University of Rajshahi

Department of Computer Science and Engineering

CSE4222 Computer Graphics Lab

# Lab Report

*From:*
Name: Md. Miju Ahmed
ID: 2010676104

*To:*
ABU RAIHAN SHOYEB
AHMED SIDDIQUE
PROFESSOR

August 22, 2025

# Contents


**1 Lab Manual** **2**
- 1.1 Lab Question . . . 2
  - 1.1.1 Experiment: . . . 2

**2 Python Code** **3**
- 2.1 Sutherland-Hodgeman Line Clipping . . . 3
  - 2.1.1 Code: . . . 3
  - 2.1.2 Output: . . . 5
- 2.2 Sutheland-Hodgeman Polygon Clipping . . . 6
  - 2.2.1 Code: . . . 6
  - 2.2.2 Output: . . . 8
- 2.3 Translation . . . 9
  - 2.3.1 Code: . . . 9
  - 2.3.2 Output: . . . 10
- 2.4 Rotation . . . 11
  - 2.4.1 Code: . . . 11
  - 2.4.2 Output: . . . 12
- 2.5 Scaling . . . 13
  - 2.5.1 Code: . . . 13
  - 2.5.2 Output: . . . 14
- 2.6 Translation and Rotation . . . 15
  - 2.6.1 Code: . . . 15
  - 2.6.2 Output: . . . 16
- 2.7 Scaling and Rotation . . . 17
  - 2.7.1 Code: . . . 17
  - 2.7.2 Output: . . . 18
- 2.8 Beizer Curve . . . 19
  - 2.8.1 Code: . . . 19
  - 2.8.2 Output: . . . 20
- 2.9 Bresenham Line Drawing . . . 21
  - 2.9.1 Code: . . . 21
  - 2.9.2 Output: . . . 22
- 2.10 Bresenham Circle Drawing . . . 23
  - 2.10.1 Code: . . . 23
  - 2.10.2 Output: . . . 24
- 2.11 Snowflake Pattern with Fractal Geometry . . . 25
  - 2.11.1 Code: . . . 25
  - 2.11.2 Output: . . . 26

# Chapter 1

# Lab Manual

## 1.1 Lab Question

### 1.1.1 Experiment:

1. Implement the Cohen-Sutherland Line Clipping algorithm
2. Implement the Sutherland-Hodgman Polygon Clipping algorithm
3. Create the Bezier Curve
4. Simulate two-dimensional geometric Translation, Rotation, and Scaling
5. Draw a line with the Bresenham Line Drawing algorithm
6. Draw a circle with the Bresenham Circle Drawing algorithm
7. Draw the Snowflake Pattern with Fractal Geometry

# Chapter 2

# Python Code

## 2.1 Sutherland-Hodgeman Line Clipping

### 2.1.1 Code:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

INSIDE=0
LEFT=1
RIGHT=2
BOTTOM=4
UP=8

def compute_outcode(x,y,xmin,ymin,xmax,ymax):
    code = INSIDE
    if x<xmin:
        code|=LEFT
    elif x>xmax:
        code|=RIGHT
    elif y<ymin:
        code|=BOTTOM
    elif y>ymax:
        code|=UP
    return code
def cohen_sutherland_clipping(x1,y1,x2,y2,xmin,ymin,xmax,ymax):
    outcode1=compute_outcode(x1,y1,xmin,ymin,xmax,ymax)
    outcode2=compute_outcode(x2,y2,xmin,ymin,xmax,ymax)
    accepted=False
    while True:
        if not(outcode1|outcode2):
            accepted=True
            break
        elif (outcode1&outcode2):
            break
        else:
            outcode_out=outcode1 if outcode1 else outcode2
            x,y=0,0
            if outcode_out&UP:
                x=x1+(x2-x1)*(ymax-y1)/(y2-y1)
                y=ymax
            elif outcode_out&BOTTOM:
                x=x1+(x2-x1)*(ymin-y1)/(y2-y1)
                y=ymin
            elif outcode_out&LEFT:
                x=xmin
                y=y1+(y2-y1)*(xmin-x1)/(x2-x1)
```

```
            elif outcode_out&RIGHT:
                x=xmax
                y=y1+(y2-y1)*(xmax-x1)/(x2-x1)
            if outcode_out==outcode1:
                x1,y1=x,y
                outcode1=compute_outcode(x1,y1,xmin,ymin,xmax,ymax)
            else:
                x2,y2=x,y
                outcode2=compute_outcode(x2,y2,xmin,ymin,xmax,ymax)
    return accepted,x1,y1,x2,y2
def visualize(window,lines):
    fig,ax=plt.subplots(1,len(lines),figsize=(5*len(lines),5))
    if len(lines)==1:
        ax=[ax]
    xmin,ymin,xmax,ymax=window
    for i,(title,line) in enumerate(lines.items()):
        x1,y1,x2,y2=line
        clip_rect=patches.Rectangle((xmin,ymin),xmax-xmin,ymax-ymin,linewidth=1.5,edgecolor='red',facecolor='none',linestyle='--')
        ax[i].add_patch(clip_rect)
        ax[i].plot([x1,x2],[y1,y2],'gray',linestyle=':',marker='o',label='Original Line')
        accepted,cl_x1,cl_y1,cl_x2,cl_y2=cohen_sutherland_clipping(x1,y1,x2,y2,xmin,ymin,xmax,ymax)
        if accepted:
            ax[i].plot([cl_x1,cl_x2],[cl_y1,cl_y2],'blue',marker='o',label="Clipped LLine")
            print(f"{title} : Accepted. Clipped to ({cl_x1:.2f},{cl_y1:.2f})-({cl_x2:.2f},{cl_y2:.2f})")
        else:
            print(f"{title} : Rejected")
        ax[i].set_title(title)
        ax[i].set_xlim(0,20)
        ax[i].set_ylim(0,20)
        ax[i].set_aspect('equal', adjustable='box')
        ax[i].legend()
        ax[i].grid(True)
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    CLIP_WINDOW = (5, 5, 15, 15)

    test_lines = {
        "1. Fully Inside": (6, 6, 14, 14),
        "2. Fully Outside (Reject)": (1, 1, 4, 4),
        "3. Crossing One Boundary": (1, 10, 10, 10),
        "4. Crossing Two Boundaries": (4, 18, 16, 2),
        "5. Diagonal Corner Crossing": (16, 16, 4, 4),
        "6. Vertical Line": (10, 1, 10, 19),
    }

    visualize(CLIP_WINDOW, test_lines)
```

### 2.1.2 Output:

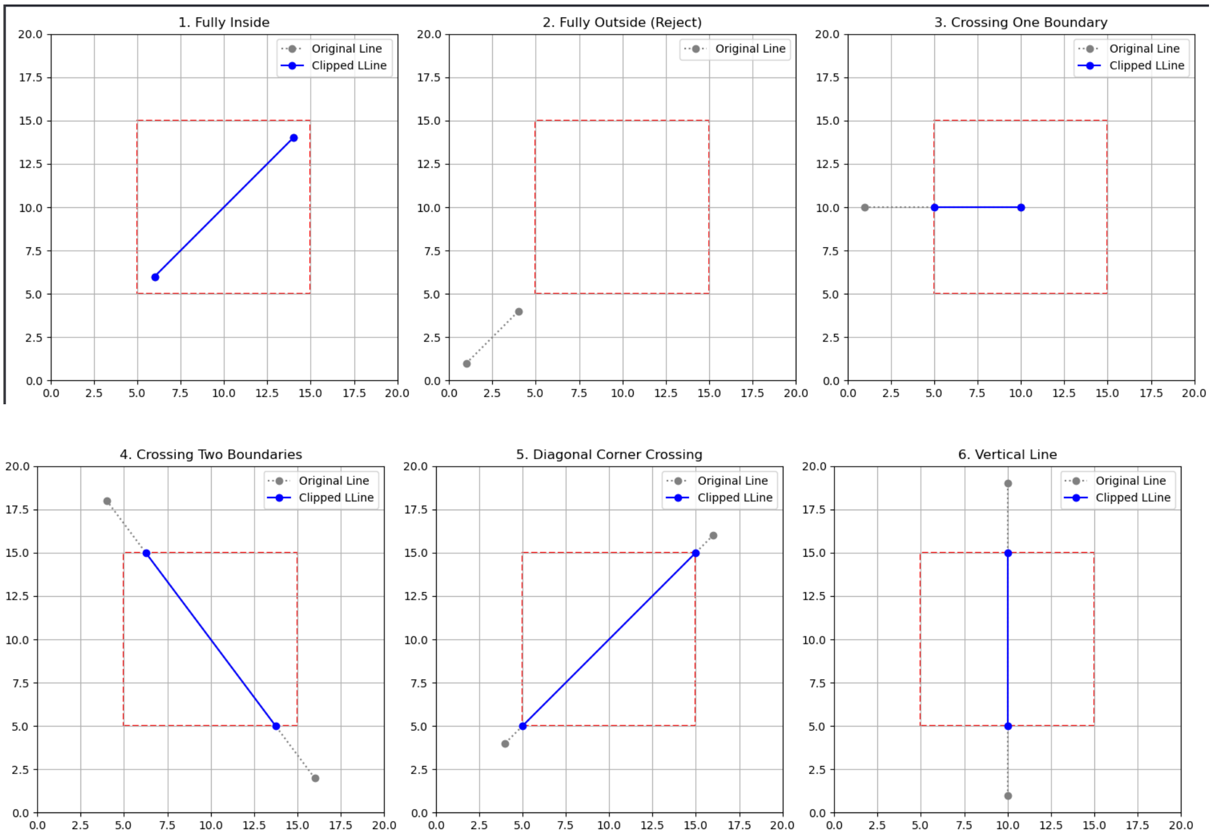


Figure 2.1: Sutherland-Hodgeman Line Clipping

## 2.2 Sutheland-Hodgeman Polygon Clipping

### 2.2.1 Code:

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

LEFT = 0
RIGHT = 1
BOTTOM = 2
TOP = 3

def is_inside(p, edge, clip_value):
    x, y = p
    if edge == LEFT:
        return x >= clip_value
    elif edge == RIGHT:
        return x <= clip_value
    elif edge == BOTTOM:
        return y >= clip_value
    elif edge == TOP:
        return y <= clip_value
    return False

def get_intersection(p1, p2, edge, clip_value):
    x1, y1 = p1
    x2, y2 = p2
    dx = x2 - x1
    dy = y2 - y1

    if edge == LEFT or edge == RIGHT:
        if dx == 0:
            return (clip_value, y1)
        y = y1 + dy * (clip_value - x1) / dx
        return (clip_value, y)

    elif edge == BOTTOM or edge == TOP:
        if dy == 0:
            return (x1, clip_value)
        x = x1 + dx * (clip_value - y1) / dy
        return (x, clip_value)

def clip_polygon_against_edge(polygon, edge, clip_value):
    output_polygon = []
    if not polygon:
        return output_polygon

    s = polygon[-1]

    for p in polygon:
        s_inside = is_inside(s, edge, clip_value)
        p_inside = is_inside(p, edge, clip_value)

        # Case 1: Both points are inside -> Add the second point
        if s_inside and p_inside:
            output_polygon.append(p)
        # Case 2: S is inside, P is outside -> Add intersection only
        elif s_inside and not p_inside:
            intersection = get_intersection(s, p, edge, clip_value)
            output_polygon.append(intersection)
        # Case 3: S is outside, P is inside -> Add intersection AND the second point
        elif not s_inside and p_inside:
```

```
            intersection = get_intersection(s, p, edge, clip_value)
            output_polygon.append(intersection)
            output_polygon.append(p)
        # Case 4: Both points are outside -> Do nothing

        s = p  # Move to the next edge
    return output_polygon

def sutherland_hodgman_polygon_clip(polygon, clip_window):
    x_min, y_min, x_max, y_max = clip_window

    clipped = clip_polygon_against_edge(polygon, LEFT, x_min)
    clipped = clip_polygon_against_edge(clipped, RIGHT, x_max)
    clipped = clip_polygon_against_edge(clipped, BOTTOM, y_min)
    clipped = clip_polygon_against_edge(clipped, TOP, y_max)

    return clipped

def plot_polygons(polygon, window, clipped):
    fig, ax = plt.subplots()

    poly_org = patches.Polygon(polygon, closed=True, edgecolor='red', facecolor='none', linestyle='--', linewidth=2, label='Original Polygon')
    ax.add_patch(poly_org)

    x_min, y_min, x_max, y_max = window
    rect = patches.Rectangle((x_min, y_min), x_max - x_min, y_max - y_min,
                             linewidth=2, facecolor='none', edgecolor='green', linestyle=':', label='Clipping Window')
    ax.add_patch(rect)

    if clipped:
        poly_clipped = patches.Polygon(clipped, closed=True, edgecolor='blue', linewidth=2, facecolor='blue', alpha=0.4, label='Clipped Polygon')
        ax.add_patch(poly_clipped)

    ax.set_title("Sutherland-Hodgman Polygon Clipping")
    ax.legend()
    ax.set_aspect('equal', 'box')

    all_points = polygon + [(x_min, y_min), (x_max, y_max)]
    all_x = [p[0] for p in all_points]
    all_y = [p[1] for p in all_points]
    plt.xlim(min(all_x) - 20, max(all_x) + 20)
    plt.ylim(min(all_y) - 20, max(all_y) + 20)

    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    subject_polygon = [(50, 150), (200, 50), (350, 150), (100, 250), (200,150)]
    clip_window = (100, 100, 300, 200)

    clipped_polygon = sutherland_hodgman_polygon_clip(subject_polygon, clip_window)

    print("Original Polygon Vertices:", subject_polygon)
    print("Clipped Polygon Vertices:", clipped_polygon)

    plot_polygons(subject_polygon, clip_window, clipped_polygon)
```

### 2.2.2 Output:

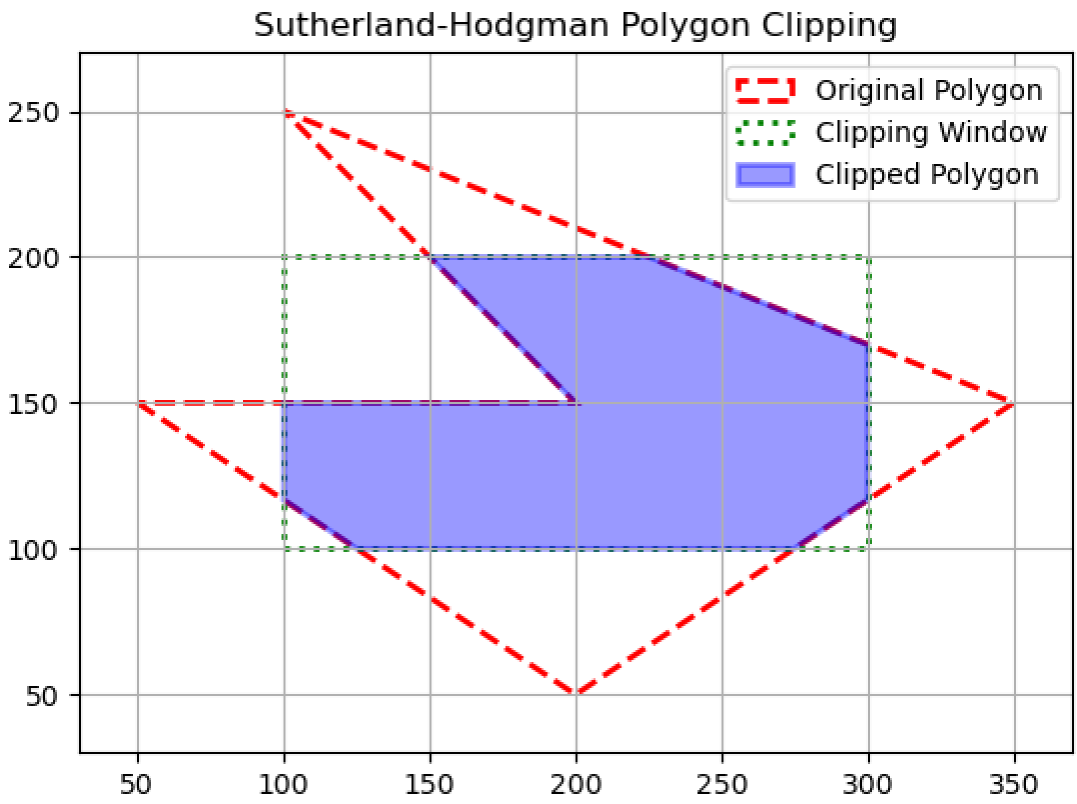


Figure 2.2: Sutherland-Hodgeman Polygon Clipping

## 2.3 Translation

### 2.3.1 Code:

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

def plot_shape(shape, title="", color='blue'):
    plt.plot(shape[:, 0], shape[:, 1], color=color, marker='o', linestyle='-',
    label=title)


def get_traslation_matrix(tx,ty):
    return np.array([
        [1,0,tx],
        [0,1,ty],
        [0,0,1]
    ])
def transformation(shape,matrix):
    homogeneous_shape=np.hstack([shape,np.ones((shape.shape[0],1))])
    transformed_shape=(matrix@homogeneous_shape.T).T
    return transformed_shape[:,:2]

if __name__=="__main__":
    house_shape=np.array([
        [0, 0], [0, 10], [5, 15], [10, 10], [10, 0], [0, 0]
    ])

    tx,ty=23,30
    translation_matrix= get_traslation_matrix(tx,ty)
    translated_house= transformation(house_shape,translation_matrix)

    plt.figure(figsize=(12,8))
    ax=plt.gca()
    ax.set_aspect('equal',adjustable='box')
    plot_shape(house_shape,"Original",'gray')
    plot_shape(translated_house,f"Translated by [{tx},{ty}]","purple")
    plt.title("2D Geometric Transformations")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.show()
```

### 2.3.2 Output:

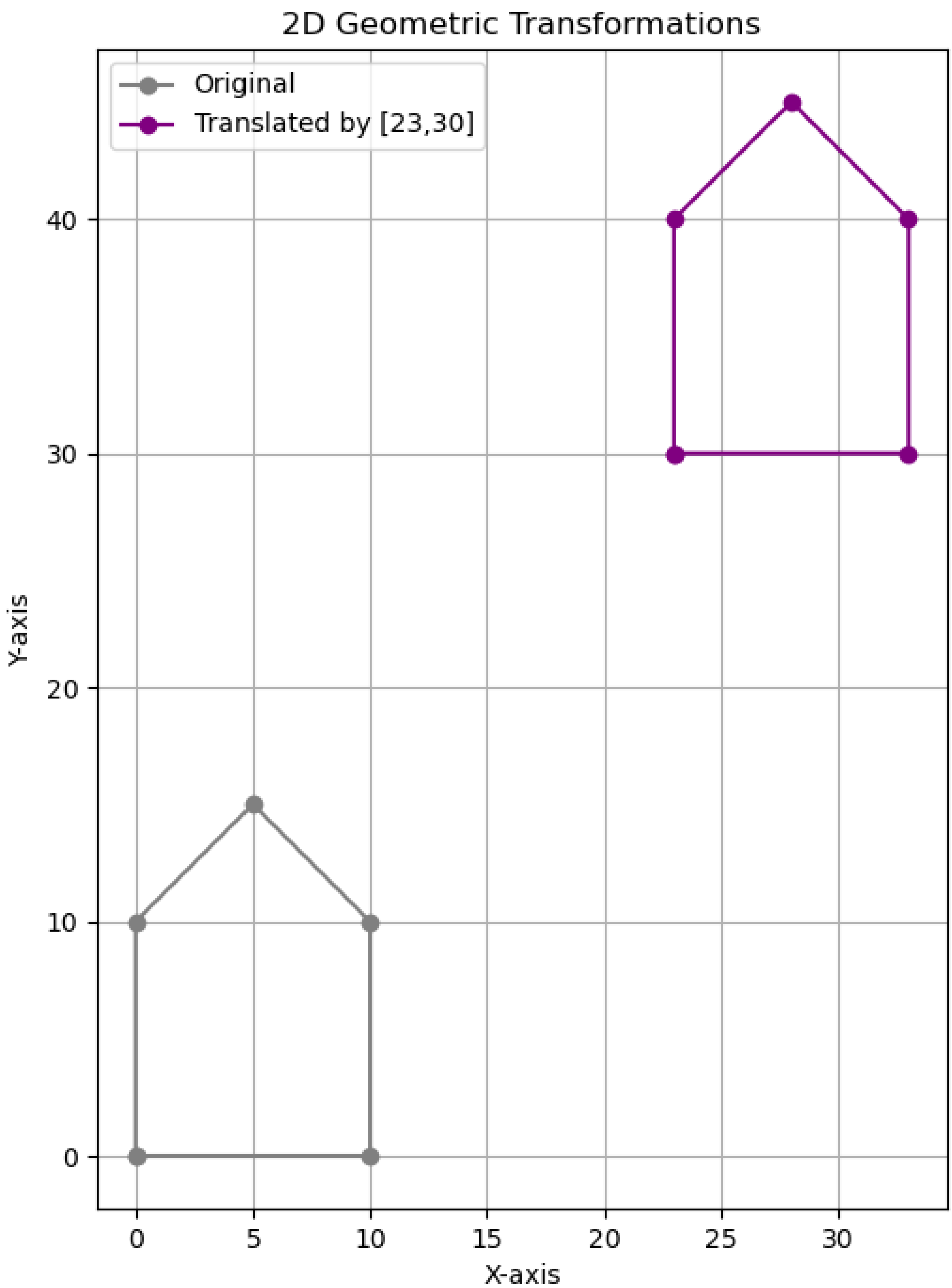


Figure 2.3: Translation

## 2.4 Rotation

### 2.4.1 Code:

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

def plot_shape(shape, title="", color='blue'):
    plt.plot(shape[:, 0], shape[:, 1], color=color, marker='o', linestyle='-',
    label=title)


def get_rotation_matrix(angles):
    cos_a=np.cos(angles)
    sin_a=np.sin(angles)
    return np.array([
        [cos_a,-sin_a,0],
        [sin_a,cos_a,0],
        [0,0,1]
    ])
def transformation(shape,matrix):
    homogeneous_shape=np.hstack([shape,np.ones((shape.shape[0],1))])
    transformed_shape=(matrix@homogeneous_shape.T).T
    return transformed_shape[:,:2]

if __name__=="__main__":
    house_shape=np.array([
        [0, 0], [0, 10], [5, 15], [10, 10], [10, 0], [0, 0]
    ])

    angles=75
    rotation_matrix= get_rotation_matrix(angles)
    rotated_house= transformation(house_shape,rotation_matrix)

    plt.figure(figsize=(12,8))
    ax=plt.gca()
    ax.set_aspect('equal',adjustable='box')
    plot_shape(house_shape,"Original",'gray')
    plot_shape(rotated_house,f"Rotated by [{angles}]","purple")
    plt.title("2D Geometric Transformations")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.show()
```

### 2.4.2 Output:

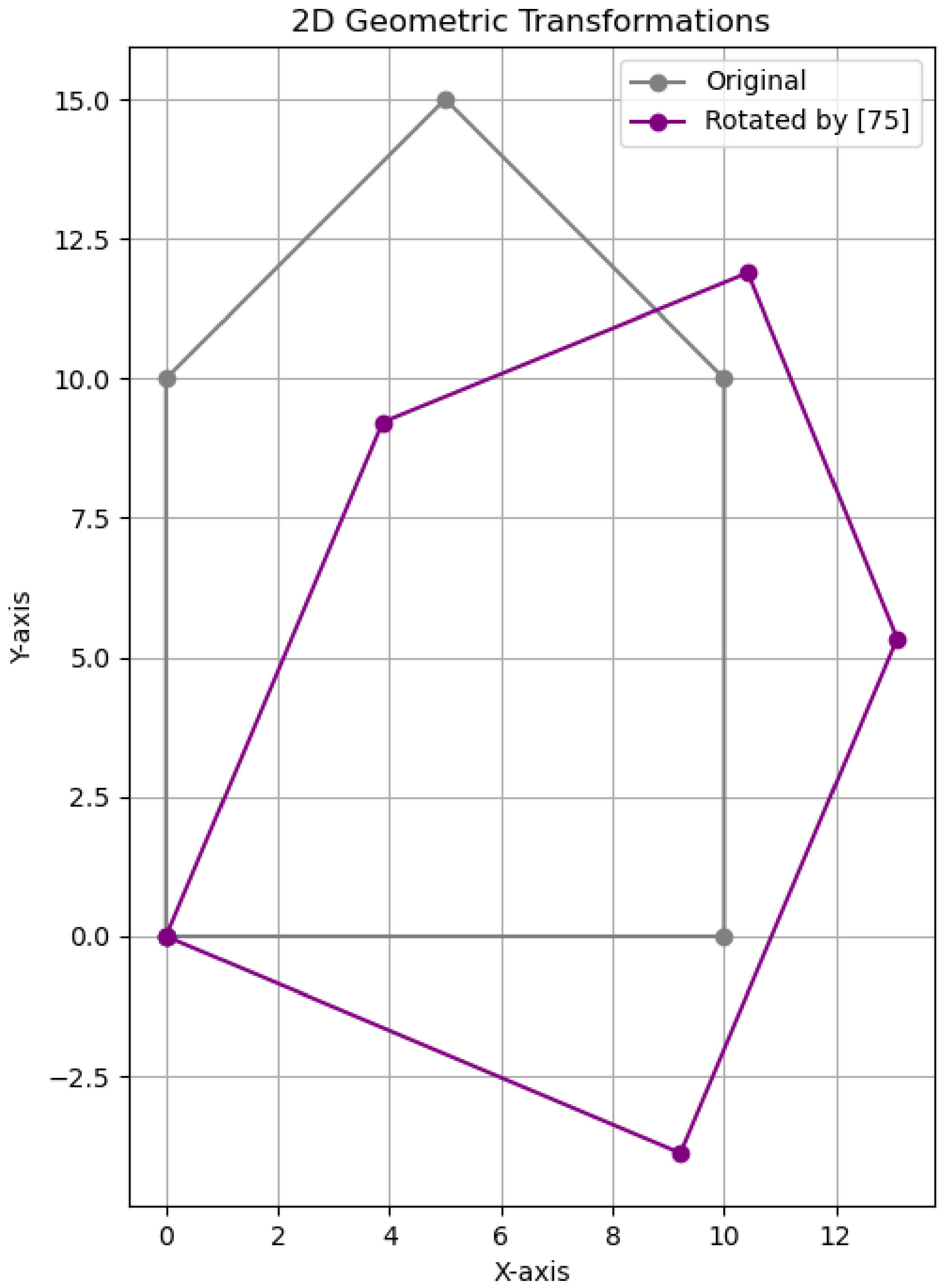


Figure 2.4: Rotation

## 2.5 Scaling

### 2.5.1 Code:

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

def plot_shape(shape, title="", color='blue'):
    plt.plot(shape[:, 0], shape[:, 1], color=color, marker='o', linestyle='-',
    label=title)


def get_scaling_matrix(sx,sy):
    return np.array([
        [sx,0,0],
        [0,sy,0],
        [0,0,1]
    ])
def transformation(shape,matrix):
    homogeneous_shape=np.hstack([shape,np.ones((shape.shape[0],1))])
    transformed_shape=(matrix@homogeneous_shape.T).T
    return transformed_shape[:,:2]

if __name__=="__main__":
    house_shape=np.array([
        [0, 0], [0, 10], [5, 15], [10, 10], [10, 0], [0, 0]
    ])

    sx,sy=2,0.5
    scaling_matrix= get_scaling_matrix(sx,sy)
    rotated_house= transformation(house_shape,scaling_matrix)

    plt.figure(figsize=(12,8))
    ax=plt.gca()
    ax.set_aspect('equal',adjustable='box')
    plot_shape(house_shape,"Original",'gray')
    plot_shape(rotated_house,f"Scaled by [{sx},{sy}]","purple")
    plt.title("2D Geometric Transformations")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.show()
```

### 2.5.2 Output:

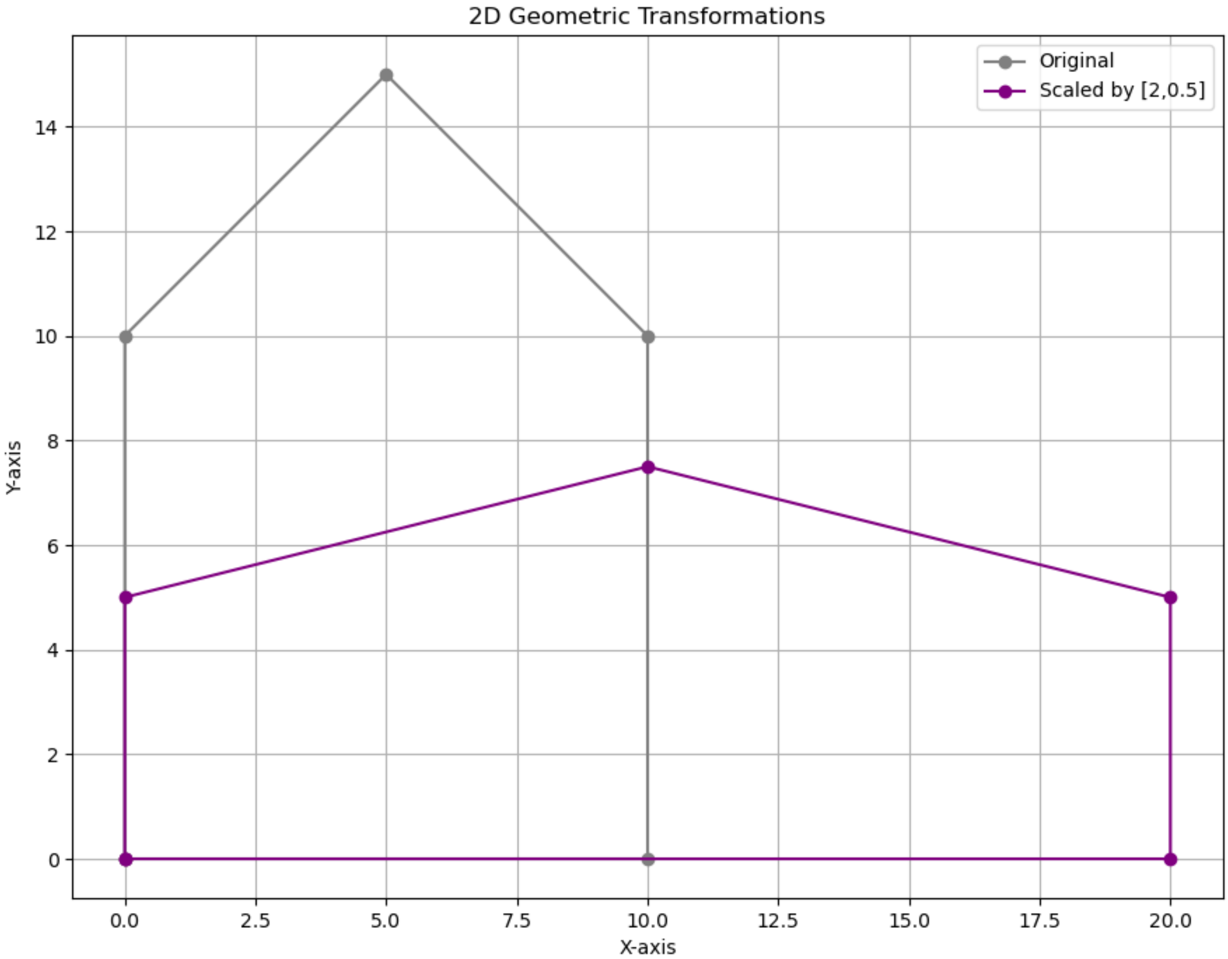


Figure 2.5: Scaling

## 2.6 Translation and Rotation

### 2.6.1 Code:

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

def plot_shape(shape, title="", color='blue'):
    plt.plot(shape[:, 0], shape[:, 1], color=color, marker='o', linestyle='-',
    label=title)


def get_traslation_matrix(tx,ty):
    return np.array([
        [1,0,tx],
        [0,1,ty],
        [0,0,1]
    ])
def get_rotation_matrix(angles):
    cos_a=np.cos(angles)
    sin_a=np.sin(angles)
    return np.array([
        [cos_a,-sin_a,0],
        [sin_a,cos_a,0],
        [0,0,1]
    ])

def transformation(shape,matrix):
    homogeneous_shape=np.hstack([shape,np.ones((shape.shape[0],1))])
    transformed_shape=(matrix@homogeneous_shape.T).T
    return transformed_shape[:,:2]

if __name__=="__main__":
    house_shape=np.array([
        [0, 0], [0, 10], [5, 15], [10, 10], [10, 0], [0, 0]
    ])

    tx,ty=5,10
    translation_matrix= get_traslation_matrix(tx,ty)
    translated_house= transformation(house_shape,translation_matrix)
    angles=-130
    rotation_matrix= get_rotation_matrix(angles)
    rotated_house= transformation(translated_house,rotation_matrix)

    plt.figure(figsize=(12,8))
    ax=plt.gca()
    ax.set_aspect('equal',adjustable='box')
    plot_shape(house_shape,"Original",'gray')
    plot_shape(rotated_house,f"Translated and Rotate the house","purple")
    plt.title("2D Geometric Transformations")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.show()
```

### 2.6.2 Output:

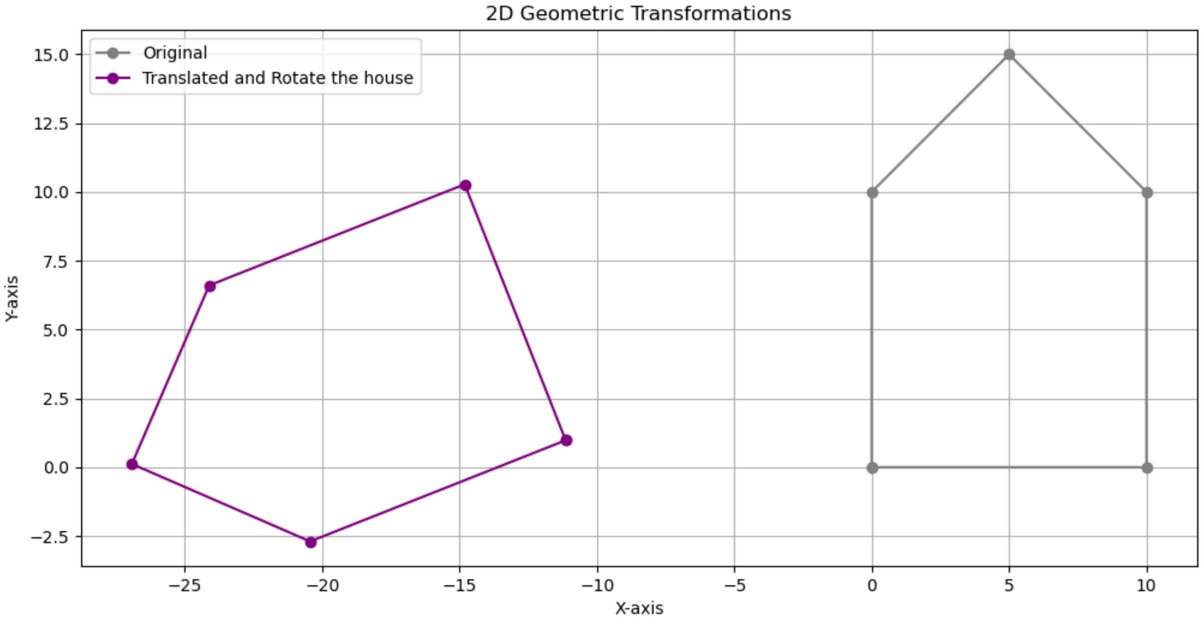


Figure 2.6: Translation and Rotation

## 2.7 Scaling and Rotation

### 2.7.1 Code:

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

def plot_shape(shape, title="", color='blue'):
    plt.plot(shape[:, 0], shape[:, 1], color=color, marker='o', linestyle='-',
    label=title)


def get_scaling_matrix(sx,sy):
    return np.array([
        [sx,0,0],
        [0,sy,0],
        [0,0,1]
    ])
def get_rotation_matrix(angles):
    cos_a=np.cos(angles)
    sin_a=np.sin(angles)
    return np.array([
        [cos_a,-sin_a,0],
        [sin_a,cos_a,0],
        [0,0,1]
    ])


def transformation(shape,matrix):
    homogeneous_shape=np.hstack([shape,np.ones((shape.shape[0],1))])
    transformed_shape=(matrix@homogeneous_shape.T).T
    return transformed_shape[:,:2]

if __name__=="__main__":
    house_shape=np.array([
        [0, 0], [0, 10], [5, 15], [10, 10], [10, 0], [0, 0]
    ])

    sx,sy=2,0.5
    scaling_matrix= get_scaling_matrix(sx,sy)
    scaled_house= transformation(house_shape,scaling_matrix)
    angles=75
    rotation_matrix= get_rotation_matrix(angles)
    scaled_rot_house= transformation(scaled_house,rotation_matrix)

    plt.figure(figsize=(12,8))
    ax=plt.gca()
    ax.set_aspect('equal',adjustable='box')
    plot_shape(house_shape,"Original",'gray')
    plot_shape(scaled_rot_house,f"Scaled and Rotated","purple")
    plt.title("2D Geometric Transformations")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.grid(True)
    plt.show()
```

### 2.7.2 Output:

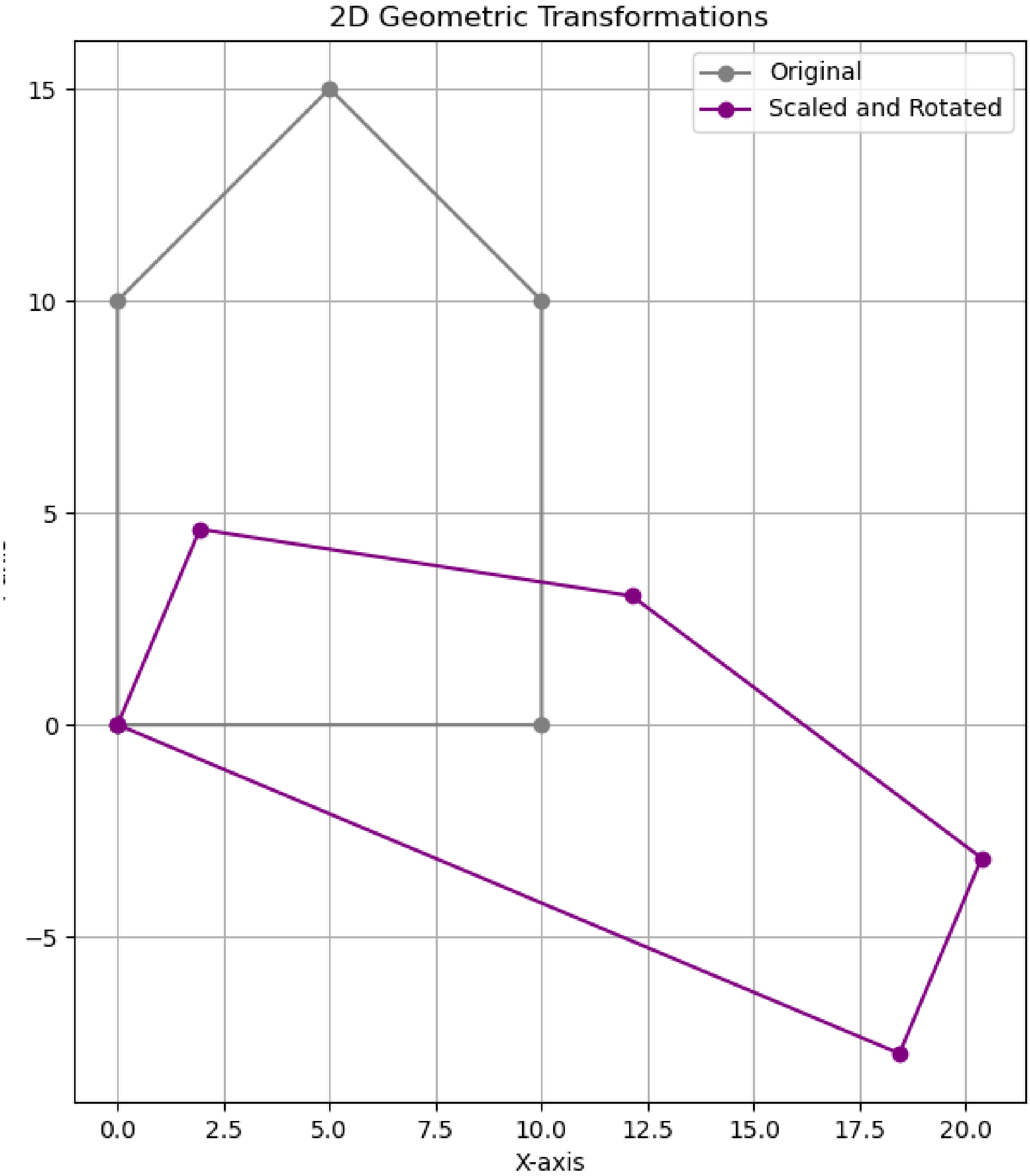


Figure 2.7: Scaling and Rotation

## 2.8 Beizer Curve

### 2.8.1 Code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import comb

def Beizer_curve(control_points,n_points=100):
    n=len(control_points)-1
    if n<1:
        raise ValueError("At least two points required.")
    control_points=np.array(control_points,dtype=float)
    t=np.linspace(0,1,n_points)
    curve=np.zeros((n_points,2))
    for i in range(n+1):
        bernstain_polygon=comb(n,i)*(t**i)*((1-t)**(n-i))
        curve+=np.outer(bernstain_polygon,control_points[i])
    return curve
def plot_curve_with_control_points(curve,control_points,title="Beizer Curve"):
    control_points=np.array(control_points)
    plt.figure(figsize=(10,8))
    plt.plot(curve[:,0],curve[:,1],"b-",lw=2,label="Beizer Curve")
    plt.plot(control_points[:,0],control_points[:,1],'ro--',label="Control
    Polygon")
    plt.plot(control_points[:,0],control_points[:,1],'go',markersize=10,label='
    Control Points')
    for i,(x,y) in enumerate(control_points):
        plt.text(x+5,y,f'P{i}',fontsize=14,verticalalignment='bottom')
    plt.title(title,fontsize=16)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.grid(True)
    plt.axis('equal')
    plt.show()
if __name__=="__main__":
    quadratic_control_points = [
        (50, 250),
        (150, -250),
        (200, 250),
        (250, -250),
        (300, 250),
        (350, -250)
    ]
    quadratic_curve=Beizer_curve(quadratic_control_points)
    plot_curve_with_control_points(quadratic_curve,quadratic_control_points,
    title="Quadratic Beizer Curve")
```

### 2.8.2 Output:

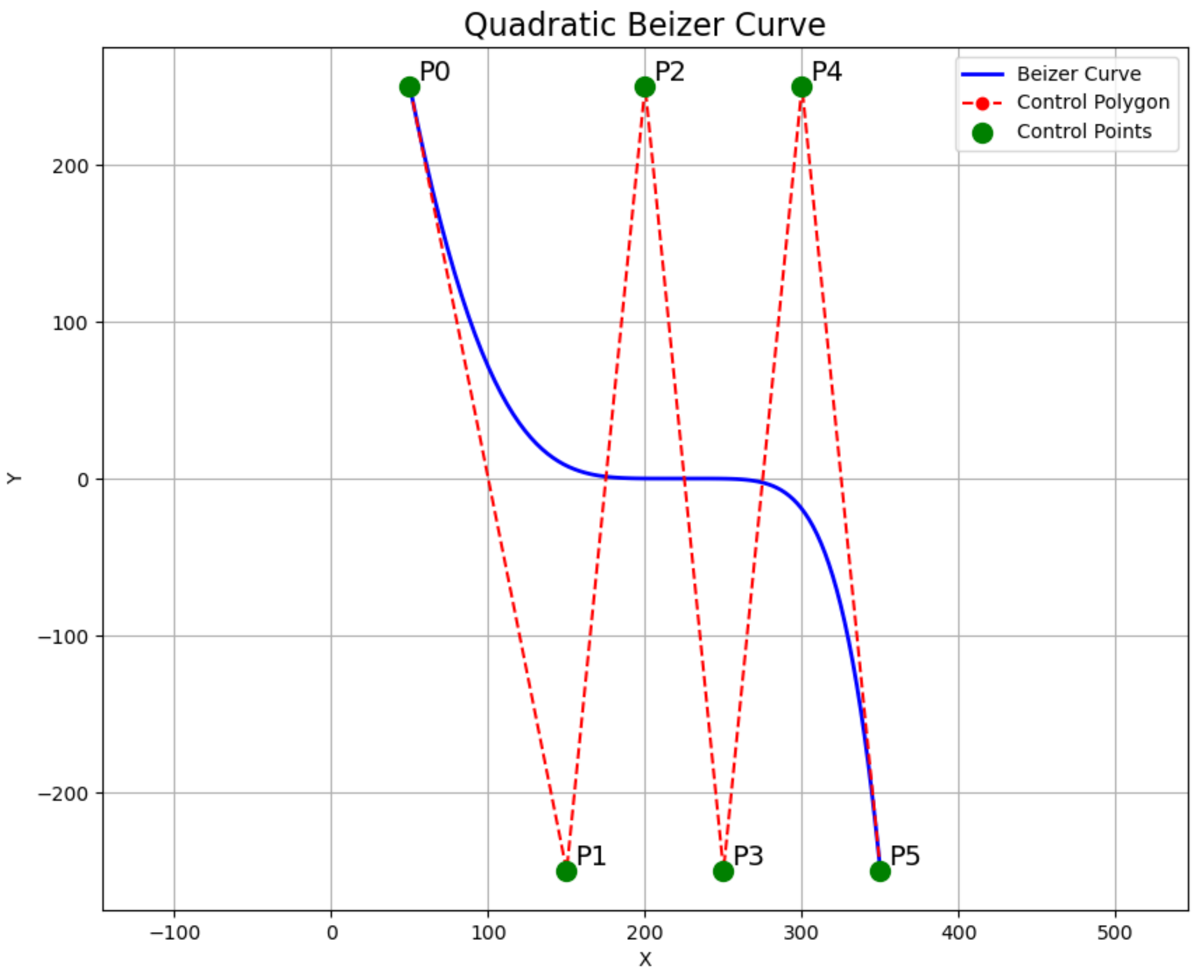


Figure 2.8: Beizer Curve

## 2.9 Bresenham Line Drawing

### 2.9.1 Code:

```

# bresenham.py
from PIL import Image

def bresenham(x0, y0, x1, y1):
    """Return list of (x,y) integer points on the line from (x0,y0) to (x1,y1).
    """
    points = []
    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, y = x0, y0

    while True:
        points.append((x, y))
        if x == x1 and y == y1:
            break
        e2 = 2 * err
        if e2 > -dy:
            err -= dy
            x += sx
        if e2 < dx:
            err += dx
            y += sy
    return points

if __name__ == "__main__":
    # Demo: draw a few lines and save as PNG
    W, H = 200, 200
    img = Image.new("RGB", (W, H), "white")
    px = img.load()

    lines = [
        (10, 190, 190, 10),  # other diagonal
    ]

    for (x0, y0, x1, y1) in lines:
        for x, y in bresenham(x0, y0, x1, y1):
            if 0 <= x < W and 0 <= y < H:
                px[x, y] = (0, 0, 0)  # black pixel

    img.save("bresenham_demo.png")
    print("Saved bresenham_demo.png")
```

### 2.9.2 Output:

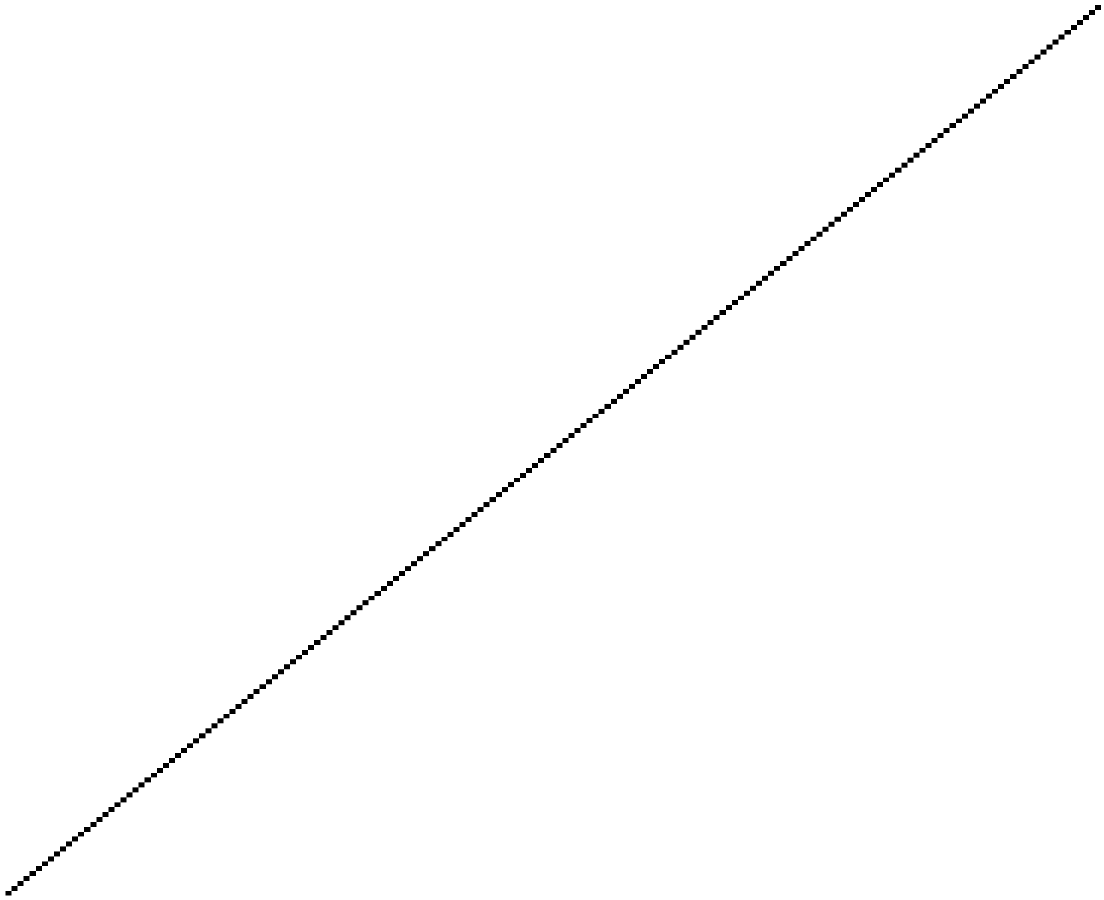

Figure 2.9: Bresenham Line Drawing

## 2.10 Bresenham Circle Drawing

### 2.10.1 Code:

```
import matplotlib.pyplot as plt

def plot_circle_points(xc,yc,x,y,points):
    points.extend([
        (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)
    ])

def Bresenham_circle(xc,yc,r):
    x=0
    y=r
    d=3-2*r
    points=[]
    while x<=y:
        plot_circle_points(xc,yc,x,y,points)
        if d<0:
            d=d+4*x+6
        else:
            d=d+4*(x-y)+10
            y=y-1
        x+=1
    return points
if __name__=="__main__":
    xc,yc,r=-10,34,50
    circle_points=Bresenham_circle(xc,yc,r)
    x_vals,y_vals=zip(*circle_points)
    plt.figure(figsize=(6,6))
    plt.scatter(x_vals, y_vals, color='blue', s=10)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.grid(True)
    plt.show()
```

### 2.10.2 Output:

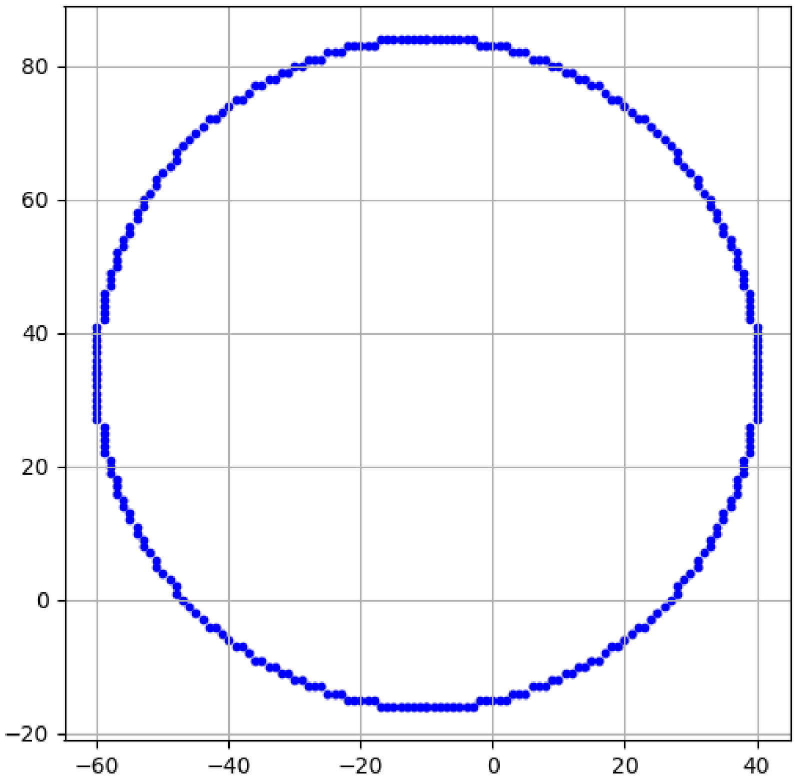


Figure 2.10: Bresenham Circle Drawing

## 2.11 Snowflake Pattern with Fractal Geometry

### 2.11.1 Code:

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

def koch_snowflake(order, scale=10):

    def initial_triangle(scale):
        # Equilateral triangle vertices (upward triangle)
        p1 = np.array([0, 0])
        p2 = np.array([scale, 0])
        p3 = np.array([scale/2, scale*np.sqrt(3)/2])
        return np.array([p1, p2, p3, p1])

    def koch_iteration(points):
        new_points = []
        for i in range(len(points)-1):
            p1 = points[i]
            p2 = points[i+1]
            delta = (p2 - p1) / 3.0
            pa = p1 + delta
            pb = p1 + 2*delta

            angle = -np.pi / 3
            peak = pa + np.array([
                delta[0]*np.cos(angle) - delta[1]*np.sin(angle),
                delta[0]*np.sin(angle) + delta[1]*np.cos(angle)
            ])

            new_points.extend([p1, pa, peak, pb])
        new_points.append(points[-1])
        return np.array(new_points)

    points = initial_triangle(scale)
    for _ in range(order):
        points = koch_iteration(points)
    return points


# Plot Koch snowflake iterations
orders = [0, 1, 2, 3, 4, 5]
fig, axes = plt.subplots(1, len(orders), figsize=(15, 3))

for ax, order in zip(axes, orders):
    points = koch_snowflake(order)
    ax.plot(points[:, 0], points[:, 1], color="blue")
    ax.set_aspect('equal')
    ax.axis("off")
    ax.set_title(f"Order {order}")

plt.show()
```

### 2.11.2 Output:

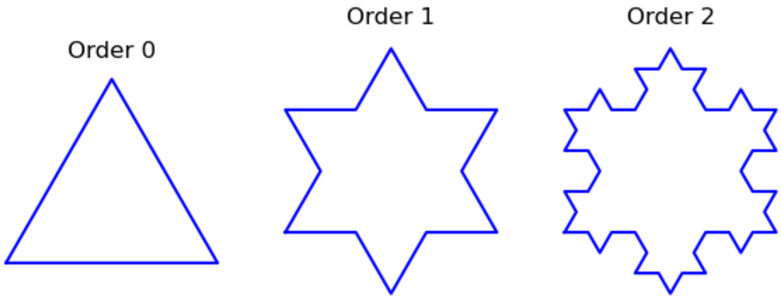


Figure 2.11: Koch Snowflake Pattern

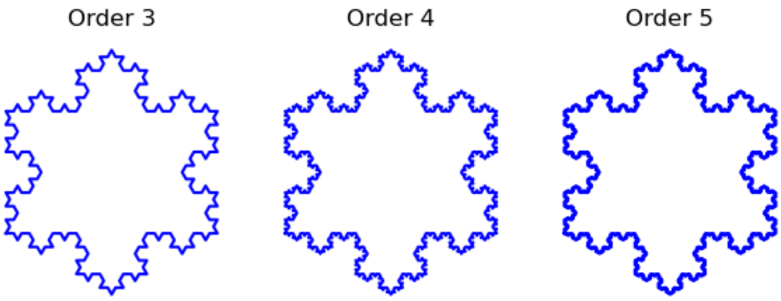


Figure 2.12: Koch Snowflake Pattern