

CentriPy, a Centroid Analysis System Using Python

MAT 122 Statics

Fall 2024

Project Team

Student ID	Student Name	Project Evaluation	
221101140	Nour Maher	Report	
		Presentation	
		Code	
221101030	youssef yehia	Report	
		Presentation	
		Code	
221100049	abdullah hossam	Report	
		Presentation	
		Code	
221101013	youssef gamal	Report	
		Presentation	
		Code	
221100128	Ahmed hassany	Report	
		Presentation	
		Code	
221100979	Yassin waleed	Report	
		Presentation	
		Code	

Under supervision of Professor: Ahmed Hazem Eltanboly

Professor: Amr

2024-2025

Project Overview

Centripy is a Python-based application designed for educational and analytical purposes.

It provides an interactive platform for users to draw, analyze, and manage various geometric shapes while exploring their individual and composite centroids.

Built using the Tkinter library, Centripy combines mathematical computations with graphical interactivity to offer an engaging experience.

The tool is particularly useful for students, engineers, and educators to understand geometric principles intuitively.

Scientific Background

What is a Centroid?

- The centroid is the geometric center or “center of mass” of a shape.
- It’s the average position of all the points in a shape or object.
- For simple geometric shapes, the centroid is determined by symmetry or calculated using mathematical formulas.

Why Are Centroids Important?

- Applications in Engineering: Centroids are critical for structural analysis, robotics, and design.
- Physics and Mathematics: Used to determine balance points and moments of inertia.
- Real-World Applications: Designing stable structures, finding centers of gravity, and optimizing shapes for efficiency.

Objectives & Project idea

1. Develop a user-friendly graphical interface for creating and manipulating geometric shapes.
2. Implement accurate mathematical algorithms to compute centroids and areas of geometric shapes.
3. Provide dynamic tools for recalculating centroids when shapes are added or removed.
4. Enable data import/export functionality for persistence and interoperability.
5. Offer a detailed help system to guide users through the application's features and functionality.

Problem:

1. Complex Shapes:

- Calculating centroids for irregular or composite shapes is significantly more challenging compared to simple, symmetric shapes.
- Irregular geometries often lack symmetry, requiring advanced mathematical techniques, such as integration, to determine the centroid accurately.

2. Visualization Issues:

- Complex geometries make it difficult to intuitively visualize the centroid's location.
- Graphical representation tools often fail to provide clarity, especially for composite or irregularly shaped objects.

3. Manual Calculations:

- Manual computation of centroids for intricate shapes can be tedious and error-prone.
- These calculations often involve detailed knowledge of calculus and require significant time investment to ensure accuracy.

4. Composite Structures:

- Combining multiple shapes into a single centroid calculation adds layers of complexity.
- The weighted average approach for composite centroids is highly dependent on the precision of individual shape properties, such as area and partial centroids.

Development Process

1. Requirements Analysis

- Identify shapes to support and define their properties.
- Specify user interactions, such as drawing shapes, deleting them, and calculating centroids.

2. Design

- Create a Shape base class with abstract methods for centroid and area calculations.
- Design derived classes for each shape, implementing specific algorithms.
- Develop a Tkinter-based GUI with a canvas for interactivity and a toolbar for control options.

3. Implementation

- Implement individual shape classes with centroid and area formulas.
- Build GUI elements, including buttons, dropdowns, and event bindings.
- Add features for importing/exporting JSON data and recalculating centroids dynamically.

4. Testing and Debugging

- Validate mathematical calculations for centroids and areas.
- Test GUI interactions for responsiveness and usability.
- Ensure compatibility and robustness for importing/exporting data.

5. Enhancements

- Added JSON serialization for data storage.
- Included a help dialog for improved user experience.

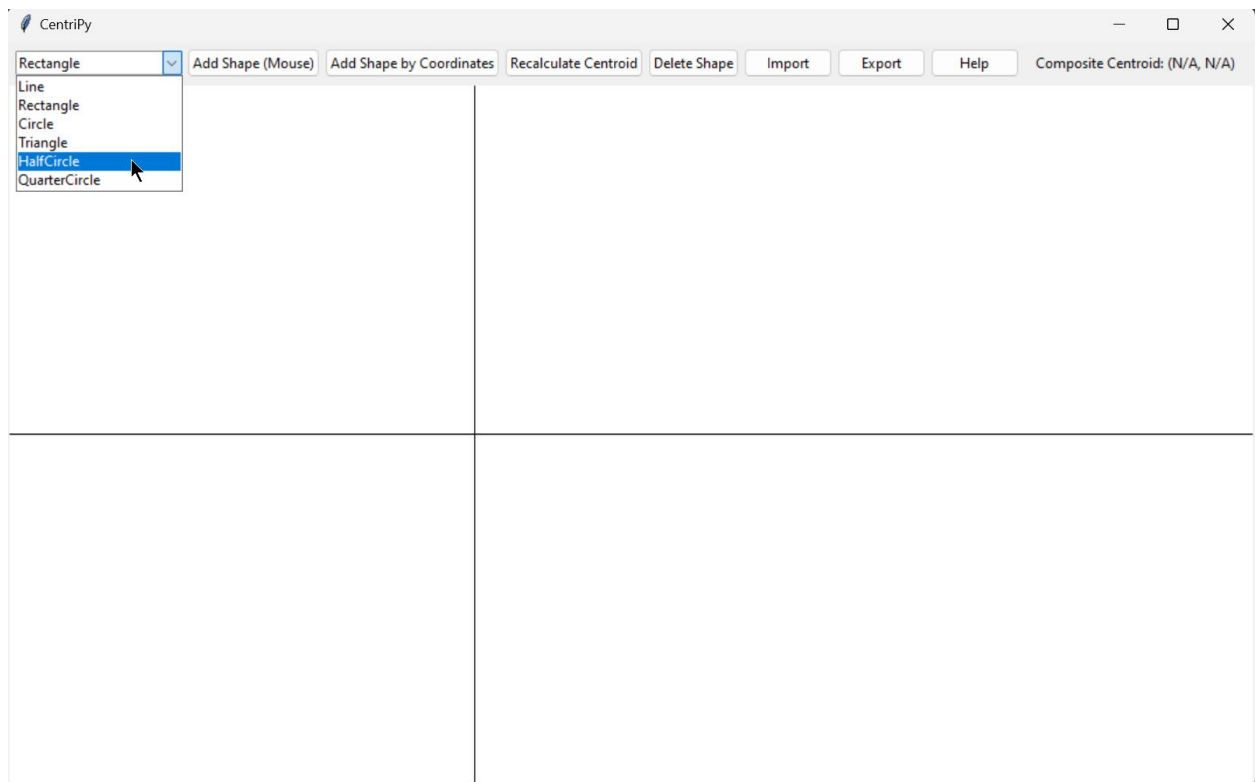
Features and Project Analysis

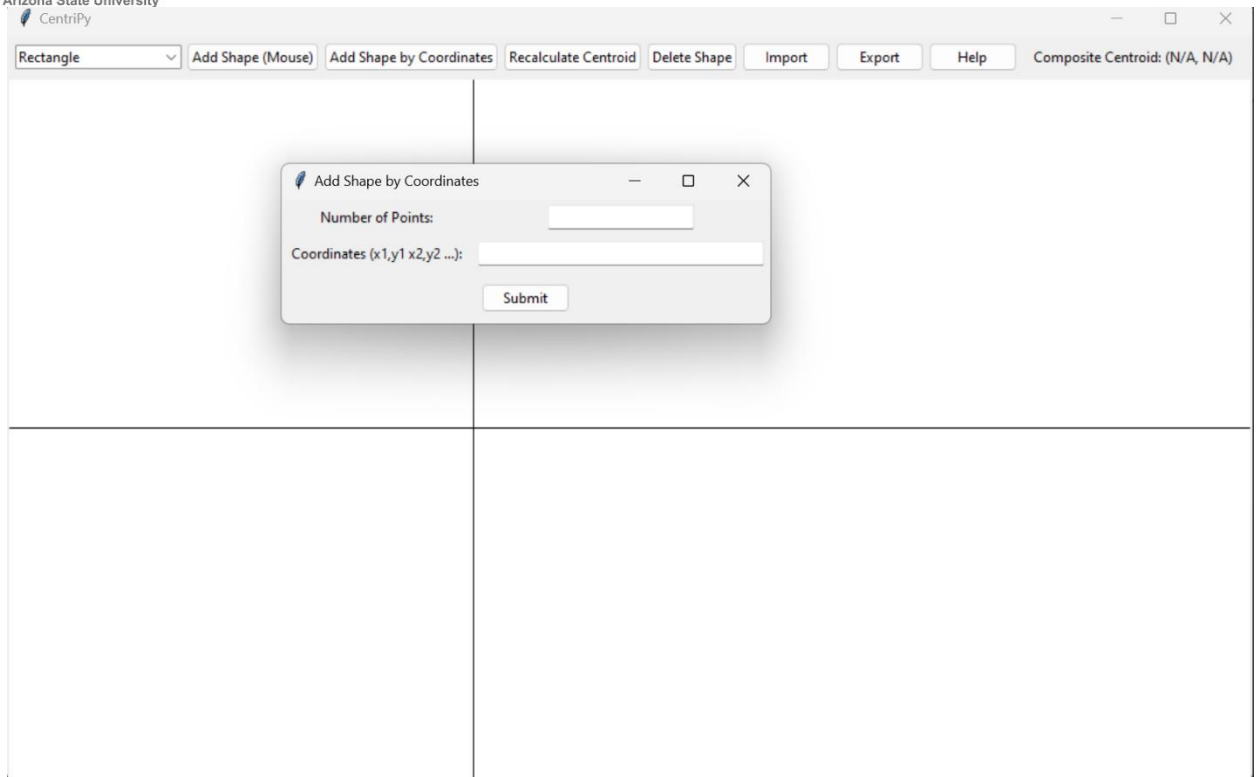
1. Supported Geometric Shapes

The application currently supports the following shapes:

- **Line:** The centroid is the midpoint of the line segment, and the area is zero.
- **Rectangle:** The centroid is the center of the rectangle, and the area is calculated as width \times height.
- **Circle:** The centroid is the center point of the circle, and the area is calculated as $\pi \times \text{radius}^2$.
- **Triangle:** The centroid is the average of its vertices, and the area is determined using the Shoelace formula.
- **Half-Circle:** The centroid is offset vertically from the flat side by $4r/(3\pi)$, and the area is half that of a full circle.
- **Quarter-Circle:** The centroid is offset diagonally from the flat sides by $4r/(3\pi)$, and the area is one-fourth that of a full circle.

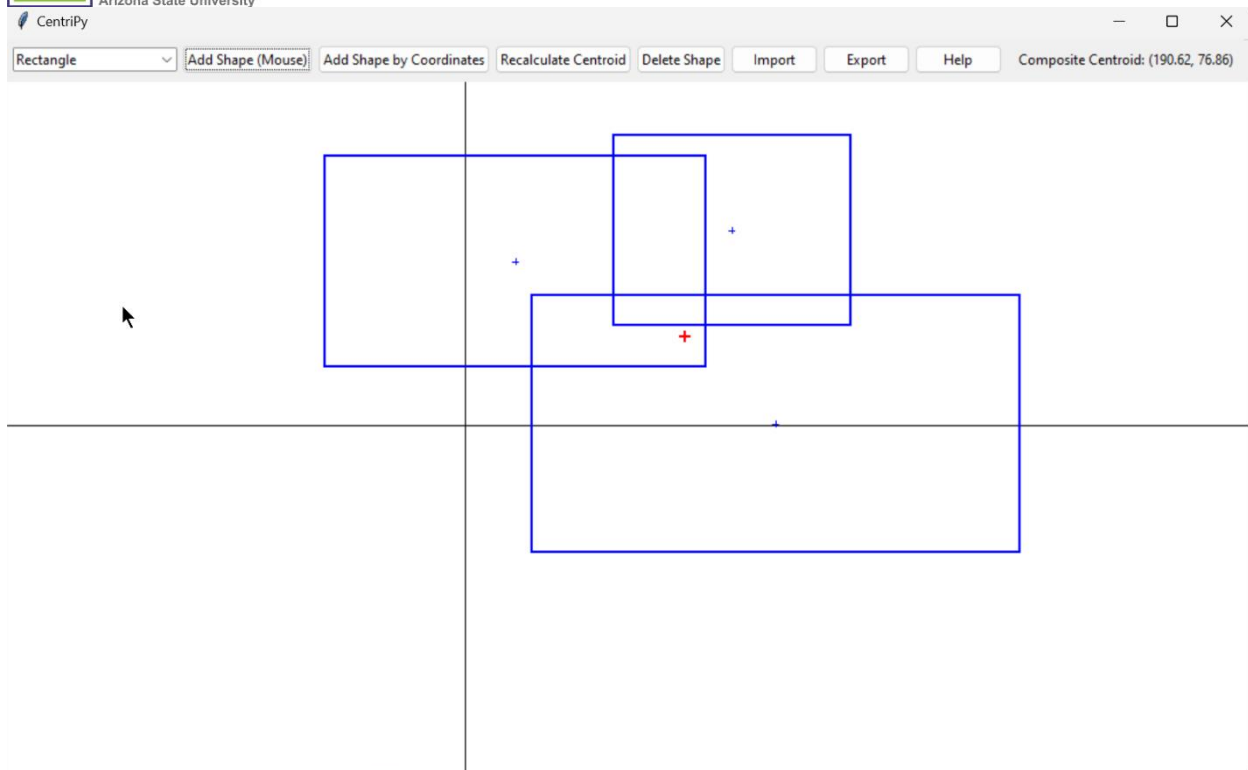
Also supports inserting shapes By coordinates if they were complex shapes (ie. Irregular shapes / higher number of vertices than 3)





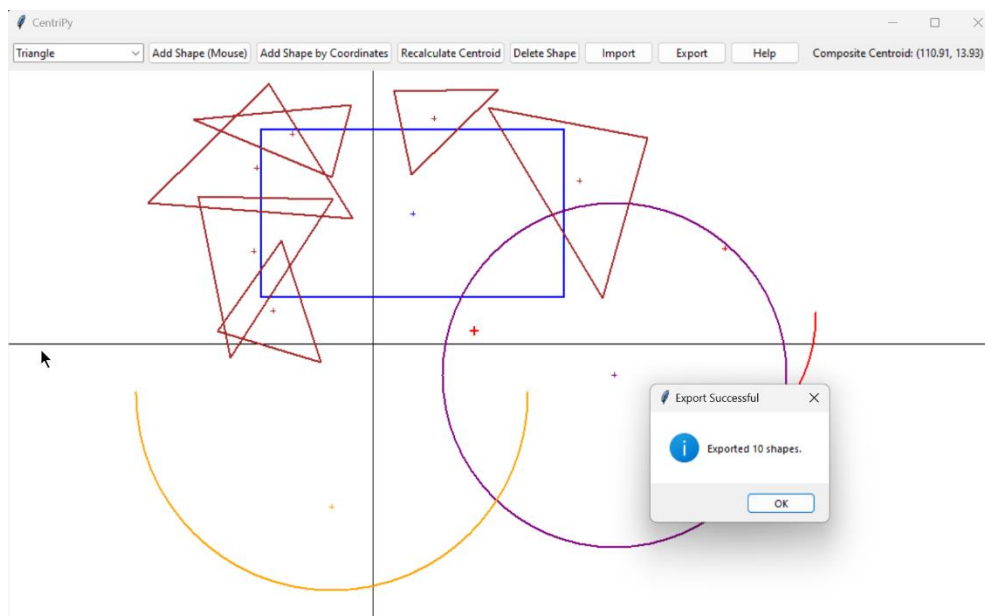
2. Graphical Interface

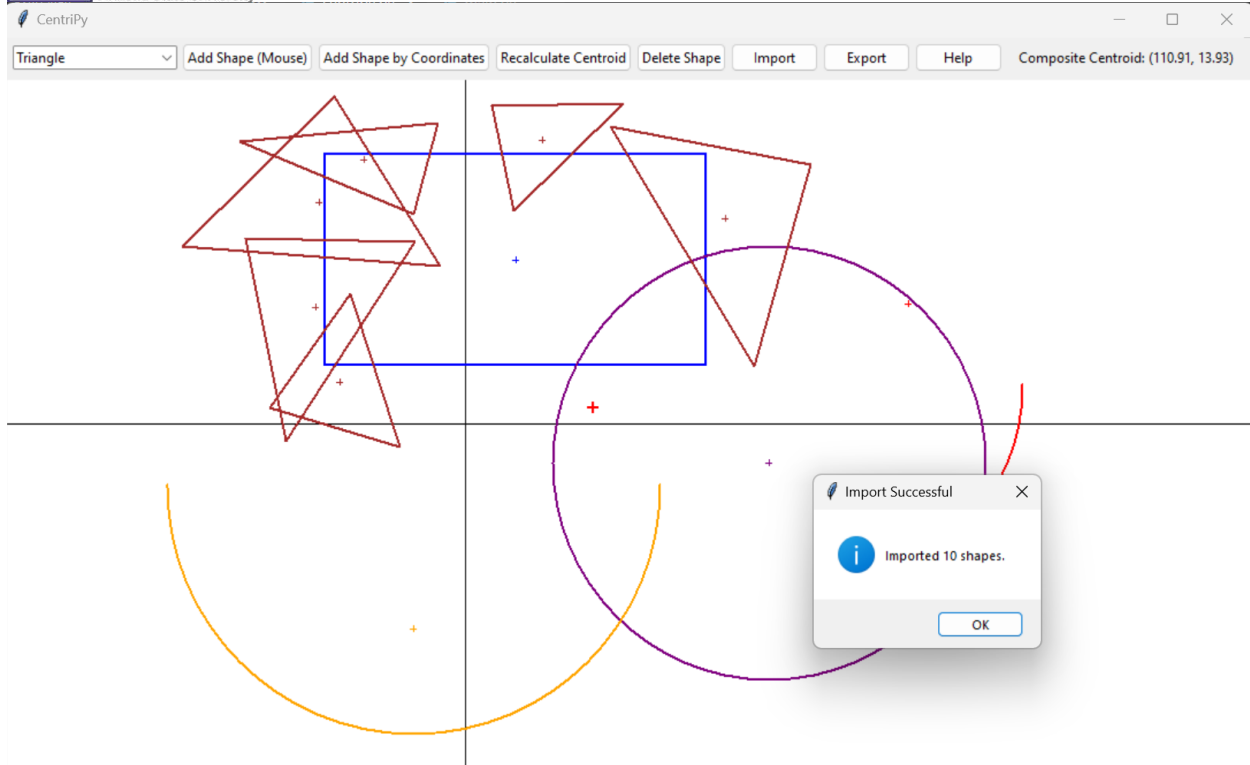
- **Canvas:** A drawing area where users can add and manipulate shapes interactively.
- **Toolbar:** Tools for selecting shapes, toggling delete mode, recalculating centroids, and managing data.
- **Axes:** Cartesian coordinate axes for spatial reference, with the origin marked on the canvas.
- **Real-Time Feedback:** The composite centroid is dynamically calculated and displayed as shapes are added or removed.



3. Data Management

- **Export:** Save the current shapes and their properties to a JSON file.
- **Import:** Load shapes from a JSON file, restoring their attributes and positions for further analysis.





4. Help System

- A detailed help dialog provides step-by-step guidance on how to use the application, including creating shapes, managing data, and understanding calculations.



Coordinates

Recalculate Centroid

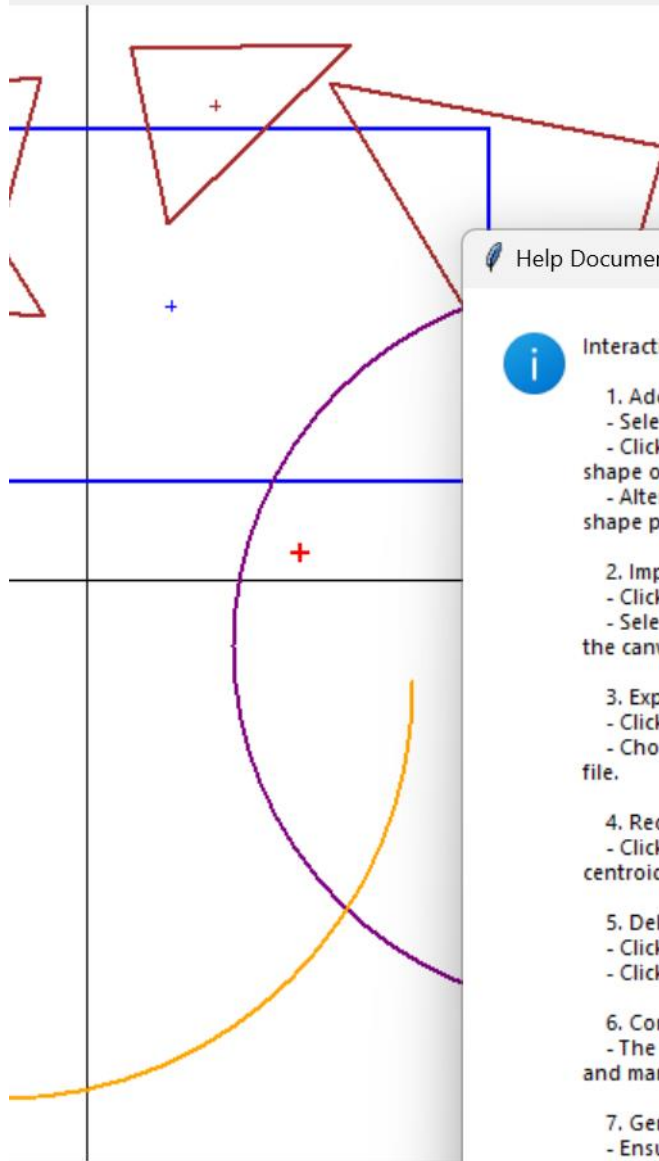
Delete Shape

Import

Export



Help


Composite Centroid: (11



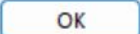
```
toolbar = ttk.Frame(self.root)
toolbar.grid()

def activate_shape(shape_type):
    """
    Activated: click on a shape to re
    : TriangleShape
    : TriangleShape
    """
```

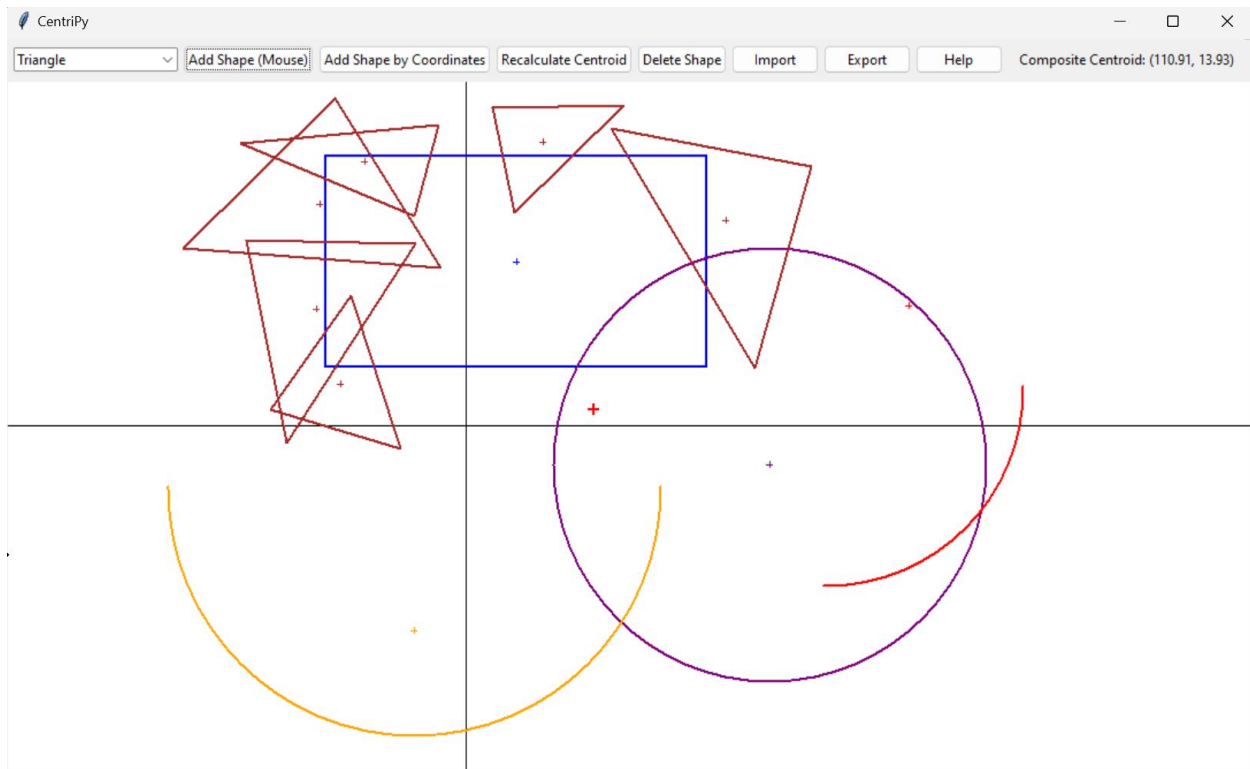
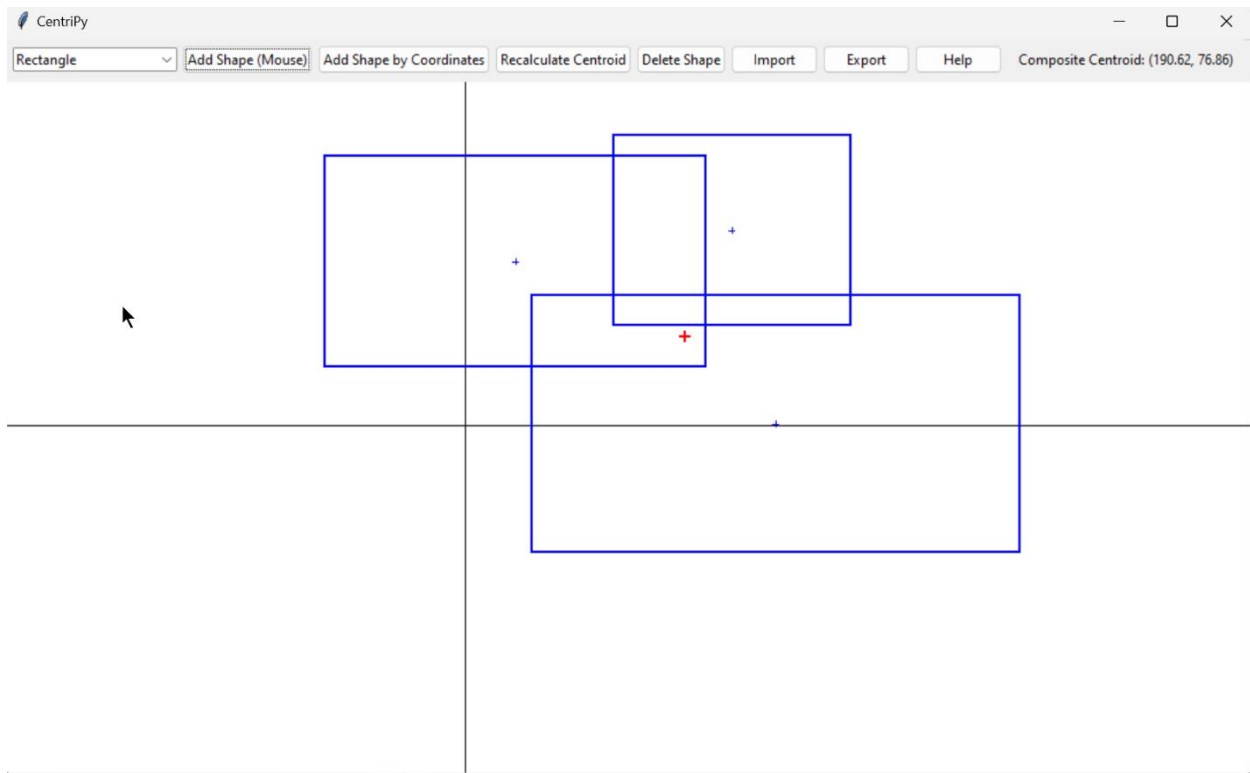
 Help Documentation 

 Interactive Centroid Finder - Help

1. Adding Shapes:
 - Select a shape from the dropdown menu.
 - Click 'Add Shape (Mouse)' and use the mouse to draw the shape on the canvas.
 - Alternatively, click 'Add Shape by Coordinates' to input shape parameters manually.
2. Importing Shapes:
 - Click the 'Import' button.
 - Select a JSON file containing shapes to load them onto the canvas.
3. Exporting Shapes:
 - Click the 'Export' button.
 - Choose a location to save the current shapes to a JSON file.
4. Recalculating Centroid:
 - Click 'Recalculate Centroid' to update the composite centroid based on all shapes.
5. Deleting Shapes:
 - Click 'Delete Shape' to enter delete mode.
 - Click on a shape to remove it from the canvas.
6. Composite Centroid:
 - The current composite centroid is displayed on the toolbar and marked with a red cross on the canvas.
7. General Tips:
 - Ensure to save your shapes using the 'Export' feature to avoid losing data.
 - Imported shapes will retain their properties and can be further manipulated.



5. Visualizations



Key Calculations

Centroid Calculations

The centroid calculations vary depending on the shape:

Rectangle:

The centroid of a rectangle is the midpoint of its diagonals.

Calculation:

$$C_x = (x_1 + x_2) / 2, \quad C_y = (y_1 + y_2) / 2$$

Here, (x_1, y_1) and (x_2, y_2) are the coordinates of two opposite corners.

Triangle:

The centroid of a triangle is the average of its three vertices.

Calculation:

$$C_x = (x_1 + x_2 + x_3) / 3, \quad C_y = (y_1 + y_2 + y_3) / 3$$

Here, (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) are the vertices of the triangle.

Circle:

The centroid of a circle is its center.

Calculation:

$$C_x = C_{\text{center},x}, \quad C_y = C_{\text{center},y}$$

Half-Circle:

For a semicircle, the centroid is offset along the y-axis by $-r + 4r / (3\pi)$ from the flat side.

Calculation:

$$C_x = C_{\text{center},x}, \quad C_y = C_{\text{center},y} + (-r + 4r / (3\pi))$$

Here, r is the radius of the semicircle.

Quarter-Circle:

For a quarter-circle, the centroid is offset diagonally by $4r / (3\pi)$.

Calculation:

$$C_x = C_{\text{center},x} + 4r / (3\pi), \quad C_y = C_{\text{center},y} + 4r / (3\pi)$$

r is the radius of the quarter-circle.

Composite Centroid

The composite centroid is calculated as a weighted average of the centroids of individual shapes based on their areas. The formula used in the code is:

$$C_{\text{composite},x} = \sum(C_{x,i} \times A_i) / \sum A_i, \quad C_{\text{composite},y} = \sum(C_{y,i} \times A_i) / \sum A_i$$

Where:

- A_i is the area of the i -th shape.
- $C_{x,i}$ and $C_{y,i}$ are the x and y coordinates of the centroid of the i -th shape.
- The total area is the sum of the areas of all shapes:

$$A_{\text{total}} = \sum A_i$$

Code Snippets

1. Adding a Rectangle

```
class RectangleShape(Shape):
    def __init__(self, x1, y1, x2, y2):
        super().__init__()
        self.x1, self.y1 = x1, y1
        self.x2, self.y2 = x2, y2

    def calculate_area(self):
        return abs(self.x2 - self.x1) * abs(self.y2 - self.y1)

    def calculate_centroid(self):
        cx = (self.x1 + self.x2) / 2
        cy = (self.y1 + self.y2) / 2
        return (cx, cy)
```

2. Calculate Centroid for half Circle

```
def calculate_area(self):
    return 0.5 * math.pi * (self.r ** 2)

def calculate_centroid(self):
    # For a top half circle with flat side at y = cy - r:
    offset = -self.r + (4 * self.r) / (3 * math.pi)
    return (self.cx, self.cy + offset)
```

Testing The app:

Used the following coordinates to verify the centroids of some complex shapes

Shape: Rectangle

- Vertex 1: -50,-25 - Vertex 2: 50,-25 - Vertex 3: 50,75 - Vertex 4: -50,75

Single input string: -50,-25 50,-25 50,75 -50,75

Shape: Irregular Hexagon

- Vertex 1: -50,0 - Vertex 2: -30,50 - Vertex 3: 30,50 - Vertex 4: 50,0 - Vertex 5: 30,-40 - Vertex 6: -30,-40

Single input string: -50,0 -30,50 30,50 50,0 30,-40 -30,-40

Shape: Irregular Pentagon

- Vertex 1: -60,0 - Vertex 2: -30,60 - Vertex 3: 30,40 - Vertex 4: 60,-10 - Vertex 5: 0,-50

Single input string: -60,0 -30,60 30,40 60,-10 0,-50

Shape: Irregular Octagon

- Vertex 1: -70,20 - Vertex 2: -50,50 - Vertex 3: 0,70 - Vertex 4: 50,50
- Vertex 5: 70,20 - Vertex 6: 50,-30 - Vertex 7: 0,-50 - Vertex 8: -50,-30

Single input string: -70,20 -50,50 0,70 50,50 70,20 50,-30 0,-50 -50,-30

Shape: Star

- Vertex 1: 0,50 - Vertex 2: 20,15 - Vertex 3: 50,15 - Vertex 4: 25,-5 - Vertex 5: 35,-50
- Vertex 6: 0,-25 - Vertex 7: -35,-50 - Vertex 8: -25,-5 - Vertex 9: -50,15 - Vertex 10: -20,15

Single input string: 0,50 20,15 50,15 25,-5 35,-50 0,-25 -35,-50 -25,-5 -50,15 -20,15

Conclusion and Future Directions

Centripy successfully combines mathematical computations with graphical interactivity, making it a valuable tool for learning and analyzing geometric principles.

Its extensible design ensures that future enhancements can be seamlessly integrated.

The project demonstrates a practical application of Python's Tkinter library for interactive software development.

Future Work

1. **Additional Shape Support:** Extend support to ellipses and custom-defined shapes.
2. **Advanced Export Options:** Add support for exporting data in formats like CSV or XML.
3. **Geometric Analysis:** Integrate additional features, such as moment of inertia and perimeter calculations.
4. **Interactive Shape Editing:** Allow users to modify shapes directly on the canvas.
5. **Modernized UI:** Update the graphical interface with frameworks like PyQt or advanced Tkinter themes for a more polished look.