MA1008 Introduction to Computational Thinking
Mini Project: Working with Polygons
Semester 1, AY 2021/2022, Week 10 – Week 13

1. Introduction

The objective of the mini project is for you to produce a program of a moderate size and depth that will require you to utilise what you have learned in this course, and more, to do something interesting. Through this, you would learn to design, manage and execute a sizable program.

2. The Project

This project is on working with polygons using interactive graphics. We all know computers can interact with users and display infinitely different pictures, either hand drawn, from cameras or synthesized using algorithms. This project exposes you to just a little of the inside of this fascinating technology – computer graphics – over the very limited domain of polygons. Yet polygons are the basic building blocks of much fantastic computer graphics imagery that you see in computer-aided design, cinematic animation, simulation displays, and more.

The Program

You are to write a program that can create, store, retrieve and manipulate polygons. Your program should have the following capabilities:

i. Polygon Creation

Create a polygon, store its data in a file, and retrieve it when needed. A polygon is represented by its vertices. You need to provide two methods of input:

a. Type in their coordinates or
b. Generate them by mouse clicks in the graphics display.

Both capabilities need to be provided in your program. You would need to setup the data structure for your polygon first. We exclude self-intersecting polygons from this project. Hence your program should raise an error and prevent the creation of a self-intersecting polygon.

ii. Polygon Edges

A polygon can have as many edges as the user desires. You need to allow two types of edges:

a. Straight. A straight edge is defined by two adjacent polygon vertices.
b. Curved. Curved edges are discussed in Appendix 1.

iii. Displaying a Polygon

Display the polygon graphically. There are two display options:

a. the outline of the polygon and
b. the interior filled with a colour.

A display may use one or both the options. The user should be able to choose the colour for both the options. You may display multiple different polygons at the same time, each with its own display options.

iv. Manipulating a polygon

There are many possible manipulations. You need to provide the following five:

a. Modify the polygon by deleting an existing vertex or inserting a new one. Then display the outcome.
b. Move the polygon by a specific distance in a specific direction. (Panning)
c. Rotate the polygon by a specific angle about a specific point.
d. Scale the polygon up or down by a specific factor.
e. Produce multiple copies of the polygon, either linearly or by rotational about a point, with or without scaling.

v. Analytical Tools using a Polygon

There are numerous useful things one can do with a polygon. Provide these in your program:

a. Find the perimeter length of the polygon.
b. Find the area of the polygon.
c. Determine if a given point is inside, outside or on the boundary of the polygon.
d. Select a polygon by pointing to it with the mouse cursor, and then manipulate it. The selection can be done using the capability of Item c above or by pointing to an edge of the polygon, which means your program needs to know if the cursor is at an edge. This capability is required when you have multiple polygons displayed and need to manipulate individual ones.
e. (Optional) Set operations: Join two overlapping polygons into one (union), or subtract one polygon from another (difference), or produce their common area (intersection). The algorithm for doing this requires the capability in Item c above also. The algorithm will be provided to those who are interested to pursue this option.

See Appendix 2 for the mathematics and algorithms for the operations in iv and v. You may provide other operations apart from those listed. List and describe them in your report.

vi. File Input and Output

Provide a data file that stores polygons that you have created. Hence your program should have two means of input:

a. The method described above in Item i Polygon Creation.
b. Reading in from file.

Therefore, you need functions for writing polygons to file and reading them back from file one by one. Note that upon reading in a polygon, your program needs to allow the user to operate on the polygon before reading in the next one.

Here are some points you should note:

i. Your screen size is fixed, but your polygon size is variable. Therefore, you should scale your display such that the polygon fits nicely in the display window.
ii. You will need a user interface for the operations you provide. Good to have it as a graphical user interface, which means you provide a menu on the graphics screen and the user selects the options in the menu via mouse clicks.
iii. Mouse clicks do not give exact locations. Hence, if you want exact coordinates when you enter polygon vertices by mouse clicks, then you can impose a grid over the display area and a mouse click snaps to the nearest grid point.

3. Resources

The main extra resource that you will need for this project is for the graphical display, for which you need the graphics library provided with Python, called **Turtle**. Turtle capabilities and functions are described fully in Section 24.1 of the Python documentation, which you can access via the IDLE interactive shell by clicking Help > Python Docs and then search for "Turtle". Appendix 3 provides some guidance on the use of Turtle.

4. Prohibitions

Apart from the graphics library, everything you need for this project can be done through what you have learnt in this course. To allow you to exercise what you have learned and to prevent you from veering wildly beyond the scope of this project, your program should:

i. Not use the *class* construct to define objects you require in your program
ii. Only use the Turtle library for graphics. You must not use *tkinter* or *matplotlib*.
iii. Not use the libraries *numpy* or *pygame*.

You may use the standard Python libraries like *math*. As for other Python libraries, please seek permission before you use them.

5. Submission

You need to submit the following:

i. **A working Python program** that gets the relevant inputs from the user or from a file and displays the polygons and allows the user to interactively work with the polygons.
ii. **A data file** containing the required data for three different polygons. Your grader will be using these data to test run your program.
iii. **A report in a Word file**, providing
   - A guide on how to run your program
   - A list of the capabilities you provide for manipulating polygons, including those listed above.
   - Pictures of your graphics window, with displays of the polygons you have created.
   - Three different pictures of multiple polygons duplicated linearly or in rotation.
   - You should also include any pictures that will help your reader comprehend and appreciate your program better. For example, if you use grids to help in generating your inputs, then a picture showing the grid with the polygon would be useful. Or in your function to determine the position of a point with respect to a polygon, you can display the polygon, the point and the line you cast to determine the position of the point, and highlight the points of intersection between the line and the polygon.
   - Highlights of the **key strengths and limitations** of the program.

The **deadline** for project submission is **23:59 hours, Friday 15 April 2022**. Please submit all your files through the course site using the same link to the project file.

6. Grading Rubrics

Here are what the graders will be looking for when grading:

i. The quality and correctness of the program and the outputs, especially the graphics display.
ii. The ease in interacting with the program, which includes specifying the input data and working with the input files.
iii. The quality and correctness of your program, which includes:
   - How easy it is to read and understand your program. This means your program should be adequately commented with good choice of variable and function names.
   - Modularisation of the program including appropriate use of functions, the design of the functions which include the appropriateness of parameters in the functions.

7. Epilogue

You should start working on the program immediately. Do not wait till the last week. What you produce depends on the time you spend on the project and your ability in creating good algorithms and writing good code.

To get started, you should first decide on how to represent a polygon in your program – remember that a polygon is defined by its vertices, which means you should be storing the vertices. This is fine for polygons with straight edges, since a straight line requires only the two end points. But there are complications with curved edges. They still require the end points, but there is extra data to define a curve. I will discuss this and more when I discuss the project in the lecture in Week 10. However, for a start, you can work with polygons with straight edges only, as this is simpler. Once you have gained confidence with straight-edge polygons, you can extend it to include curved edges.

This is an individual project. You may consult the tutors and discuss with your classmates, but everyone must write their own program. Any programs deemed to have been copies of each other will be penalised heavily, regardless of who is doing the copying.

If in doubt or in difficulty, always ask. And ask early!

Good news: Dr. Arif, one of our seasoned tutors, will be available for consultation every Thursday afternoon from 1:30 pm in CAE Lab 2.

Appendix 1: How to specify curves

There are many different types of curves one can use to define curved edges. Here, we offer only two which you may use in your program. You may use other curve types, but then you would need to provide the mathematical formulation in your report.

1. Circular arcs

A circular arc is a part of a circle subtended over a certain angle at the centre. There are different ways to specify a circular arc:

i. The centre, the start point, and the angle the arc subtends at the centre. When the angle is positive, the arc goes anticlockwise, and clockwise when the angle is negative.
ii. Three points – the start point, the end point and an intermediate point which the arc goes through. You can compute for the centre and the radius of the arc from the three points.

There are other ways to specify an arc. You may provide your own.

Given the centre of the arc at (cx, cy) and radius r, a point (x, y) on the arc at angle θ is given by

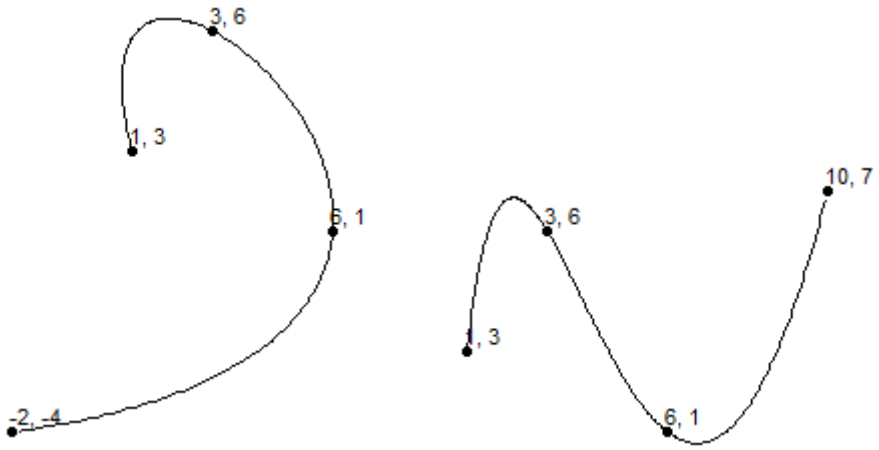
$x = r \cos \theta + cx$
$y = r \sin \theta + cy$

2. Cubic curves

A cubic curve is a parametric cubic of the form $\mathbf{P}(t) = \mathbf{B}_3 t^3 + \mathbf{B}_2 t^2 + \mathbf{B}_1 t + \mathbf{B}_0$, where $\mathbf{P} = (x(t), y(t))$ is a point on the curve at parameter value t, $\mathbf{B}_3$, $\mathbf{B}_2$, $\mathbf{B}_1$ and $\mathbf{B}_0$ are the unknown vector coefficients. We need to solve for these four unknows to obtain the curve. Four vector unknowns require four vector conditions to solve for them. For the formulae given below, t lies in the range $0 \le t \le 3$, and the four vector conditions are four input points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ at t = 0, 1, 2 and 3 respectively. The formulae for the **B**'s in terms of the inputs are (derivation omitted):

$\mathbf{B}_0 = \mathbf{P}_0$
$\mathbf{B}_1 = (2\mathbf{P}_3 - 9\mathbf{P}_2 + 18\mathbf{P}_1 - 11\mathbf{P}_0) / 6$
$\mathbf{B}_2 = -(\mathbf{P}_3 - 4\mathbf{P}_2 + 5\mathbf{P}_1 - 2\mathbf{P}_0) / 2$
$\mathbf{B}_3 = (\mathbf{P}_3 - 3\mathbf{P}_2 + 3\mathbf{P}_1 - \mathbf{P}_0) / 6$

Below are two examples of curves produced using the formulae. The dots are the input points.



Appendix 2: Relevant mathematics and algorithms

1. Translation and rotation of a vector

   A vertex is a position vector (x, y). To translate a polygon, we translate all its vertices equally. If the translation vector is (δx, δy), then the point after translation is ($x+\delta x$, $y+\delta y$).

   The rotation of a vertex (x, y) about a point (p, q) by an angle θ radians is given by the matrix operation:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x-p \\ y-q \end{bmatrix} + \begin{bmatrix} p \\ q \end{bmatrix}$$

2. Perimeter length of a polygon

   The perimeter length of a polygon is the sum of the lengths of its individual edges. For a straight edge between vertices with coordinates ($x1$, $y1$) and ($x2$, $y2$), the length is $\sqrt{((x_2-x_1)^2+(y_2-y_1)^2)}$.

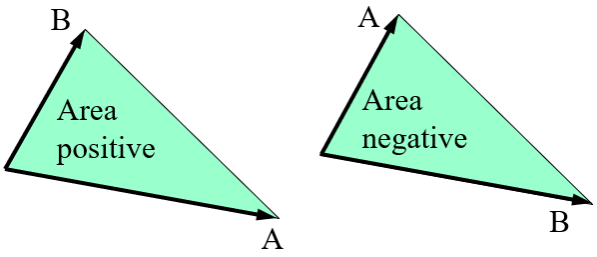
   For a circular arc, the arc length is $2r\theta$, where r is the radius of the arc and θ is the angle in radians subtended by the arc at its centre.

   For a cubic curve, the formula for the arc length is not simple. An easy method is to obtain a good approximation by approximating the curve with a suitably large number of connected short segments. Note that the curve equation $\mathbf{P}(t) = \mathbf{B_3}t_3 + \mathbf{B_2}t_2 + \mathbf{B_1}t + \mathbf{B_0}$ uses the parameter t, with t lying between 0 and 3. This means that a value of t between 0 and 3 gives a point on the curve. You can generate 100 points, say, on the curve by stepping from 0 to 3 for t at 3/100 interval. Summing the length of each segment between two successive points gives you an approximation to the curve length. You can improve the approximation by increasing the number of segments, but that increases the amount of computation.

3. The area of a polygon

   There are different methods for the area of a polygon. The method introduced here requires the signed area of a triangle. Referring to the diagram, **A** and **B** are two vectors that form two sides of a triangle. The area of the triangle can be determined using vector algebra thus:

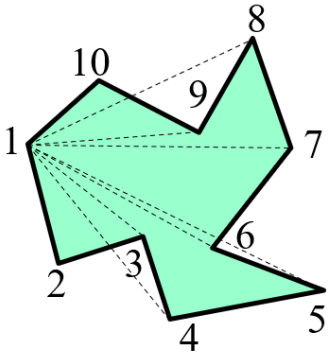
   Area = (**A** x **B**) · **N**



   where **N** is the normal to the plane of the triangle. The area is positive or negative depending on the positions of the two vectors. In 3D, we can let

   A = (A_x, A_y, 0)
   B = (B_x, B_y, 0)
   N = (0, 0, 1)

   Then Area = (0, 0, $A_xB_y - A_yB_x$) · (0, 0, 1) = $A_xB_y - A_yB_x$.



   Note that the vector going from Point **P** = (P_x, P_y) to Point **Q** = (Q_x, Q_y) is **Q** – **P** = (Q_x–P_x, Q_y–P_y).

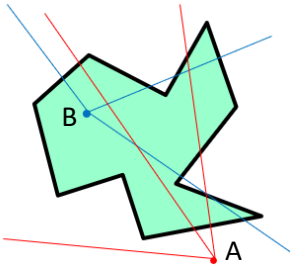
   For a polygon such as the one shown in the diagram, we can anchor at Point 1, and use it to form triangles with every subsequent pair of consecutive points, Δ123, Δ134, Δ145 … etc. in sequential order, until the very last point. These triangles may have overlapping areas or areas outside the polygon. Each has a positive or negative area

depending on the orientation of the triangle (clockwise or anticlockwise). Summing the areas of these triangles gives the area of the polygon as the negative areas cancel out the double-counted areas due to overlap and the extra areas not in the polygon. The algorithm for finding the area of the polygon is

```
TotalArea = 0
for i from 2 to n-1
  TotalArea = TotalArea + area of triangle formed by vertices 1, i and i+1.
```

There remains the question of dealing with curves. This is straightforward if we maintain the same approximation earlier using short segments for finding the curve length. This applies for both circular arcs and cubic curves. We simply treat each segment as an edge of the polygon and apply the same triangle area formula and algorithm as above.

4. Algorithm for determining if a point lies in a polygon



We can check whether a point is inside, outside or on the boundary of a polygon by counting the number of intersections a line from the point makes with the boundary of the polygon. In the diagram, Point A is outside the polygon and the three lines from it intersect the boundary of the polygon 0, 2 and 4 times, all even. Point B is inside the polygon and the three lines from it intersect the boundary of the polygon 1, 3 and 3 times, all odd. Generally, lines from points inside would intersect the polygon boundary an odd number of times and lines from points outside would intersect the boundary an even number of times, including zero. if a point is on the boundary of the polygon, then the point must be equal to a point of intersection.

Hence the method for determining the position of a point with respect to a polygon is to cast a line in a chosen direction (any direction, theoretically to infinity to ensure that it travels through the polygon) from the point and check if the number of intersections this line makes with the boundary of the polygon is odd or even.

This requires the mathematics for the intersection of two lines, and then be able to determine if the point of intersection lies within the polygon edge; if it does there is an intersection, otherwise there isn't.

To compute for the intersection, we use the parametric equation of a straight line: $\mathbf{P} = \mathbf{P0} + \mathbf{V}t$, where $\mathbf{P} = (x, y)$ is a point on the line, $\mathbf{P0}$ is a known point on the line and $\mathbf{V}$ is the direction vector of the line and t is the parameter. If we have the two end points $\mathbf{E1}$ and $\mathbf{E2}$ of the line, then we can set $\mathbf{P0} = \mathbf{E1}$, and $\mathbf{V} = \mathbf{E2} - \mathbf{E1}$.

Since we have two lines, then there are two equations:

$\mathbf{P_1} = \mathbf{P0_1} + \mathbf{V_1}t_1$ and $\mathbf{P_2} = \mathbf{P0_2} + \mathbf{V_2}t_2$

At the point of intersection, $\mathbf{P_1} = \mathbf{P_2}$. Therefore

$\mathbf{P0_1} + \mathbf{V_1}t_1 = \mathbf{P0_2} + \mathbf{V_2}t_2$

This is in fact two equations, one each for the x and y components. Therefore, we have two equations and two unknowns, t_1 and t_2. The equations can be easily solved for the values of t_1 and t_2. Assuming that the equation for $\mathbf{P_1}$ is for the edge of the polygon, then, to tell if the point of intersection lies within the edge, all we need to do is to check if $0 \le t_1 \le 1$ (prove for yourself why this is so). If it is, the point of intersection lies in the edge, otherwise it does not. There is complication when the point of intersection is at an end of a polygon edge because you would get two points of intersection for the two edges that meet at that end point. Instead of trying to resolve this complication, simply cast a different line since we can freely choose the direction.

Appendix 3: Some guidance on the use of Turtle

Turtle is a graphics library provided with Python. It contains the functions required for drawing, such as the creation of lines, shapes and posting of text, and controlling their attributes such as location, colour, and more. You should study the different Turtle functions required for your display. There are many Turtle functions, you only need to learn the ones you require. It is therefore necessary that you work out what you need beforehand.

In this project, you will be doing some interactive graphics, basically using mouse clicks on the screen to create polygon vertices or select menu options. These two videos provide good guides for such interactive events:
https://www.techwithtim.net/tutorials/python-module-walk-throughs/turtle-module/key-presses-events/
https://www.techwithtim.net/tutorials/python-module-walk-throughs/turtle-module/drawing-with-mouse/

You will need to draw curves in the project. Turtle provides only basic functions for drawing straight lines. You can break a curve down to many connected short straight-line segments. By drawing these segments, you get what looks like your curve. The quality of the curve depends on how many segments you use to approximate it. You can experiment to find out how many segments make a good approximation. Note that if you have more segments, then there is more computation, and it takes more time to draw. This is the same sort of approximation mentioned earlier for calculating curve length and polygon area.

You need other graphical functions such as those for posting text and controlling colours. These are again available in Turtle. You can find the definition of these functions in the Turtle documentation.

By default, Turtle draws slowly to allow us to see how the drawing is done. However, for this project, you should draw quickly. So make sure that you set the drawing speed to the fastest possible, again doable by calling the appropriate Turtle function with the appropriate parameter value. You should hide the turtle too.