ANNAMALAI UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.E. [COMPUTER SCIENCE AND ENGINEERING]

SEMESTER - V

CSCP508 - COMPUTER GRAPHICS AND MULTIMEDIA LAB

LABORATORY MANUAL

(July 2025 – November 2025)

LAB IN-CHARGE

Dr. G. Arulselvi, Associate Professor

Department of Computer Science and Engineering

Annamalai University

VISION

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

MISSION

- Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.

- Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.

- Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs.

- Expose the students to the emerging technological advancements for meeting the demands of the industry.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO	PEO Statements
PEO1	To prepare the graduates with the potential to get employed in the right role and/or become entrepreneurs to contribute to the society.
PEO2	To provide the graduates with the requisite knowledge to pursue higher education and carry out research in the field of Computer Science.
PEO3	To equip the graduates with the skills required to stay motivated and adapt to the dynamically changing world so as to remain successful in their career.
PEO4	To train the graduates to communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

LIST OF EXPERIMENTS

IMPLEMENTATION OF COMPUTER GRAPHICS USING PYTHON

1. Implementation of Line drawing Algorithm :
    (a). Bresenham's line drawing algorithm.
    (b). DDA (Digital Differential Analyser) Line drawing Algorithm.
2. Implementation of Circle drawing Algorithm :
    (a). Midpoint Circle drawing Algorithm.
    (b). Bresenham's Circle drawing Algorithm.
3. Implementation of Bresenham's Ellipse Drawing Algorithm.
4. Implementation of 2D Transformations :
    (a). Translation, Rotation, Scaling.
    (b). Reflection and Shearing.
5. Implementation of 2D Line Clipping Algorithm :
    (a). Cohen-Sutherland Algorithm.
    (b). Liang-Barsky Algorithm.
6. Polygon clipping using Sutherland-Hodgeman Algorithm.
7. Implementation of 3D Transformations - Translation,Rotation,Scaling.
8. Implementation of 2D Animation (using Timer,Loop,simple animation):
    (a). Bouncing Ball.
    (b). Car movement.
9. Implementation of 3D Animation - Human Facial Expressions:
    (a) Smile. (b) Sad. (c) Surprise.
10. Drawing Three Dimensional Objects and Scenes using OpenGL.

MULTIMEDIA

I. GIMP

1. Implementation of Logo Creation
2. Implementation of Text Animation

II. AUDACITY

1. Audio Signal Processing: Silencing, Trimming, and Duplicating
2. Applying Advanced Effects to Audio Signals

III. WINDOWS MOVIE MAKER

1. Applying Visual Effects to Videos
2. Creating and Adding Titles in Video Clips

IV. SWISH

1. Implementation of Dynamic Text Effects
2. Designing a Pre-Loader Animation

V. FLASH

1. Transforming Object Shapes
2. Implementing Image Masking for Viewing Effects

VI. PHOTO IMPACT

1. Advanced Text Effects Implementation
2. Image Slicing and Segmentation Techniques

LAB INCHARGES:

A Batch – Dr. G. ARULSELVI

B Batch – Dr. A. KANTHIMATHINATHAN

Head of the Department

Course Outcomes:

At the end of this course, the students will be able to

1. Implement 2D and 3D shape drawing algorithms, transformations and its applications.
2. Develop applications on image, sound and video using editing tools such as GIMP, Audacity, Windows Movie Maker, Swish, Flash, etc.
3. Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering

Mapping of Course Outcomes with Programme Outcomes												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1	3	2	1	-	1	-	-	-	-	-	-	-
CO2	1	1	3	1	3	-	-	-	-	-	-	-
CO3	2	2	-	-	-	-	-	-	-	2	-	2

Rubric for CO3 in Laboratory Courses					
Rubric	**Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks**				
	Up To 2.5 Marks	**Up To 5 Marks**	**Up To 7.5 Marks**	**Up To 10 marks**	**Up To 2.5 Marks**
Demonstrate an ability to listen and answer the viva questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.	Poor listening and communication skills. Failed to relate the programming skills needed for solving the problem.	Showed better communication skill by relating the problem with the programming skills acquired but the description showed serious errors.	Demonstrated good communication skills by relating the problem with the programming skills acquired with few errors.	Demonstrated excellent communication skills by relating the problem with the programming skills acquired and have been successful in tailoring the description.	Demonstrate an ability to listen and answer the Viva Questions related to programming skills needed for solving real-world problems in Computer Science and Engineering.

TABLE OF CONTENTS

GRAPHICS

GIMP

AUDACITY

WINDOWS MOVIE MAKER

SWISH

FLASH

PHOTO IMPACT

EX.NO.1.A	IMPLEMENTATION OF LINE DRAWING ALGORITHM DDA (DIGITAL DIFFERENTIAL ANALYSER) LINE

AIM:

To draw a line between two points using DDA Line Drawing Algorithm in Python.

ALGORITHM:

1. Start
2. Accept the two endpoints of the line.
3. Calculate the number of steps using dx and dy.
4. Calculate increment values for x and y.
5. Starting from the first point, keep adding the increments and plot each point.
6. Stop

PROGRAM:

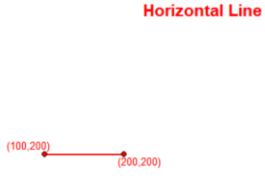
```
import tkinter as tk

print("--- LINE TYPE MENU ---")
print("1. Horizontal Line")
print("2. Vertical Line")
print("3. Forward Slanting (/)")
print("4. Backward Slanting (\\)")
choice = input("Enter your line type choice (1-4): ")

print("\nEnter starting point:")
x1 = int(input("x1: "))
y1 = int(input("y1: "))
length = int(input("Enter length of line: "))

if choice == '1':
    x2 = x1 + length
    y2 = y1
    label = "Horizontal Line"
elif choice == '2':
    x2 = x1
    y2 = y1 + length
    label = "Vertical Line"
elif choice == '3':
    x2 = x1 + length
    y2 = y1 - length
```

```
    label = "Forward Slanting (/)"
elif choice == '4':
    x2 = x1 + length
    y2 = y1 + length
    label = "Backward Slanting (\\)"
else:
    print("Invalid choice")
    exit()

print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
color_choice = input("Choose color (1-5): ")
color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
color = color_map.get(color_choice, 'black')

print("\nPattern Options: 1. Solid  2. Dashed  3. Dotted")
pattern_choice = input("Choose pattern (1-3): ")
pattern_map = {'1': 'solid', '2': 'dashed', '3': 'dotted'}
pattern = pattern_map.get(pattern_choice, 'solid')

root = tk.Tk()
root.title("DDA Line Drawing - No Functions")
canvas = tk.Canvas(root, width=600, height=600, bg="white")
canvas.pack()

canvas.create_text(300, 20, text=label, fill=color, font=("Helvetica", 16, "bold"))

dx = x2 - x1
dy = y2 - y1
steps = abs(dx) if abs(dx) > abs(dy) else abs(dy)
x_inc = dx / steps
y_inc = dy / steps

x = x1
y = y1

i = 0
while i <= steps:
    xi = round(x)
    yi = round(y)

    if pattern == 'solid':
        canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)
```

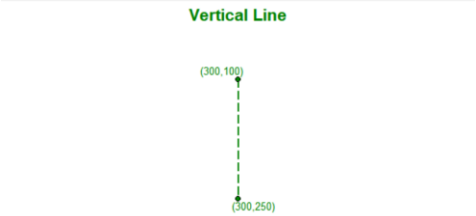
```
    elif pattern == 'dashed':
        if i % 15 < 10:
            canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)
    elif pattern == 'dotted':
        if i % 6 == 0:
            canvas.create_rectangle(xi, yi, xi+1, yi+1, outline=color)

    x += x_inc
    y += y_inc
    i += 1

canvas.create_oval(x1 - 3, y1 - 3, x1 + 3, y1 + 3, fill=color)
canvas.create_oval(x2 - 3, y2 - 3, x2 + 3, y2 + 3, fill=color)

canvas.create_text(x1 - 20, y1 - 10, text=f"({x1},{y1})", fill=color, font=("Arial", 10))
canvas.create_text(x2 + 20, y2 + 10, text=f"({x2},{y2})", fill=color, font=("Arial", 10))
root.mainloop()
```

OUTPUT:

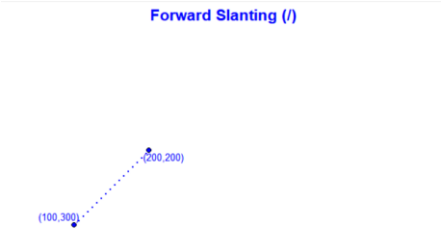
```
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 1
Enter starting point:
x1: 100
y1: 200
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1
```



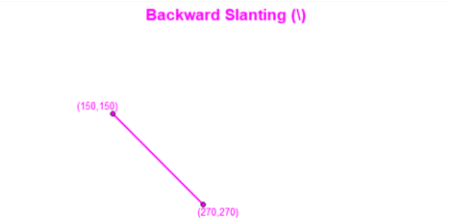
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 2
Enter starting point:
x1: 300
y1: 100
Enter length of line: 150
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 2



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 3
Enter starting point:
x1: 100
y1: 300
Enter length of line: 100
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 3



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 4
Enter starting point:
x1: 150
y1: 150
Enter length of line: 120
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1



Result:

Thus the Python program to draw a line using DDA algorithm is implemented and executed successfully.

EX.NO.1.B	IMPLEMENTATION OF LINE DRAWING ALGORITHM BRESENHAM'S LINE DRAWING ALGORITHM

AIM:

To draw a Line using Bresenham's Line drawing algorithm using Python program.

ALGORITHM:

1. Start
2. Accept the two endpoints of the line.
3. Initialize decision parameters to choose the next pixel.
4. Use only integer addition to move in the closest pixel direction.
5. Plot each point from start to end based on the decision value.
6. Stop

PROGRAM:

```
import tkinter as tk


print("--- LINE TYPE MENU ---")
print("1. Horizontal Line")
print("2. Vertical Line")
print("3. Forward Slanting (/)")
print("4. Backward Slanting (\\)")


choice = input("Enter your line type choice (1-4): ")


print("\nEnter Starting Point:")
x0 = int(input("x0: "))
y0 = int(input("y0: "))
length = int(input("Enter length of line: "))


print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
color_choice = input("Choose color (1-5): ")
color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
color = color_map.get(color_choice, 'black')
```

```
print("\nPattern Options: 1. Solid  2. Dashed  3. Dotted")
pattern_choice = input("Choose pattern (1-3): ")
pattern_map = {'1': 'solid', '2': 'dashed', '3': 'dotted'}
pattern = pattern_map.get(pattern_choice, 'solid')

if choice == '1':
    x1 = x0 + length
    y1 = y0
    label = "Horizontal Line"
elif choice == '2':
    x1 = x0
    y1 = y0 + length
    label = "Vertical Line"
elif choice == '3':
    x1 = x0 + length
    y1 = y0 - length
    label = "Forward Slanting (/)"
elif choice == '4':
    x1 = x0 + length
    y1 = y0 + length
    label = "Backward Slanting (\\)"
else:
    print("Invalid line type choice!")
    exit(

root = tk.Tk()
root.title("Bresenham Line Drawing - With Color & Pattern")
canvas = tk.Canvas(root, width=800, height=800, bg="white")
canvas.pack()

canvas.create_text(400, 20, text=label, fill=color, font=("Helvetica", 16, "bold"))
```

```
dx = abs(x1 - x0)
dy = abs(y1 - y0)
sx = 1 if x0 < x1 else -1
sy = 1 if y0 < y1 else -1
err = dx - dy

x = x0
y = y0
step_count = 0

while True:
    # Apply pattern
    if pattern == 'solid':
        canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)
    elif pattern == 'dashed':
        if step_count % 15 < 10:
            canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)
    elif pattern == 'dotted':
        if step_count % 6 == 0:
            canvas.create_rectangle(x, y, x + 1, y + 1, outline=color)

    if x == x1 and y == y1:
        break

    e2 = 2 * err
    if e2 > -dy:
        err -= dy
        x += sx
    if e2 < dx:
        err += dx
        y += sy
```

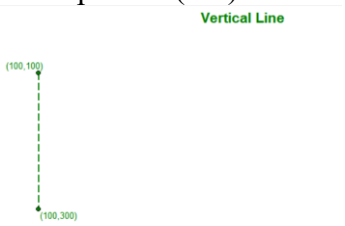
```
    step_count += 1


canvas.create_oval(x0 - 3, y0 - 3, x0 + 3, y0 + 3, fill=color)
canvas.create_oval(x1 - 3, y1 - 3, x1 + 3, y1 + 3, fill=color)


canvas.create_text(x0 - 20, y0 - 10, text=f"({x0},{y0})", fill=color, font=("Arial", 10))
canvas.create_text(x1 + 30, y1 + 10, text=f"({x1},{y1})", fill=color, font=("Arial", 10))
root.mainloop()
```

SAMPLE INPUT/OUTPUT:

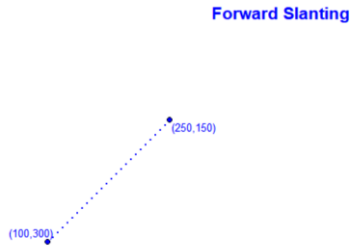
```
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 1
Enter Starting Point:
x1: 100
y1: 150
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1
```


```
--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 2
Enter Starting Point:
x1: 100
y1: 100
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2
Pattern Options: 1. Solid 2. Dashed 3. Dotted
```

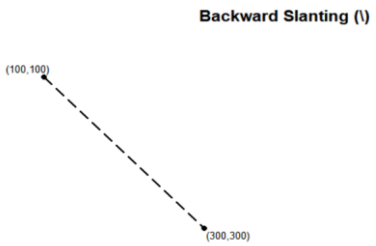
Choose pattern (1-3): 3



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 3
Enter Starting Point:
x1: 100
y1: 300
Enter length of line: 150
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 1



--- LINE TYPE MENU ---
1. Horizontal Line
2. Vertical Line
3. Forward Slanding (/)
4. Backward Slanding (\)
Enter your line type choice (1-4): 4
Enter Starting Point:
x1: 100
y1: 100
Enter length of line: 200
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4
Pattern Options: 1. Solid 2. Dashed 3. Dotted
Choose pattern (1-3): 2



RESULT:

Thus the Python program to draw a line using Bresenham's algorithm is implemented and executed successfully.

EX.NO.2(a)	Implementation of Midpoint Circle drawing Algorithm.

AIM:

To draw an Circle using Midpoint drawing algorithm using Python progam.

ALGORITHM:

1. Start
2. Accept radius and center point.
3. Start at the top of the circle (0, r).
4. Use a decision parameter to find the next pixel.
5. Use symmetry to draw eight points at once.
6. Continue until x > y.
7. Stop

PROGRAM:

```
import tkinter as tk

import math


def draw_point(canvas, xc, yc, x, y, color):
    for dx, dy in [(x, y), (-x, y), (x, -y), (-x, -y),
                   (y, x), (-y, x), (y, -x), (-y, -x)]:
        canvas.create_oval(xc+dx, yc+dy, xc+dx+1, yc+dy+1, fill=color, outline=color)


def midpoint_circle(canvas, xc, yc, r, color):
    x = 0
    y = r
    p = 1 - r
    draw_point(canvas, xc, yc, x, y, color)

    while x < y:
        x += 1
```

```
        if p < 0:
            p += 2 * x + 1
        else:
            y -= 1
            p += 2 * (x - y) + 1
        draw_point(canvas, xc, yc, x, y, color)

def draw_internal_pattern(canvas, xc, yc, r, pattern, color):
    if pattern == 'radial':
        for angle in range(0, 360, 15):
            rad = math.radians(angle)
            x = xc + int(r * math.cos(rad))
            y = yc + int(r * math.sin(rad))
            canvas.create_line(xc, yc, x, y, fill=color)

    elif pattern == 'horizontal':
        for dy in range(-r, r+1, 10):
            y = yc + dy
            span = int((r**2 - dy**2) ** 0.5)
            canvas.create_line(xc - span, y, xc + span, y, fill=color)

    elif pattern == 'vertical':
        for dx in range(-r, r+1, 10):
            x = xc + dx
            span = int((r**2 - dx**2) ** 0.5)
            canvas.create_line(x, yc - span, x, yc + span, fill=color)

    elif pattern == 'cross':
```

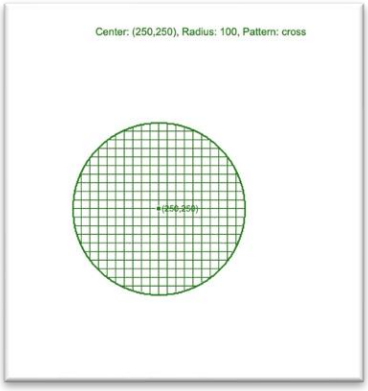
```
        draw_internal_pattern(canvas, xc, yc, r, 'horizontal', color)
        draw_internal_pattern(canvas, xc, yc, r, 'vertical', color)

def get_user_input():
    print("\n--- Midpoint Circle with Internal Patterns ---")
    xc = int(input("Enter center X: "))
    yc = int(input("Enter center Y: "))
    r = int(input("Enter radius: "))

    print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
    color_choice = input("Choose color (1-5): ")
    color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
    color = color_map.get(color_choice, 'black')

    print("\nPattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")
    pattern_choice = input("Choose internal pattern (1-5): ")
    pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}
    pattern = pattern_map.get(pattern_choice, 'none')

    return xc, yc, r, color, pattern

def draw_circle_with_pattern(xc, yc, r, color, pattern):
    root = tk.Tk()
    root.title("Circle with Patterns Inside")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    canvas.create_text(300, 20, text="Midpoint Circle with Patterns", font=("Helvetica", 16, "bold"))
```

```
    canvas.create_text(300, 45, text=f"Center: ({xc},{yc}), Radius: {r}, Pattern: {pattern}", font=("Arial", 12), fill=color)

    midpoint_circle(canvas, xc, yc, r, color)

    if pattern != 'none':
        draw_internal_pattern(canvas, xc, yc, r, pattern, color)

    canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
    canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)

    root.mainloop()

# Main loop
while True:
    user_input = get_user_input()
    draw_circle_with_pattern(*user_input)
    again = input("\nDraw another circle? (y/n): ")
    if again.lower() != 'y':
        break
```

OUTPUT:
--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2

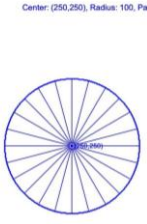
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 5



--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 3

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 2



--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

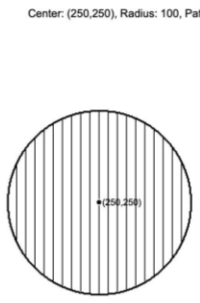
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3

--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

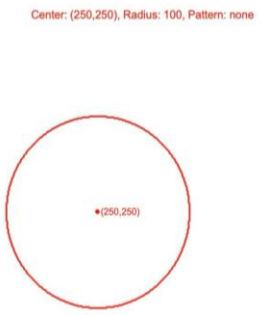
Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



--- MidPoint Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 1



RESULT:

Thus the program to draw an Circle using MidPoint Drawing algorithm is implemented and executed successfully.

EX.NO.2(b)	Implementation of Bresenham's Circle drawing Algorithm.

AIM:

To draw an Circle using Bresenham's drawing algorithm using Python progam.

ALGORITHM:

1. Start
2. Accept radius and center coordinates.
3. Initialize variables to track the circle's edge.
4. Use integer decision parameters to determine next pixel.
5. Plot all 8 symmetric points.
6. Loop until x > y.
7. Stop

PROGRAM:

```
import tkinter as tk

import math

print("\n--- Bresenham Circle with Patterns ---")

xc = int(input("Enter center X: "))

yc = int(input("Enter center Y: "))

r = int(input("Enter radius: "))


print("\nColor Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")

color_choice = input("Choose color (1-5): ")

color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}

color = color_map.get(color_choice, 'black')


print("\nPattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")

pattern_choice = input("Choose internal pattern (1-5): ")

pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}

pattern = pattern_map.get(pattern_choice, 'none')
```

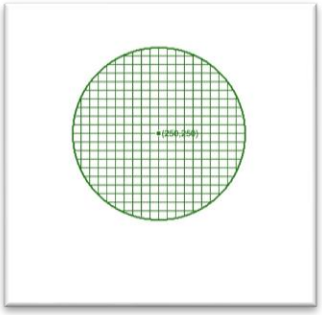
```
root = tk.Tk()
root.title("Bresenham Circle with Patterns")
canvas = tk.Canvas(root, width=600, height=600, bg="white")
canvas.pack()

x = 0
y = r
d = 3 - 2 * r

points = [(x, y), (-x, y), (x, -y), (-x, -y),
          (y, x), (-y, x), (y, -x), (-y, -x)]
for dx, dy in points:
    canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)

while x <= y:
    x += 1
    if d <= 0:
        d = d + 4 * x + 6
    else:
        y -= 1
        d = d + 4 * (x - y) + 10

    points = [(x, y), (-x, y), (x, -y), (-x, -y),
              (y, x), (-y, x), (y, -x), (-y, -x)]
    for dx, dy in points:
        canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)
```

```
if pattern == 'radial':
    for angle in range(0, 360, 15):
        rad = math.radians(angle)
        x1 = xc + int(r * math.cos(rad))
        y1 = yc + int(r * math.sin(rad))
        canvas.create_line(xc, yc, x1, y1, fill=color)

elif pattern == 'horizontal':
    for dy in range(-r, r+1, 10):
        y = yc + dy
        span = int((r**2 - dy**2) ** 0.5)
        canvas.create_line(xc - span, y, xc + span, y, fill=color)

elif pattern == 'vertical':
    for dx in range(-r, r+1, 10):
        x = xc + dx
        span = int((r**2 - dx**2) ** 0.5)
        canvas.create_line(x, yc - span, x, yc + span, fill=color)

elif pattern == 'cross':

    for dy in range(-r, r+1, 10):
        y = yc + dy
        span = int((r**2 - dy**2) ** 0.5)
        canvas.create_line(xc - span, y, xc + span, y, fill=color)

    for dx in range(-r, r+1, 10):
        x = xc + dx
```

```
        span = int((r**2 - dx**2) ** 0.5)
        canvas.create_line(x, yc - span, x, yc + span, fill=color)

canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)

root.mainloop()
```

OUTPUT:

```
--- Bresenham's Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta
Choose color (1-5): 2

Pattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross
Choose internal pattern (1-5): 5
```



```
--- Bresenham's Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta
Choose color (1-5): 3

Pattern Inside Circle: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross
Choose internal pattern (1-5): 2
```

---Bresenham’s Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

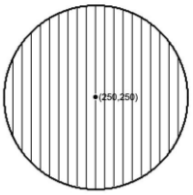
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3

--- Bresenham’s Circle with Patterns ---
Enter center X: 250
Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



--- Bresenham’s Circle with Patterns ---
Enter center X: 250

Enter center Y: 250
Enter radius: 100

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1

Pattern Inside Circle: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 1

RESULT:

Thus the program to draw an Circle using Bresenham's Drawing algorithm is implemented and executed successfully.

EX.NO.3	Implementation of Bresenham's Ellipse Drawing Algorithm.

AIM:

To draw a Ellipse Using a Bresenham's ellipse drawing algorithm.

ALGORITHM:

1. Start
2. Accept center and radii along X and Y axes.
3. Divide the drawing into two regions (slope < 1 and > 1).
4. Use decision parameters to move across pixels in both regions.
5. Plot symmetric points in all 4 quadrants.
6. Continue until full ellipse is drawn.
7. Stop

PROGRAM:

```
import tkinter as tk
import math

def draw_ellipse_points(canvas, xc, yc, x, y, color):
    points = [(x, y), (-x, y), (x, -y), (-x, -y)]
    for dx, dy in points:
        canvas.create_oval(xc + dx, yc + dy, xc + dx + 1, yc + dy + 1, fill=color, outline=color)

def bresenham_ellipse(canvas, xc, yc, rx, ry, color):
    x = 0
    y = ry

    rx2 = rx * rx
    ry2 = ry * ry
    two_rx2 = 2 * rx2
    two_ry2 = 2 * ry2

    px = 0
    py = two_rx2 * y
    p1 = ry2 - (rx2 * ry) + (0.25 * rx2)

    while px < py:
        draw_ellipse_points(canvas, xc, yc, x, y, color)
```

```
            x += 1
            px += two_ry2
            if p1 < 0:
                p1 += ry2 + px
            else:
                y -= 1
                py -= two_rx2
                p1 += ry2 + px - py

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

        while y >= 0:
            draw_ellipse_points(canvas, xc, yc, x, y, color)
            y -= 1
            py -= two_rx2
            if p2 > 0:
                p2 += rx2 - py
            else:
                x += 1
                px += two_ry2
                p2 += rx2 - py + px

    def draw_internal_pattern(canvas, xc, yc, rx, ry, pattern, color):
        if pattern == 'radial':
            for angle in range(0, 360, 15):
                rad = math.radians(angle)
                x = xc + int(rx * math.cos(rad))
                y = yc + int(ry * math.sin(rad))
                canvas.create_line(xc, yc, x, y, fill=color)
        elif pattern == 'horizontal':
            for dy in range(-ry, ry + 1, 10):
                y = yc + dy
                span = int(rx * ((1 - (dy / ry) ** 2) ** 0.5))
                canvas.create_line(xc - span, y, xc + span, y, fill=color)
        elif pattern == 'vertical':
            for dx in range(-rx, rx + 1, 10):
                x = xc + dx
                span = int(ry * ((1 - (dx / rx) ** 2) ** 0.5))
                canvas.create_line(x, yc - span, x, yc + span, fill=color)
        elif pattern == 'cross':
            draw_internal_pattern(canvas, xc, yc, rx, ry, 'horizontal', color)
            draw_internal_pattern(canvas, xc, yc, rx, ry, 'vertical', color)
```

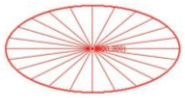
```
def get_user_input():
    xc = int(input("Enter center X: "))
    yc = int(input("Enter center Y: "))
    rx = int(input("Enter X radius (horizontal): "))
    ry = int(input("Enter Y radius (vertical): "))

    print("Color Options: 1. Red  2. Green  3. Blue  4. Black  5. Magenta")
    color_choice = input("Choose color (1-5): ")
    color_map = {'1': 'red', '2': 'green', '3': 'blue', '4': 'black', '5': 'magenta'}
    color = color_map.get(color_choice, 'black')

    print("Pattern Inside Ellipse: 1. None  2. Radial  3. Horizontal  4. Vertical  5. Cross")
    pattern_choice = input("Choose internal pattern (1-5): ")
    pattern_map = {'1': 'none', '2': 'radial', '3': 'horizontal', '4': 'vertical', '5': 'cross'}
    pattern = pattern_map.get(pattern_choice, 'none')

    return xc, yc, rx, ry, color, pattern

def draw_bresenham_ellipse_with_pattern(xc, yc, rx, ry, color, pattern):
    root = tk.Tk()
    root.title("Bresenham Ellipse with Patterns")
    canvas = tk.Canvas(root, width=700, height=700, bg="white")
    canvas.pack()

    bresenham_ellipse(canvas, xc, yc, rx, ry, color)

    if pattern != 'none':
        draw_internal_pattern(canvas, xc, yc, rx, ry, pattern, color)

    canvas.create_oval(xc - 3, yc - 3, xc + 3, yc + 3, fill=color)
    canvas.create_text(xc + 25, yc, text=f"({xc},{yc})", font=("Arial", 10), fill=color)

    root.mainloop()

while True:
    details = get_user_input()
    draw_bresenham_ellipse_with_pattern(*details)
    again = input("Draw another ellipse? (y/n): ")
    if again.lower() != 'y':
        break
```

OUTPUT:

--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 60

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 1

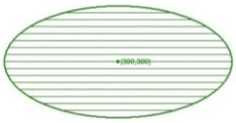
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 2



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 80

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 2

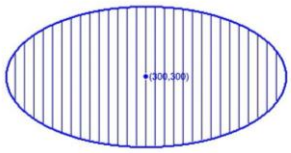
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 3



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 150
Enter Y radius (vertical): 75

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 3

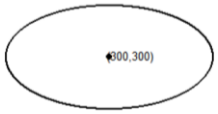
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 4



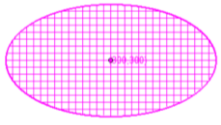
-- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 120
Enter Y radius (vertical): 60

Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 4

Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 1



--- Bresenham Ellipse Drawing with Patterns ---
Enter center X: 300
Enter center Y: 300
Enter X radius (horizontal): 150
Enter Y radius (vertical): 80
Color Options: 1. Red 2. Green 3. Blue 4. Black 5. Magenta
Choose color (1-5): 5
Pattern Inside Ellipse: 1. None 2. Radial 3. Horizontal 4. Vertical 5. Cross
Choose internal pattern (1-5): 5



RESULT:

Thus the program to draw line, circle and ellipse attributes is implemented and executed successfully.

EX.NO.4	TWO DIMENSIONAL TRANSFORMATIONS - TRANSLATION, ROTATION, SCALING, REFLECTION AND SHEAR

AIM:

To implement 2D transformations like Translation, Rotation, Scaling, Reflection, Shear using Python program.

ALGORITHM:

1. Start
2. Input the object's original coordinates.
3. Apply translation by adding Tx and Ty.
4. Apply scaling by multiplying with Sx and Sy.
5. Apply rotation using angle θ.
6. Reflect across X, Y, or origin by changing signs.
7. Apply shearing using shearing factors.
8. Display the transformed object.
9. Stop

PROGRAM:

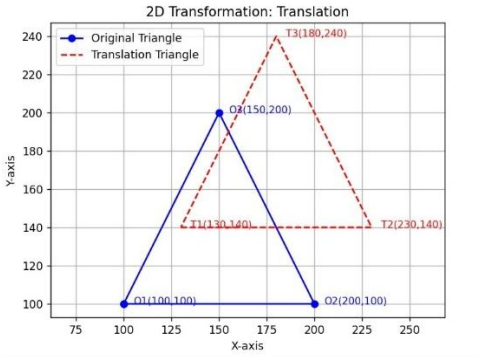
```
import matplotlib.pyplot as plt
import math

def plot_triangle(original, transformed, transformation_name):
    ox, oy = zip(*original)
    tx, ty = zip(*transformed)

    # Close triangles
    ox += (ox[0],)
    oy += (oy[0],)
    tx += (tx[0],)
    ty += (ty[0],)

    plt.figure()
    plt.title(f"2D Transformation: {transformation_name}")
    plt.plot(ox, oy, 'bo-', label='Original Triangle')
    plt.plot(tx, ty, 'r--', label=f'{transformation_name} Triangle')

    # Show vertex coordinates
    for i, (x, y) in enumerate(original):
        plt.text(x + 5, y, f"O{i+1}({x},{y})", fontsize=9, color='blue')
```

```
    for i, (x, y) in enumerate(transformed):
        plt.text(x + 5, y, f"T{i+1}({x},{y})", fontsize=9, color='red')

    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.axis("equal")
    plt.show()

def translate(triangle, tx, ty):
    return [(x + tx, y + ty) for x, y in triangle]

def rotate(triangle, angle):
    rad = math.radians(angle)
    return [(round(x * math.cos(rad) - y * math.sin(rad)),
             round(x * math.sin(rad) + y * math.cos(rad))) for x, y in triangle]

def scale(triangle, sx, sy):
    return [(x * sx, y * sy) for x, y in triangle]

def shear(triangle, shx, shy):
    return [(x + shx * y, y + shy * x) for x, y in triangle]

def reflect(triangle, axis):
    if axis == 'x':
        return [(x, -y) for x, y in triangle]
    elif axis == 'y':
        return [(-x, y) for x, y in triangle]
    elif axis == 'xy':
        return [(-x, -y) for x, y in triangle]
    else:
        print("Invalid axis for reflection.")
        return triangle

# Main
triangle = [(100, 100), (200, 100), (150, 200)]

print("\n--- 2D Transformation Menu ---")
print("1. Translation")
print("2. Rotation")
print("3. Scaling")
```

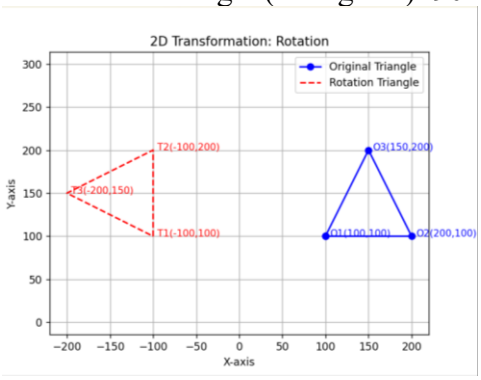
```
print("4. Shearing")
print("5. Reflection")
choice = input("Enter your choice (1-5): ")

if choice == '1':
    tx = int(input("Enter translation in X (tx): "))
    ty = int(input("Enter translation in Y (ty): "))
    transformed = translate(triangle, tx, ty)
    plot_triangle(triangle, transformed, "Translation")

elif choice == '2':
    angle = int(input("Enter rotation angle (in degrees): "))
    transformed = rotate(triangle, angle)
    plot_triangle(triangle, transformed, "Rotation")

elif choice == '3':
    sx = float(input("Enter scaling factor in X (sx): "))
    sy = float(input("Enter scaling factor in Y (sy): "))
    transformed = scale(triangle, sx, sy)
    plot_triangle(triangle, transformed, "Scaling")

elif choice == '4':
    shx = float(input("Enter shearing factor in X (shx): "))
    shy = float(input("Enter shearing factor in Y (shy): "))
    transformed = shear(triangle, shx, shy)
    plot_triangle(triangle, transformed, "Shearing")

elif choice == '5':
    axis = input("Enter reflection axis (x/y/xy): ").lower()
    transformed = reflect(triangle, axis)
    plot_triangle(triangle, transformed, "Reflection")

else:
    print("Invalid choice.")
```

OUTPUT:

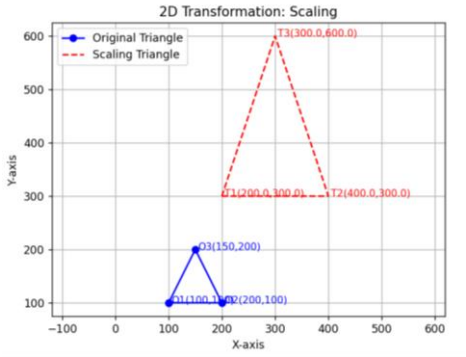
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 1
Enter translation in X (tx): 30
Enter translation in Y (ty): 40



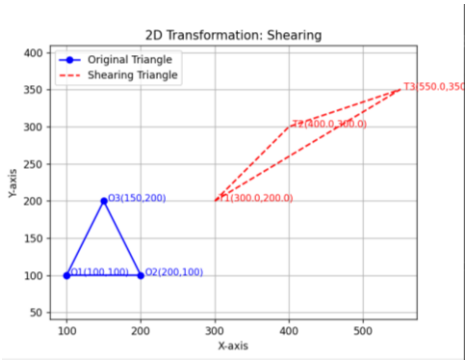
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 2
Enter rotation angle (in degrees): 90



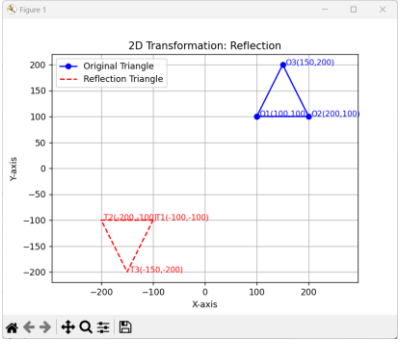
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 3
Enter scaling factor in X (sx): 2
Enter scaling factor in Y (sy): 3



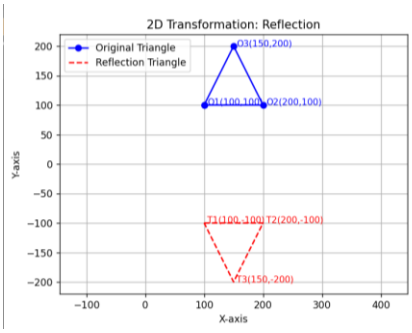
--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 4
Enter shearing factor in X (shx): 2
Enter shearing factor in Y (shy): 1



--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 5
Enter reflection axis (x/y/xy): xy



--- 2D Transformation Menu ---
1. Translation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
Enter your choice (1-5): 5
Enter reflection axis (x/y/xy): x



RESULT:

Thus the program to implement 2D transformations is executed successfully.

EX.NO.5 A	IMPLEMMENTATION OF 2D LINE CLIPPING ALGORITHM - COHEN-SUTHERLAND 2D LINE CLIPPING ALGORITHM

AIM:

To implement Cohen-Sutherland 2D line clipping algorithm using Python program.

ALGORITHM:

1. Start
2. Define clipping window and input line endpoints.
3. Assign region codes to both endpoints.
4. If both codes are 0000 → accept.
5. If logical AND of codes ≠ 0000 → reject.
6. Else, calculate intersection and repeat checks.
7. Stop

PROGRAM:

```
import tkinter as tk

INSIDE, LEFT, RIGHT, BOTTOM, TOP = 0, 1, 2, 4, 8

def compute_code(x, y, x_min, y_min, x_max, y_max):
    code = INSIDE
    if x < x_min: code |= LEFT
    elif x > x_max: code |= RIGHT
    if y < y_min: code |= BOTTOM
    elif y > y_max: code |= TOP
    return code

def cohen_sutherland_clip(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
    code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)
    accept = False

    while True:
        if code1 == 0 and code2 == 0:
            accept = True
            break
        elif (code1 & code2) != 0:
```

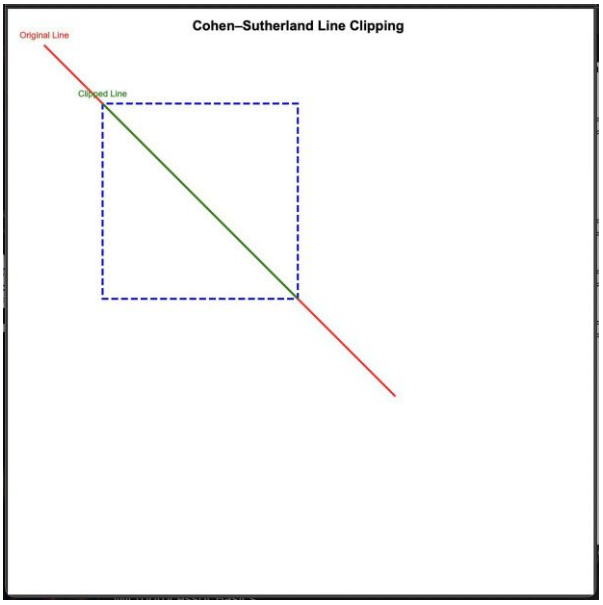
```
            break
        else:
            code_out = code1 if code1 != 0 else code2
            if code_out & TOP:
                x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1)
                y = y_max
            elif code_out & BOTTOM:
                x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1)
                y = y_min
            elif code_out & RIGHT:
                y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1)
                x = x_max
            elif code_out & LEFT:
                y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1)
                x = x_min

            if code_out == code1:
                x1, y1 = x, y
                code1 = compute_code(x1, y1, x_min, y_min, x_max, y_max)
            else:
                x2, y2 = x, y
                code2 = compute_code(x2, y2, x_min, y_min, x_max, y_max)

    return accept, x1, y1, x2, y2

def draw_result(x1, y1, x2, y2, clipped, x_min, y_min, x_max, y_max):
    root = tk.Tk()
    root.title("Cohen-Sutherland Line Clipping")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    # Draw clipping window
    canvas.create_rectangle(x_min, y_min, x_max, y_max, outline="blue", dash=(5, 5), width=2)
    canvas.create_text(300, 20, text="Cohen–Sutherland Line Clipping", fill="black", font=("Arial", 14,
"bold"))

    # Original line in red
```

```
    canvas.create_line(orig_x1, orig_y1, orig_x2, orig_y2, fill="red", width=2)
    canvas.create_text(orig_x1, orig_y1 - 10, text="Original Line", fill="red", font=("Arial", 9))

    if clipped:
        # Clipped line in green
        canvas.create_line(x1, y1, x2, y2, fill="green", width=2)
        canvas.create_text(x1, y1 - 10, text="Clipped Line", fill="green", font=("Arial", 9))
    else:
        canvas.create_text(300, 560, text="Line Rejected (Outside Window)", fill="red", font=("Arial", 12,
"bold"))

    root.mainloop()

# -------- Console Input --------
print("Enter Line Coordinates:")
orig_x1 = int(input("x1: "))
orig_y1 = int(input("y1: "))
orig_x2 = int(input("x2: "))
orig_y2 = int(input("y2: "))

print("\nEnter Clipping Window:")
x_min = int(input("x_min: "))
y_min = int(input("y_min: "))
x_max = int(input("x_max: "))
y_max = int(input("y_max: "))

# Clip the line
accept, cx1, cy1, cx2, cy2 = cohen_sutherland_clip(orig_x1, orig_y1, orig_x2, orig_y2, x_min, y_min,
x_max, y_max)

# Draw output
draw_result(cx1, cy1, cx2, cy2, accept, x_min, y_min, x_max, y_max)
```

OUTPUT:

Enter Line Coordinates:
x1: 40
y1: 40
x2: 400
y2: 400

Enter Clipping Window:
x_min: 100
y_min: 100
x_max: 300
y_max: 300



RESULT:

Thus the program to implement Cohen-Sutherland 2D line clipping is executed successfully.

E X.NO. 5 B IMPLEMENTATION OF 2D LINE CLIPPING ALGORITHM: LIANG-BARSKY ALGORITHM.

AIM:

To implement the 2D Line using a liang-barsky algorithm.

ALGORITHM:

1. Start
2. Accept endpoints and clipping window.
3. Convert line to parametric form.
4. Calculate p and q values for edges.
5. Determine entry and exit points using t values.
6. If valid t values, clip and draw the line.
7. Stop

PROGRAM:

```
import tkinter as tk

def liang_barsky(x1, y1, x2, y2, x_min, y_min, x_max, y_max):
    dx = x2 - x1
    dy = y2 - y1
    p = [-dx, dx, -dy, dy]
    q = [x1 - x_min, x_max - x1, y1 - y_min, y_max - y1]

    u1 = 0.0
    u2 = 1.0

    for i in range(4):
        if p[i] == 0:
            if q[i] < 0:
                return False, x1, y1, x2, y2
        else:
            u = q[i] / p[i]
            if p[i] < 0:
                if u > u1:
                    u1 = u
            else:
                if u < u2:
                    u2 = u

    if u1 > u2:
        return False, x1, y1, x2, y2

    x1_clip = x1 + u1 * dx
```

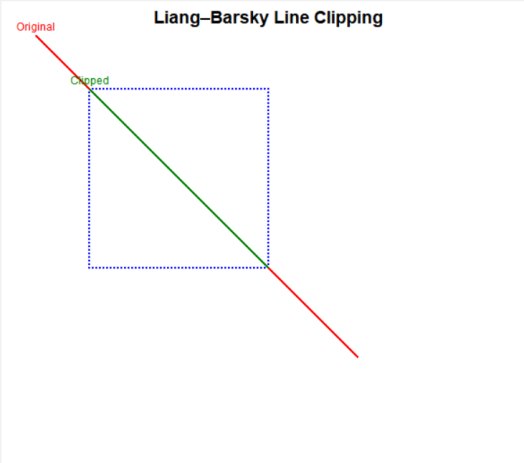
```
    y1_clip = y1 + u1 * dy
    x2_clip = x1 + u2 * dx
    y2_clip = y1 + u2 * dy

    return True, x1_clip, y1_clip, x2_clip, y2_clip

def draw_result(x1, y1, x2, y2, clipped, x_min, y_min, x_max, y_max):
    root = tk.Tk()
    root.title("Liang–Barsky Line Clipping Algorithm")
    canvas = tk.Canvas(root, width=600, height=600, bg="white")
    canvas.pack()

    # Draw clipping rectangle
    canvas.create_rectangle(x_min, y_min, x_max, y_max, outline="blue", dash=(5, 5), width=2)
    canvas.create_text(300, 20, text="Liang–Barsky Line Clipping", fill="black", font=("Arial", 14,
"bold"))

    # Draw original line in red
    canvas.create_line(orig_x1, orig_y1, orig_x2, orig_y2, fill="red", width=2)
    canvas.create_text(orig_x1, orig_y1 - 10, text="Original", fill="red", font=("Arial", 9))

    if clipped:
        # Draw clipped line in green
        canvas.create_line(x1, y1, x2, y2, fill="green", width=2)
        canvas.create_text(x1, y1 - 10, text="Clipped", fill="green", font=("Arial", 9))
    else:
        canvas.create_text(300, 560, text="Line Rejected (Outside)", fill="red", font=("Arial", 12, "bold"))

    root.mainloop()

# -------- Console Input --------
print("Enter Line Coordinates:")
orig_x1 = int(input("x1: "))
orig_y1 = int(input("y1: "))
orig_x2 = int(input("x2: "))
orig_y2 = int(input("y2: "))

print("\nEnter Clipping Window:")
x_min = int(input("x_min: "))
y_min = int(input("y_min: "))
x_max = int(input("x_max: "))
y_max = int(input("y_max: "))

# Clip the line using Liang-Barsky
accept, cx1, cy1, cx2, cy2 = liang_barsky(orig_x1, orig_y1, orig_x2, orig_y2, x_min, y_min, x_max,
y_max)

# Show result
```

draw_result(cx1, cy1, cx2, cy2, accept, x_min, y_min, x_max, y_max)

OUTPUT:

Enter Line Coordinates:
x1: 40
y1: 40
x2: 400
y2: 400

Enter Clipping Window:
x_min: 100
y_min: 100
x_max: 300
y_max: 300



RESULT:

Thus the python program to implement the 2D Line using a liang-barsky algorithm was successfully executed.

EX.NO.6	POLYGON CLIPPING USING SUTHERLAND-HODGEMAN ALGORITHM

AIM:

To implement Sutherland-Hodgeman polygon clipping algorithm using Python program.

ALGORITHM:

1. Start
2. Input polygon vertices and clipping window.
3. Clip the polygon edge-by-edge for all 4 window edges.
4. Check each edge: inside/outside combinations.
5. Add resulting points to output list.
6. After final edge, draw the clipped polygon.
7. Stop

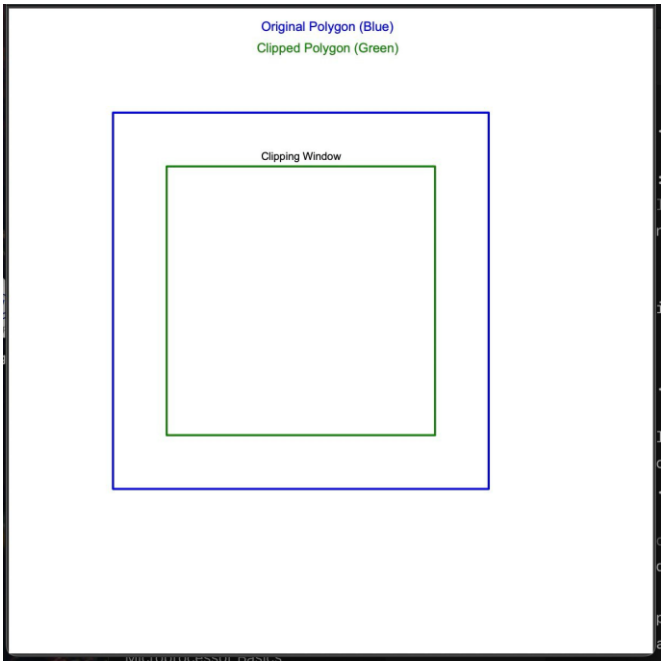
PROGRAM:

```
import tkinter as tk

# Clipping boundaries
clip_window = {'xmin': 150, 'ymin': 150, 'xmax': 400, 'ymax': 400}

def inside(p, edge):
    x, y = p
    if edge == 'LEFT':
        return x >= clip_window['xmin']
    elif edge == 'RIGHT':
        return x <= clip_window['xmax']
    elif edge == 'BOTTOM':
        return y >= clip_window['ymin']
    elif edge == 'TOP':
        return y <= clip_window['ymax']

def intersect(p1, p2, edge):
    x1, y1 = p1
    x2, y2 = p2

    if edge == 'LEFT':
        x = clip_window['xmin']
        y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
    elif edge == 'RIGHT':
        x = clip_window['xmax']
        y = y1 + (y2 - y1) * (x - x1) / (x2 - x1)
    elif edge == 'BOTTOM':
```

```
        y = clip_window['ymin']
        x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
    elif edge == 'TOP':
        y = clip_window['ymax']
        x = x1 + (x2 - x1) * (y - y1) / (y2 - y1)
    return (x, y)

def suth_hodg_clip(polygon, edges=['LEFT', 'RIGHT', 'BOTTOM', 'TOP']):
    output_list = polygon
    for edge in edges:
        input_list = output_list
        output_list = []
        if not input_list:
            break
        s = input_list[-1]

        for e in input_list:
            if inside(e, edge):
                if inside(s, edge):
                    output_list.append(e)
                else:
                    output_list.append(intersect(s, e, edge))
                    output_list.append(e)
            elif inside(s, edge):
                output_list.append(intersect(s, e, edge))
            s = e
    return output_list

def draw_polygon(canvas, points, outline_color, fill_color="", width=2):
    if len(points) >= 2:
        canvas.create_polygon(points, outline=outline_color, fill=fill_color, width=width)

def draw_clipping_window(canvas):
    canvas.create_rectangle(
        clip_window['xmin'], clip_window['ymin'],
        clip_window['xmax'], clip_window['ymax'],
        outline='black', dash=(4, 2)
    )
    canvas.create_text(275, 140, text="Clipping Window", fill="black", font=('Arial', 10))

def main():
    # Sample polygon
    polygon_points = [(100, 100), (450, 100), (450, 450), (100, 450)]

    root = tk.Tk()
    root.title("Sutherland-Hodgman Polygon Clipping")

    canvas = tk.Canvas(root, width=600, height=600, bg='white')
    canvas.pack()
```

```
    draw_clipping_window(canvas)
    draw_polygon(canvas, polygon_points, outline_color="blue", fill_color="", width=2)
    canvas.create_text(300, 20, text="Original Polygon (Blue)", fill="blue", font=('Arial', 12))

    # Perform clipping
    clipped_polygon = suth_hodg_clip(polygon_points)

    if clipped_polygon:
        draw_polygon(canvas, clipped_polygon, outline_color="green", fill_color="", width=2)
        canvas.create_text(300, 40, text="Clipped Polygon (Green)", fill="green", font=('Arial', 12))

    root.mainloop()

main()
```

OUTPUT:



RESULT:

Thus Polygon Clipping using Sutherland Hodgeman algorithm is implemented and output is verified.

EX.NO.7	THREE DIMENSIONAL TRANSFORMATIONS – TRANSLATION, ROTATION AND SCALING

AIM:

To implement 3D transformations using Python program.

ALGORITHM:

1. Start
2. Accept coordinates of the 3D object.
3. Apply translation by adding Tx, Ty, Tz.
4. Apply scaling using Sx, Sy, Sz.
5. Rotate around X, Y, or Z using rotation matrices.
6. Display transformed object using 3D plotting.
7. Stop

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define a cube (8 vertices)
cube = np.array([
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [0, 1, 1],
    [1, 0, 0],
    [1, 0, 1],
    [1, 1, 0],
    [1, 1, 1]
])

# Define edges for visualization
edges = [
    (0,1), (0,2), (0,4), (1,3), (1,5),
    (2,3), (2,6), (3,7), (4,5), (4,6),
    (5,7), (6,7)
]

def plot_3d(original, transformed, title):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title(title)
```

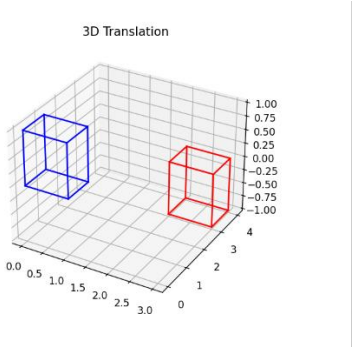
```
    # Draw original in blue
    for edge in edges:
        p1 = original[edge[0]]
        p2 = original[edge[1]]
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], color='blue')

    # Draw transformed in red
    for edge in edges:
        p1 = transformed[edge[0]]
        p2 = transformed[edge[1]]
        ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], color='red')

    plt.show()

# Transformation Functions
def translate(cube, tx, ty, tz):
    matrix = np.array([
        [1, 0, 0, tx],
        [0, 1, 0, ty],
        [0, 0, 1, tz],
        [0, 0, 0, 1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def scale(cube, sx, sy, sz):
    matrix = np.array([
        [sx, 0,  0,  0],
        [0,  sy, 0,  0],
        [0,  0,  sz, 0],
        [0,  0,  0,  1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_x(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
        [1, 0,           0,          0],
        [0, np.cos(angle), -np.sin(angle), 0],
        [0, np.sin(angle),  np.cos(angle), 0],
        [0, 0,           0,          1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_y(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
```

```
        [ np.cos(angle), 0, np.sin(angle), 0],
        [ 0,             1, 0,             0],
        [-np.sin(angle), 0, np.cos(angle), 0],
        [ 0,             0, 0,             1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

def rotate_z(cube, angle_deg):
    angle = np.radians(angle_deg)
    matrix = np.array([
        [np.cos(angle), -np.sin(angle), 0, 0],
        [np.sin(angle),  np.cos(angle), 0, 0],
        [0,              0,             1, 0],
        [0,              0,             0, 1]
    ])
    points = np.hstack((cube, np.ones((cube.shape[0], 1))))
    return (matrix @ points.T).T[:, :3]

# Menu
while True:
    print("\n--- 3D Transformation Menu ---")
    print("1. Translation")
    print("2. Scaling")
    print("3. Rotation about X-axis")
    print("4. Rotation about Y-axis")
    print("5. Rotation about Z-axis")
    print("6. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        tx = float(input("Translate X by: "))
        ty = float(input("Translate Y by: "))
        tz = float(input("Translate Z by: "))
        new_cube = translate(cube, tx, ty, tz)
        plot_3d(cube, new_cube, "3D Translation")
    elif choice == 2:
        sx = float(input("Scale X by: "))
        sy = float(input("Scale Y by: "))
        sz = float(input("Scale Z by: "))
        new_cube = scale(cube, sx, sy, sz)
        plot_3d(cube, new_cube, "3D Scaling")
    elif choice == 3:
        angle = float(input("Enter rotation angle (X-axis): "))
        new_cube = rotate_x(cube, angle)
        plot_3d(cube, new_cube, "Rotation about X-axis")
    elif choice == 4:
        angle = float(input("Enter rotation angle (Y-axis): "))
```

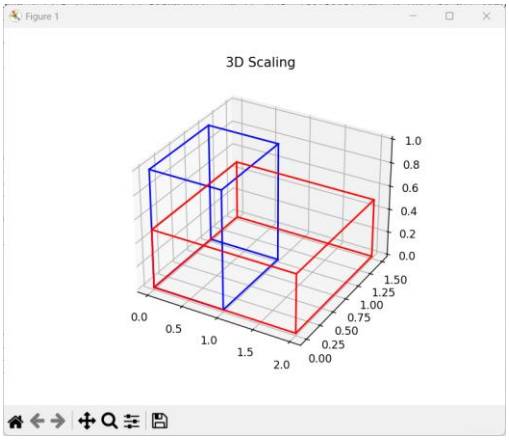
```
        new_cube = rotate_y(cube, angle)
        plot_3d(cube, new_cube, "Rotation about Y-axis")
    elif choice == 5:
        angle = float(input("Enter rotation angle (Z-axis): "))
        new_cube = rotate_z(cube, angle)
        plot_3d(cube, new_cube, "Rotation about Z-axis")
    elif choice == 6:
        print("Exiting program.")
        break
    else:
        print("Invalid choice. Try again.")
```

OUTPUT:

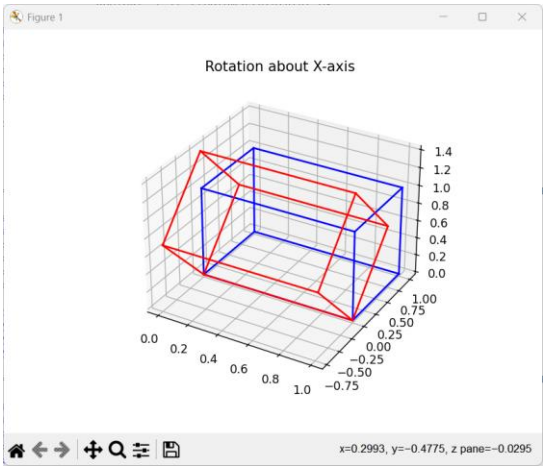
```
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 1
Translate X by: 2
Translate Y by: 3
Translate Z by: -1
```



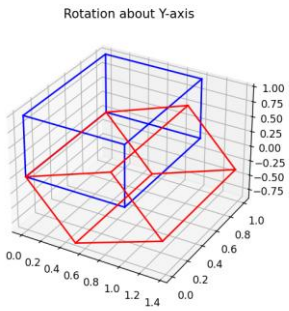

```
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 2
Scale X by: 2
Scale Y by: 1.5
Scale Z by: 0.5
```



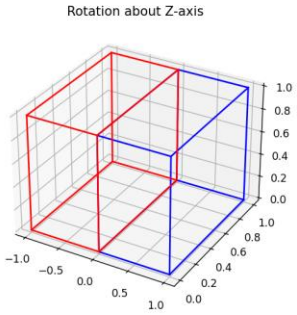
--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 3
Enter rotation angle (X-axis): 45



--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 4
Enter rotation angle (Y-axis): 60



--- 3D Transformation Menu ---
1. Translation
2. Scaling
3. Rotation about X-axis
4. Rotation about Y-axis
5. Rotation about Z-axis
6. Exit
Enter your choice: 5
Enter rotation angle (Z-axis): 90



RESULT:

Thus, the 3D Transformations on a cube are implemented and the output is verified.

EX. NO. 8A IMPLEMENTATION OF 2D ANIMATION (USING TIMER, LOOP, SIMPLE ANIMATION) – BOUNCING BALL)

AIM:

To implement the 2D Animation (using timer, Loop, Simple animation) Bouncing Ball

ALGORITHM:

1. Start
2. Draw the ball at initial position.
3. Use loop or timer to move it.
4. Detect boundaries and reverse direction if needed.
5. Redraw ball in new position continuously.
6. Stop

PROGRAM:

```
import time
import os

# Console window size (box for ball to bounce inside)
width = 40
height = 10

# Ball properties
x = 2
y = 2
dx = 1
dy = 1

def print_ball(x, y):
    os.system('cls' if os.name == 'nt' else 'clear')  # Clear console
    for i in range(height):
        for j in range(width):
            if i == y and j == x:
                print("O", end="")  # Ball
            else:
                print(" ", end="")
        print()
    print(f"Ball position: ({x},{y})")

# Animation loop
while True:
    print_ball(x, y)
    time.sleep(0.1)  # Delay

    # Update position
```

```
    x += dx
    y += dy

    # Check for bounce
    if x <= 0 or x >= width - 1:
        dx = -dx
    if y <= 0 or y >= height - 1:
        dy = -dy
```

OUTPUT:

RESULT:

Thus the above Python program to generate fractal images is executed.

EX. NO. 9	DRAWING THREE DIMENSIONAL OBJECTS AND SCENES USING OPENGL

AIM:

To generate 3D objects and scenes using OpenGl functions

ALGORITHM:

Step 1: Start the program.

Step 2: Include GL/glut.h.

Step 3: Initialize the glut functions.

Step 4: Enable the necessary GL-flags.

Step 5: Draw two icosahedrons.

Step 6: Apply rendering for icosahedrons.

Step 7: Apply blending effects to icosahedrons.

Step 8: Stop the program.

PROCEDURE:

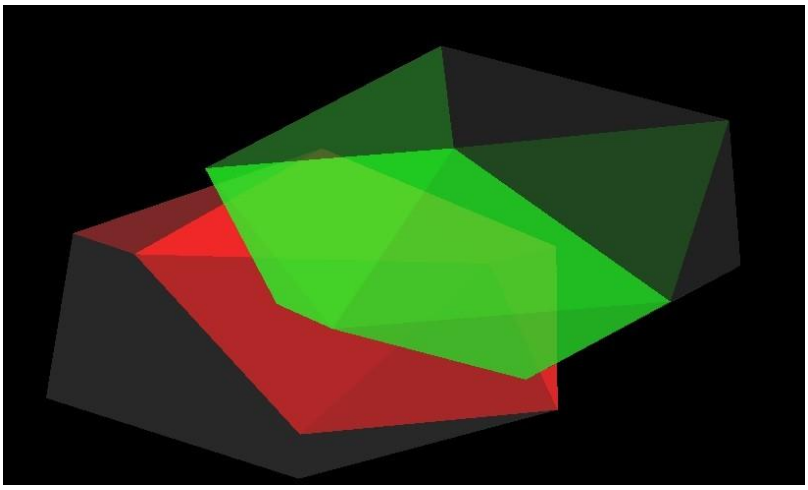
```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
GLfloat light0_ambient[] ={0.2, 0.2, 0.2, 1.0};
GLfloat light0_diffuse[] ={0.0, 0.0, 0.0, 1.0};
GLfloat light1_diffuse[] ={1.0, 0.0, 0.0, 1.0};
GLfloat light1_position[] ={1.0, 1.0, 1.0, 0.0};
GLfloat light2_diffuse[] ={0.0, 1.0, 0.0, 1.0};
GLfloat light2_position[] ={-1.0, -1.0, 1.0, 0.0};
float s = 0.0;
GLfloat angle1 = 0.0, angle2 = 0.0;
void
output(GLfloat x, GLfloat y, char *text)
{
  char *p;
```

```
  glPushMatrix();
  glTranslatef(x, y, 0);
  for (p = text; *p; p++)
    glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
  glPopMatrix();
}
Void display(void)
{
  static GLfloat amb[] ={0.4, 0.4, 0.4, 0.0};
  static GLfloat dif[] ={1.0, 1.0, 1.0, 0.0};
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_LIGHT1);
  glDisable(GL_LIGHT2);
  amb[3] = dif[3] = cos(s) / 2.0 + 0.5;
  glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, dif);
  glPushMatrix();
  glTranslatef(-0.3, -0.3, 0.0);
  glRotatef(angle1, 1.0, 5.0, 0.0);
  glCallList(1);        /* render ico display list */
  glPopMatrix();
  glClear(GL_DEPTH_BUFFER_BIT);
  glEnable(GL_LIGHT2);
  glDisable(GL_LIGHT1);
  amb[3] = dif[3] = 0.5 - cos(s * .95) / 2.0;
  glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, dif);
  glPushMatrix();
  glTranslatef(0.3, 0.3, 0.0);
  glRotatef(angle2, 1.0, 0.0, 5.0);
  glCallList(1);        /* render ico display list */
  glPopMatrix();
  glPushAttrib(GL_ENABLE_BIT);
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_LIGHTING);
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluOrtho2D(0, 1500, 0, 1500);
  glMatrixMode(GL_MODELVIEW);
```

```
  glPushMatrix();
  glLoadIdentity();
  /* Rotate text slightly to help show jaggies. */
  glRotatef(4, 0.0, 0.0, 1.0);
  output(200, 225, "This is antialiased.");
  glDisable(GL_LINE_SMOOTH);
  glDisable(GL_BLEND);
  output(160, 100, "This text is not.");
  glPopMatrix();
  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glPopAttrib();
  glMatrixMode(GL_MODELVIEW);
  glutSwapBuffers();
}
void
idle(void)
{
  angle1 = (GLfloat) fmod(angle1 + 0.8, 360.0);
  angle2 = (GLfloat) fmod(angle2 + 1.1, 360.0);
  s += 0.05;
  glutPostRedisplay();
}
void
visible(int vis)
{
  if (vis == GLUT_VISIBLE)
    glutIdleFunc(idle);
  else
    glutIdleFunc(NULL);
}
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("blender");
  glutDisplayFunc(display);
  glutVisibilityFunc(visible);
  glNewList(1, GL_COMPILE);  /* create ico display list */
  glutSolidIcosahedron();
```

```
  glEndList();
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
  glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
  glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
  glLightfv(GL_LIGHT2, GL_DIFFUSE, light2_diffuse);
  glLightfv(GL_LIGHT2, GL_POSITION, light2_position);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_CULL_FACE);
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glEnable(GL_LINE_SMOOTH);
  glLineWidth(2.0);
  glMatrixMode(GL_PROJECTION);
  gluPerspective( /* field of view in degree */ 40.0,/* aspect ratio */ 1.0,/* Z near */ 1.0,
   /* Z far    */ 10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 5.0,  /* eye is at (0,0,5) */0.0, 0.0, 0.0,      /* center is at (0,0,0) */0.0,
   1.0, 0.);      /* up is in positive Y direction */
  glTranslatef(0.0, 0.6, -1.0);
glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```

OUTPUT:



RESULT:

Thus the 3D objects and scenes are successfully generated using openGL functions.

Ex. No. 10	CREATING CHESS BOARD USING OPENGL

AIM:

To create a Chess Board using OpenGl functions.

ALGORITHM:

Step 1: Start the program.

Step 2: Include GL/glut.h.

Step 3: Initialize the glut functions.

Step 4: Enable the necessary GL-flags.

Step 5: Draw a square box.

Step 6: Repeat step5 to form squares in 8 rows and 8 columns constituting a chessboard.

Step 7: Stop.

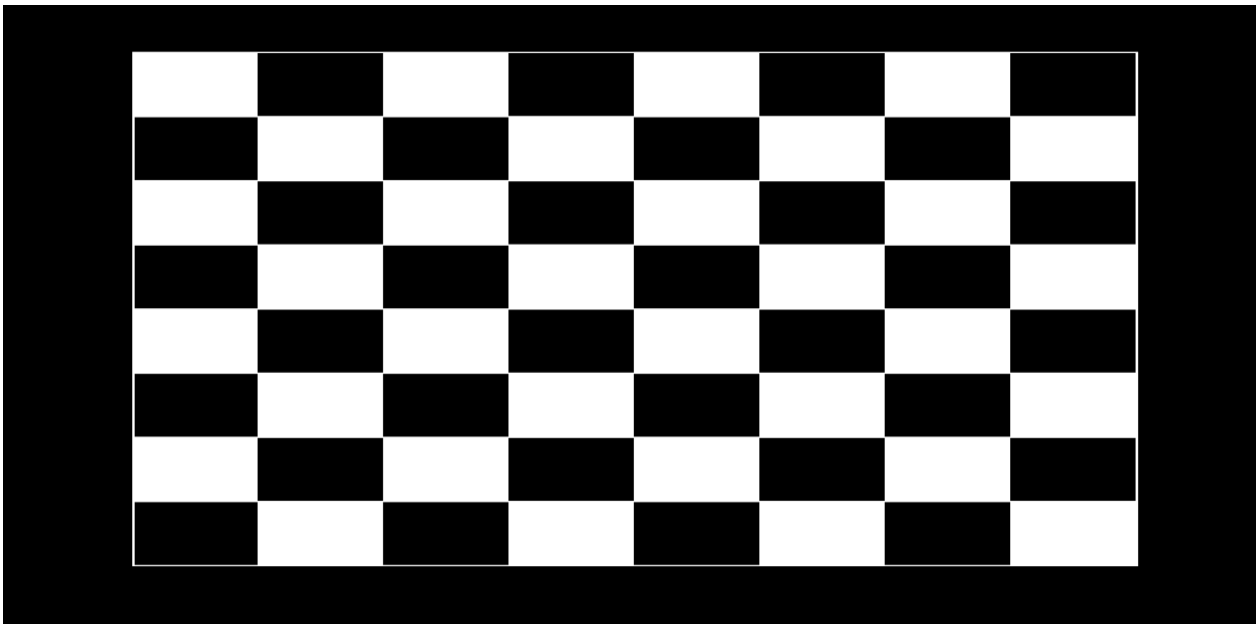
PROGRAM:

```
 #include<windows.h>
#include<glut.h>
int c = 0;
void init()
{
// For displaying the window color
glClearColor(0, 1, 1, 0);
// Choosing the type of projection
glMatrixMode(GL_PROJECTION);
// for setting the transformation which here is 2D
gluOrtho2D(0, 800, 0,600);
}

void drawSquare(GLint x1, GLint y1, GLint x2, GLint y2, GLint x3, GLint y3, GLint x4, GLint
y4)
{
// if color is 0 then draw white box and change value of color = 1
```

```
if (c == 0)
{
glColor3f(1, 1, 1); // white color value is 1 1 1
c = 1;
}
// if color is 1 then draw black box and change value of color = 0
else
{
glColor3f(0, 0, 0); // black color value is 0 0 0
c = 0;
}

// Draw Square
glBegin(GL_POLYGON);
glVertex2i(x1, y1);
glVertex2i(x2, y2);
glVertex2i(x3, y3);
glVertex2i(x4, y4);
glEnd();
}
void chessboard()
{
glClear(GL_COLOR_BUFFER_BIT); // Clear display window
GLint x, y;

for (x = 0; x <= 800; x += 100)
{
for (y = 0; y <= 600; y += 75)
{
drawSquare(x, y + 75, x + 100, y + 75, x + 100, y, x, y);
}
}
// Process all OpenGL routine s as quickly as possible
glFlush();
}

int main(int agrc, char ** argv)
{
// Initialize GLUT
glutInit(&agrc, argv);
```

```
// Set display mode
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
// Set top - left display window position.
glutInitWindowPosition(100, 100);
// Set display window width and height
glutInitWindowSize(800, 600);
// Create display window with the given title
glutCreateWindow("Chess Board using OpenGL in C++");
// Execute initialization procedure
init();
// Send graphics to display window
glutDisplayFunc(chessboard);
// Display everything and wait.
glutMainLoop();
}
```

OUTPUT:



RESULT:

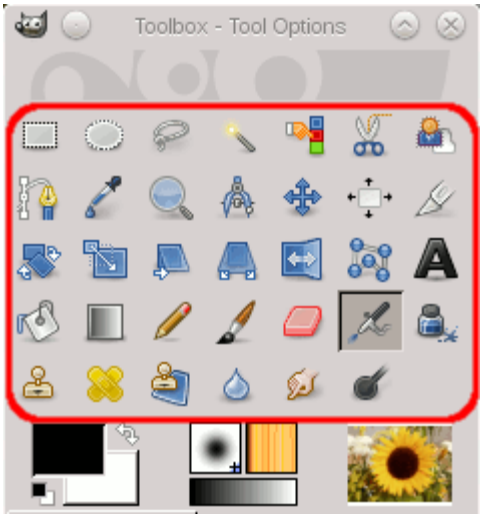
Thus the program to create a chess board is executed successfully.

INTRODUCTION TO GIMP

GIMP provides a comprehensive toolbox in order to quickly perform basic tasks such as making selections or drawing paths. The many tools contained within GIMP's toolbox are discussed in detail here.

GIMP has a diverse assortment of tools that let you perform a large variety of tasks. The tools can be thought of as falling into five categories:

- *Selection tools*, which specify or modify the portion of the image that will be affected by subsequent actions;
- *Paint tools*, which alter the colors in some part of the image;
- *Transform tools*, which alter the geometry of the image;
- *Color tools*, which alter the distribution of colors across the entire image;
- *Other tools*, which don't fall into the other four categories.



Most tools can be activated by clicking on an icon in the Toolbox. By default, some tools are accessible only via the menus (namely the Color tools are accessible only either as Colors or as Tools → Colors). Every tool, in fact, can be activated from the *Tools* menu; also, every tool can be activated from the keyboard using an accelerator key.

In the default setup, created when GIMP is first installed, not all tools show icons in the Toolbox: the Color tools are omitted. You can customize the set of tools that are shown in the Toolbox through Edit → Preferences → Toolbox. There are two reasons you might want to do this: first, if you only rarely use a tool, it might be easier to find the tools you want if the distracting icon is removed; second, if you use the Color tools a lot, you might find it convenient to have icons for them easily available. In any case, regardless of the Toolbox, you can always access any tool at any time using the Tools menu from an image menubar.

The shape of the cursor changes when it is inside an image, to one that indicates which tool is active (if in Preferences you have set Image Windows → Mouse Pointers → Pointer mode → Tool icon).

EX.NO.1	LOGO CREATION

AIM :

To create a logo using Gimp tool.

ALGORITHM / PROCEDURE:

Step 1 : Open GIMP icon.

Step 2 : Select file -> new.

Step 3 : Set height, width and resolution of your logo.

Step 4 : Select Bucket fill tool -> select pattern fill->select any pattern and click on your workspace to set background of your logo.

Step 5 : Select text tool and type text for your logo.

Step 6 : Select color icon to change color of your text.

Step 7 : To insert an image select file -> open as layers and select your image and use scale tool to resize image.

Step 8 : If you want to add more effects on your logo select ->filters ->select any options.

Step 9 : After designing your logo select file ->export ->select file format and location and save your logo.

OUTPUT:

RESULT:

Thus the logo is created successfully using GIMP.

EX.NO.2	TEXT ANIMATION

AIM:

To create text animation using GIMP.

ALGORITHM / PROCEDURE:

Step 1: Open GIMP icon.

Step 2: Select file -> new.

Step 3: Set height, width and resolution of your workspace.

Step 4: Select Bucket fill tool -> select pattern fill->select any pattern and click on your workspace to set background.

Step 5: Select text tool and type texts.

Step 6: Select color icon to change color of your text.

Step 7: Select filter ->animation->select any option(blend, spinning globe,waves,rippling,burn-in) to animate text.

Step 8: Select filter ->play back.

Step 9: To save file->export->select file format->save. Applying spinning globe.

OUTPUT:

RESULT:

Thus text animation is created successfully using GIMP.

INTRODUCTION TO AUDACITY

Control Tool Bar:

Editing tools:

Selection tool – For selecting the range of audio you want to edit or listen to

Envelope tool – For changing the volume over driver

Draw tool – For modifying individual samples

Zoom tool– for zooming in and out.

Time shift tool – For sliding tracks left or right

Multi tool – lets you access all of these tools at once depending on the location of the mouse and the keys you are holding down.

Audio control button:

Skip to Start – moves the cursor to time 0. If you press Play at this point, you project will play from the beginning.

Play – starts playing audio at the cursor position. If some audio is selected, only the selection is played.

Loop –if you hold down the Shift key, the Play button changes to a Loop button, which lets you keep playing the selection over

Record – starts recording audio at the project sample rate (the sample rate in the lower–left corner of the window). The new track begin at the current cursor position, so click the "Skip to Start" button first if you want the track to begin at time 0.

Pause – temporarily stops playback or recording until you press pause again.

Stop – stops recording or playing. You must do this before applying effects, saving or exporting.

Skip to End – moves the cursor to the end of the last track.

Mixer tool bar:

The Mixer Toolbar has three controls, used to set the voh1me levels of your audio device and choose the input source. The leftmost slider controls the output voh1me, the other slider controls the recording voh1me, and the control on the right lets you choose the input source (such as "Microphone", "Line In", "Audio CD", etc.). Use the Record Level Meter to set the correct level
Changing these controls has no effect on the audio data in you· project - in other words it doesn't matter what the output volume level is when you Export or Save a project - the end result is the same.

Edit toolbar:

All of the buttons on this toolbar perform action - and with couple of exceptions, they're on just shortcuts of existing menu items to save you
Holding the mouse over a tool will show a “tooltip” in case you forgot which one is which.

 Cut.

 Copy.

 Paste.

 Trim away the audio outside the selection.

 Silence the selected audio.

 Undo.

 Redo.

 Zoom In.

 Zoom out.

 Fit selection in window - zooms until the selection just fits inside the window.

 Fit project in window - zooms until all of the audio just fits inside the window.

EX.NO.1	SILENCING TRIMMING AND DUPLICATING AUDIO SIGNAL

AIM:

To apply silence, trim and duplicate effects for a selected audio signal.

ALGORITHM / PROCEDURE:

A. Silencing:

1. Open Audacity and open a new audio file.
2. Select a particular region to be silence and go to Edit -› Silence or Ctrl + L.
3. To duplicate a particular part, select the part and go to Edit -›Copy and paste it in any place of the audio track.
4. Play the track using the play button in the toolbar.
5. Save the audio file.

B. Trimming :

1. Open Audacity and open a new audio file.
2. To zoom in to get a closer look at the waveform, first choose the *Selection Tool* , then click near the point you're interested in, then click the *Zoom In* button to see the detail you need.
3. Select a particular region to be trimmed using button.
4. Select  cut icon from the toolbar.
5. Play the track using the play button in the toolbar.

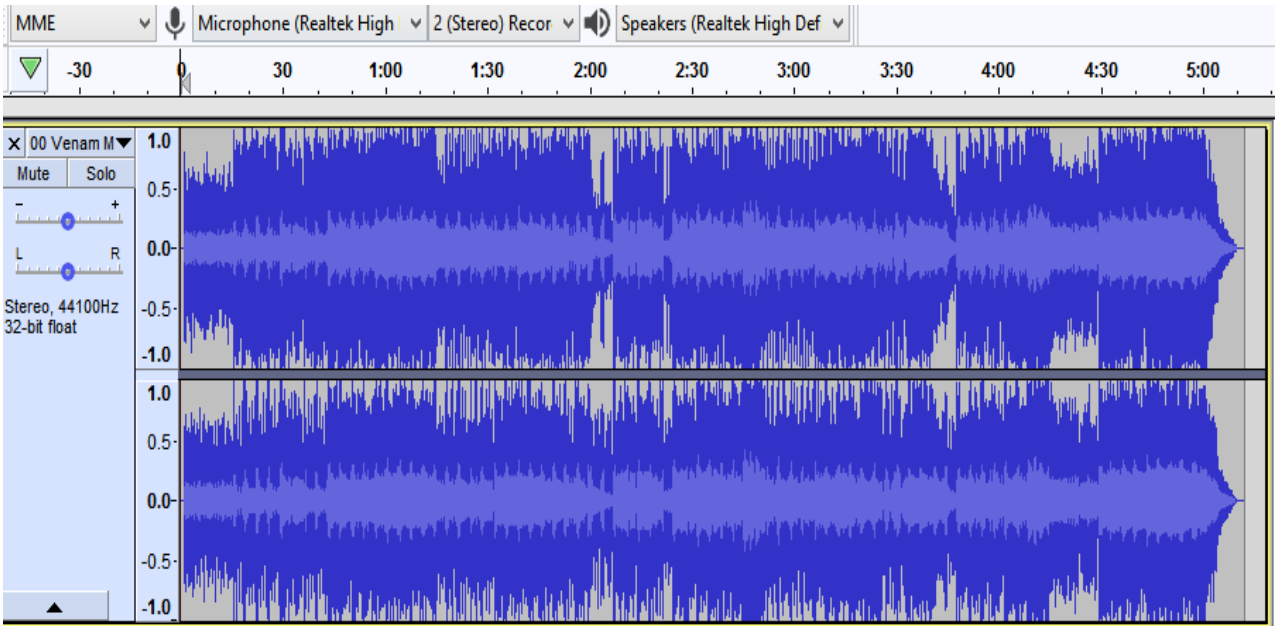
6. Save the audio signal.

C. Duplicating :

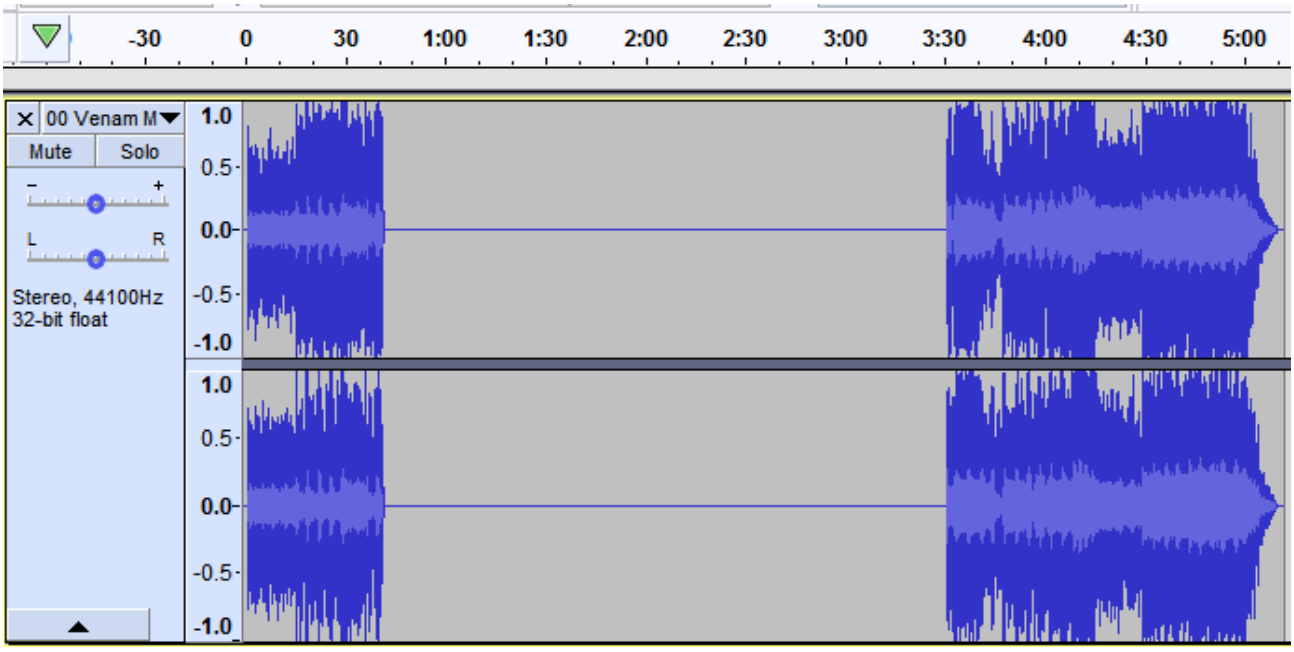
1. Open Audacity and open a new audio file.
2. To zoom in to get a closer look at the waveform, first choose the *Selection Tool* I , then click near the point you're interested in, then click the *Zoom In* button to see the detail you need.
3. Select a particular region to be duplicated using I button.
4. Play the track using the play button in the toolbar.
5. Save the audio signal.

OUTPUT:

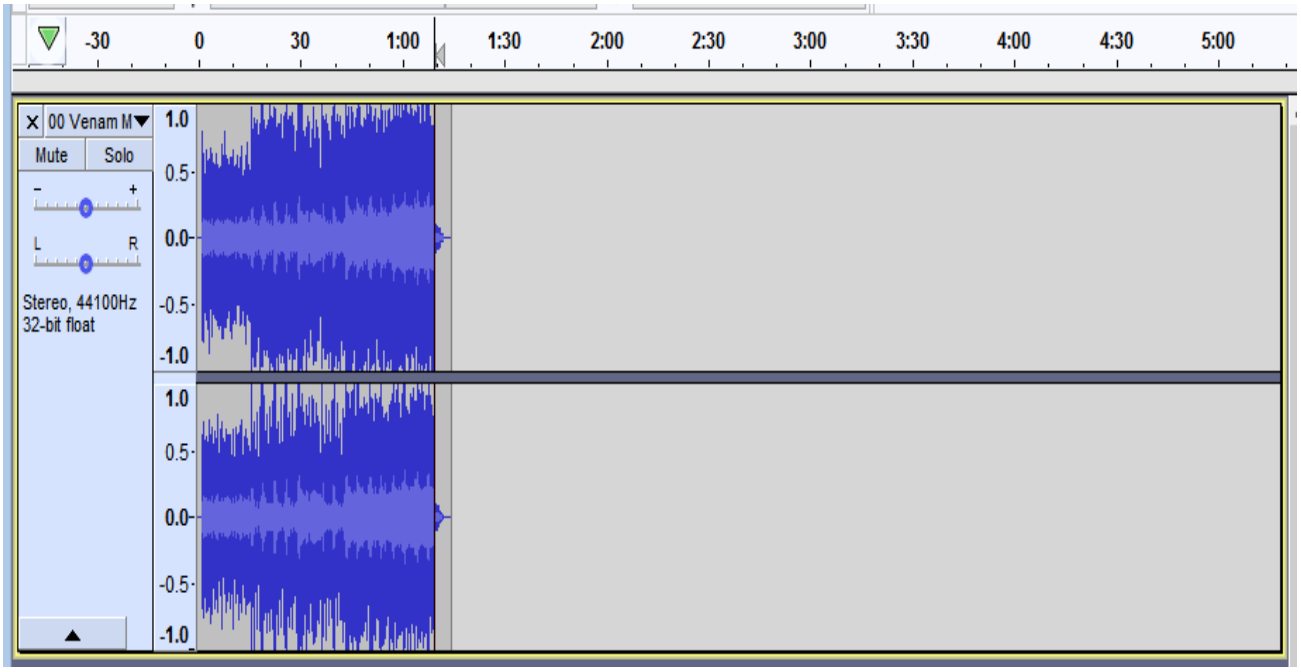
i) Original audio signal:



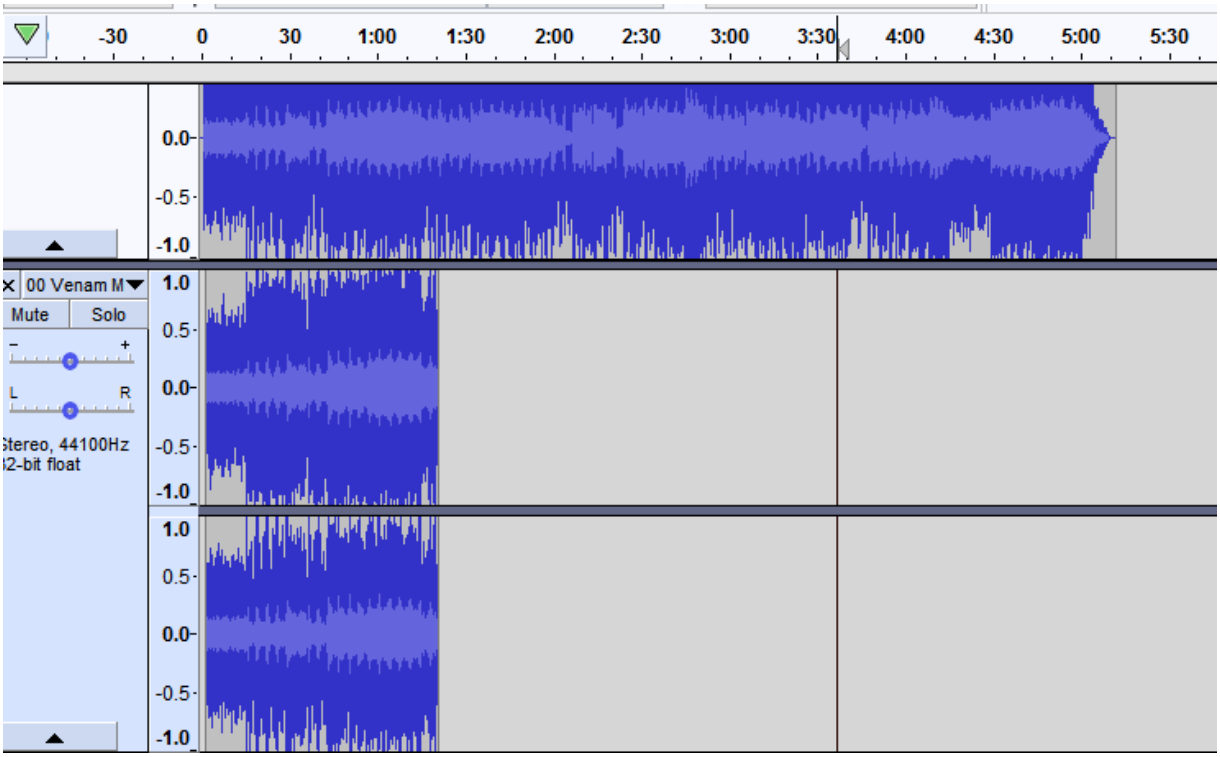
ii) After Silencing



iii) After Trimming:



C) Duplicating



RESULT :

Thus an audio source has been duplicated and silenced at a particular part using Audacity.

EX.NO.2	ADVANCED EFFECTS TO AUDIO SIGNAL

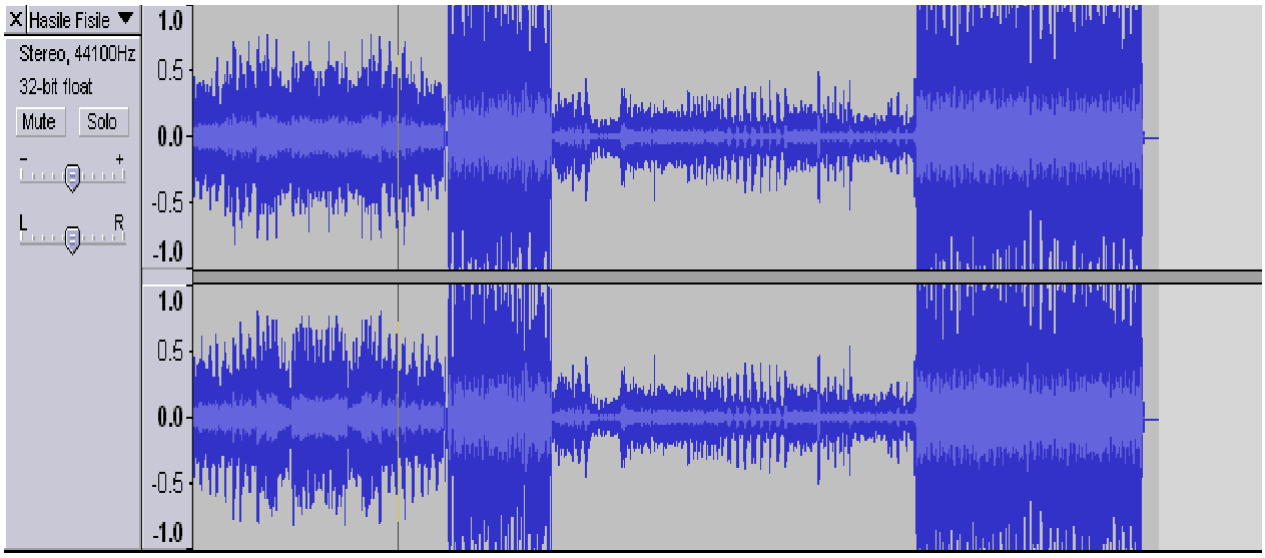
AIM :

To apply various audio transitions to an audio signal.

ALGORITHM / PROCEDURE:

1. Open Audacity and open a new audio track.
2. Select a part of it and go to Effect-›Reverse. Play to hear the effect.
3. Select another part of the track and go to Effect -› WahWah and set the value as required. Play to hear the effect.
4. Apply various effects of user defined.
5. Audio effects have now been created.

OUTPUT :



RESULT:

Thus various effects have been applied to an audio signal.

WINDOWS MOVIE MAKER

EX.NO.1	APPLICATION OF SOME EFFECTS TO VIDEO

AIM:

To apply some effects to video using windows movie maker.

ALGORITHM / PROCEDURE:

Step 1: Open movie maker

Step 2: To add video and images select add videos and photos icon.

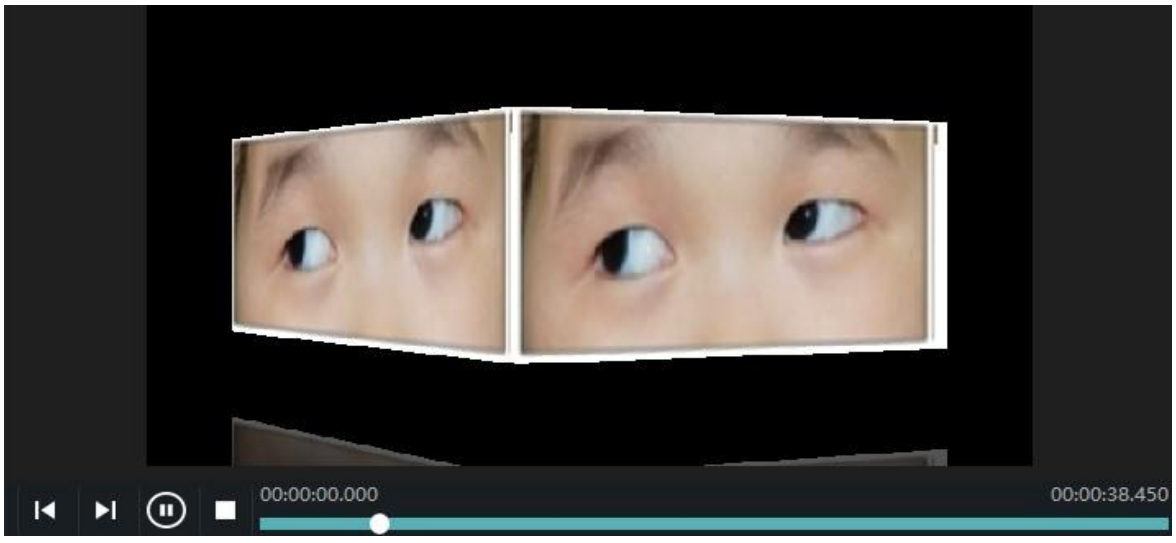
Step 3: To add music select add music icon.

Step 4: Your selected images/videos and music displayed on bottom of your window. here you can manage your movie.

Step 5: To apply effects click animation and visual effects icon.

Step 6: After applying all effects select export to save.

OUTPUT:



Original video

After applying grayscale effect

RESULT:

Thus the effect to video is implemented successfully using windows movie maker.

EX.NO.2	CREATING TITLES IN VIDEO

AIM:

To create titles in videos using windows movie maker.

ALGORITHM / PROCEDURE:

Step 1: Open movie maker

Step 2: To add video and images select add videos and photos icon.

Step 3: To add effects on your video refer the procedure of previous exercise.

Step 4: Select text/credit icon->select title->add title for your movie

Step 5: To save click export from file and name your movie and then save.

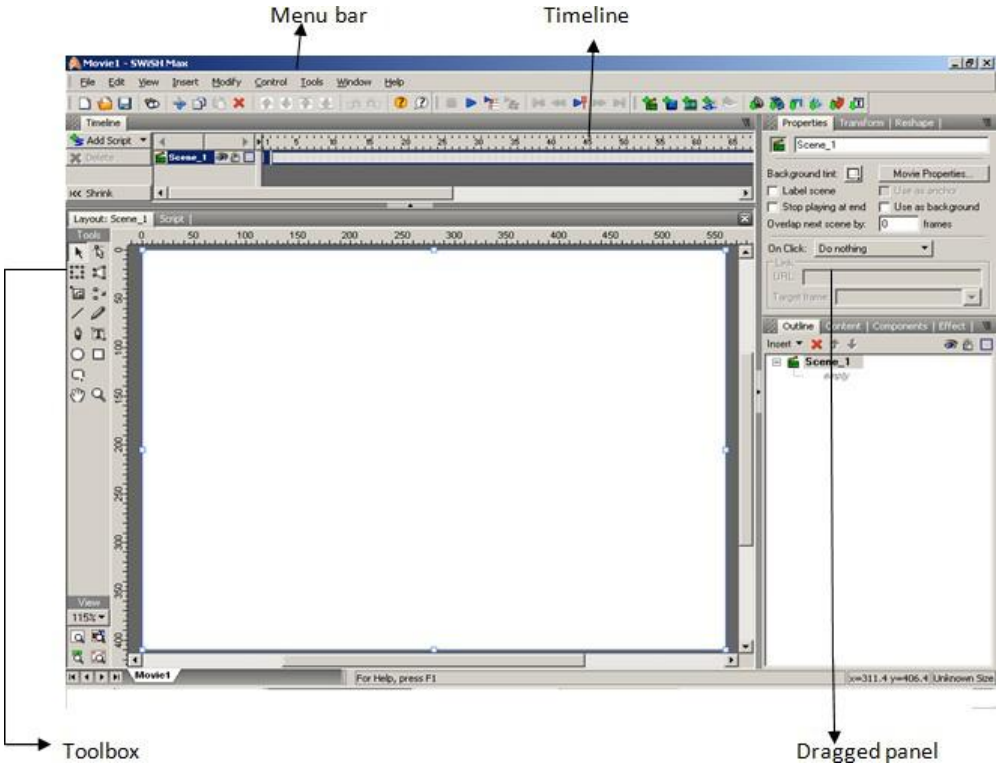
OUTPUT:

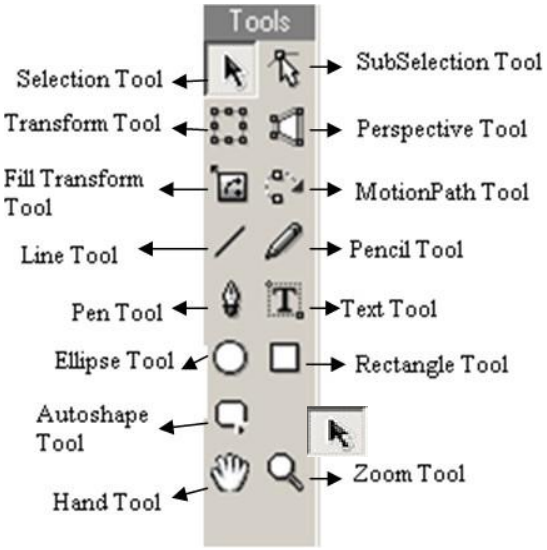
RESULT:

Thus titles in videos is created successfully using windows movie maker.

INTRODUCTION TO SWISH MAX 2

Swish Max 2 is easy to use and produces complex animations with text, images, graphics, video and sound. Swish Max has tools for creating lines, rectangles, ellipses, vector and freehand curves, motion paths, movie clips, rollover buttons, and input forms all in an intuitive easy-to-use interface. Swish Max also includes a large range of preset multimedia effects, components and vector art.





Tools:

Tools are modal commands that determine what Swish Max does when you click and drag the mouse on the workspace. You can select a tool by clicking one of the options in the Tool box, which is located in the top-left of the Layout panel. You can only select one tool at a time.

Selection Tool (V)

The selection tool is used for selecting entire Objects and transforming graphic objects by moving or reshaping the object.

Subselection Tool  **(A)**

It reshapes graphic Objects by clicking and dragging the vertices the object outline path.

Transform Tool  **(Q)**

It transforms graphic Objects by clicking and dragging.

Perspective Tool (I) 

Dragging the handles provided modifies the Perspective of the Object.

Fill Transform (F) 

It transforms an Object's gradient or image fill without transforming the Object.

Motion Path (M) 

It plots a Motion Path for the selected Object by clicking and dragging.

Line Tool (L) 

It draws a line.

Pencil Tool (N) 

It draws a freehand line object.

Pen Tool (P) 

The Pen tool draws a set of connected curves or line segments.

Text Tool (T) 

The Text tool draws a Text Object.

Ellipse Tool (E) 

It draws an ellipse or circle.

Rectangle Tool (R) 

It draws a rectangle or square.

AutoShapes 

AutoShapes allows you to draw a number of predetermined shapes.

Hand Tool (H) 

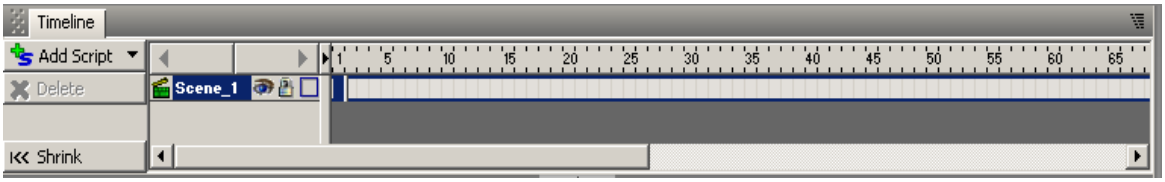
Pans around the workspace in the Layout pannel by clicking and dragging.

Zoom Tool (Z) 

It Zooms in/out of the workspace in the Layout pannel using the mouse.

Timeline:

The 'Timeline' Panel contains time-based properties for the current Scene. The Scene is made up of a series of Frames, in the same way that a motion picture is

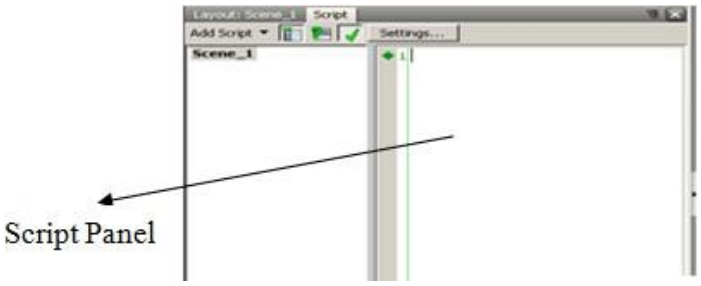


made up of Frames. The Timeline is a visual representation of the Frames with the first Frame at the left and last Frame at the right

Scene: A Scene is a collection of Objects that are animated over a number of Frames.

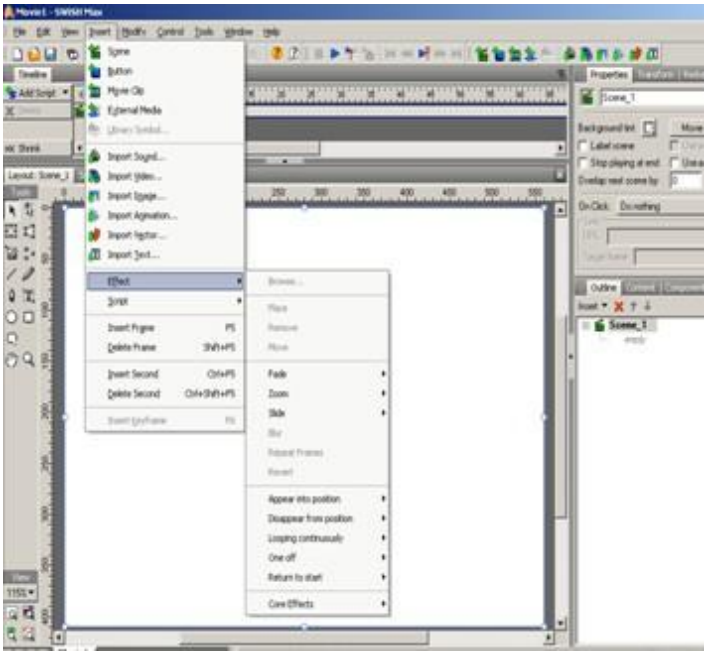
Scripting:

Scripting can be used to define Actions that will occur at a specific Frame, when two Objects collide, or when some other external input is applied to the Movie.



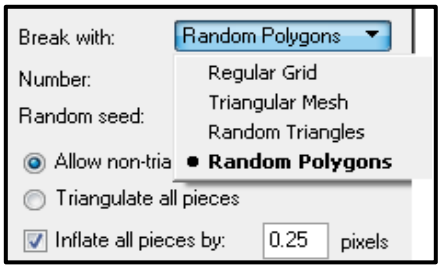
Effects:

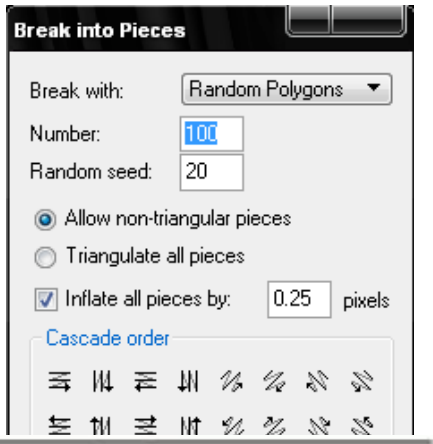
Effects are animations that change the appearance of an Object over time. You can add, modify and coordinate Effects using the Timeline panel.



Events:

All Actions are triggered in response to an Event. Any Scripting object can have Events associated with it. When an Event occurs, the Event handling routine executes any Actions that are defined for that Event.



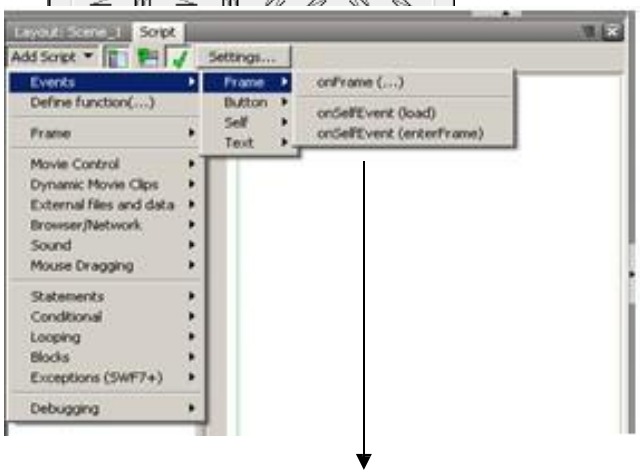


There are three types of events:

Frame

- Button
- Self

Input Text Objects also have the additional Events on(changed) onSelfEventChanged available.



Frame Event is selected

EX.NO.1	TEXT EFFECTS

AIM :

To apply various text effects to the predefined formats.

ALGORITHM \ PROCEDURE :

1. Create a Static text object and place it on the Stage.
2. In the Outline Panel, right-click on the text object and select *Copy Object*. Then, use the **Edit menu** and select *Paste in Place.* You should now have two text objects at the exact same location on the stage.
3. Select the bottom text object and use **Modify -› Break -› Break into pieces**.
4. In the Break into Pieces settings window, use the drop-down menu next to *Break With: ´* and select **Random Polygons**.
5. Allow non-triangular pieces or Triangulate all pieces… again, this is up to you as it is difficult to tell any difference at such small sizes.
6. Inflate all pieces by **0.25 pixels**. This helps seal the scenes in between in each piece.
7. When SWiSHmax has finished breaking the object into pieces, you should end up with a Group.

8. In the Timeline Panel, right-click at Frame 1 on the row for the top text object and select the **Place effect**.
9. Next, right-click at Frame 1 on the row for the Group object and select the **Remove effect**.
10. Next, right-click at Frame 20 on the row for the top text object and select the **Remove effect**.
11. Next, right-click at Frame 20 on the row for the Group object and select **Disappear from Position | Wild – Splatter.**

OUTPUT:

RESULT :

Thus the dust effect has been applied to the custom text using Swish.

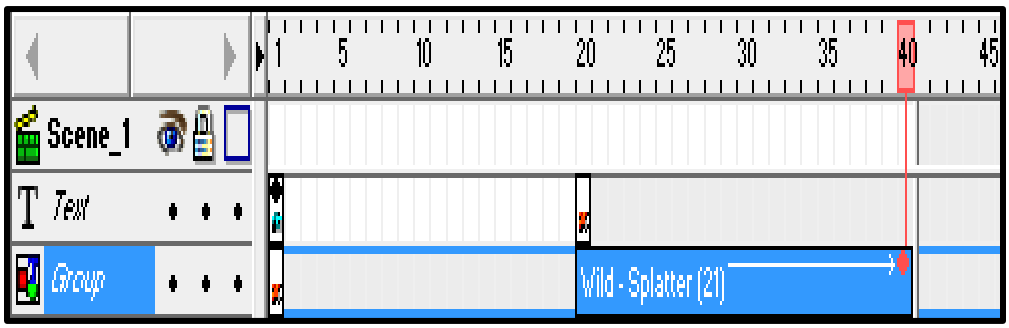
EX.NO.2	PRELOADER

AIM:

To create a pre-loader effect manually using Swish.

ALGORITHM \ PROCEDURE :

1. Create a Dynamic TextBox name it as myMsgObj and name its variable myMsg.
2. Create a rectangle for 20x20 pixels.
3. Then convert it into movie clip .



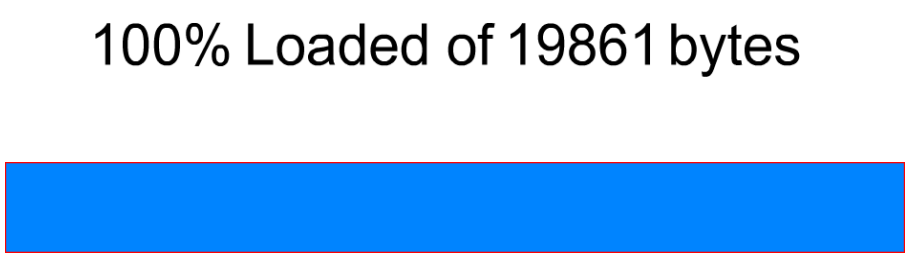
4. Name the movie clip as ‘ progBar ’.
5. Then type the following coding on Script -> Scene_1:

```
        onFrame(1)
{
   stop();
}
onSelfEvent(enterFrame)
{
```

```
    pLoaded=this.percentLoaded();
    tBytes=this.getBytesTotal();
    myMsg=pLoaded+"%Loaded of "+tBytes+" bytes";
    progBar_width=pLoaded;
    if(pLoaded==100)
    play();
}
```

6. The preloader is added.

OUTPUT :



RESULT :

Thus the pre-loader effect has been created using Swish.

FLASH

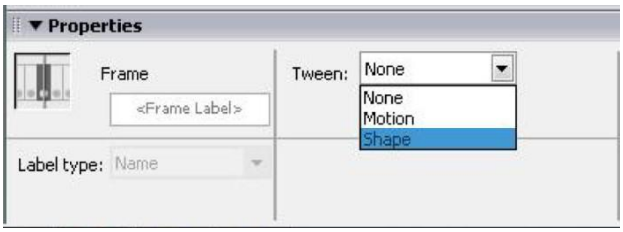
EX.NO.1 CHANGING THE SHAPE OF THE OBJECT

AIM:

To create Shape Tweening using Flash.

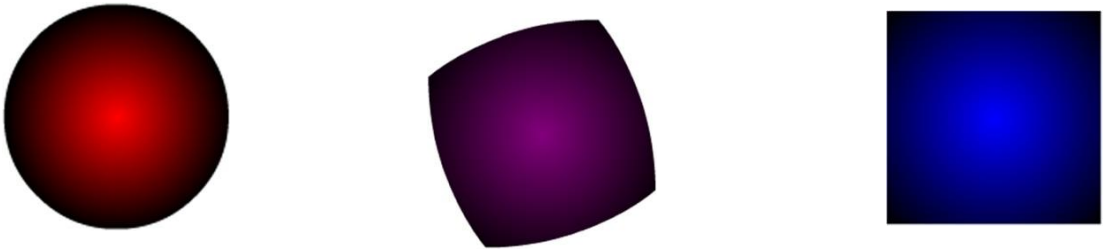
ALGORITHM \ PROCEDURE:

1. Create a New Flash Document.
2. Click the *Oval Tool*  on the Tools.
3. Using the *Fill Color* Option  select the attractive color to the Shape.
4. Create a Circle Object on Frame 1.
5. Click the 40th Frame and press *F6.*
6. Now Delete the Circle Object then Create a new Rectangle Object by using *Rectangle Tool.* Then  change the color of the Rectangle Object using Fill Color.
7. Select the 20th Frame on Timeline, then Change the Tween option to the Shape on the Properties window.



8. Now Shape Tweening is completed. Press *Ctrl + Enter.*

Shape tweening:



RESULT:

Thus the shape tweening has been created using Flash.

EX.NO. 2	IMAGE VIEWING USING MASK

AIM:

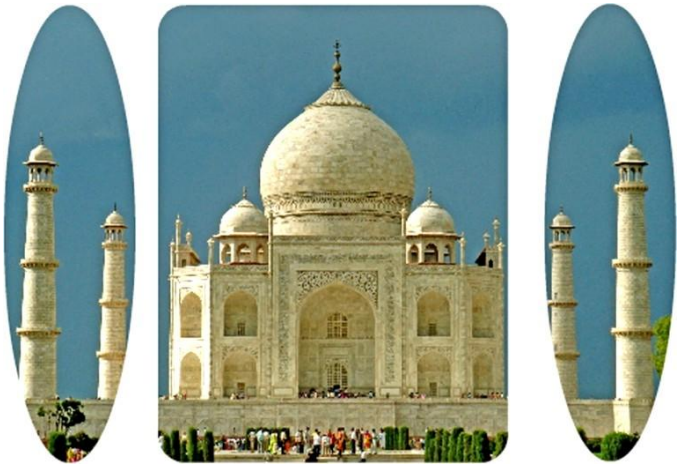
To create Masking using Flash.

ALGORITHM \ PROCEDURE:

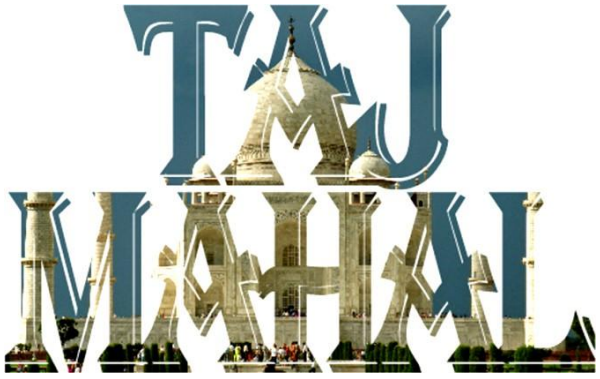
1. Create a New Flash Document.
2. Import any Image on Frame 1.
3. Create Motion Tweening for that Image. So that the Image can move from left to right side of the screen.
4. Insert keyFrame.
5. Now Lock the *Layer 1* using  *Lock.*
6. Insert another *Layer 2*,  and create any Objects (Ovel/ Rectangle/Any Shapes) for **Object Masking** , Or Type any Text with different Font & size for **Text Masking.**
7. Right click on Layer 2 then select *Mask*.
8. Now the Masking is created. Run the animation using *Ctrl + Enter.*

OUTPUT:

Object Masking…



Text Masking…



RESULT:

Thus the Masking effecting has been created using Flash.

PHOTO IMPACT

EX.NO.1	TEXT EFFECTS

AIM:

To create custom effects to the pre defined font styles.

ALGORITHM \ PROCEDURE:

1. Open a new image file.
2. Create any text with user defined font style and size.
3. Select the text and Select *Object -> convert Object Type ->From Text/Image to Path.*
4. Switch to the *Path Edit Tool*  on the Tool Panel, you will then see from the wireframe structure that the text has become a path and can be edited as such.
5. Use the dots to create custom design.

Output:

Before:

After:

RESULT:

Thus custom effect has been applied to the pre defined font using Ulead Photo Impact.

EX.NO.2 IMAGE SLICING

AIM :

To slice a portion from a picture and merge it into another using Ulead Photo Impact.

ALGORITHM \ PROCEDURE :

1. Open an image from which a specific portion has to be sliced.
2. Select the image and go to *Object -> Extract Object.*
3. By custom brush size and color, sketch out the portion to be parted and click next.
4. Click *extract* and select the portion which should not to be sliced and click next.
5. Adjust *transparency* and other details and click next.
6. Now the *portion to be sliced* alone is selected.
7. Import and background image and drag the selected image to the background image.
8. Now the user defined image is ready.

OUTPUT :

BEFORE

AFTER

RESULT:

Thus photo slicing and merging has been done using Ulead PhotoImpact.