Computer Graphics and Animation

20192

Programming Assignment 7

Adjie Ghusa Mahendra | 001201800130
Gilang Martadinata | 001201800022
Kenny Susanto | 001201800047

Table of Contents

I. Introduction

This program allows the user to add, edit, and delete polygons and polylines. The polygons and polylines are implemented using linked lists. The language used to implement this program is Python.

II. Basic Theory

Polygon

A polygon is a closed figure where the edges are all line segments. Each edge must intersects exactly two other edges at their endpoints. The edges must not have a straight line in common and have a common endpoint. A polygon is usually named after the number of edges it has, a polygon with n-edges is called a n-gon. E.g. The building which houses the United States Department of Defense is called pentagon since it has 5 edges.

A polygon has the following properties:

- Edge
  - An edge is a line segment that makes up the polygon
- Vertex
  - A vertex is a point where two edges meet.
- Diagonal
  - A diagonal is a line connecting two vertices that are not adjacent.
- Interior angle
  - An interior angle is an angle formed by two adjacent edges inside the polygon.
- Exterior angle
  - An exterior angle is an angle formed by two adjacent edges outside the polygon.
- Area
  - An area is the number of square units it takes to completely fill a regular polygon.
- Perimeter
  - A perimeter is the total distance around the outside of a polygon.

Polygon also has several categories:

- Convex polygon

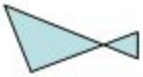
  A convex polygon is a polygon whose interior angles are less than 180 degrees. Any two points in a polygon can be connected by an inside line. The picture below shows an example of a convex polygon.

- Concave polygon

  A concave polygon is a polygon whose one or more interior angles are greater than 180 degrees. Not true of all point pairs inside the polygon. The picture below shows an example of a concave polygon.

- Self-intersecting polygon

  A self-intersecting polygon is a polygon where at least one edge crosses over another edge, creating multiple smaller polygons. The picture below shows an example of a self-intersecting polygon.



Polyline

A polyline is a list of points, where line segments are drawn between consecutive points. A polyline can be created by specifying the endpoints of each segment. In practical implementation, a polyline can be treated as a single object or divided into a list of segments.

Perimeter and Area of Polygon

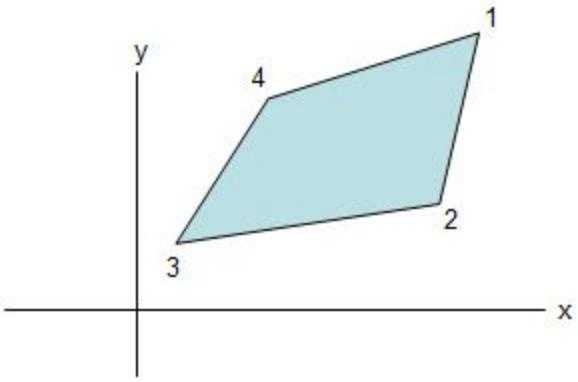
Perimeter of Polygon

A perimeter of a polygon can be obtained by adding together the length of each edge. For example, a quadrilateral whose edges are 12, 6, 9 and 8, the perimeter is the sum of these, or:

$$\text{Perimeter} = 12 + 6 + 9 + 8 = 35$$

Area of polygon

To calculate the area of a polygon, the vertices of a polygon must be defined as a set of points and the direction of the polygon must be defined as either clockwise or counterclockwise. The picture below is the illustration of said polygon.



The area of a polygon is then given by the formula:

$$\left| \frac{(x_1 y_2 - y_1 x_2) + (x_2 y_3 - y_2 x_3) \ldots\ldots + (x_n y_1 - y_n x_1)}{2} \right|$$

Where x_n is the x coordinate of vertex n and y_n is the y coordinate of vertex n. The last term of the formula means the calculation wraps around back to the beginning at the first vertex. The result of this area formula is absolute, meaning it will always be positive.

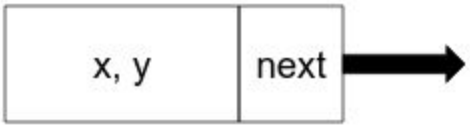
Linked List Representation of Polygon/Polyline

A polygon or a polyline can be represented as a list of points. The list can be implemented using linked lists. Technically a point contains x and y variable, a polygon or a polyline contains a list of points, a list of polygons or polylines contains a list of polygons or polylines. Below is the implementation of each object using linked lists:

- Point
  A point contains x and y variable. X variable is used to store x coordinate of the point and Y variable is used to store y coordinate of the point on the screen. Below is the example of a point data structure:

  ```
  Type Point
  {
          int x;
          int y;
          Point next;
  }
  ```

  In the data structure above, variable x and y are represented as integers. There is also the next variable represented as a pointer pointing to the next point (if necessary). Below is the illustration of a point:

- Polygon or polyline
  A polygon or polyline contains a list of points. Each point represents a vertex in a polygon or polyline. Below is the example of a polygon data structure:

  ```
  Type Polygon
  {
          Point head;
          Polygon next;
  }
  ```

In the data structure above, the variable head is represented as a point this variable contains the starting point of a polygon or polyline. And there is also the next variable represented as a pointer pointing to the next polygon or polyline (if necessary). Below is the illustration of a polygon or polyline:

- List of polygons or polylines
  A list of polygons or polylines contains a list of polygons or polylines. Each polygon or polyline represents a single object on the screen. Below is the example of a list of polygons or polylines data structure:

```
Type ListPoly
{
        Polygon head;
}
```

In the data structure above, the variable head is represented as a polygon, this variable contains the starting or the first polygon of the list. Below is the illustration of a list of polygons or polylines:

Vertex Processing of a Polygon

Vertex Processing processes every vertex of a polygon. In this process, a vertex is processed individually. The pseudocode of this process:

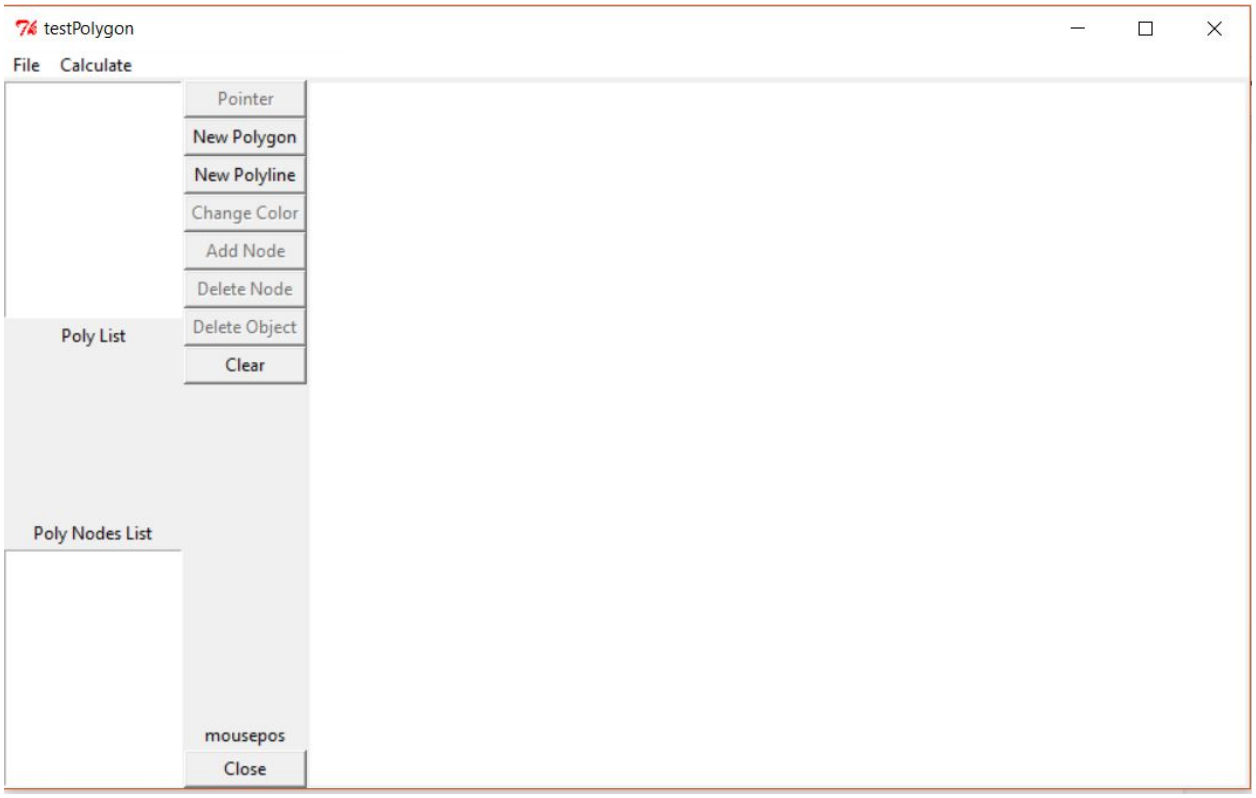
```
For each P
        Process(P)
Next P
```

Edge Processing of a polygon

Edge Processing processes each edge between the vertex of the polygon. An edge can also be called a side of a polygon. In this process, each edge or side is processed individually as the main focus of this process from the first edge or side to the last. For example, in finding the length of a line (edge) of a polygon, it is possible to directly calculate the solution from the properties of the line (edge) itself. The pseudocode of this process:

```
For each Edge
        Process(Edge)
Next Edge
```

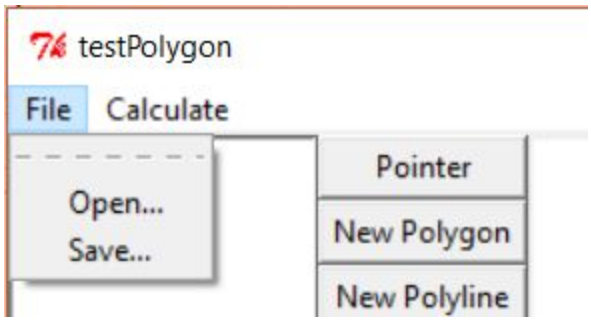
III. Implementation



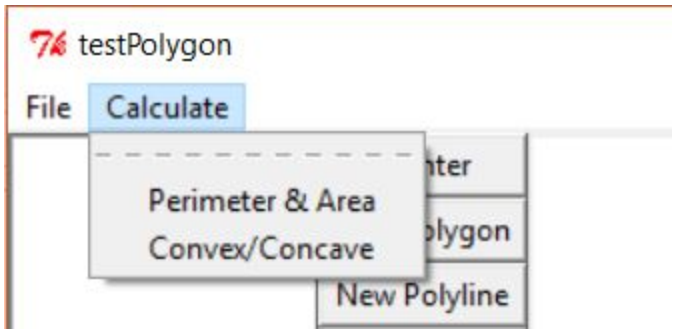
The main interface of the application contains blank canvas to draw polygons and polylines, several buttons to manipulate the drawing, a list box which contains list of polygon and its nodes, close button, mouse position tracker and file and calculate menu buttons.

- **The Canvas** is a rectangular area intended for drawing polygons.
- There are several buttons to manipulate the drawing:
    1. **Pointer**: Used for deselecting polygon
    2. **New Polygon**: Used for drawing a new polygon
    3. **New Polyline**: Used for drawing a new polyline
    4. **Change color**: Used for changing the color of the drawing
    5. **Add Node**: Used for adding a new node after a selected node
    6. **Delete Node**: Used for deleting a selected node
    7. **Delete Object**: Used for deleting a selected object
    8. **Clear**: Used for clearing the canvas and all stored objects
- There are two list boxes:
    1. **Poly List**: Showing the list of polygons and can be used for selecting a drawn polygon which will show the node's pointer of its nodes
    2. **Poly Nodes List**: Showing the list of polygon's nodes
- **The close** button is the button for closing the application window.

- **Mouse Position** tracker is a text label for tracking the coordinates of the mouse pointer which can assist to find the coordinates of the nodes.



- **Open** is used to find and open .txt files that contains previously saved file
- **Save** is used to save the current state of drawing into .txt files



- **Perimeter & Area** is used to calculate the perimeter & area of all objects on canvas.
- **Convex/Concave** is used to determine all objects on canvas whether they are convex or concave.

IV. Design

Data structures implemented in this program are Polylist, Polygon, Polyline, Node, and Point. Below is the explanation for each data structure implemented in this program:

- **Point**, is used to represent a point/dot with x and y value

```
Point(0, 0)
```

- **Node**, is used to represent a node with a Point containing x and y

```
Node(Point(100, 100))
```

- **Polyline**, is used to represent a polyline with a collection of Nodes respectively containing x and y using linked list structure

```
newPoly = Polyline()
```

- **Polygon**, is used to represent a polygon with a collection of Nodes respectively containing x and y using linked list structure

```
newpoly = Polygon()
```

- **Polylist**, is used to represent a list of polygons and polylines with a collection of polygons and polylines using linked list structure

```
self.polylist = Polylist()
```

Global variables **used** in the program are listed below (unused variable will not be explained below):

- **o**, is used to initiate default point on drawing polygon or polyline
- **polylist**, is the declaration of Polylist (collection of polygons and polylines)
- **arrCounter**, is used to count current total objects stored
- **selectedP**, is used for passing int of current selected poly from poly list box
- **selectedN**, is used for passing int of current selected node for editing location of each node
- **selectedNobj**, is used for passing Node of current selected node for editing location of each node
- **rectslist**, is used to list nodes representation as interactive rectangles on the canvas
- **lineslist**, is used to list lines of each polygon or polyline
- **rect**, is used for passing Rectangle object of current selected rectangle for editing location of each node
- **tmplines**, is used to list sub collection (a collection of lines of a polygon or polyline) to then store in lineslist
- **tmpcolor**, is used to store list of colors of a polygon or polyline
- **shape**, is used to store the shape of a polygon or polyline from saved file

```
self.o = Point(0, 0)
self.polylist = Polylist()
# self.polyArr = []
self.arrCounter = -1
self.current = None
self.selectedP = 0  # passing selected polygon (int)
self.selectedN = None  # passing selected node (int)
self.selectedNobj = None  # passing selected node (object)
self.rectslist = []  # list of polygon rectangles formed on nodes
self.lineslist = []
self.rect = None
self.tmplines = []
self.tmpcolor = []
self.shape = "polygon"
```

Implemented bonuses are listed below:

- **Calculate the perimeter and area of a polygon**

A polygon vertex is represented as a point containing x and y value for the coordinate of the point on canvas. Each pair of coordinates is processed in an area summation algorithm that looks like this:

$$A = A + \frac{(x_1 \times y_0 - y_1 \times x_0)}{2}$$

$$A = A + \frac{(x_2 \times y_1 - y_2 \times x_1)}{2}$$

$$\vdots$$

The mathematical expression for the area of a polygon would be:

$$Polygon\ Area\ A = \frac{1}{2} \sum_{i=m}^{n-1} x_{i+1} \times y_i - y_{i-1} \times x_i$$

Formally, the perimeter length is the sum of the vector magnitudes of each edge of a polygon. The mathematical expression is written as:

$$Perimeter\ Length\ P = \sum_{i=m}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

- **Determining a polygon is convex or concave**

This program implemented an algorithm to determine whether a polygon is convex or not based on the following four statements:

1. The sign of the vector cross product indicates whether the second vector is clockwise (positive) or counterclockwise (negative) with respect to the first vector, in a standard right-handed coordinate system.
2. If the vector cross product is zero, the two vectors are collinear (collinear vector means vector that is parallel to one line or lying on one line).
3. If the vector cross product between consecutive pairs of edge vectors in a polygon has differing signs (ignoring zeros), the polygon must be concave.
4. If the signs of the x and y components of the edge vectors are examined, (again ignoring zeros), consecutively along the polygon, as a circular list, there must be exactly two sign changes, or the polygon is concave.

Statements 1, 2 and 3 are known from basic vector algebra. Statements 3 and 4 combined, is equivalent to calculating the angle between each pair of consecutive edges in the polygon, and verifying that both are all in the same orientation (counterclockwise or clockwise), and that the sum of the angles is 360° (so that self-intersecting polygons can be correctly detected), except that this algorithm only consider four separate directions (the four quadrants in a standard coordinate system). Below is the pseudocode of this algorithm:

```
Function isconvex(vertexlist):
   If (# of vertices in 'vertexlist' < 3), Then
      Return FALSE
   End If

   Let  wSign = 0
# First nonzero orientation (positive or negative)

   Let  xSign = 0
   Let  xFirstSign = 0
# Sign of first nonzero edge vector x
   Let  xFlips = 0
 # Number of sign changes in x

   Let  ySign = 0
   Let  yFirstSign = 0
# Sign of first nonzero edge vector y
   Let  yFlips = 0
 # Number of sign changes in y

   Let  curr = vertexlist[N-1]
# Second-to-last vertex
   Let  next = vertexlist[N]
# Last vertex

   For  v  in  vertexlist:
# Each vertex, in order
      Let  prev = curr        # Previous vertex
      Let  curr = next        # Current vertex
      Let  next = v          # Next vertex

      # Previous edge vector ("before"):
      Let  bx = curr.x - prev.x
      Let  by = curr.y - prev.y

      # Next edge vector ("after"):
      Let  ax = next.x - curr.x
      Let  ay = next.y - curr.y

# Calculate sign flips using the next edge vector
                ("after"),
      # recording the first sign.
      If ax > 0, Then
         If xSign == 0, Then
            xFirstSign = +1
         Else If xSign < 0, Then
            xFlips = xFlips + 1
         End If
         xSign = +1
      Else If ax < 0, Then
         If xSign == 0, Then
            xFirstSign = -1
         Else If xSign > 0, Then
            xFlips = xFlips + 1
         End If
         xSign = -1
      End If

      If xFlips > 2, Then
         Return FALSE
      End If
```

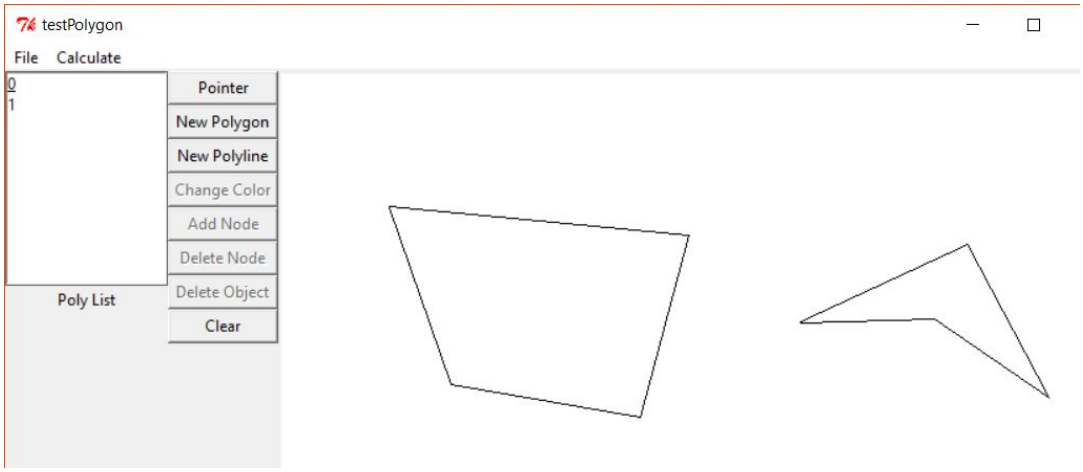
```
      If ay > 0, Then
         If ySign == 0, Then
            yFirstSign = +1
         Else If ySign < 0, Then
            yFlips = yFlips + 1
         End If
         ySign = +1
      Else If ay < 0, Then
         If ySign == 0, Then
            yFirstSign = -1
         Else If ySign > 0, Then
            yFlips = yFlips + 1
         End If
         ySign = -1
      End If

      If yFlips > 2, Then
         Return FALSE
      End If

# Find out the orientation of this pair of edges,
# and ensure it does not differ from previous ones.
      w = bx*ay - ax*by
      If (wSign == 0) and (w != 0), Then
         wSign = w
      Else If (wSign > 0) and (w < 0), Then
         Return FALSE
      Else If (wSign < 0) and (w > 0), Then
         Return FALSE
      End If
   End For

   # Final/wraparound sign flips:
    If (xSign != 0) and (xFirstSign != 0) and (xSign !=
xFirstSign), Then
      xFlips = xFlips + 1
   End If
    If (ySign != 0) and (yFirstSign != 0) and (ySign !=
yFirstSign), Then
      yFlips = yFlips + 1
   End If

   # Concave polygons have two sign flips along each
axis.
   If (xFlips != 2) or (yFlips != 2), Then
      Return FALSE
   End If

   # This is a convex polygon.
   Return TRUE
End Function
```

The implementation of this algorithm in the program is slightly different to match present variables and data structures. Please refer to the source code of this program.

V. Evaluation

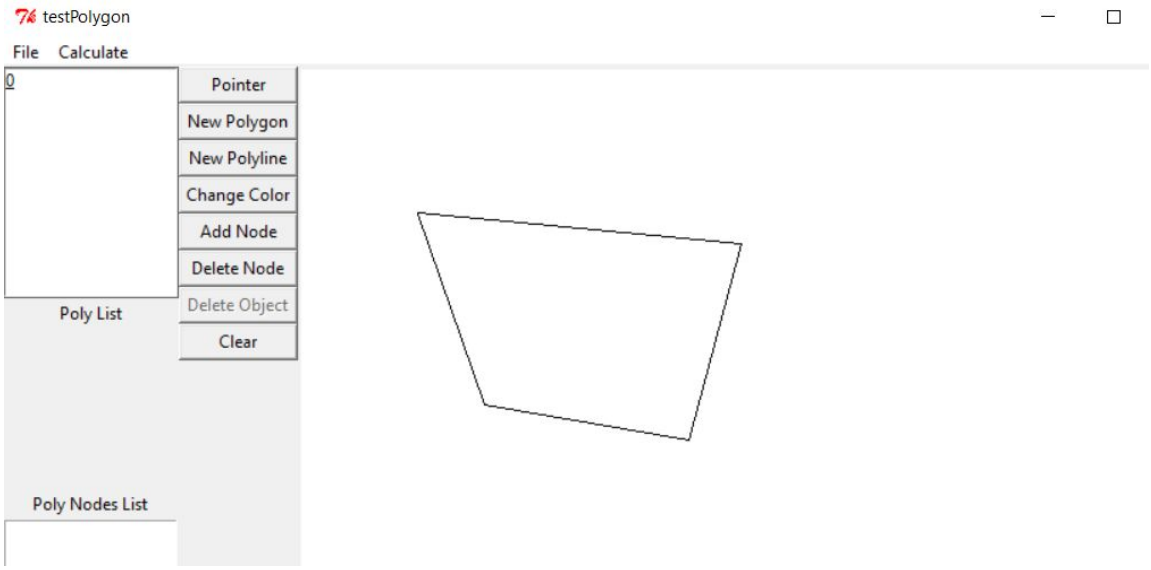
- **Add and Delete a polygon**
  - Adding Polygon

    This process turns out to be successfully done. It successfully added one polygon or more.



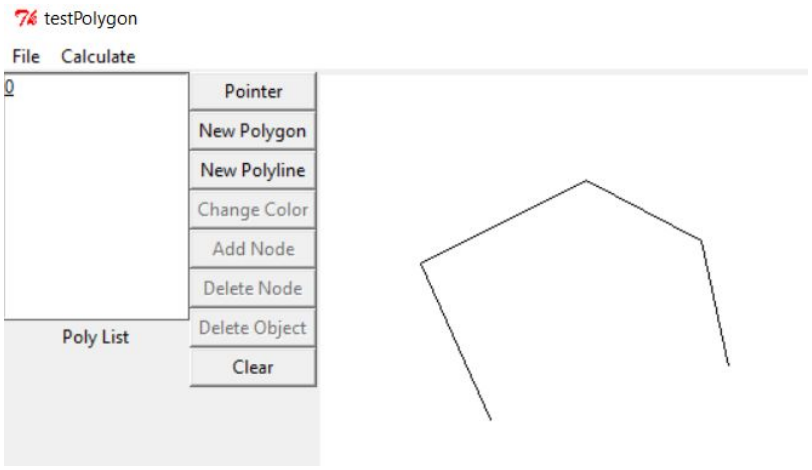
  - Deleting Polygon

    This process is also successfully done as shown below. It deleted one of the drawn polygons.



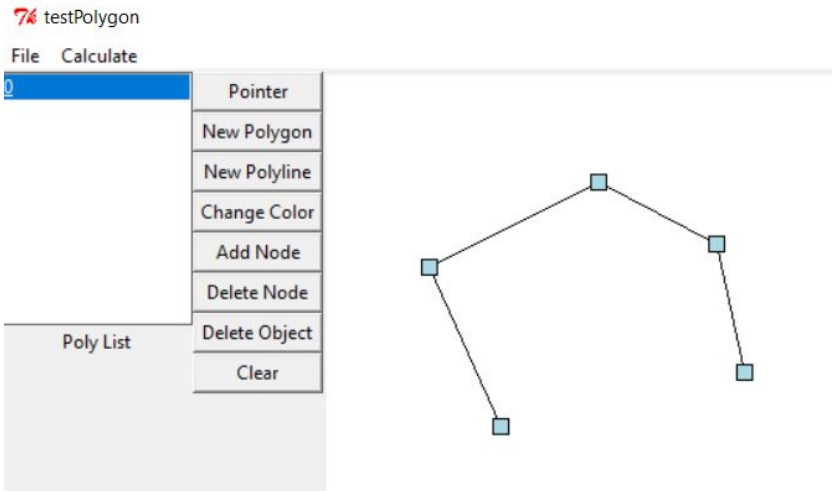
- **Add and Delete a polyline**
    - Adding Polyline

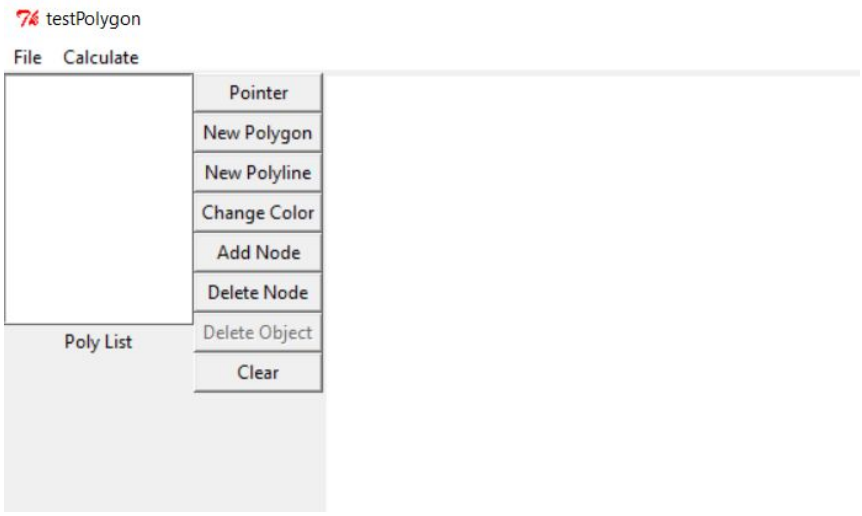
      This process is successfully done as shown below.



    - Deleting Polyline

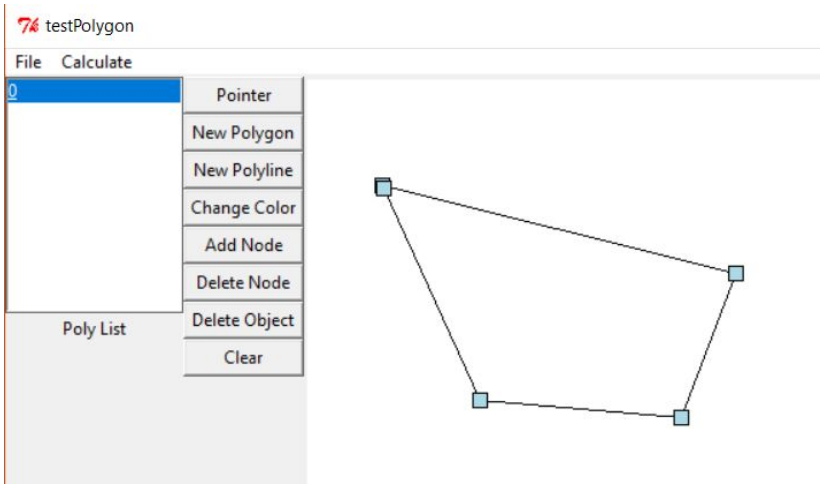
      The selected polyline is deleted as shown below.



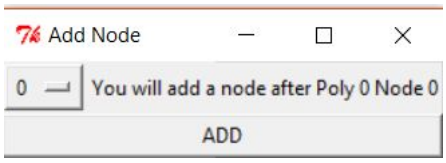


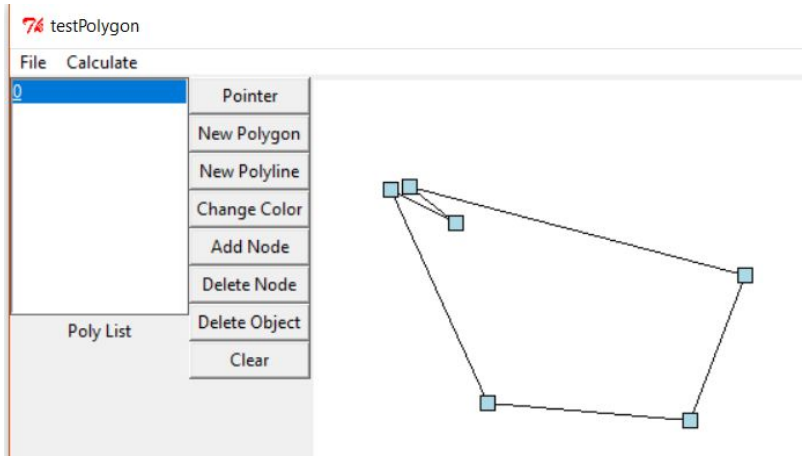
- **Add Vertex to an existing Polygon/Polyline**

  As can be seen below, this test case is executed successfully. Including the subtest cases.

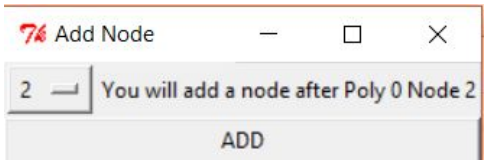


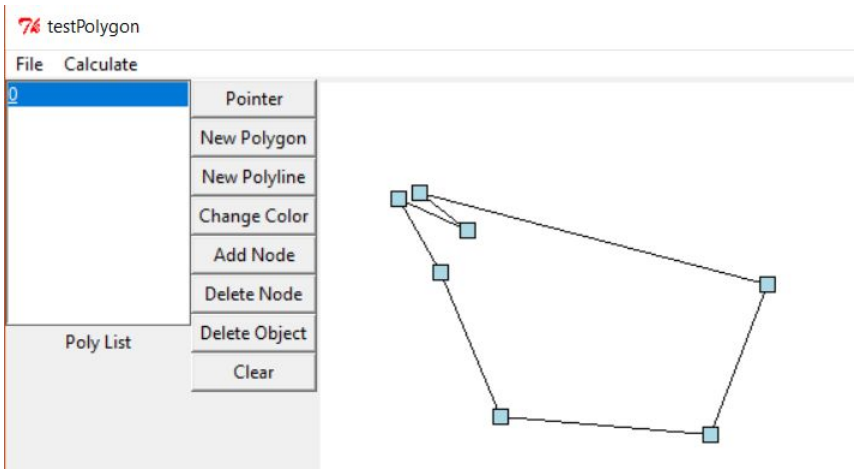
  - Adding as First vertex



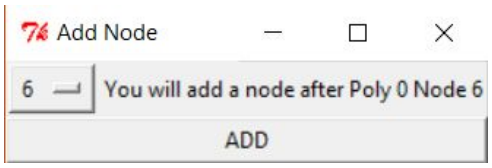


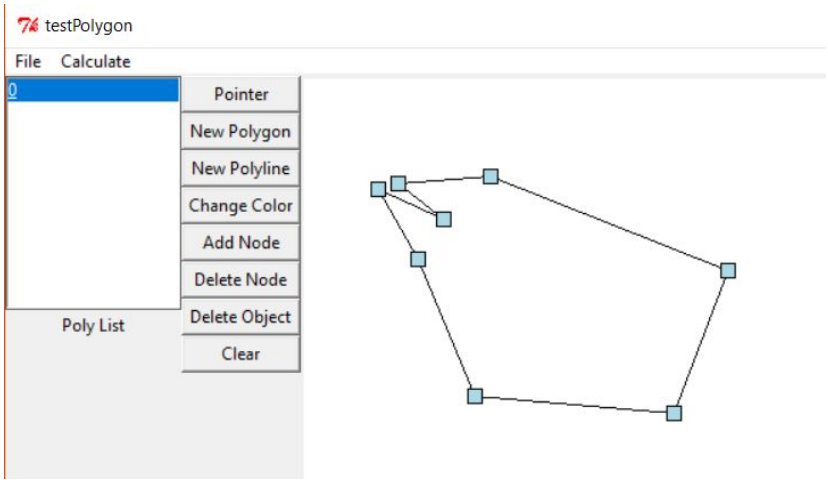
- Adding as Middle vertex





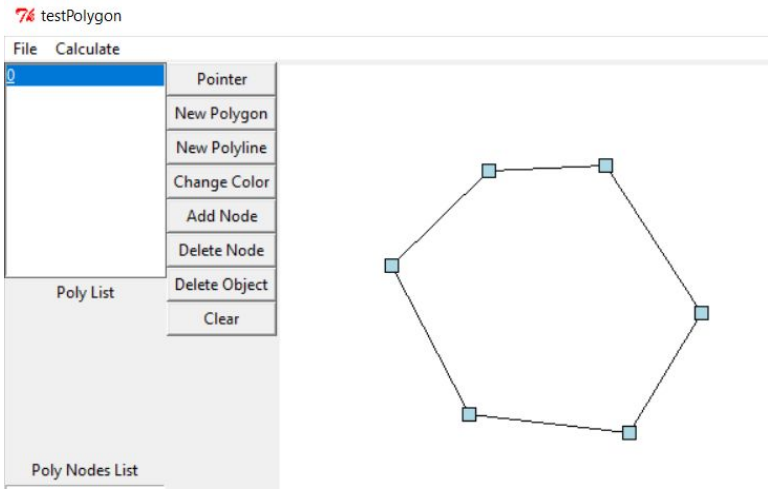
- Adding as Last Vertex

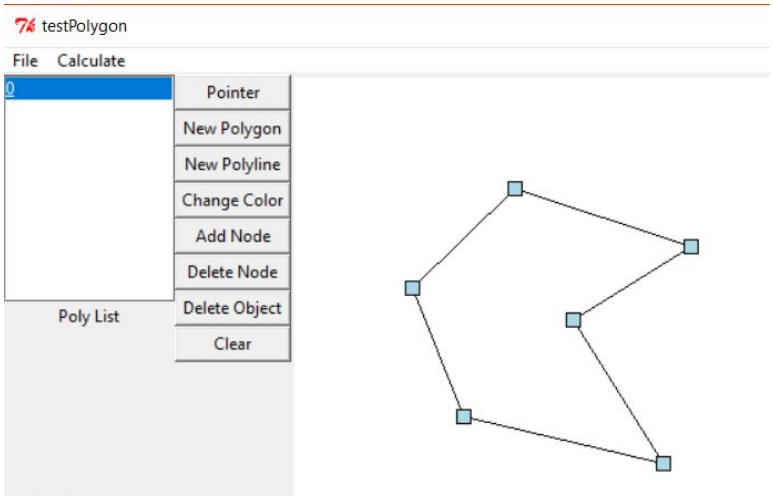




- **Change the Vertex location on an existing Polygon/Polyline**

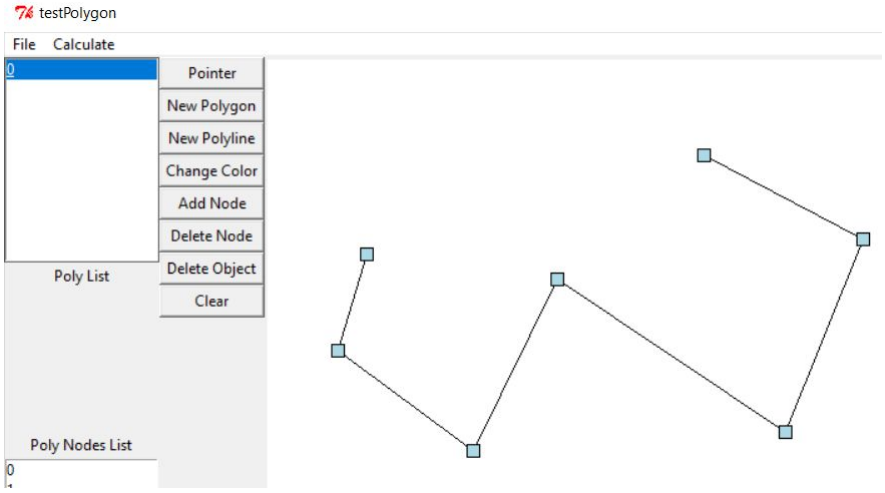
  As can be seen below, this test case is executed successfully. Attached screenshot of before and after comparison of the polygon.



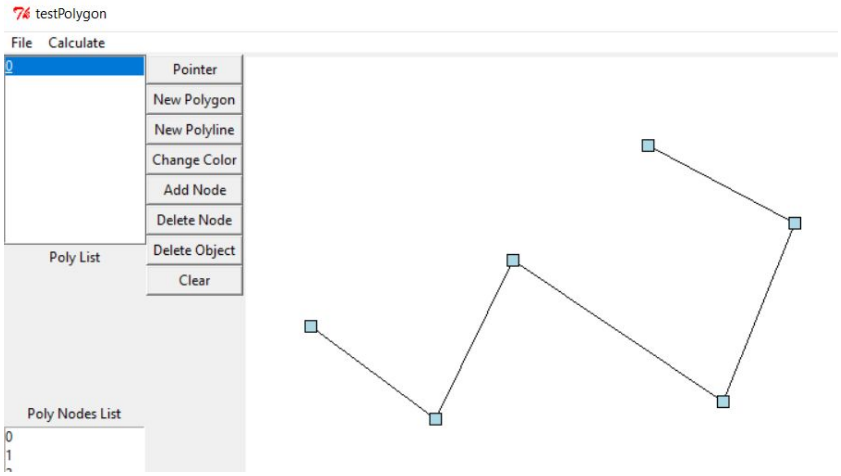


- **Delete a vertex from a Polygon/Polyline**

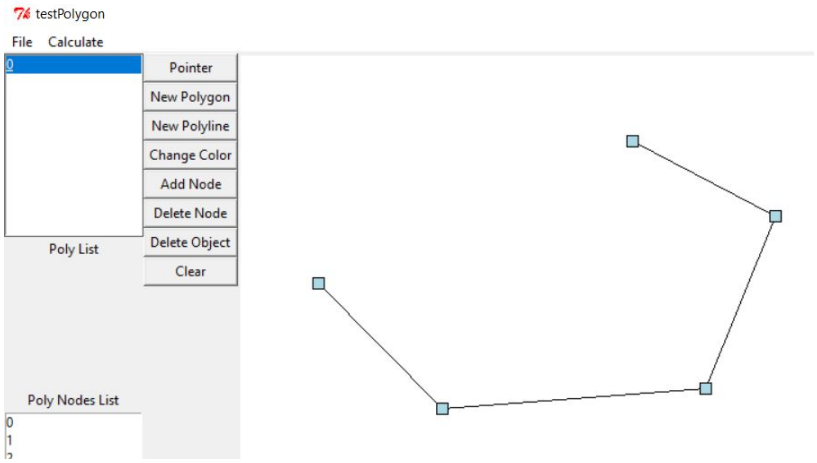
  As can be seen below, this test case is executed successfully.



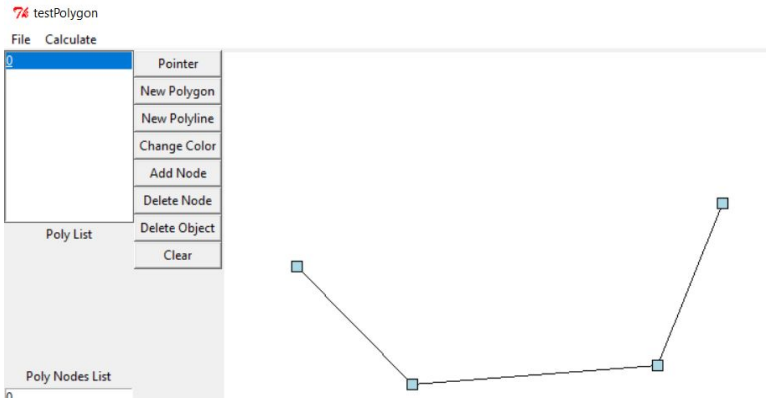
- Delete the First vertex



- Delete the Middle vertex

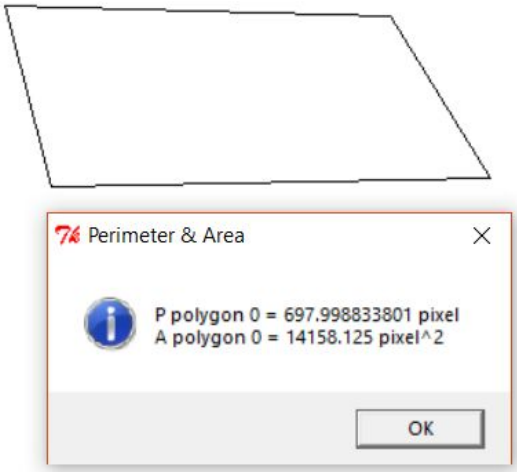


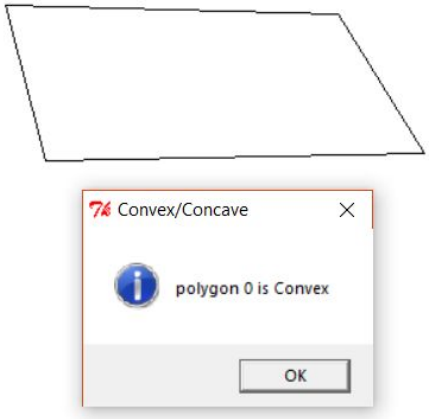
- Delete the Last vertex



- **Bonus Test Case**

This program can determine whether the polygon is convex or concave and calculate the perimeter and area of the polygon without a problem.





VI. Work Log

Work log table

Date	Activity / Progress	Personnel Involved
31 January 2020 - 2 February 2020	Developing prototype	Adjie, Gilang, Kenny
2 February 2020 - 4 February 2020	Processing weekly report	Adjie, Gilang, Kenny
7 February 2020 - 9 February 2020	Finishing program and report	Adjie, Gilang, Kenny
14 February 2020 - 16 February 2020	Fixing program defects and finishing report	Adjie, Gilang, Kenny

Summary of the requirements given

Here is the summary of the requirements given to finish this programming assignment with its explanation whether the requirement is fully implemented, partially, or not at all. **Draw a polygon on the screen** is fully implemented. **Draw a polyline on the screen** is fully implemented. **Delete a polygon/polyline** is fully implemented. **Add a vertex to a polygon/polyline. The user can select which existing vertices will be adjacent to the added vertex** is fully implemented. **Edit the location of a vertex on a polygon/polyline** is fully implemented. **Delete a vertex from a polygon/polyline. The user can select which vertex to delete** is fully implemented. **Change the color of a polygon** is fully implemented. **Clear the screen, this deletes all polygons and polylines** is fully implemented. **Save/load data of polygons and polylines to/from a file** is fully implemented. **Determine whether a polygon is convex or not** is fully implemented. **Calculate the perimeter and area of a polygon** is fully implemented. In short, all requirements are all fully implemented in this program.

VII. Conclusion and Remarks

This program is finished with all the requirements fulfilled. The bonus features also have been implemented in this program and executed properly as intended. This program also proved that linked lists can be used to represent polygon or polylist.

Hard work, curiosity, and diligence are needed to finish this assignment. Overall, the assignment is challenging enough and it is convenient to be able to pick a programming language deliberately for this program. This programming assignment has been insightful and beneficial for the experience of the members on this team.