LAB MANUAL

ON

COMPUTER GRAPHICS AND VISUALIZATION [ENCT 201]

Year: II, Part: I

PREPARED BY:

Er. SUSHANT PANDEY

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

HIMALAYA COLLEGE OF ENGINEERING

Copyright

Introduction of Course

Computer Graphics and Visualization (ENCT 201) deals with generating, manipulating and displaying graphical information using computers. The course introduces fundamental algorithms for line and curve drawing, clipping, viewing and geometric transformations in 2D and 3D.

In the lab component, students will:

- Implement standard rasterisation algorithms such as DDA, Bresenham, midpoint circle and ellipse.
- Use Python as a platform to write, run and visualise algorithms.
- Apply 2D and 3D geometric transformations using homogeneous coordinates.

Credit

- Practical Credit: 3 hours per week
- Practical Marks: 50 marks

Objective of Course

The objectives are:

- To understand basic raster graphics concepts and scan conversion algorithms.
- To implement line and curve drawing algorithms using Python.
- To apply 2D and 3D transformations on geometric objects.
- To visualise and interpret the effect of transformations.

List of Experiments

General Guidelines and Safety Instructions

1. Strictly adhere to lab timings and follow instructions from the instructor. Clarify doubts immediately.
2. Do not bring bags or personal items into the lab unless permitted.
3. Study the lab manual procedures before starting experiments.
4. Verify code syntax and logic before execution to avoid system crashes.
5. Save work frequently and maintain backups to prevent data loss.
6. Report any hardware/software issues to the instructor immediately.
7. Maintain silence and discipline in the lab.
8. Do not eat or drink near computers.
9. Do not install unauthorized software or make system changes.
10. Avoid running unnecessary applications.
11. Avoid plagiarism and write original code.
12. Avoid downloading unverified files.
13. Do not engage in hacking or unethical programming practices.

Rules for Artificial Intelligence (AI) Usage

1. Avoid over-reliance on AI for problem-solving; develop your own coding skills.
2. Always analyze and understand AI-generated code before using it.
3. Verify before you trust AI-generated code.
4. Mention AI contributions in your work when applicable.

Lab Report Submission Guidelines

1) Pre-Report Submission

Purpose: Demonstrate that students have prepared for the experiment by understanding its objectives, methodology and theoretical background.

Deadline: Before the start of the lab session.

Required Contents:

a) Experiment title

b) Objectives

c) Software required

d) Theoretical background

e) Algorithm or steps

f) Preparatory work (e.g., sample code snippets)

2) Post Lab Report Submission

Purpose: Summarise findings and outcomes of the experiment, including analysis and interpretation of results.

Deadline: Next lab session.

Required Contents:

a) Observations (plots, screenshots, outputs)

b) Analysis and calculations

c) Discussion (interpretation, errors, limitations)

d) Conclusion and recommendations

Note:

i) 20% deduction if the report is submitted one working day late; not accepted after that.

ii) Reports must be signed by the lab instructor; students should maintain a log book.

Conduction of Practical Examination / Project

- **Lab Examination:** Covers all lab topics.
- **Viva:** Based on lab activities and concepts.
- **Project:** Implementation of a CGV mini-project with demonstration.

Mark Distribution

Attendance	Lab Performance	Lab Report	Viva	Lab Exam	Lab Project	Total Marks (50)
5	5	10	5	10	15	50

Note:

i) Minimum attendance of 80% is required.

ii) All pre- and post-lab reports must be submitted to be eligible for evaluation.

iii) All lab reports should be compiled and bound with proper cover page and index.

LAB 1: BASIC PYTHON PROGRAMMING AND INTRODUCTION TO MATPLOTLIB

Objective(s)

i) To learn basic Python syntax required for Computer Graphics.

ii) To understand variables, data types, operators, conditions, loops and lists.

iii) To write simple Python programs using input/output and control structures.

iv) To plot basic graphs using the `matplotlib` library.

Software(s) Required

- Python 3.x
- Jupyter Notebook / VS Code / PyCharm
- Python libraries: `matplotlib`, `numpy` (for later labs)

Theory

Python is a high-level, interpreted language with simple, readable syntax. In this course, Python is used as the main language to implement and visualise computer graphics algorithms.

Key concepts:

- **Indentation:** Python uses indentation (spaces) instead of braces {} to define blocks.
- **Variables:** Created by assignment (no explicit type declaration).
- **Data Types:** `int`, `float`, `str`, `bool`, `list`, etc.
- **Control Flow:** `if`, `elif`, `else`, `for`, `while`.
- **Functions:** Defined using `def` keyword.
- **Libraries:** Imported using `import` statement (e.g. `import matplotlib.pyplot as plt`).

Basic Syntax and Variables

```
# Single-line comment starts with '#'

# Output using print()
print("Hello World")

# Variables and basic types
x = 10          # integer
pi = 3.14       # float
course = "CGV"  # string
is_bool = True  # boolean

print(x, pi, course, is_bool)
```

Operators and Expressions

- **Arithmetic:** +, -, *, /, // (integer division), % (modulus), ** (power)
- **Comparison:** ==, !=, <, >, <=, >=
- **Logical:** and, or, not

```
a = 1
b = 2

x = a // b   # Integer division -> 0
print("Integer division:", x)

y = a / b    # Floating point division -> 0.5
print("Float division:", y)

print("a == b ?", a == b)
print("a != b ?", a != b)
print("a < b  ?", a < b)
```

Input and Output

```
# Reading input from user (uncomment when running
   interactively)
# a = int(input("Enter a number: "))
# b = int(input("Enter second number: "))
# c = a + b
# print("Sum =", c)

# f-string for formatted output
name = "Sushant"
marks = 85
print(f"Student {name} scored {marks} marks.")
```

Conditional Statements

```
a = 5
b = 3

if a > b:
    print("a is greater than b")
elif a < b:
    print("a is less than b")
else:
    print("a and b are equal")

# Combined condition using logical operators
if a != b or a == b:
    print("This condition is always True in this example")
```

Loops in Python

While Loop

```
x = 20
while x < 100:
    print("hello from while loop, x =", x)
    x = x + 20   # or x += 20
```

For Loop with `range()`

```
# range(start, stop) generates numbers from start to stop-1
for i in range(1, 5):
    print("i =", i)
```

Lists and Indexing

Lists store multiple values of possibly different types.

```
names = ["A", "B", "C", "D"]

# Access by index (0-based)
print(names[0])
print(names[1])
print(names[2])
print(names[3])

# Loop through list elements
for name in names:
    print("Name:", name)

# Building a list using input
values = []
for i in range(3):
    num = float(input(f"Enter number {i+1}: "))
    values.append(num)

print("You entered:", values)
```

Functions

Functions group reusable code.

```
def add_numbers(x, y):
    """Return the sum of x and y."""
    result = x + y
    return result

s = add_numbers(5, 7)
```

```
print("Sum =", s)
```

Basic Plotting with Matplotlib

Line Plot

```
import matplotlib.pyplot as plt

# Data points
x = [1, 2, 3, 4, 5]
y = [10, 5, 12, 8, 15]

plt.plot(x, y, marker='o', linestyle='-', color='red')
plt.title("Simple Line Plot in Matplotlib")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.grid(True)
plt.show()
```

Bar Plot

```
# Bar graph of 5 values
labels = ["A", "B", "C", "D", "E"]
values = [5, 7, 3, 9, 4]

plt.bar(labels, values)
plt.title("Simple Bar Graph")
plt.xlabel("Category")
plt.ylabel("Value")
plt.grid(axis="y")
plt.show()
```

Lab Tasks

1. Write a Python program to input five numbers and print the largest number.
2. Write a program to print the multiplication table of any number (from 1 to 10).
3. Create a list of 10 integers and print:

    - all even numbers
    - all odd numbers

4. Using `matplotlib`, plot:
    - a simple line graph of your choice
    - a bar graph of marks of 5 subjects

Outcomes

- Students will understand core Python syntax needed for later graphics labs.
- Students will be able to use conditions, loops and lists in basic programs.
- Students will know how to create simple plots with `matplotlib`.

Lab Assignment

1. Draw a sine wave using `matplotlib` for x from 0 to 2π.
2. Write a function-based Python program to compute the factorial of a number.
3. Write a Python script that prints all prime numbers between 1 and 100.

LAB 2: IMPLEMENTATION OF DDA LINE DRAWING ALGORITHM (PYTHON)

Objective(s)

i) To understand the Digital Differential Analyzer (DDA) line drawing algorithm.

ii) To implement the DDA algorithm in Python and visualise the generated pixels.

iii) To compare DDA with the analytical line equation.

Software(s) Required

Python 3, `matplotlib`, any IDE or Jupyter Notebook.

Theory

In raster graphics, a straight line must be approximated using discrete pixels. The DDA algorithm is an incremental method that uses the line slope to step along the dominant axis and compute intermediate points.

Given two endpoints (x_1, y_1) and (x_2, y_2), we define:

$$dx = x_2 - x_1, \quad dy = y_2 - y_1$$

The number of steps is chosen as:

$$\text{steps} = \max(|dx|, |dy|)$$

The increment in each step is:

$$x_{\text{inc}} = \frac{dx}{\text{steps}}, \quad y_{\text{inc}} = \frac{dy}{\text{steps}}$$

Starting from (x_1, y_1), the algorithm adds these increments repeatedly and rounds to the nearest pixel.

Algorithm: DDA Line Drawing

1. Read starting point (x_1, y_1) and ending point (x_2, y_2).

2. Compute dx, dy and number of steps.
3. Compute x_{inc} and y_{inc}.
4. Initialise $x = x_1,\ y = y_1$.
5. For $k = 0$ to steps:
   - Plot pixel at $(\text{round}(x), \text{round}(y))$.
   - Update $x = x + x_{\text{inc}},\ y = y + y_{\text{inc}}$.

Sample Python Code

```
import matplotlib.pyplot as plt

def dda_line(x1, y1, x2, y2):
    xes, yes = [], []

    dx = x2 - x1
    dy = y2 - y1

    steps = int(max(abs(dx), abs(dy)))

    x_inc = dx / steps
    y_inc = dy / steps

    x = x1
    y = y1

    for _ in range(steps + 1):
        xes.append(round(x))
        yes.append(round(y))
        x += x_inc
        y += y_inc

    return xes, yes


def plot_dda(x1, y1, x2, y2):
    xes, yes = dda_line(x1, y1, x2, y2)
    plt.figure(figsize=(6, 6))
    plt.plot(xes, yes, marker='o', linestyle='-', color='
       blue')
    plt.title("DDA Line Drawing Algorithm")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.axis('equal')
    plt.show()


# Example
plot_dda(2, 3, 15, 9)
```

Lab Tasks

1. Implement the DDA algorithm as a function that returns x and y coordinate lists.
2. Draw lines with different slopes: $m < 1, m > 1$, horizontal, vertical and negative slope.
3. Modify the program to take endpoints as user input.

Outcomes

Students will be able to implement DDA and understand incremental plotting of raster lines.

Result Analysis

Compare the plotted points with the theoretical line equation. Comment on any visual staircasing and rounding effects.

Lab Assignment

1. Extend the DDA program to draw a rectangle given two opposite corners.

2. Use DDA to draw the axes of a simple coordinate system (X and Y axes).

LAB 3: IMPLEMENTATION OF BRESENHAM LINE DRAWING ALGORITHM (PYTHON)

Objective(s)

i) To understand Bresenham's line drawing algorithm.

ii) To implement Bresenham's algorithm in Python.

iii) To compare Bresenham with DDA in terms of arithmetic operations and output.

Software(s) Required

Python 3, `matplotlib`.

Theory

Bresenham's algorithm is an efficient incremental line rasterisation algorithm that uses only integer arithmetic. A decision parameter controls whether the next pixel is chosen in the same row/column or diagonally.

Algorithm Overview

- Calculate $dx = |x_2 - x_1|$ and $dy = |y_2 - y_1|$.
- Determine step directions s_x and s_y.
- Use different update rules depending on whether the line is more horizontal ($dx \geq dy$) or more vertical ($dx < dy$).

Sample Python Code

```python
import matplotlib.pyplot as plt

def bresenham_line(x1, y1, x2, y2):
    xes, yes = [], []

    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    sx = 1 if x2 >= x1 else -1
    sy = 1 if y2 >= y1 else -1

    x, y = x1, y1

    if dx >= dy:
        p = 2 * dy - dx
        for _ in range(dx + 1):
            xes.append(x)
            yes.append(y)
            x += sx
            if p >= 0:
                y += sy
                p += 2 * dy - 2 * dx
            else:
                p += 2 * dy
    else:
        p = 2 * dx - dy
        for _ in range(dy + 1):
            xes.append(x)
            yes.append(y)
            y += sy
            if p >= 0:
                x += sx
                p += 2 * dx - 2 * dy
            else:
                p += 2 * dx

    return xes, yes


def plot_bresenham(x1, y1, x2, y2):
    xes, yes = bresenham_line(x1, y1, x2, y2)
    plt.figure(figsize=(6, 6))
    plt.plot(xes, yes, marker='o', linestyle='-', color='
       green')
    plt.title("Bresenham Line Drawing Algorithm")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.axis('equal')
    plt.show()


# Example
plot_bresenham(2, 3, 10, 8)
```

Lab Tasks

1. Implement Bresenham's algorithm as shown.
2. Draw lines for different octants and compare visually with DDA lines.
3. Compare the number of integer additions and multiplications used by DDA and Bresenham.

LAB 4: IMPLEMENTATION OF MIDPOINT CIRCLE DRAWING ALGORITHM

Objective(s)

i) To understand the midpoint circle algorithm.

ii) To use 8-way symmetry to efficiently rasterise a circle.

iii) To implement the algorithm in Python and visualise the result.

Software(s) Required

Python 3, `matplotlib`.

Theory

A circle with radius r centred at (x_c, y_c) satisfies:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

The midpoint circle algorithm starts at $(0, r)$ and moves in the first octant, choosing between two candidate pixels at each step based on a decision parameter. Using symmetry, 8 pixels are plotted for each computed point.

Sample Python Code

```
import matplotlib.pyplot as plt

def plot_circle_points(xc, yc, x, y, xes, yes):
    pts = [
        ( x + xc,  y + yc),
        (-x + xc,  y + yc),
        ( x + xc, -y + yc),
        (-x + xc, -y + yc),
        ( y + xc,  x + yc),
        (-y + xc,  x + yc),
        ( y + xc, -x + yc),
        (-y + xc, -x + yc),
    ]
    for px, py in pts:
        xes.append(px)
        yes.append(py)


def midpoint_circle(r, xc=0, yc=0):
    x = 0
    y = r
    p = 1 - r
    xes, yes = [], []

    plot_circle_points(xc, yc, x, y, xes, yes)

    while x < y:
        x += 1
        if p < 0:
            p = p + 2 * x + 1
        else:
            y -= 1
            p = p + 2 * (x - y) + 1
        plot_circle_points(xc, yc, x, y, xes, yes)

    return xes, yes


def plot_midpoint_circle(r, xc=0, yc=0):
    xes, yes = midpoint_circle(r, xc, yc)
    plt.figure(figsize=(6, 6))
    plt.scatter(xes, yes, marker='.', color='red')
    plt.title("Midpoint Circle Algorithm")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.axis('equal')
    plt.show()
```

```
# Example
plot_midpoint_circle(20, 0, 0)
```

Lab Tasks

1. Implement the midpoint circle algorithm as above.
2. Draw circles with different radii and centres.
3. Use the algorithm to draw concentric circles (target pattern).

LAB 5: IMPLEMENTATION OF MIDPOINT ELLIPSE DRAWING ALGORITHM

Objective(s)

i) To understand the midpoint ellipse algorithm.

ii) To process Region 1 and Region 2 separately for an ellipse.

iii) To implement the algorithm in Python using 4-way symmetry.

Software(s) Required

Python 3, `matplotlib`.

Theory

An ellipse centred at (x_c, y_c) with radii r_x and r_y satisfies:

$$\frac{(x-x_c)^2}{r_x^2} + \frac{(y-y_c)^2}{r_y^2} = 1$$

The midpoint ellipse algorithm divides the first quadrant into two regions based on the slope magnitude. Different decision parameters are used in each region, and 4-way symmetry generates points in all quadrants.

Sample Python Code

```
import matplotlib.pyplot as plt

def plot_ellipse_points(xc, yc, x, y, xes, yes):
    pts = [
        ( x + xc,  y + yc),
        (-x + xc,  y + yc),
        ( x + xc, -y + yc),
        (-x + xc, -y + yc),
    ]
    for px, py in pts:
        xes.append(px)
        yes.append(py)


def midpoint_ellipse(rx, ry, xc=0, yc=0):
    rx2 = rx * rx
    ry2 = ry * ry

    x = 0
    y = ry

    xes, yes = [], []

    # Region 1
    p1 = ry2 - (rx2 * ry) + 0.25 * rx2
    plot_ellipse_points(xc, yc, x, y, xes, yes)

    while 2 * ry2 * x <= 2 * rx2 * y:
        x += 1
        if p1 < 0:
            p1 += 2 * ry2 * x + ry2
        else:
            y -= 1
            p1 += 2 * ry2 * x - 2 * rx2 * y + ry2
        plot_ellipse_points(xc, yc, x, y, xes, yes)

    # Region 2
    p2 = (ry2 * (x + 0.5) ** 2) + (rx2 * (y - 1) ** 2) - (
       rx2 * ry2)

    while y >= 0:
        if p2 > 0:
            y -= 1
            p2 -= 2 * rx2 * y + rx2
        else:
            x += 1
            y -= 1
            p2 += 2 * ry2 * x - 2 * rx2 * y + rx2
        plot_ellipse_points(xc, yc, x, y, xes, yes)
    return xes, yes


def plot_midpoint_ellipse(rx, ry, xc=0, yc=0):
    xes, yes = midpoint_ellipse(rx, ry, xc, yc)
    plt.figure(figsize=(6, 6))
    plt.scatter(xes, yes, marker='.', color='purple')
    plt.title("Midpoint Ellipse Algorithm")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.grid(True)
    plt.axis('equal')
    plt.show()


# Example
plot_midpoint_ellipse(30, 15, 0, 0)
```

Lab Tasks

1. Implement the midpoint ellipse algorithm.
2. Draw ellipses with different radii and centres.
3. Compare the point spacing in Region 1 and Region 2.

LAB 6: 2D GEOMETRIC TRANSFORMATIONS USING HOMOGENEOUS COORDINATES

Objective(s)

i) To understand 2D transformations (translation, scaling, rotation) using homogeneous coordinates.

ii) To implement fixed-point scaling of a line using composite transformations.

iii) To visualise the effect of scaling and rotation on a line using Python.

Software(s) Required

Python 3, `numpy`, `matplotlib`.

Theory

In homogeneous coordinates, a 2D point (x, y) is represented as $(x, y, 1)^T$. Basic 3×3 transformation matrices:

Translation by (t_x, t_y):

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling by (S_x, S_y):

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation by angle θ:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Fixed-point scaling about (x_f, y_f) is implemented as:

$$M = T(x_f, y_f)\, S(S_x, S_y)\, T(-x_f, -y_f)$$

Sample Python Code

```
import numpy as np
import matplotlib.pyplot as plt


def bresenham_line(x0, y0, x1, y1):
    xes, yes = [], []
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    sx = 1 if x1 >= x0 else -1
    sy = 1 if y1 >= y0 else -1
    x, y = x0, y0

    if dx >= dy:
        p = 2 * dy - dx
        for _ in range(dx + 1):
            xes.append(x)
            yes.append(y)
            x += sx
            if p >= 0:
                y += sy
                p += 2 * dy - 2 * dx
            else:
                p += 2 * dy
    else:
        p = 2 * dx - dy
        for _ in range(dy + 1):
            xes.append(x)
            yes.append(y)
            y += sy
            if p >= 0:
                x += sx
                p += 2 * dx - 2 * dy
            else:
                p += 2 * dx

    return np.array(xes), np.array(yes)


def apply_2d_transformation(x_coords, y_coords,
   transformation_matrix):
    points = np.vstack([x_coords, y_coords, np.ones_like(
       x_coords)])
    transformed_points = transformation_matrix @ points
    return transformed_points[0], transformed_points[1]


def plot_line_with_transformations(x0, y0, x1, y1):
    x_orig, y_orig = bresenham_line(x0, y0, x1, y1)

    # Fixed point (start of line)
    xf, yf = x0, y0

    scaling_matrix = np.array([
        [2,   0,   0],
        [0, 0.5,   0],
        [0,   0,   1]
    ])

    T_to_origin = np.array([
        [1, 0, -xf],
        [0, 1, -yf],
        [0, 0,  1]
    ])

    T_back = np.array([
        [1, 0, xf],
        [0, 1, yf],
        [0, 0, 1]
    ])

    composite_matrix = T_back @ scaling_matrix @ T_to_origin
    theta = np.pi / 4  # 45 degrees
    rotation_matrix = np.array([
        [np.cos(theta), -np.sin(theta), 0],
        [np.sin(theta),  np.cos(theta), 0],
        [0,              0,             1]
    ])

    composite_matrix = rotation_matrix @ composite_matrix

    x_transformed, y_transformed = apply_2d_transformation(
        x_orig, y_orig, composite_matrix
    )

    plt.figure(figsize=(8, 6))
    plt.plot(x_orig, y_orig, marker='*', color='blue',
             linestyle='-', label='Original Line')
    plt.plot(x_transformed, y_transformed, marker='o',
             color='red', linestyle='--', label='Transformed
                 Line')

    plt.title("Bresenham Line with 2D Transformations")
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.legend()
    plt.grid(True)
    plt.axis('equal')
    plt.show()


# Example Usage
plot_line_with_transformations(2, 3, 10, 8)
```

Lab Tasks

1. Implement fixed-point scaling of a line about its starting point.
2. Change the fixed point to the midpoint of the line and observe the difference.
3. Implement pure rotation about the origin using the given rotation matrix.

LAB 7: 3D GEOMETRIC TRANSFORMATIONS AND VISUALIZATION

Objective(s)

i) To introduce 3D homogeneous coordinates and 4×4 transformation matrices.

ii) To apply translation, scaling and rotation to simple 3D objects.

iii) To visualise 3D transformations via 2D projections in Python.

Software(s) Required

Python 3, `numpy`, `matplotlib` (with `mpl_toolkits.mplot3d`).

Theory

In 3D graphics, a point (x, y, z) is represented in homogeneous coordinates as $(x, y, z, 1)^T$. Transformations are expressed as 4×4 matrices.

Translation by (t_x, t_y, t_z):

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z-axis by θ:

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composite transformations are formed by multiplying matrices in the order of application.

Sample Python Code: Transforming a Cube

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401


def make_unit_cube():
    pts = np.array([
        [0, 0, 0, 1],
        [1, 0, 0, 1],
        [1, 1, 0, 1],
        [0, 1, 0, 1],
        [0, 0, 1, 1],
        [1, 0, 1, 1],
        [1, 1, 1, 1],
        [0, 1, 1, 1],
    ]).T
    return pts


def plot_cube(ax, pts, style='b-'):
    edges = [
        (0, 1), (1, 2), (2, 3), (3, 0),
        (4, 5), (5, 6), (6, 7), (7, 4),
        (0, 4), (1, 5), (2, 6), (3, 7)
    ]
    xs, ys, zs = pts[0], pts[1], pts[2]
    for i, j in edges:
        ax.plot(
            [xs[i], xs[j]],
            [ys[i], ys[j]],
            [zs[i], zs[j]],
            style
        )


def transform_points(pts, M):
    return M @ pts
```

```
def example_3d_transform():
    cube = make_unit_cube()

    S = np.array([
        [2,   0,   0,   0],
        [0, 1.5,   0,   0],
        [0,   0, 0.5,  0],
        [0,   0,   0,   1]
    ])

    T = np.array([
        [1, 0, 0, 2],
        [0, 1, 0, 1],
        [0, 0, 1, 3],
        [0, 0, 0, 1]
    ])

    theta = np.pi / 6
    Rz = np.array([
        [np.cos(theta), -np.sin(theta), 0, 0],
        [np.sin(theta),  np.cos(theta), 0, 0],
        [0,              0,             1, 0],
        [0,              0,             0, 1]
    ])

    M = T @ Rz @ S
    cube_t = transform_points(cube, M)

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    plot_cube(ax, cube, style='b-')
    plot_cube(ax, cube_t, style='r--')

    ax.set_title("3D Transformations of a Cube")
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.view_init(elev=20, azim=30)
    plt.show()


# Example
example_3d_transform()
```

Lab Tasks

1. Construct a cube and draw it using `matplotlib`.
2. Apply scaling, rotation and translation, and display both original and transformed cubes.
3. Try different viewing angles using `ax.view_init()` and observe the projection.

Lab Assignment

1. Create a simple 3D house model (cube plus pyramid roof) and apply transformations.
2. Implement rotations about the X-axis and Y-axis and compare their effects with rotation about the Z-axis.

LAB 8: UNITY BASICS I – SETTING UP THE ROLL-A-BALL GAME

Objective(s)

i) To get familiar with the Unity Hub and Unity Editor interface.

ii) To create a new 3D Unity project using the Roll-a-Ball style setup.

iii) To construct a simple 3D scene with a ground plane, player object, camera and basic materials.

iv) To understand the concepts of GameObjects, Components and the Transform in Unity.

Software(s) Required

- Unity Hub (latest version)
- Unity Editor (Unity 6 / compatible LTS version)

Theory

7.1 Unity Hub, Project and Scene

Unity Hub is used to manage Unity Editor installations and projects. A **Unity Project** contains all assets, scenes and settings for a game. In the Roll-a-Ball learning project, a new 3D project is created using the *Universal 3D* template.[1]

A **Scene** is like a level or environment that contains GameObjects. For this lab, we create a scene (e.g. `MiniGame`) where the entire Roll-a-Ball game is built.

7.2 GameObjects and Components

Everything in a Unity scene is a **GameObject**. GameObjects do not do anything by themselves; their behaviour comes from **Components**. Common components:

- **Transform** – position, rotation and scale of the GameObject.

[1]Based on Unity's "3D Beginner: Roll-a-Ball" project and the "Setting up the Game" tutorial.:contentReference[oaicite:0]index=0

- **Mesh Filter / Mesh Renderer** – define the shape and how it is drawn.
- **Collider** – used for physics collisions.
- **Rigidbody** – enables physics simulation (gravity, forces, etc.).

When creating the Roll-a-Ball game, we use:

- A plane GameObject for the ground.
- A sphere GameObject as the player.
- A camera GameObject to view the scene.

7.3 Transform and 3D Coordinate System

Unity uses a left-handed 3D coordinate system:

- X-axis: left/right
- Y-axis: up/down
- Z-axis: forward/back

The **Transform** component stores:

- **Position** (x, y, z) – where the object is in the scene.
- **Rotation** – orientation in degrees.
- **Scale** – size in each direction.

In Roll-a-Ball:

- The ground (plane) is typically centred at $(0, 0, 0)$ and scaled to form a play area.
- The player sphere is placed slightly above the ground, e.g. $(0, 0.5, 0)$, so that it rests on the plane.[2]

[2]Typical setup described in Unity Roll-a-Ball resources.:contentReference[oaicite:1]index=1

7.4 Setting up the Roll-a-Ball Scene

Following the Roll-a-Ball "Setting up the Game" tutorial::contentReference[oaicite:2]index=2

1. **Create a new Unity Project**
   - Open Unity Hub → **New Project**.
   - Select the **Universal 3D** template.
   - Name the project (e.g. `Rollaball`) and choose a location.
   - Click **Create Project**.
2. **Create and save a Scene**
   - In Unity Editor, go to `File → New Scene`.
   - Choose a basic 3D scene template if prompted.
   - Save as `MiniGame.unity` inside a `Scenes` folder.
3. **Create the Ground**
   - In the Hierarchy, create `GameObject → 3D Object → Plane`.
   - Rename it to `Ground`.
   - Reset its Position to $(0, 0, 0)$.
   - Scale it to create a larger play area, e.g. $(x = 5, y = 1, z = 5)$ or similar.
4. **Create the Player**
   - In the Hierarchy, create `3D Object → Sphere`.
   - Rename it to `Player`.
   - Set its Position to $(0, 0.5, 0)$ so that it sits on the Ground.
   - Add a **Rigidbody** component (via `Add Component`) to enable physics.
5. **Adjust Lighting and Sky**
   - Adjust the Directional Light rotation to get clear lighting.
   - Optionally modify the Skybox or background colour from the Lighting settings.
6. **Set up the Camera**

- Select the Main Camera.
- Move it to a position that looks down at the Ground and Player, e.g. ($x = 0, y = 10, z = -10$) and rotate it to view the play area.
- Use `Game` view to check the final framing.

7. **Add Basic Materials (Optional)**
   - In the Project window, create new Materials for Ground and Player.
   - Change their colours and drag them onto the respective GameObjects.

Lab Tasks

1. Install Unity (via Unity Hub) and create a new 3D project named `Rollaball`.
2. Create and save a scene named `MiniGame` inside a `Scenes` folder.
3. Add a `Ground` plane, centre it at $(0, 0, 0)$ and scale it to form the play area.
4. Add a `Player` sphere, place it at $(0, 0.5, 0)$ and add a Rigidbody component.
5. Adjust the Main Camera so that both Ground and Player are clearly visible in the Game view.
6. Assign at least two different materials (colours) to the Ground and Player.

Outcomes

After completing this lab, students will:

- Understand the basic workflow of creating and saving a Unity project and scene.
- Be able to add and manipulate GameObjects using Transforms.
- Recognise the role of components (Transform, Mesh, Collider, Rigidbody).
- Have a working 3D scene ready for scripting in the next lab.

Lab Assignment

1. Add four cube walls around the Ground to form a closed play area (arena). Position and scale them appropriately.
2. Experiment with different camera angles (top-down, diagonal) and note which one provides the best view for a rolling ball game.
3. Save screenshots of your Scene view and Game view and attach them to your report.

LAB 9: UNITY BASICS II – PLAYER MOVEMENT, COLLECTIBLES AND UI

Objective(s)

i) To control the player sphere using a C# script and physics (Rigidbody).

ii) To create collectibles and detect collisions using trigger colliders.

iii) To display score and messages using a simple Unity UI.

iv) To understand the basic game loop: input → movement → collision → score update.

Software(s) Required

- Unity Hub and Unity Editor (same project as Lab 7)
- Text editor / code editor (Unity's built-in editor, VS Code, or similar)

Theory

This lab follows the next steps of the Roll-a-Ball learning project: moving the player, setting up the play area, creating collectibles, detecting collisions with them, and displaying score using UI elements.[3]

8.1 Player Movement with Rigidbody

The player sphere has a **Rigidbody** component which is controlled via physics. A C# script (e.g. `PlayerController.cs`) is used to:

- Read input from the keyboard (horizontal and vertical axes).
- Convert input into a direction vector.
- Apply a force on the Rigidbody to roll the ball.

[3]Concepts adapted from Unity's "3D Beginner: Roll-a-Ball" tutorials on moving the player, creating collectibles, collision detection, UI text and building the game.:contentReference[oaicite:3]index=3

Key concepts:

- `Input.GetAxis("Horizontal")` and `Input.GetAxis("Vertical")` provide smoothed input values.
- `Rigidbody.AddForce()` applies a force to move the ball.
- `FixedUpdate()` is used for physics-related updates.

8.2 Play Area and Collectibles

The play area is surrounded by walls (cubes) to prevent the ball from falling off. **Collectibles** (e.g. small cubes or capsules) are placed across the ground. Each collectible:

- Has a collider set as **Is Trigger**.
- May have a simple rotation script for visual effect.
- Is tagged with a tag like `"PickUp"` to identify it in code.

8.3 Collision Detection with Triggers

When the player collides with a collectible, the ball should "pick up" the item:

- The Player has a Rigidbody and a Collider.
- The Collectible has a Collider with **Is Trigger** enabled.
- The Player's script implements `OnTriggerEnter(Collider other)` to detect when it passes through a trigger.

Inside `OnTriggerEnter`, the script can:

- Disable the collectible GameObject.
- Increase the score.
- Update UI text to display the new score.

8.4 Simple UI for Score and Messages

Unity's UI system uses a **Canvas** GameObject containing UI elements. In Roll-a-Ball:

- A Text or TextMeshPro element displays the current score.
- Another text field can display a "You Win!" message when all collectibles are collected.

The Player script holds references to these UI elements and updates their text accordingly.

Sample Code: Player Controller

C# Script: `PlayerController.cs`

```
using UnityEngine;
using UnityEngine.UI;   // For UI Text

public class PlayerController : MonoBehaviour
{
    public float speed = 10f;
    public Text scoreText;
    public Text winText;

    private Rigidbody rb;
    private int score = 0;
    private int totalPickups;

    void Start()
    {
        rb = GetComponent<Rigidbody>();

        // Count all objects tagged as "PickUp"
        GameObject[] pickups = GameObject.
           FindGameObjectsWithTag("PickUp");
        totalPickups = pickups.Length;

        score = 0;
        UpdateScoreText();
        winText.text = "";
    }

    void FixedUpdate()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical   = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f,
            moveVertical);

        rb.AddForce(movement * speed);
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("PickUp"))
        {
            other.gameObject.SetActive(false);   // Hide
               collectible
            score++;
            UpdateScoreText();
        }
    }

    void UpdateScoreText()
    {
        scoreText.text = "Score: " + score.ToString();

        if (score >= totalPickups)
        {
            winText.text = "You Win!";
        }
    }
}
```

8.5 Steps to Attach the Script

1. Create a new `C# Script` named `PlayerController` in the Project window.
2. Double-click to open it in the code editor and replace content with the above code.
3. Attach the script to the `Player` GameObject (drag it onto the Player in the Inspector).
4. In the Inspector, set:
   - `Speed` (e.g. 10).
   - Drag the Score Text UI element onto the `Score Text` field.
   - Drag the Win Text UI element onto the `Win Text` field.

Lab Tasks

1. **Player Movement**
   - Add a `PlayerController.cs` script to the Player.
   - Implement movement using `Input.GetAxis` and `Rigidbody.AddForce`.
   - Test movement in Play mode.
2. **Collectibles Setup**
   - Create a small cube or capsule GameObject and name it `PickUp`.
   - Add a Collider and enable **Is Trigger**.
   - Tag it as `"PickUp"`.
   - Duplicate it to place multiple collectibles across the Ground.
3. **UI Setup**
   - Create a Canvas with Text (or TextMeshPro) UI elements for score and win message.
   - Link them to the PlayerController script.
   - Run the game and verify that the score increments and the win message appears when all collectibles are collected.

Outcomes

After completing this lab, students will:

- Understand basic player control using physics-based movement in Unity.
- Know how to create and manage collectibles using trigger colliders and tags.
- Be able to update UI elements from a script to reflect game state (score and win text).
- Have a simple but complete interactive prototype (Roll-a-Ball core gameplay).

Lab Assignment

1. Add a simple rotation script to the `PickUp` objects so they spin continuously.
2. Create a timer UI that counts the time taken to collect all pickups.
3. Modify the game so that if the ball falls off the Ground (e.g. below a certain Y value), the scene is reloaded and the game restarts.

Additional Notes

This page is reserved for additional notes, observations or corrections made during the lab sessions.

Signature of Instructor:

Signature of Student:

Viva and Evaluation Remarks

Viva Questions Discussed:

- ______________________________
- ______________________________
- ______________________________
- ______________________________

Instructor's Remarks:

- ______________________________
- ______________________________
- ______________________________

Signature of Instructor: