```python
import turtle
import time

# Setup screen and turtle
screen = turtle.Screen()
screen.setup(700, 700)
screen.title("Line and Circle Drawing")
pen = turtle.Turtle()
pen.speed(0)
pen.hideturtle()
pen.penup()

PIXEL_SIZE = 10

def set_pixel(x, y, color="black"):
    pen.goto(x, y)
    pen.dot(PIXEL_SIZE, color)

def draw_line(x1, y1, x2, y2, color="blue"):
    dx = x2 - x1
    dy = y2 - y1
    steps = int(max(abs(dx), abs(dy)))
    x_inc = dx / steps
    y_inc = dy / steps
    x, y = x1, y1
    for _ in range(steps + 1):
        set_pixel(round(x), round(y), color)
        x += x_inc
        y += y_inc
        time.sleep(0.005)

def draw_circle(cx, cy, r, color="green"):
    x = 0
    y = r
    p = 1 - r

    def plot_points(xc, yc, x, y):
        points = [
            (xc + x, yc + y), (xc - x, yc + y),
            (xc + x, yc - y), (xc - x, yc - y),
            (xc + y, yc + x), (xc - y, yc + x),
            (xc + y, yc - x), (xc - y, yc - x)
        ]
```

```
        for px, py in points:
            set_pixel(px, py, color)

    plot_points(cx, cy, x, y)

    while x < y:
        x += 1
        if p < 0:
            p += 2 * x + 1
        else:
            y -= 1
            p += 2 * (x - y) + 1
        plot_points(cx, cy, x, y)
        time.sleep(0.01)

# Set coordinate system so (0,0) is center of screen
screen.setworldcoordinates(-350, -350, 350, 350)

# Draw line first
draw_line(-150, -150, 150, 150)

# Draw circle centered at (0,0)
draw_circle(0, 0, 100)

screen.exitonclick()
=
```

Program 2

```
def plot_line_bresenham(x1, y1, x2, y2, grid_size=10):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
    dx = x2 - x1
    dy = y2 - y1
    x, y = x1, y1
    p = 2 * dy - dx
    while x <= x2:
        if 0 <= x < grid_size and 0 <= y < grid_size:
            grid[grid_size - 1 - y][x] = 'X'
        x += 1
        if p < 0:
            p = p + 2 * dy
        else:
```

```
            y += 1
            p = p + 2 * (dy - dx)
    print(f"Bresenham's Line from ({x1}, {y1}) to ({x2}, {y2}):\n")
    for row in grid:
        print(" ".join(row))


def plot_line_dda(x1, y1, x2, y2, grid_size=10):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
    dx = x2 - x1
    dy = y2 - y1
    steps = max(abs(dx), abs(dy))
    x_inc = dx / steps
    y_inc = dy / steps
    x, y = x1, y1
    for _ in range(steps + 1):
        xi = int(round(x))
        yi = int(round(y))
        if 0 <= xi < grid_size and 0 <= yi < grid_size:
            grid[grid_size - 1 - yi][xi] = 'X'
        x += x_inc
        y += y_inc
    print(f"DDA Line from ({x1}, {y1}) to ({x2}, {y2}):\n")
    for row in grid:
        print(" ".join(row))


plot_line_bresenham(2, 2, 8, 6)
print("\n" + "-"*40 + "\n")
plot_line_dda(1, 7, 7, 3)
```

Program 3

```
def plot_circle_points(grid, xc, yc, x, y, grid_size):
    points = [
        (xc + x, yc + y), (xc - x, yc + y),
        (xc + x, yc - y), (xc - x, yc - y),
        (xc + y, yc + x), (xc - y, yc + x),
        (xc + y, yc - x), (xc - y, yc - x)
    ]
    for px, py in points:
        if 0 <= px < grid_size and 0 <= py < grid_size:
```

```
            grid[py][px] = 'X'  # Note: row = y, col = x


def plot_circle_bresenham(xc, yc, r, grid_size=20):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
    x, y = 0, r
    p = 3 - 2 * r

    while x <= y:
        plot_circle_points(grid, xc, yc, x, y, grid_size)
        if p < 0:
            p += 4 * x + 6
        else:
            y -= 1
            p += 4 * (x - y) + 10
        x += 1

    return grid


def plot_circle_midpoint(xc, yc, r, grid_size=20):
    grid = [['.' for _ in range(grid_size)] for _ in range(grid_size)]
    x, y = 0, r
    p = 1 - r

    while x <= y:
        plot_circle_points(grid, xc, yc, x, y, grid_size)
        if p < 0:
            p += 2 * x + 3
        else:
            y -= 1
            p += 2 * (x - y) + 5
        x += 1

    return grid


def print_grid(grid, title):
    print(f"\n{title}")
    for row in reversed(grid):  # Reverse to show origin (0,0) at
bottom-left
        print(" ".join(row))
```

```
def draw_circle(center, radius, grid_size=20):
    xc, yc = center
    print(f"Center: ({xc}, {yc}) | Radius: {radius} | Grid:
{grid_size}x{grid_size}")

    # Bresenham
    grid_bresenham = plot_circle_bresenham(xc, yc, radius, grid_size)
    print_grid(grid_bresenham, "Bresenham's Circle Drawing")

    print("\n" + "-" * 40)

    # Midpoint
    grid_midpoint = plot_circle_midpoint(xc, yc, radius, grid_size)
    print_grid(grid_midpoint, "Mid-Point Circle Drawing")

    print("\n" + "-" * 40 + "\n")


# Example usage
draw_circle(center=(10, 10), radius=6, grid_size=21)
```

Program 4

```
import math

# ---------- Helper Function ----------
def display_points(points, title):
    print(f"\n{title}")
    for i in range(len(points)):
        print(f"Point {i+1}: {points[i]}")

# ---------- 2D Transformations ----------
def translate_2d(points, tx, ty):
    return [(x + tx, y + ty) for x, y in points]

def scale_2d(points, sx, sy):
    return [(x * sx, y * sy) for x, y in points]

def rotate_2d(points, angle_deg):
    angle_rad = math.radians(angle_deg)
    cos_a = math.cos(angle_rad)
```

```python
    sin_a = math.sin(angle_rad)
    return [(x * cos_a - y * sin_a, x * sin_a + y * cos_a) for x, y in
points]

def shear_2d(points, shx, shy):
    result = []
    for x, y in points:
        x_new = x + shx * y
        y_new = y + shy * x
        result.append((x_new, y_new))
    return result

# ---------- 3D Transformations ----------
def translate_3d(points, tx, ty, tz):
    return [(x + tx, y + ty, z + tz) for x, y, z in points]

def scale_3d(points, sx, sy, sz):
    return [(x * sx, y * sy, z * sz) for x, y, z in points]

def rotate_3d_x(points, angle_deg):
    angle_rad = math.radians(angle_deg)
    cos_a = math.cos(angle_rad)
    sin_a = math.sin(angle_rad)
    result = []
    for x, y, z in points:
        y_new = y * cos_a - z * sin_a
        z_new = y * sin_a + z * cos_a
        result.append((x, y_new, z_new))
    return result

def rotate_3d_y(points, angle_deg):
    angle_rad = math.radians(angle_deg)
    cos_a = math.cos(angle_rad)
    sin_a = math.sin(angle_rad)
    result = []
    for x, y, z in points:
        x_new = x * cos_a + z * sin_a
        z_new = -x * sin_a + z * cos_a
        result.append((x_new, y, z_new))
    return result

def rotate_3d_z(points, angle_deg):
    angle_rad = math.radians(angle_deg)
```

```python
    cos_a = math.cos(angle_rad)
    sin_a = math.sin(angle_rad)
    result = []
    for x, y, z in points:
        x_new = x * cos_a - y * sin_a
        y_new = x * sin_a + y * cos_a
        result.append((x_new, y_new, z))
    return result

def shear_3d(points, shxy=0, shxz=0, shyx=0, shyz=0, shzx=0, shzy=0):
    result = []
    for x, y, z in points:
        x_new = x + shxy * y + shxz * z
        y_new = y + shyx * x + shyz * z
        z_new = z + shzx * x + shzy * y
        result.append((x_new, y_new, z_new))
    return result

# ---------- Main Program ----------
if __name__ == "__main__":
    # ===== 2D Transformation =====
    print("==== 2D TRANSFORMATIONS ====")
    points_2d = [(0, 0), (2, 0), (1, 2)]
    display_points(points_2d, "Original 2D Points")

    translated_2d = translate_2d(points_2d, 3, 2)
    display_points(translated_2d, "After Translation (tx=3, ty=2)")

    scaled_2d = scale_2d(points_2d, 2, 1.5)
    display_points(scaled_2d, "After Scaling (sx=2, sy=1.5)")

    rotated_2d = rotate_2d(points_2d, 45)
    display_points(rotated_2d, "After Rotation (45°)")

    sheared_2d = shear_2d(points_2d, shx=0.5, shy=0.2)
    display_points(sheared_2d, "After Shearing (shx=0.5, shy=0.2)")

    # ===== 3D Transformation =====
    print("\n==== 3D TRANSFORMATIONS ====")
    points_3d = [(1, 1, 1), (2, 1, 1), (1, 2, 1), (1, 1, 2)]
    display_points(points_3d, "Original 3D Points")

    translated_3d = translate_3d(points_3d, 2, 3, 4)
```

```
    display_points(translated_3d, "After Translation (tx=2, ty=3, 
tz=4)")

    scaled_3d = scale_3d(points_3d, 2, 0.5, 1.5)
    display_points(scaled_3d, "After Scaling (sx=2, sy=0.5, sz=1.5)")

    rotated_x = rotate_3d_x(points_3d, 45)
    display_points(rotated_x, "After Rotation about X-axis (45°)")

    rotated_y = rotate_3d_y(points_3d, 45)
    display_points(rotated_y, "After Rotation about Y-axis (45°)")

    rotated_z = rotate_3d_z(points_3d, 45)
    display_points(rotated_z, "After Rotation about Z-axis (45°)")

    sheared_3d = shear_3d(points_3d, shxy=0.3, shyz=0.2, shzx=0.4)
    display_points(sheared_3d, "After Shearing (shxy=0.3, shyz=0.2, 
shzx=0.4)")
```

Program 5

```
import matplotlib.pyplot as plt

# Clipping window boundaries
X_MIN, Y_MIN = 0, 0
X_MAX, Y_MAX = 10, 10

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

# Compute region code
def compute_code(x, y):
    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

# Cohen-Sutherland algorithm
```

```python
def cohen_sutherland_line_clip(x1, y1, x2, y2):
    code1, code2 = compute_code(x1, y1), compute_code(x2, y2)
    while True:
        if code1 == 0 and code2 == 0:       # Both inside
            return (x1, y1, x2, y2)
        elif (code1 & code2) != 0:          # Both outside same region
            return None
        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)
            else:
                x2, y2 = x, y
                code2 = compute_code(x2, y2)

# Draw window and lines
def draw(x1, y1, x2, y2):
    plt.plot([X_MIN, X_MAX, X_MAX, X_MIN, X_MIN],
             [Y_MIN, Y_MIN, Y_MAX, Y_MAX, Y_MIN], 'k-')  # Window
    plt.plot([x1, x2], [y1, y2], 'b--')  # Original line
    res = cohen_sutherland_line_clip(x1, y1, x2, y2)
    if res:
        plt.plot([res[0], res[2]], [res[1], res[3]], 'r-', lw=2)
    plt.axis([-2, 12, -2, 12])
    plt.grid(True)
    plt.show()

# Example
draw(2, 3, 12, 5)
```

Program 6

```
import pygame, sys

pygame.init()

WIDTH, HEIGHT = 600, 400
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Quadratic Bézier Curve (3 points)")

WHITE = (255, 255, 255)
RED = (255,   0,   0)
BLUE = (0,   0, 255)

# Three control points: start, control, end
points = [(100, 300), (300, 50), (500, 300)]
drag = None

def bezier_quad(p0, p1, p2, t):
    """Return (x, y) for quadratic Bezier at parameter t."""
    u = 1 - t
    # Quadratic Bézier formula: (1-u)^2 P0 + 2u(1-u) P1 + u^2 P2
    x = u * u * p0[0] + 2 * u * t * p1[0] + t * t * p2[0]
    y = u * u * p0[1] + 2 * u * t * p1[1] + t * t * p2[1]
    return int(x), int(y)

def draw():
    screen.fill(WHITE)
    # Draw control polygon (lines connecting the points)
    pygame.draw.lines(screen, RED, False, points, 2)
    # Draw control points
    for p in points:
        pygame.draw.circle(screen, RED, p, 6)
    # Draw the Bézier curve by sampling many points
    for i in range(101):
        t = i / 100
        pygame.draw.circle(screen, BLUE, bezier_quad(points[0],
points[1], points[2], t), 2)
    pygame.display.flip()
```

```
while True:
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if e.type == pygame.MOUSEBUTTONDOWN:
            for i, p in enumerate(points):
                if abs(e.pos[0] - p[0]) < 8 and abs(e.pos[1] - p[1]) <
8:
                    drag = i
        if e.type == pygame.MOUSEBUTTONUP:
            drag = None
        if e.type == pygame.MOUSEMOTION and drag is not None:
            points[drag] = e.pos

    draw()
```