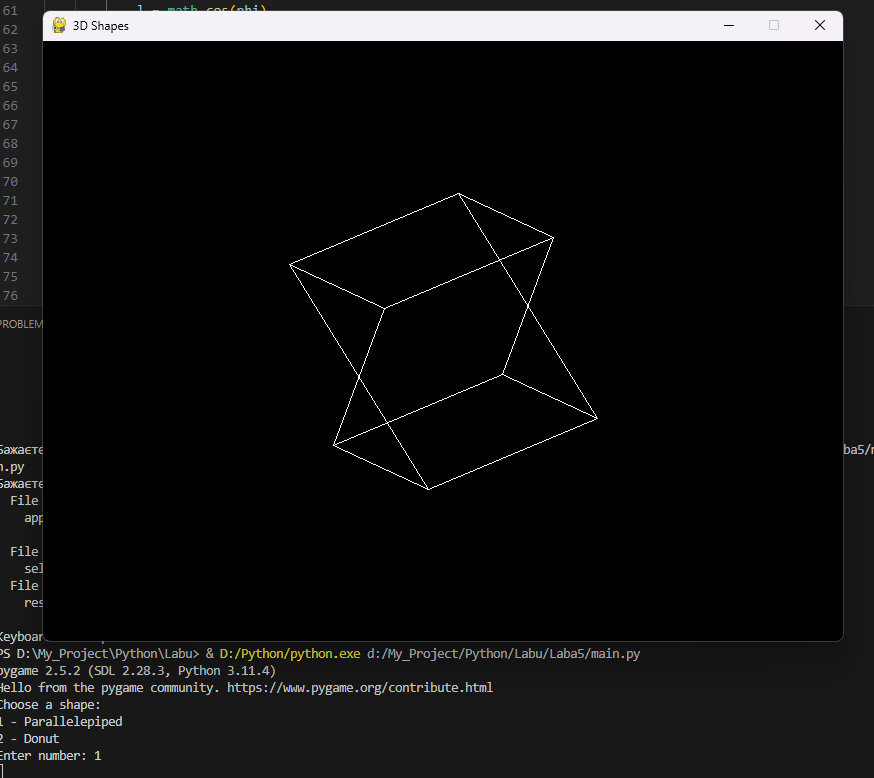
    clock.tick(60)

    for event in pygame.event.get():

        if event.type == pygame.QUIT or (event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE):

            running = False

pygame.quit()



*Рис.1 – Результат виконання програми*

**Висновок:** на даній лабораторній роботі я створив високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволить користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті