التحويلات الأساسية في 2D

تُعد التحويلات الأساسية الرسوم ثنائية البعد من العمليات الجوهرية التي تتيح تعديل الكائنات الرسومية من حيث الموقع، الحجم، والتوجه.

تشمل هذه التحويلات الإزاحة رسوم Translation، التكبير/التصغير Scaling، والدوران Rotation، وأحياناً القص Shearing.

تُستخدم هذه العمليات لتحريك الأشكال، تغيير أحجامها، أو تدويرها في الفضاء ثنائي الأبعاد، وهي أساسية في تطبيقات مثل الألعاب، برامج التصميم، والرسوم المتحركة.

تعتمد هذه التحويلات على الرياضيات الخطية، وخاصة مصفوفات التحويل، التي تتيح تسلسل العمليات بكفاءة.

في هذا القسم، سنتوسع في شرح هذه التحويلات، مع التركيز على الصيغ الرياضية، تنفيذها برمجياً، والتحديات المرتبطة بها، مع بعض الأمثلة العملية لتوضيح كيفية تطبيق هذه التحويلات في سياقات مختلفة.

التطبيقات العملية

- ✓ **الألعاب** : تحريك الشخصيات أو الكاميرا.
- ✓ **برامج التصميم** : مثل Adobe Illustrator لتعديل الأشكال.
- ✓ **الرسوم المتحركة** : لإنشاء حركات سلسة.
- ✓ **CAD** : لتصميم الأجزاء بدقة.

التحديات

- ☹ **التراكم الرقمي** : العمليات العائمة قد تؤدي إلى أخطاء تراكمية عند التكرار.
- ☹ **الأداء** : الدوران والقص يتطلبان عمليات مثلثية مكلفة. حسابياً
- ☹ **التحويل حول نقطة غير الأصل** : يتطلب إزاحة الأصل، تطبيق التحويل، ثم إعادة الإزاحة.

أنواع التحويلات الأساسية

1- الإزاحة Translation

الإزاحة هي تحريك كائن من موقع إلى آخر دون تغيير حجمه أو توجهه، يتم تمثيل النقطة x,y بعد الإزاحة بـ x', y' كالتالي:

$$x' = x + tx$$

$$y' = y + ty$$

حيث tx و ty هما مقدار الإزاحة في المحورين x و y

الخوارزمية:

```
def Translate(x, y, tx, ty):
    # إضافة مقدار الإزاحة مباشرة إلى الإحداثيات
    x_new = x + tx
    y_new = y + ty
    return x_new, y_new
```

هذه العملية بسيطة للغاية، حيث تضيف قيم الإزاحة إلى الإحداثيات الأصلية.

مثال ذلك: إذا كانت النقطة (2، 3) و tx=5، ty=4 فإن النقطة الجديدة هي (8, 6).

2- التكبير/التصغير Scaling

التكبير يغير حجم الكائن دون تغيير موقعه النسبي أو توجهه، يتم ضرب الإحداثيات في معاملات التكبير.

الصيغة:

$$x' = x * sx$$

$$y' = y * sy$$

حيث sx و sy هما معاملات التكبير في المحورين x و y

مثال ذلك: إذا كانت النقطة (3، 2) و sx=2 ، sy=0.5 فإن النقطة الجديدة هي (4, 1.5)

الخوارزمية:

```
def Scale(x, y, sx, sy):
    # ضرب الإحداثيات في معاملات التكبير
    x_new = x * sx
    y_new = y * sy
    return x_new, y_new
```

إذا كان $sx > 1$ ، يزداد الحجم، وإذا كان $sx < 1$ ، ينخفض.

إذا كان $sx \neq sy$ ، يحدث تكبير غير متجانس، مما قد يشوه الشكل (مثلاً تحويل دائرة إلى شكل بيضوي).

3- الدوران Rotation

الدوران يغير توجه الكائن حول نقطة مرجعية، عادة الأصل (0,0)

يتم استخدام الزاوية θ لتحديد مقدار الدوران.

الصيغة:

$$x' = x * \cos(\theta) - y * \sin(\theta)$$

$$y' = x * \sin(\theta) + y * \cos(\theta)$$

حيث θ هي زاوية الدوران.

الخوارزمية:

```
def Rotate(x, y, theta):
# استخدام الدوال المثلثية لتدوير النقطة
    x_new = x * cos(theta) - y * sin(theta)
    y_new = x * sin(theta) + y * cos(theta)
    return x_new, y_new
```

العمليات المثلثية sin و cos مكلفة حسابياً، لذا يُفضل تخزين القيم مسبقاً للزوايا الشائعة.

مثال ذلك : إذا كانت النقطة (0، 1) و زاوية الدوران $\theta=90°$، فإن النقطة الجديدة هي تقريباً (0,1).

4- القص Shearing

القص يشوه الكائن باتجاه معين، مما يغير شكله دون تغيير حجمه.

الصيغة:

$$x' = x + shx * y$$

$$y' = y + shy * x$$

حيث shx و shy هما معاملا القص.

الخوارزمية:

```
def Shear(x, y, shx, shy):
    # إضافة تأثير القص بناءً على الإحداثيات الأخرى
    x_new = x + shx * y
    y_new = y + shy * x
    return x_new, y_new
```

مثال: إذا كانت النقطة (2 ، 1) و shx=0.5 ، shy=0 فإن النقطة الجديدة هي (2 , 2)

قد يؤدي القص إلى تشوه غير مرغوب إذا لم يتم التحكم في معاملاته بدقة.

5- الدوران حول نقطة معينة

الخوارزمية:

```
def RotateAroundPoint(x, y, cx, cy, theta):
    # (cx, cy) نقل النقطة إلى الأصل نسبة إلى
    x_temp = x - cx
    y_temp = y - cy
    # تطبيق الدوران
    x_new = x_temp * cos(theta) - y_temp * sin(theta) + cx
    y_new = x_temp * sin(theta) + y_temp * cos(theta) + cy
    return x_new, y_new
```

يجب التأكد من دقة النقطة المرجعية (cx, cy) لتجنب انحراف الدوران.

ما تم شرحة من تحويلات هو لنقطة واحدة فقط ، الآن سنرى كيف يمكن تعميم التحويلات لأكثر من نقطة.

6- التعميم

6-1- تعميم الإزاحة لـ مستقيم:

استخدمانا الدالة Translate لإزاحة نقطة ، المستقيم مكون من نقطتين (x1,y1) ، (x2,y2) نُزيح كل نقطة باستخدام نفس الدالة.

```
def TranslateLine(x1, y1, x2, y2, tx, ty):
# إزاحة النقطة الأولى باستخدام نفس الدالة
    x1_new, y1_new = Translate(x1, y1, tx, ty)
# إزاحة النقطة الثانية باستخدام نفس الدالة
    x2_new, y2_new = Translate(x2, y2, tx, ty)
# النتيجة
    return x1_new, y1_new, x2_new, y2_new
```

6-2- تعميم الإزاحة لـ مضلع:

المضلع = قائمة من النقاط points ، نُزيح كل نقطة في القائمة باستخدام Translate

points: [(x1,y1), (x2,y2), …], tx, ty: مقدار الإزاحة

```
def TranslatePolygon(points, tx, ty):
  # قائمة فارغة لتخزين النقاط الجديدة
    translated_points = []
  # حلقة على كل نقطة في المضلع
    for x, y in points:
      # إزاحة النقطة الحالية باستخدام نفس الدالة
        x_new, y_new = Translate(x, y, tx, ty)
      # إضافة النقطة الجديدة إلى القائمة
        translated_points.append((x_new, y_new))
    return translated_points
```

6-3- تعميم التكبير / التصغير لـ مضلع Polygon

المضلع = قائمة من النقاط points: [(x1,y1), (x2,y2), …]

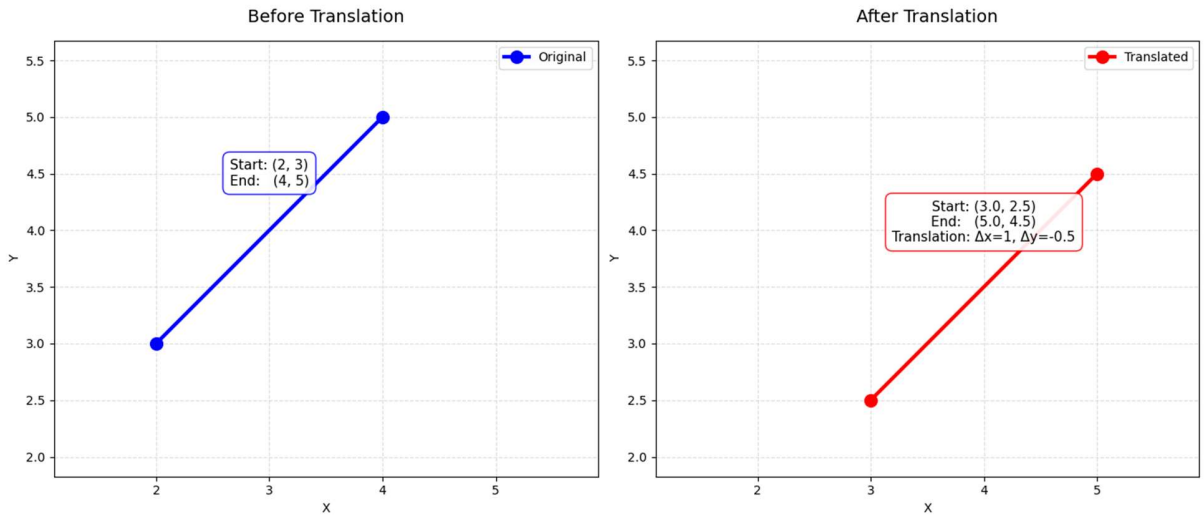
نُكبّر / نصغر كل نقطة في القائمة بالمعاملات sx, sy بالنسبة للأصل (0,0)

ترجع: قائمة جديدة من النقاط المُكبّرة أو المصغرة

```
def ScalePolygon(points, sx, sy):
   scaled_points[] =
   for x, y in points:
      x_new, y_new = Scale(x, y, sx, sy)
      scaled_points.append((x_new, y_new))
   return scaled_points
```

دالة لإزاحة مستقيم:

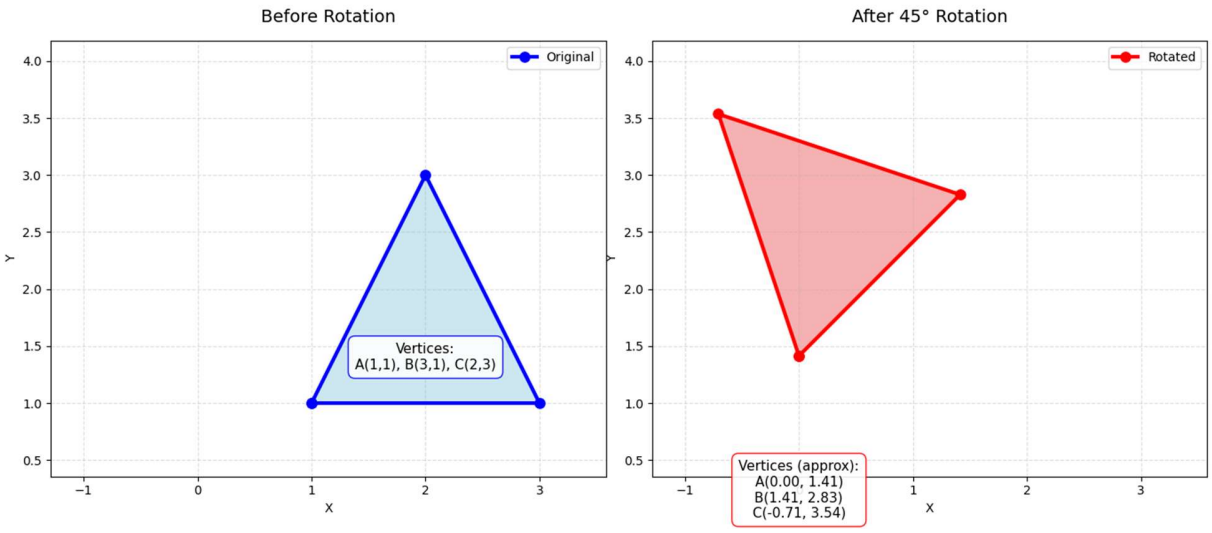
```
# ------------------------------------------------
# 1. Translation functions
# ------------------------------------------------
def Translate(x, y, tx, ty):
    """Translate point (x, y) by (tx, ty)."""
    return x + tx, y + ty

def TranslateLine(x1, y1, x2, y2, tx, ty):
    """Translate line segment."""
    x1n, y1n = Translate(x1, y1, tx, ty)
    x2n, y2n = Translate(x2, y2, tx, ty)
    return x1n, y1n, x2n, y2n

# ------------------------------------------------
# 2. Short line + small translation
# ------------------------------------------------
x1, y1 = 2, 3      # start point (shorter line)
x2, y2 = 4, 5      # end point   (length = 2√2 ≈ 2.8)
tx, ty = 1, -0.5   # small translation

# Translated line
x1n, y1n, x2n, y2n = TranslateLine(x1, y1, x2, y2, tx, ty)

# ------------------------------------------------
# 3. All points
# ------------------------------------------------
all_x = np.array([x1, x2, x1n, x2n])
all_y = np.array([y1, y2, y1n, y2n])

# ------------------------------------------------
# 4. Axis limits with 40% padding
# ------------------------------------------------
margin = 0.4  # 40% padding
x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()

x_range = max(x_max - x_min, 1)
y_range = max(y_max - y_min, 1)

x_lim = [x_min - x_range * margin, x_max + x_range * margin]
y_lim = [y_min - y_range * margin, y_max + y_range * margin]
```



دالة لدوران مثلث:

```
# ------------------------------------------------
# 1. Rotation of a single point (around origin)
# ------------------------------------------------
def Rotate(x, y, theta_deg):
    """
    Rotate point (x, y) by theta_deg degrees (counter-clockwise) around (0,0).
    Returns (x_new, y_new).
    """
    theta = radians(theta_deg)
    x_new = x * cos(theta) - y * sin(theta)
    y_new = x * sin(theta) + y * cos(theta)
    return x_new, y_new

# ------------------------------------------------
# 2. Rotate a triangle (list of 3 points)
# ------------------------------------------------
def RotateTriangle(points, theta_deg):
    """
    Rotate a triangle given as list of tuples: [(x1,y1), (x2,y2), (x3,y3)]
    Returns new list of rotated points.
    """
    rotated = []
    for x, y in points:
        x_new, y_new = Rotate(x, y, theta_deg)
        rotated.append((x_new, y_new))
    return rotated

# ------------------------------------------------
```

```
# 3. Original triangle (small, centered near origin)
# ------------------------------------------------
triangle = [
    (1.0, 1.0),   # vertex A
    (3.0, 1.0),   # vertex B
    (2.0, 3.0)    # vertex C
]

theta_deg = 45  # rotation angle

# ------------------------------------------------
# 4. Rotated triangle
# ------------------------------------------------
rotated_triangle = RotateTriangle(triangle, theta_deg)

# ------------------------------------------------
# 5. Prepare data for plotting (close the polygon)
# ------------------------------------------------
def close_polygon(pts):
    x, y = zip(*pts)
    return list(x) + [x[0]], list(y) + [y[0]]

orig_x, orig_y = close_polygon(triangle)
rot_x,  rot_y  = close_polygon(rotated_triangle)

# ------------------------------------------------
# 6. Same axis limits for both sub-plots
# ------------------------------------------------
all_x = np.array(orig_x + rot_x)
all_y = np.array(orig_y + rot_y)

margin = 0.4
x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()

x_range = max(x_max - x_min, 1)
y_range = max(y_max - y_min, 1)

x_lim = [x_min - x_range * margin, x_max + x_range * margin]
y_lim = [y_min - y_range * margin, y_max + y_range * margin]
```



دالة لتغير حجم دائرة:

```
import matplotlib.pyplot as plt
import numpy as np
# ------------------------------
# 1. الدوال الأساسية
# ------------------------------
def Scale(x, y, sx, sy):
    """تكبير نقطة"""
    return x * sx, y * sy

def ScaleCircle(cx, cy, r, sx, sy):
    """تكبير دائرة (تقريب دائري)"""
    cx_new, cy_new = Scale(cx, cy, sx, sy)
    r_new = r * ((sx + sy) / 2)      # متوسط التكبير
    return cx_new, cy_new, r_new
# ------------------------------
# 2. دائرة أصلية
# ------------------------------
cx, cy = 2, 3
r = 2
# 1.5× رأسي ،2× تكبير أفقي
sx, sy = 2.0, 1.5

# ------------------------------
# 3. تكبير الدائرة
# ------------------------------
cx_new, cy_new, r_new = ScaleCircle(cx, cy, r, sx, sy)

# ------------------------------
```

```python
# 4. إنشاء نقاط المحيط
# ------------------------------
theta = np.linspace(0, 2*np.pi, 200)

# الأصلية
x_orig = cx + r * np.cos(theta)
y_orig = cy + r * np.sin(theta)

# بعد التكبير
x_new = cx_new + r_new * np.cos(theta)
y_new = cy_new + r_new * np.sin(theta)

# ------------------------------
# 5. حساب حدود المحاور الموحدة
# ------------------------------
all_x = np.concatenate([x_orig, x_new])
all_y = np.concatenate([y_orig, y_new])

x_min, x_max = all_x.min(), all_x.max()
y_min, y_max = all_y.min(), all_y.max()

# إضافة هامش 10%
margin_x = (x_max - x_min) * 0.1
margin_y = (y_max - y_min) * 0.1

x_lim = [x_min - margin_x, x_max + margin_x]
y_lim = [y_min - margin_y, y_max + margin_y]
```