

Computer Graphics and Animation

Programming Assignment 9

Drawing Polygon - Linked List



Created by:

Natasya Oktavioni Susanto (001201900024)

Renaldo Fareza Tambunan (001201900084)

Samuel (001201900005)

Table of Contents

Table of Contents	2
I. Introduction	3
II. Basic Theory	4
III. Implementation	11
IV. Design	13
V. Evaluation	21
VI. Work Log	32
VII. Conclusion and Remarks	34
References	35

I. Introduction

This is a program that allows users to draw polygons and polylines. There are some other features such as editing, adding, and deleting vertices, also coloring the polygons, as well as clearing the screen/canvas. We are using VB .NET as our programming language to create this application. The canvas is built using a built-in system, that is bitmap, that means every object created in this program is defined by pixels. The screen coordinates origin is started on the upper-left corner of the window. We decided to use this programming language because it is much more efficient and appears more user-friendly.

This program is implemented using a linked list as the data structure. The implementation using a linked list means that we are storing items in the form of nodes into the list. The advantages of using linked lists over arrays here are the objects are easier to allocate and be accessed with the pointer, also there is no need to assign or determine the size of the medium.

II. Basic Theory

POLYGON

Polygon is a two-dimensional shape made of straight line segments connected to each other end to end. These line segments are called edges. Triangles, quadrilaterals, pentagons, and hexagons are all examples of polygons. The name is made up by how many sides the shapes have. For instance, a triangle has three sides connecting to each other endpoints, and a quadrilateral has four sides. A polygon is a closed figure, so if the sides are not fully connected, then it can't be called a polygon, hence it is a polyline.

There are several types of the polygon, regular polygon means every edge is creating the same angle and having the same size, otherwise, it is an irregular polygon. There is also a convex polygon that does not have an internal angle greater than 180° . The contrast of a convex polygon called a concave polygon. There is also a simple polygon that every edge will not cross itself, otherwise, it is a complex polygon.

A polygon has the following basic properties:

- a. Edge: An edge is a line segment on the boundary, sometimes referred to as side.
- b. Vertex: A vertex is a point at which two edges of a polygon meet.
- c. Exterior Angle: The exterior angle is the angle formed between any side of a polygon and a line drawn from the opposite side. All the exterior angles of a polygon add up to 360° .
- d. Interior Angle: The interior angle is measured the same as the exterior angle and all the angles are added up to 180° .

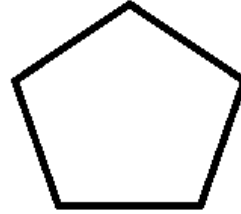
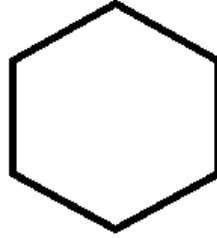
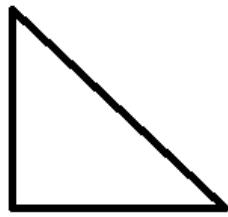
POLYLINE

A polyline is a connected sequence of line segments created as a single object. By specifying the endpoints of each segment, a polyline can be drawn. Unlike polygons, polylines are not closed.

CALCULATE THE PERIMETER AND AREA OF A POLYGON

Perimeter of Polygons

Perimeter is the distance around the exterior of any closed geometric shape. To calculate the perimeter of a polygon, we need to add up the lengths of all sides. That means for a triangle, the number of sides that will be added up is three, four numbers for a quadrilateral, and so on. Every line in a polygon has two endpoints, to get the length of a line, we can perform a square root of $dx^2 + dy^2$. The total length of the perimeter can be obtained by performing this mathematical expression below.



P: Perimeter

$$P = \sum_{i=0}^{n-1} \sqrt{(X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2}$$

Area of Polygons

The vertices of an irregular polygon can be presented using Cartesian coordinates of x, y. For example a list of coordinates of the vertices of our polygon as follows:

x_1, y_1

x_2, y_2

x_3, y_3

...

x_{n-1}, y_{n-1}

x_n, y_n

x_1, y_1

As shown above, the first coordinate which is x_1 and y_1 appear again at the end of the list, because it is a closed geometric shape, the last coordinate needs to connect with the first coordinate. Next, we can express the area of a polygon as $\frac{1}{2}$ times the sum of the determinants of the 2x2 matrices:

$$\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$$

$$\begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix}$$

...

...	$\begin{array}{ c c c } \hline \lceil & x_{n-1} & y_{n-1} & \rceil \\ \hline \lfloor & x_n & y_n & \rfloor \end{array}$	$\begin{array}{ c c c } \hline \lceil & x_n & y_n & \rceil \\ \hline \lfloor & x_1 & y_1 & \rfloor \end{array}$
-----	-------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------

The matrices above will refer to determinants with the value: $x_1 \times y_2 - y_1 \times x_2$. Each pair of coordinates is processed by an area summation algorithm shown below:

$$\begin{aligned}
 A &= A + \frac{(x_1 \times y_2) - (y_1 \times x_2)}{2} \\
 A &= A + \frac{(x_2 \times y_3) - (y_2 \times x_3)}{2} \\
 &\dots \\
 A &= A + \frac{(x_{n-1} \times y_n) - (y_{n-1} \times x_n)}{2} \\
 A &= A + \frac{(x_n \times y_1) - (y_n \times x_1)}{2}
 \end{aligned}$$

Or in another form:

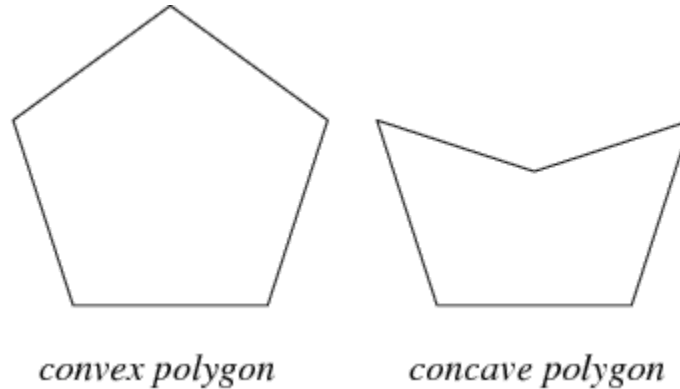
$$A = \frac{1}{2} \left\{ \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} + \begin{bmatrix} x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} + \dots + \begin{bmatrix} x_{n-1} & y_{n-1} \\ x_n & y_n \end{bmatrix} + \begin{bmatrix} x_n & y_n \\ x_1 & y_1 \end{bmatrix} \right\}$$

Based on several mathematical expressions above, we can conclude that to find out the area of a polygon, we can use the following mathematical expressions.

$$A = \frac{1}{2} \left(\left(\sum_{i=1}^{n-1} (x_i \times y_{i+1}) - (y_i \times x_{i+1}) \right) + (x_n \times y_1) - (y_n \times x_1) \right)$$

POLYGON CONVEXITY

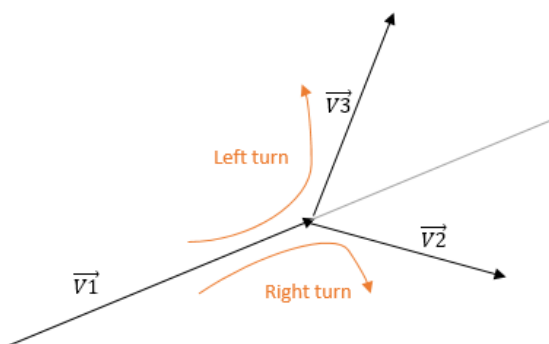
Polygon can be categorized by its type, whether a polygon is a convex polygon or a concave polygon.



- Convex Polygon: Every line is drawn through a polygon, that is not tangent to an edge or corner, meets the polygon's boundary twice. As a result, all of the interior angles are less than 180 degrees.
- Concave Polygon: A concave polygon is defined as a polygon with one or more interior angles greater than 180 degrees. A vertex appears like it has been pushed in towards the polygon's interior.

To determine the convexity of a polygon, we need to know the cross product result of the vector. The vector cross product indicates whether the second vector is clockwise (negative or less than 0) or anticlockwise (positive or greater than 0) with respect to the first vector. If the result of the vector cross product is zero, the two vectors are collinear.

If the result of the vector cross product between all consecutive pairs of edge vectors in a polygon are mixed of the clockwise and anticlockwise turns, then the polygon must be a non-convex or a concave polygon. In contrast, if all the turns resulting from the vector cross product are in the same orientation, they are all clockwise or anticlockwise, then the polygon we examined must be a convex polygon.



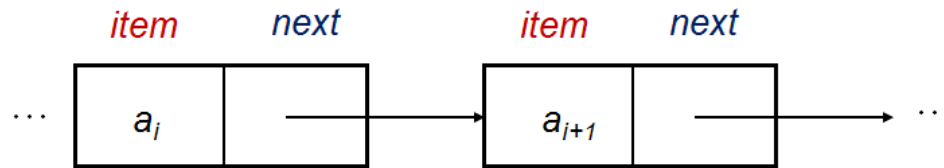
$$f(\vec{v}_1, \vec{v}_2) = x_1y_2 - x_2y_1$$

$$f(\vec{v}_1, \vec{v}_2) > 0 \rightarrow \text{Left turn}$$

$$f(\vec{v}_1, \vec{v}_2) < 0 \rightarrow \text{Right turn}$$

REPRESENT A POLYGON/POLYLINE USING LINKED LIST

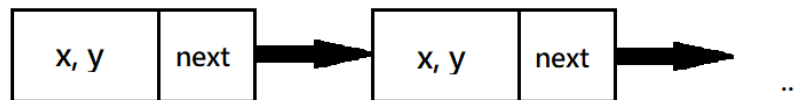
In our program, polygons or polylines are represented using a linked list. In the linked list, the items are ordered by index from 0 to $N-1$ and associate item i with its neighbor item which is $i+1$ through a pointer.



A linked list of a polygon or polyline consists of a list of points, in which each point contains x and y variables. Each of the objects will be implemented using linked list as shown below:

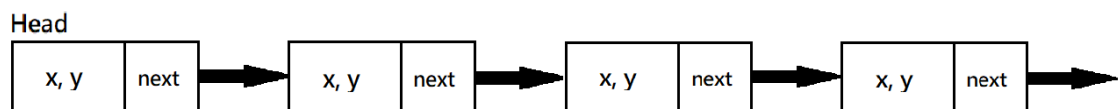
- Points

As we mentioned above, a point of a polygon or polyline contains x and y variables. These variables indicate the coordinate of the point that will be displayed on the screen. The items which are coordinate x and y are stored inside each node of the linked list, and each node has the next variable that points to the next point. Below is the illustration of the linked list:



- Polygons/Polylines

The following picture illustrates the linked list for the polygon or polylines.

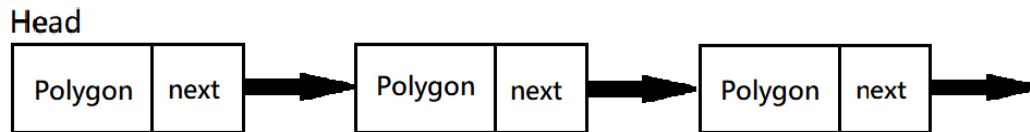


The start point of a polygon or polyline is represented by a head variable inside the linked list. The pointer pointing to the next vertex is also represented by the next variable. Each item in the linked list represents a vertex that will form a polygon or polyline.

- List of Polygons/Polylines

For multi-polygon, a linked list is needed to store polygon and polyline that are displayed on the screen. Each node in the linked list will consist of polygon and

polyline. The head variable will store the first polygon of the list and the next variable is pointing to the next item which is the second polygon and so on.



VERTEX PROCESSING

Vertex processing is a procedure to process using every vertex that we have. Examples of vertex processing are printing vertex coordinates, scaling, and also translating. General pseudocode for vertex processing using linked list is:

```

Edge = Poly.First
While Edge is Not Null Then
    Process(Edge.Info)
    Edge.Next
End While
  
```

Poly.First usually known as Head that will be pointing to the first element. We use a while loop to traverse every element inside the linked list. The value is stored inside Edge.Info, after processing the value, will go to the next element and stop until the last element is pointing to nothing. That's how the whole vertex is processed.

EDGE PROCESSING

Edge processing is a procedure to process an edge, it also can be stated that we need 2 vertices to proceed the edge. Edge processing is needed for drawing, calculating perimeter, also clipping. General pseudocode for edge processing using linked list is:

```

firstEdge = Poly.First
While firstEdge is Not Null Then
    If firstEdge is Not Null Then
        secEdge = firstEdge.Next
    Else
        secEdge = Poly.First
    End If
    Process (firstEdge, secEdge)
    firstEdge = firstEdge.Next
End While
  
```

Because the edge needs to be processed, two vertices are needed, as well as two variables to store the value. The first variable called firstEdge will contain the value of the Head of the linked list. Next, we use a while loop to traverse every element inside the linked list. Then

we have to check the firstEdge variable. If it is not in the last element yet, the second variable, secEdge will contain the value of the next element of firstEdge, otherwise, it will contain the value of the Head. In order to process the last edge, we need two Head values, especially in a polygon. This edge is the one that will connect the first and the last point together. After proceeding with the edge that we want, the firstEdge variable will contain a new value, which is the next value of the edge itself.

III. Implementation

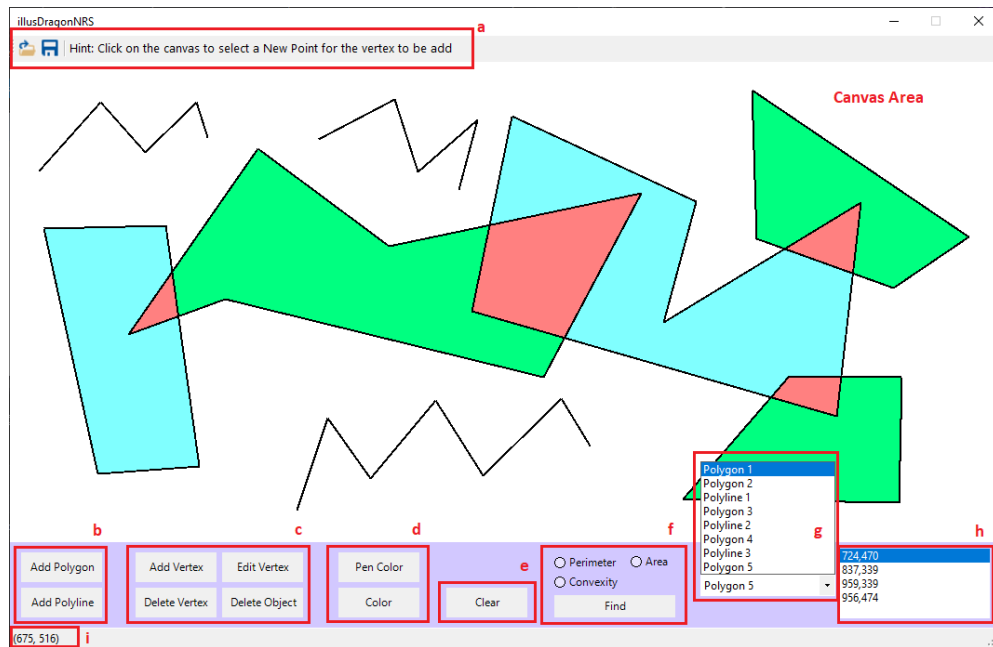


Figure shown above is the main interface of the application. This interface is divided into 3 sections, the first one in the top area, followed with the canvas area, and the bottom part with several red squares.

1. In the 'a' area, there are two buttons to open a file and save a file. Besides the button, there will also be a hint shown for some button clicked.
 - **"Save File"** button will open a save file dialog and let the user choose the directory to save a file. The file will be in txt file and will include the poly name and its vertices.
 - **"Open File"** button will open an open file dialog and let the user choose the file and it has to be in txt file. There will be several cases with this button, if the canvas contains drawing and the user would like to open a file, the application will ask the user to save the previous drawing or leave it and open a file immediately.
2. The canvas area is the area to draw polygons and/or polylines, also with the color using mouse click events.
3. This bottom area is divided into several button section, there are:
 - **b:** This includes two buttons, button "Polygon" or "Add Polygon" and button "Polyline" or "Add Polyline".
 - **"Polygon"** button is a button to draw a polygon. Different from the button "Polyline", button "Polygon" has additional features, there is by clicking the right button in the mouse, it will automatically connect the last point with the start point. After the polygon is created, the button label will change into "Add Polygon".

- **"Polyline"** button is a button to draw a polyline. After the line have been drawn, the button label will change into "Add Polyline"
- **c:** These four buttons will activate or enable if at least one polygon or one polyline has been drawn. These four buttons are:
 - **"Add Vertex"** button is to add a new vertex from the existing polygon vertices by choosing one vertex from the vertices list list box. After this button is clicked, the user has to select a new point in the canvas area and will automatically redraw the new polygon.
 - **"Edit Vertex"** button is to edit a vertex that has been selected in the vertices list Listbox. After this button is clicked, the user has to select a new point in the canvas area and will automatically redraw the polygon.
 - **"Delete Vertex"** button is to delete the selected vertex in the vertices list. After this button is clicked, the application will automatically redraw the new polygon.
 - **"Delete Object"** button is to delete the selected polygon in the polygon list combo box. After this button is clicked, the application will automatically redraw all the rest polygons/polylines.
- **d:** This is coloring sections. This section containing:
 - **"Pen Color"** button is to select the next line color. By clicking this button, there will be a color dialog shown.
 - **"Color"** button is to give the polygon color using the flood fill method. This button will be activated or enabled when at least one polygon has been drawn.
- **e:** This section contains the **"Clear"** button. This button is to delete every polygon/polyline in the canvas area, and also clear every point that has been stored in the linked list.
- **f:** This section contains three radio buttons and one button. This section will be activated or enabled if there is at least one polygon/polyline drawn.
 - **"Perimeter"** radio button: By selecting this radio button, it will show the perimeter of the selected polygon. The result will be shown in a message box.
 - **"Area"** radio button: By selecting this radio button, it will show the area of the selected polygon. The result will be displayed in a message box.
 - **"Convexity"** radio button: By selecting this radio button, it will show the convexity of the selected polygon. The result will be displayed in a message box. The result will be either convex or concave.
 - **"Find"** button is to show the result of the selected radio button.
- **g:** This section contains the **"Polygon List"** combo box. The selected polygon/polyline in the combo box will refer to several actions.
- **h:** This section contains the **"Vertices List"** list box based on the selected polygon in the "Polygon List" combo box.
- **i:** This section located in the bottom left is to track the current mouse position over the canvas.

IV. Design

Main data structure used in the program.

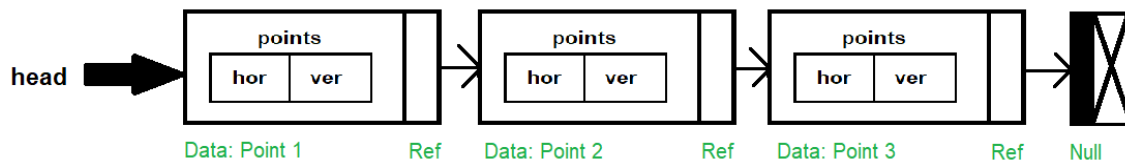
- Point Representation



Points in these programs are represented as a structure. Basically, a structure associates one or more elements with each other and with the structure itself. When you declare a structure, it becomes a composite data type, and you can declare variables of that type. The structure is useful where the implementation of a single object holds several related pieces of information or element. This structure's advantages can act as an object. In this case, the point will consist of two elements: X (horizontal) and Y (vertical).

```
Public Structure Points
    Dim hor As Double
    Dim ver As Double
    .
    .
End Structure
```

- Polygons/Polylines Representation



Polygons or polylines are represented as collections of vertices that are interconnected for each vertex by a Linked List data structure. The linked list is linear data structures sequentially interconnected for each node that consists of data and has an address to the next node. In this case, the linked list is a collection of the nodes with a point as data. The link for each node directly connected to the next node by referring to the node itself.

```
Public Class NPoint
    Private data As Points
    Private ref As NPoint
    .
    .
End Class
```

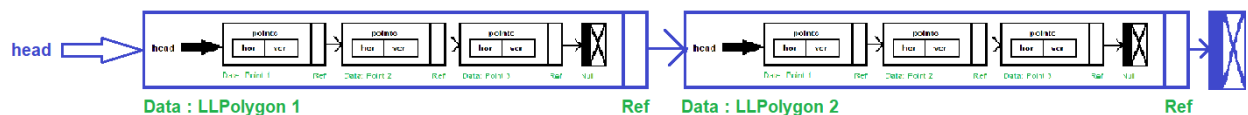
The way to declare the node as a class of Node Point (NPoint) and for each node consist of the data which are represented as points and the reference purposely for the next node of point. Then, in creating a LinkedList, there are two requirements: the list must have a head with a format of node (NPoint) to track the LinkedList node by node and the null point as a mark at the end of the list. First thing first, generating a LinkedList would be assigning head value to be null, because the linked list is still empty.

```
Public Class LLPolygon
    Private head As NPoint

    Public Sub init()
        head = Nothing
    End Sub
    .
    .
End Class
```

While creating a linked list, there are need an initialization by a function and it can be implemented while the node already created and continue with other operation in LinkedList such insertFirst, insertLast, insertAfter, findNode, findPrevNode, and deleteNode.

- Collection of Polygons/Polylines Representation



The collection of polygon/polylines are represented as the same as the collection of points which is used as a LinkedList data structure. The difference is where the data in the node of each polygon are the collection of vertices. The code shows below:

```
Public Class NPolygon
    Private polyName As String
    Private polyData As LLPolygon
    Private polyRef As NPolygon
    .
    .
End Class
```

The declaration of a node of the polygon consists of a name, data which are the LinkedList of vertex, and reference formatted as Node Polygon itself to make a relation for the next node. Then, in creating a LinkedList, it will be the same requirement that must have a head and null value. It totally used to do all operations in LinkedList.

```

Public Class LLPolygons
    Private polyHead As NPolygon

    Public Sub init()
        polyHead = Nothing
    End Sub
End Class

```

After initialization of LinkedList, basically, there are some operations such as insertFirst, insertLast, insertAfter, findNode, findPrevNode, deleteNode, and traverse to access the value. In this LinkedList need some special approach to get point value, there will need to double traverse or nested while where the first traverse the node of polygon and second traverse the LinkedList of the vertex by visiting the node by node.

Global Variables

```

Dim bmp As Bitmap
Dim graphics As Graphics
Dim startpoint, lastpoint As Point
Dim draw, polygon,
    colorize, edit, add As Boolean
Dim listPolygon As New LLPolygons
Dim node As NPolygon
Dim Poly As New LLPolygon
Dim Vertex As NPoint
Dim p, deletePoint, editPoint, addPoint As Points
Dim color, polyColor As Color
Dim numberPolygon As Integer = 1
Dim numberPolyline As Integer = 1

```

Dim bmp As Bitmap

→ Used for creating a canvas that build-in system that commonly works with images defined by pixel.

```
bmp = New Bitmap(canvas.Width, canvas.Height)
```

Dim graphics As Graphics

→ Used for basic various draw operations from graphics objects. Examples: DrawLine, DrawCircle, etc.

```
graphics.DrawLine(New Pen(color, 2), lastpoint.X, lastpoint.Y, e.X, e.Y)
```

Dim startpoint, lastpoint As Point

→ Used for holding the coordinate point that consists of x and y coordinates but this is an object that VB.Net already provides to draw through the System.Drawing properties.

```
graphics.DrawLine(New Pen(color, 2), lastpoint.X, lastpoint.Y, startpoint.X, startpoint.Y)
```

Dim draw, polygon, colorize, edit, add As Boolean

→ Used for controlling the action needed, 'draw' will determine if the user can draw a line by adding a vertex, and 'polygon' will determine the object either polygon or polyline.

```
If draw = True Then      If polygon = True Then
```

```
Dim listPolygon As New LLPolygons
```

→ Used for deceleration of LinkedList of collection of polygons (multi-polygons).

```
node = listPolygon.infoPolyHead
```

```
Dim node As NPolygon
```

→ Used for declaration of Node Polygon that formatted as a node for LLPolygons

```
node = node.infoPolyRef
```

```
Dim Poly As New LLPolygon
```

→ Used for declaration of LinkedList of collection of vertex (a polygon)

```
Poly.addLastPoint(p)
```

```
Dim Vertex As NPoint
```

→ Used for declaration of Node Vertex that formatted as a node for LLPolygon

```
Vertex = node.infoPolyData.infoHead
```

```
Dim p, deletePoint, editPoint, addPoint As Points
```

→ Used for passing the value of selected points from ListBox or Canvas

```
p.SetPoint(e.X, e.Y)
```

```
Dim color, polyColor As Color
```

→ Used for passing the value of the color that is already provided from VB.Net by ARGB format.

```
polyColor = dialogColor.Color
```

```
Dim numberPolygon As Integer = 1
```

→ Used for counting the number of polygons and assign the default value is 1 (one)

```
numberPolygon = CInt(res(1)) + 1
```

```
Dim numberPolyline As Integer = 1
```

→ Used for counting the number of polylines and assign the default value is 1 (one)

```
numberPolyline = CInt(res(1)) + 1
```

Bonus Implementation

- **Determine the polygon is convex or concave**

The way to determine the polygon is convex or concave with traversing each node that needs edge processing.

#checking the selected object are polygon or polyline

```
If res(0) <> "Polygon" Then
```

```
    MessageBox.Show("Not a Polygon, cannot find convexity",  
    "Notice", MessageBoxButtons.OK, MessageBoxIcon.Warning)
```

```
Else
```

```
    #assign and reset the value to be default
```

```
    resNeg = False
```

```
    resPos = False
```

```
#traversing the LinkedList of Polygons
```

```
While Not (node Is Nothing)
```



```

#checking the polygon name match/not to the data in node
If node.infoPolyName = name Then
    Vertex = node.infoPolyData.infoHead
    Dim start As NPoint = Vertex
    #traversing the LinkedList of Vertex
    While Not (Vertex Is Nothing)
        Dim A As NPoint = Vertex
        Dim B As NPoint = Vertex.infoRef
        Dim C As NPoint = B.infoRef
        #check condition while the traverse already reach
        the last node
        If C Is Nothing Then
            #Doing edge processing by find crossProduct for
            each of three points
            resCrossProduct = crossProduct(A.infoX,
            A.infoY, B.infoX, B.infoY, start.infoX,
            start.infoY)
            #checking whether the result is minus or
            positif
            If resCrossProduct < 0 Then
                resNeg = True
            Else
                resPos = True
            End If
            #checking if both of the result is True so
            the polygon is concave
            If resNeg = True And resPos = True Then
                MessageBox.Show("This Polygon is Concave",
                "Convexity", MessageBoxButtons.OK,
                MessageBoxIcon.Information)
                Exit Sub
            End If
            Exit While
        End If
        #Doing edge processing by find crossProduct for
        each of three points
        resCrossProduct = crossProduct(A.infoX, A.infoY,
        B.infoX, B.infoY, C.infoX, C.infoY)
        If resCrossProduct > 0 Then
            resPos = True
        ElseIf resCrossProduct < 0 Then
            resNeg = True
        End If
        #checking if both of the result is True so the
        polygon is concave
    
```

```

        If resNeg = True And resPos = True Then
            MsgBox.Show("This Polygon is Concave",
                "Convexity", MessageBoxButtons.OK,
                MessageBoxIcon.Information)
            Exit Sub
        End If
        #access to next node
        Vertex = Vertex.infoRef
    End While
End If
#access to next node
node = node.infoPolyRef
End While

#if there nothing condition that doesn't match before,
automatically the polygon is convex
MsgBox.Show("This Polygon is Convex", "Convexity",
    MessageBoxButtons.OK, MessageBoxIcon.Information)
End If

```

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

- **Perimeter:**

The calculation of perimeter can be implemented in polyline and polygon by traversing the each node of LinkedList. But, there some special cases in polygon additional edge processing at the end cause of polygon is a geometric object.

#traversing the LinkedList of Polygons

While Not (node Is Nothing)

 #checking the polygon name match/not to the data in node

 If node.infoPolyName = name Then

 #starting the head of LL of vertex

 Vertex = node.infoPolyData.infoHead

 #get value of x and y of the first vertex

 firstX = Vertex.infoX

 firstY = Vertex.infoY

 #traversing the LinkedList of Vertex

 While Not (Vertex Is Nothing)

 #get value of x and y of previous node

 startX = Vertex.infoX

 startY = Vertex.infoY

 #access to next node

 Vertex = Vertex.infoRef

 #if reach the end of node will end the traverse

 If Vertex Is Nothing Then

 Exit While

 End If

```

        #get value of x and y the current of node
        endX = Vertex.infoX
        endY = Vertex.infoY
        #get value of dx and dy by (end point - start
        point)
        deltaX = endX - startX
        deltaY = endY - startY
        #calculating and get the total of perimeter
        every single node
        perimeter += (deltaX * deltaX + deltaY *
        deltaY) ^ 0.5
    End While
    #calculating the additional part in polygon case
    If res(0) = "Polygon" Then
        #get value of dx and dy by (first point - start
        point)
        deltaX = firstX - startX
        deltaY = firstY - startY
        #calculating and get the total of perimeter
        every single node
        perimeter += (deltaX * deltaX + deltaY *
        deltaY) ^ 0.5
    End If
End If
#access to next node
node = node.infoPolyRef
End While

```

→ Area:

The calculation area can be only implemented into the polygon as a geometric object. For this case, the calculation area with LinkedList data structure needs edge processing by accessing each node of LLPolygons and each node of LLPolygon. The implementation will start by traversing the LLPolygons by determining which polygon will be calculated and go inside of LLPolygon that access all nodes to get the value of point/vertex in each polygon. After traverse calculation starting by adding area value with $(X_{curr} * \Delta Y) - (Y_{curr} * \Delta X)$ for each iteration and after out from the iteration will calculate the last point into the first point. At last, we multiply the sum of every calculation with a half. In this calculation, we use Δ other than the original equation to prevent a two big integer multiplication, however, it is still resulting in the same result as the original equation.

```

#traversing the LinkedList of Polygons
While Not (node Is Nothing)
    #checking the polygon name match/not to the data in node
    If node.infoPolyName = name Then
        #starting the head of LL of vertex

```

```

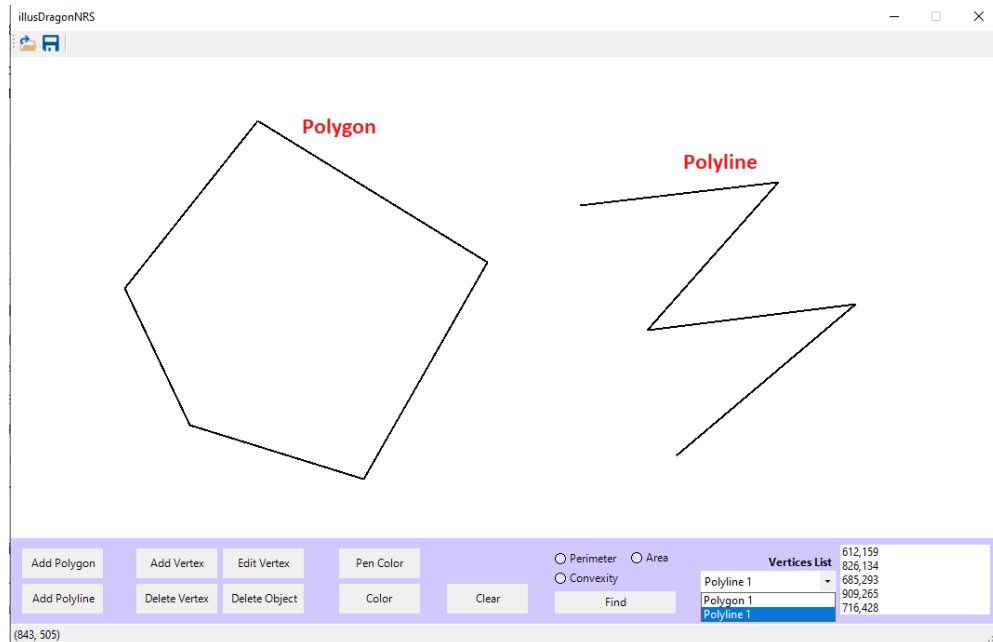
Vertex = node.infoPolyData.infoHead
#get value of x and y of the first vertex
firstX = Vertex.infoX
firstY = Vertex.infoY
#traversing the LinkedList of Vertex
While Not (Vertex Is Nothing)
    #get value of x and y of previous node
    startX = Vertex.infoX
    startY = Vertex.infoY
    #access to next node
    Vertex = Vertex.infoRef
    #if reach the end of node will end the traverse
    If Vertex Is Nothing Then
        Exit While
    End If
    #get value of x and y the current of node
    endX = Vertex.infoX
    endY = Vertex.infoY
    #get value of dx and dy by (end point - start
    point)
    deltaX = endX - startX
    deltaY = endY - startY
    #calculating and get the total of area every
    single node
    area += (startX * deltaY) - (startY * deltaX)
End While
#calculate the dx and dy by (first point - start
point)for the last and head node
deltaX = firstX - startX
deltaY = firstY - startY
#calculating and get the total of area at the last
and head node
area += (startX * deltaY) - (startY * deltaX)
End If
#access to next node
node = node.infoPolyRef
End While
#at the end multiple the area by ½
area *= 0.5

```

V. Evaluation

Test Case 1: Add a polygon/polyline

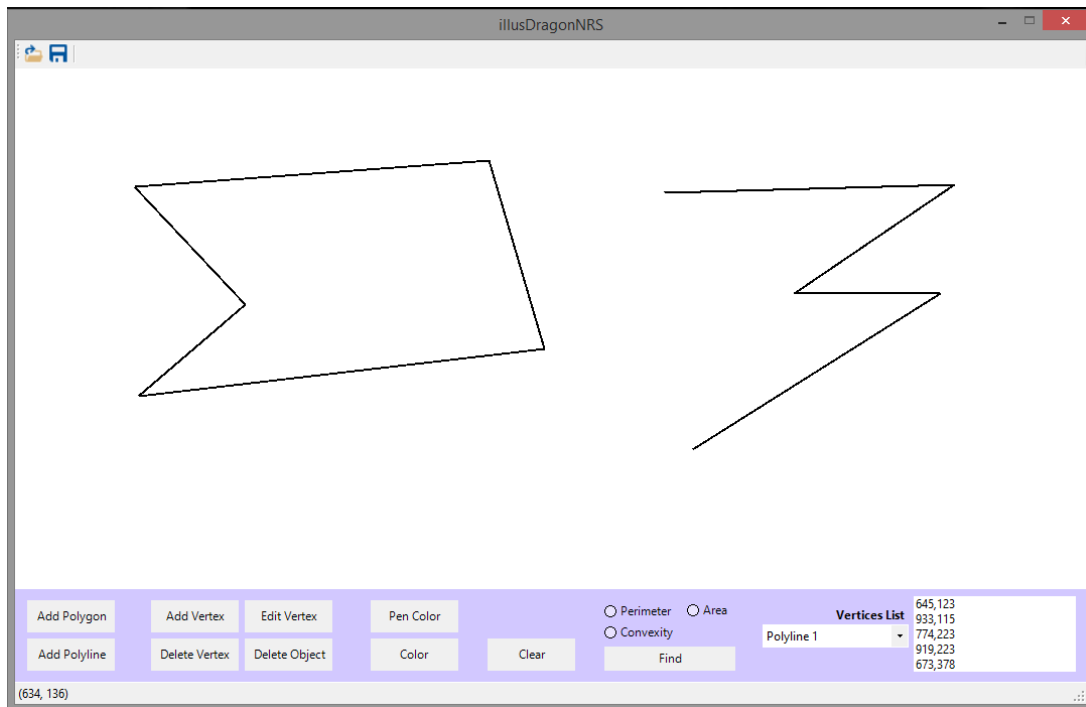
As the figure attached below, shows that this application successfully draws a polygon or/and polyline at the same canvas.



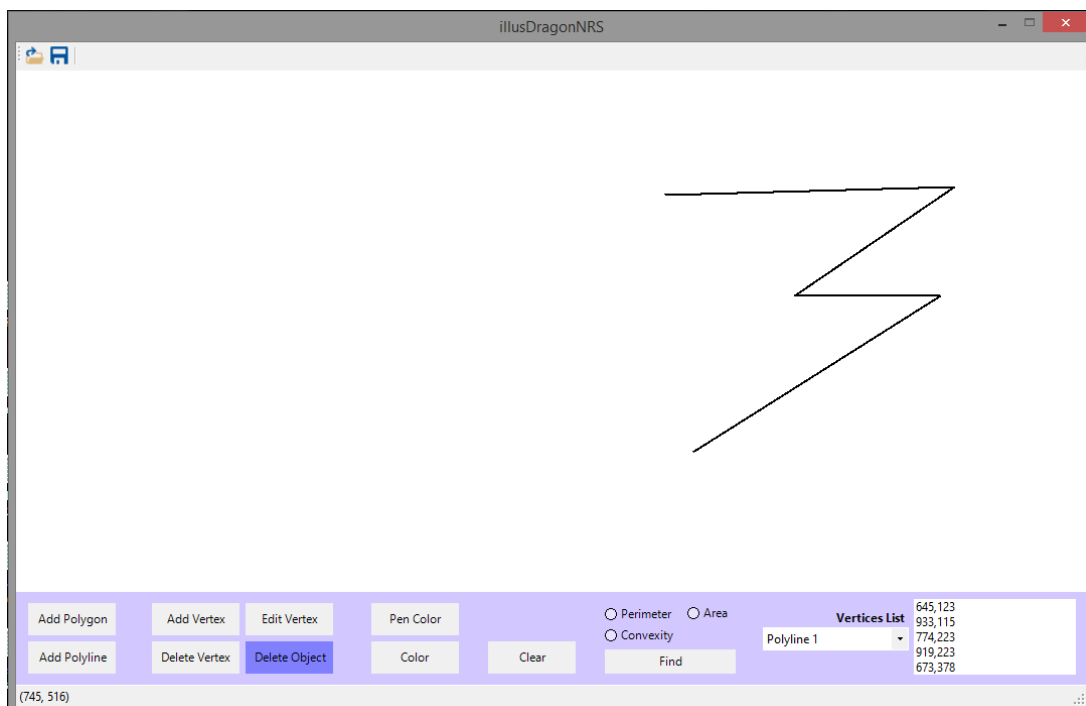
Test Case 2: Delete a polygon/polyline

The following figure shows deleting a selected polygon/polyline. The application successfully ran this test case, to delete selected objects that we want to delete.

Initial picture



The result after delete the object



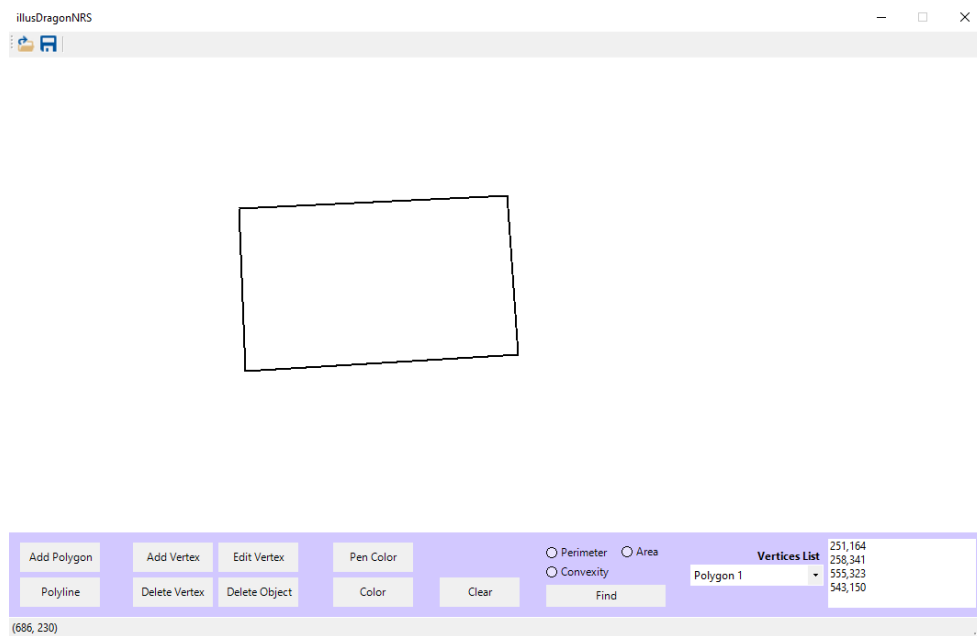
Test Case 3: Adding a vertex to existing polygon/polyline

As in the following figure, there are several initial polygons and the result after adding a vertex based on the sub-test cases. The application successfully implemented this test case.

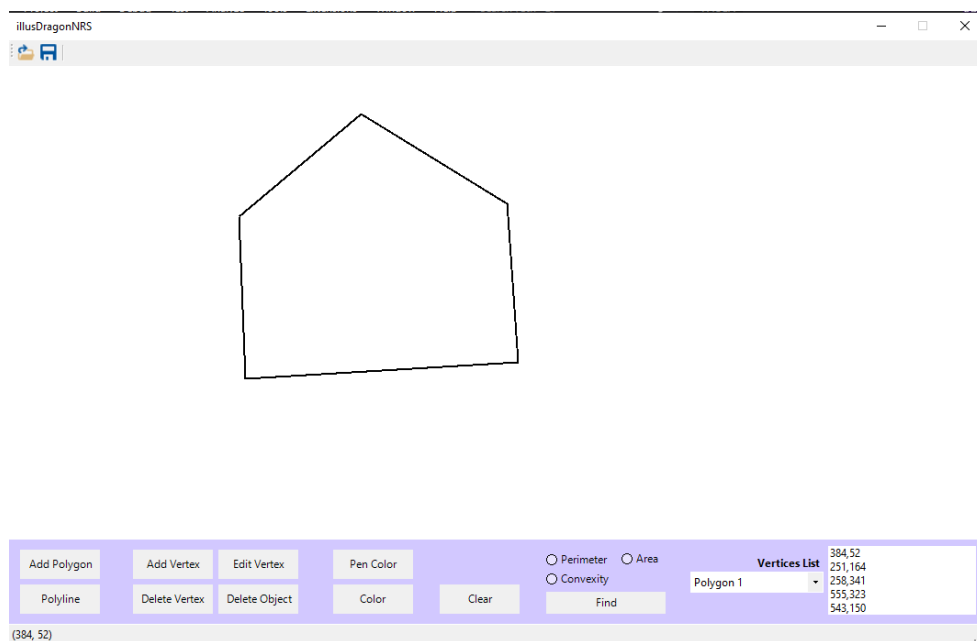
- Test Case 3a: Add a vertex as the first vertex

This test case is successfully implemented in the figure shown below, the first vertex is 251,164 and add a new vertex as the first vertex is 384,52.

Initial Polygon



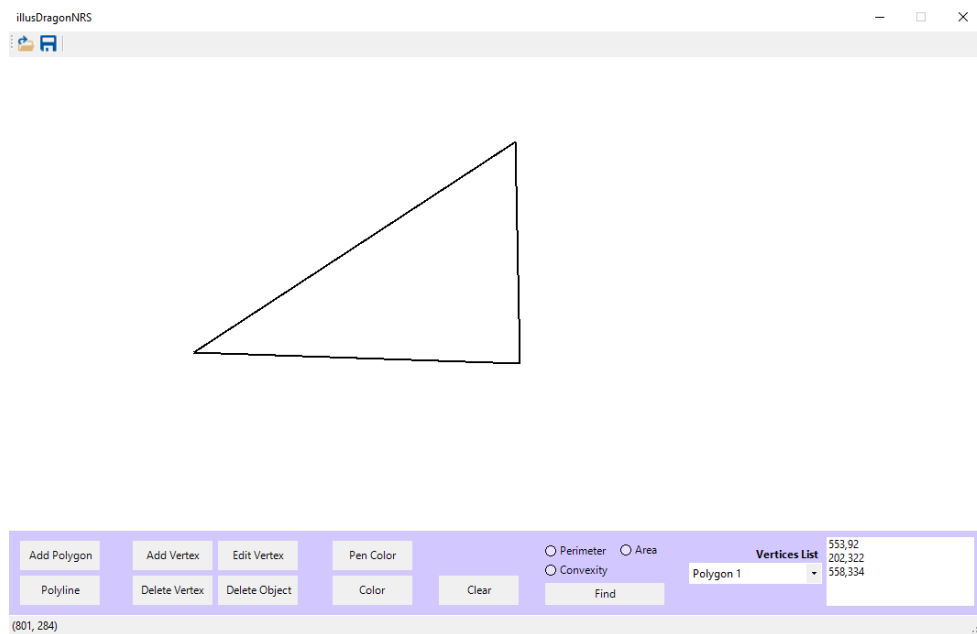
Result after adding vertex as the first vertex



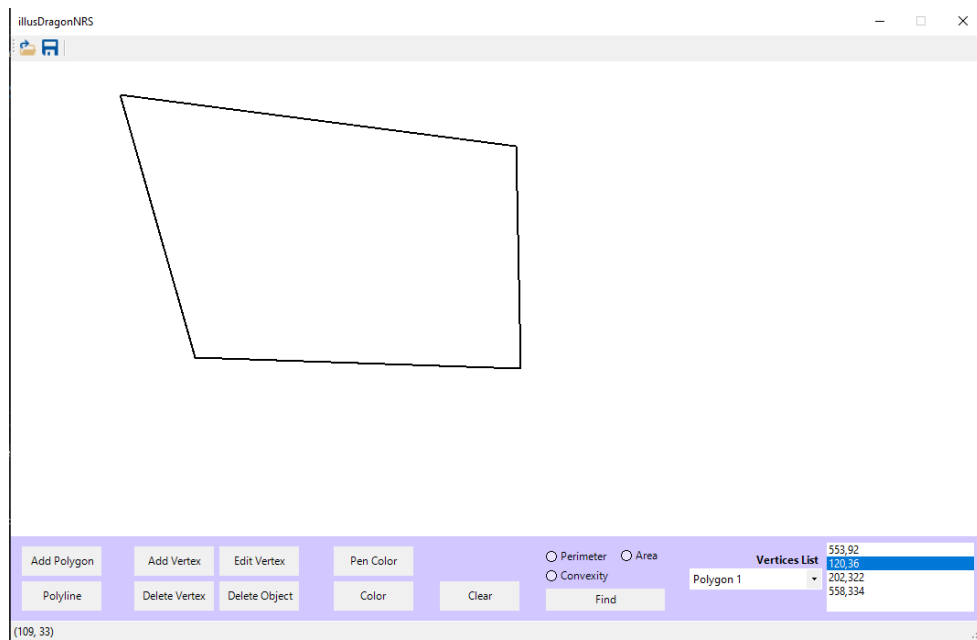
- Test Case 3b: Add a vertex as a “middle” vertex

This test case is successfully implemented to add a new vertex in the middle of all vertices.

Initial Polygon



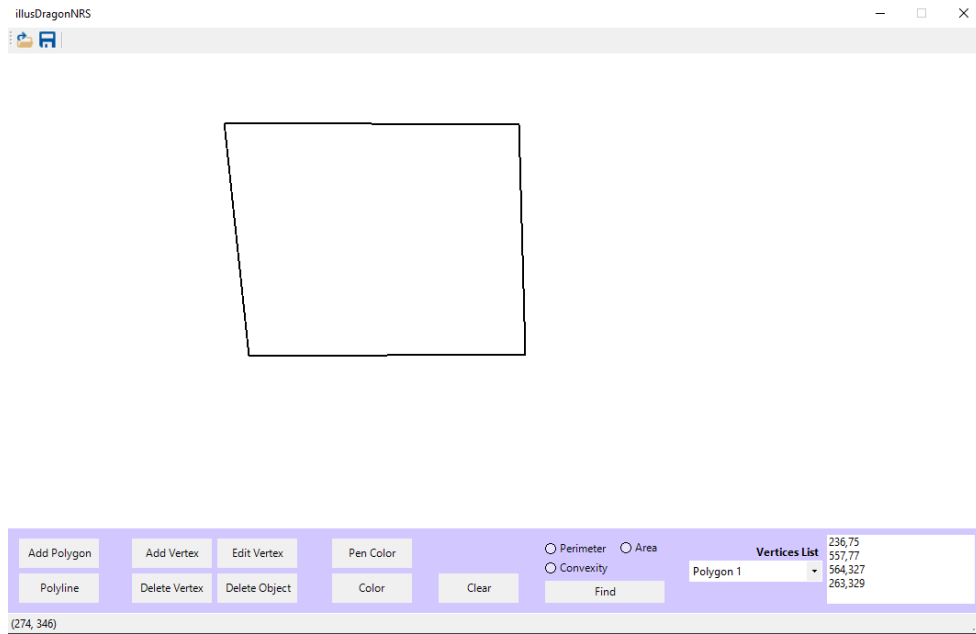
Result after adding a vertex as a “middle” vertex



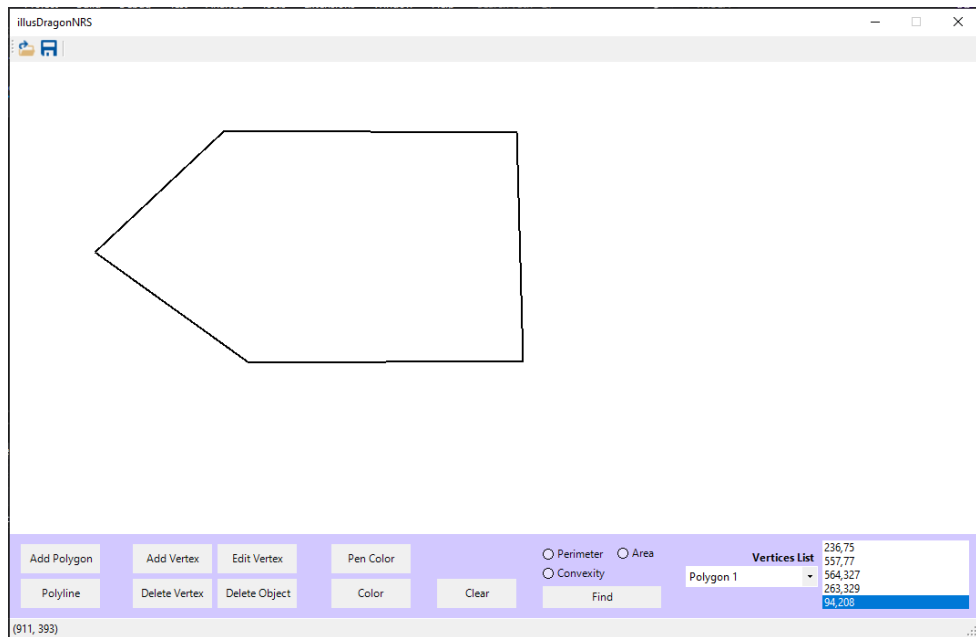
- Test Case 3c: Add a vertex as the last vertex

This test case is successfully implemented in the figure shown below. Initially, the last vertex is 263,329 and we add a new vertex as the last vertex is 94,208.

Initial Polygon



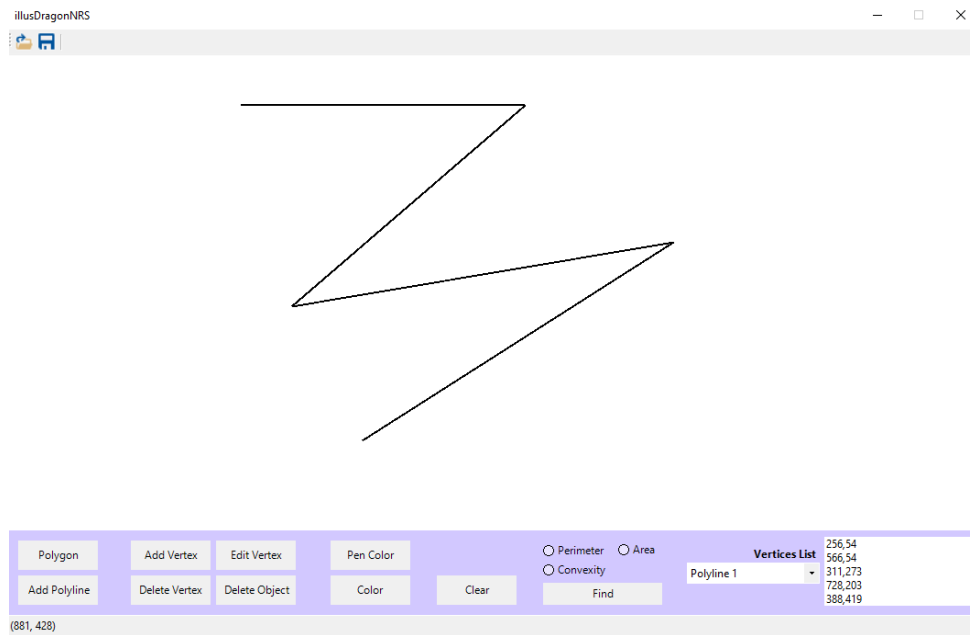
Result after adding a vertex as the last vertex



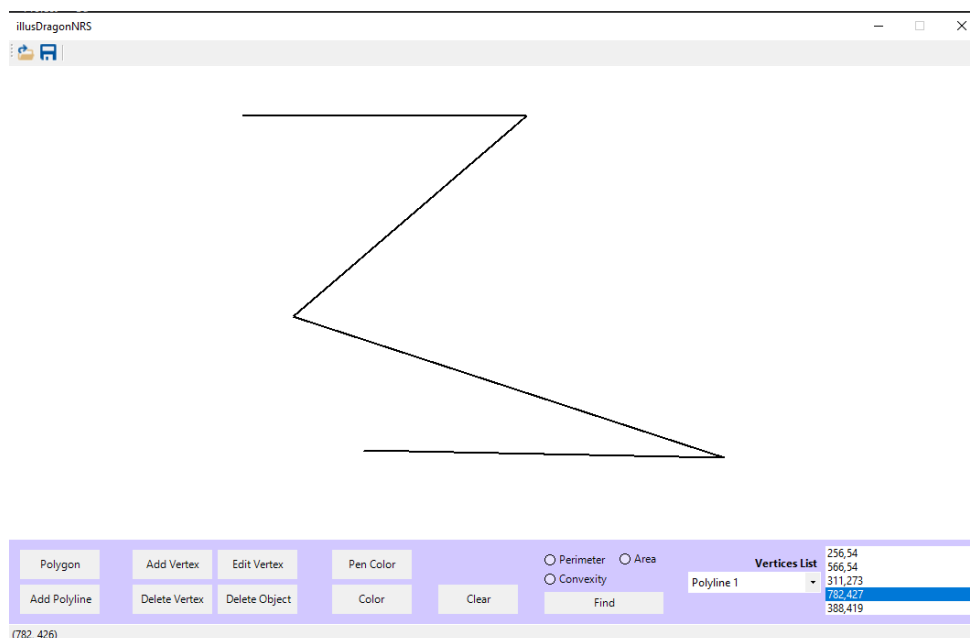
Test Case 4: Changing a location of a vertex from existing polygon/polyline

As the following figure attached below, it is shown that the application is able to change the location of the existing vertex of polygon/polyline. In the figure below, we would like to change the fourth vertex from 728,203 to 782,427. The figure shown below is using the polyline as the example. This test case is also able to change the location of a vertex from a polygon.

Initial Polyline



Result after changing a location of the vertex



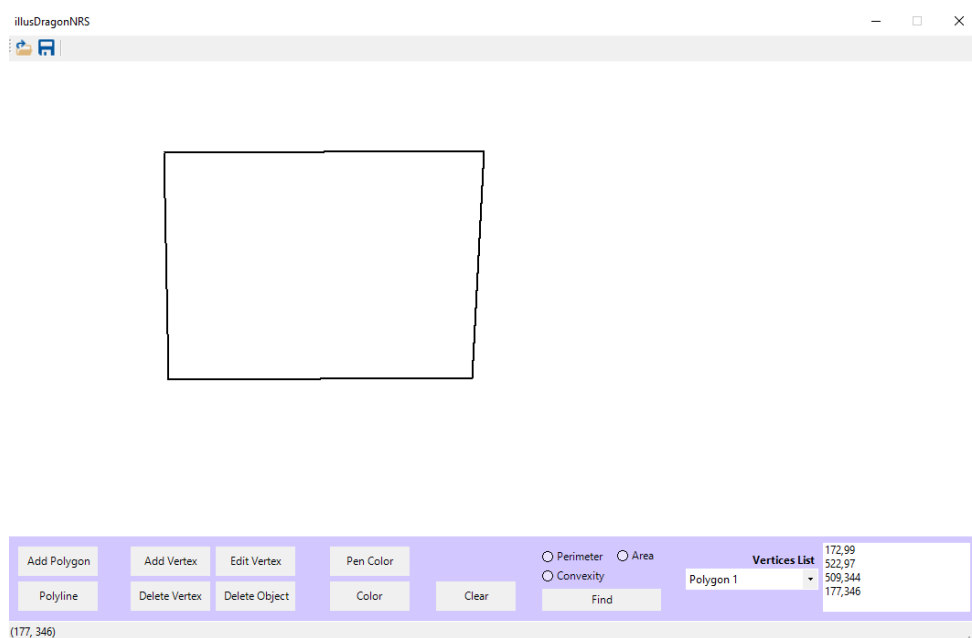
Test Case 5: Deleting a vertex to existing polygon/polyline

As in the following figure, there are several initial polygons and the result after deleting a vertex based on the sub-test cases. The application successfully implemented this test case.

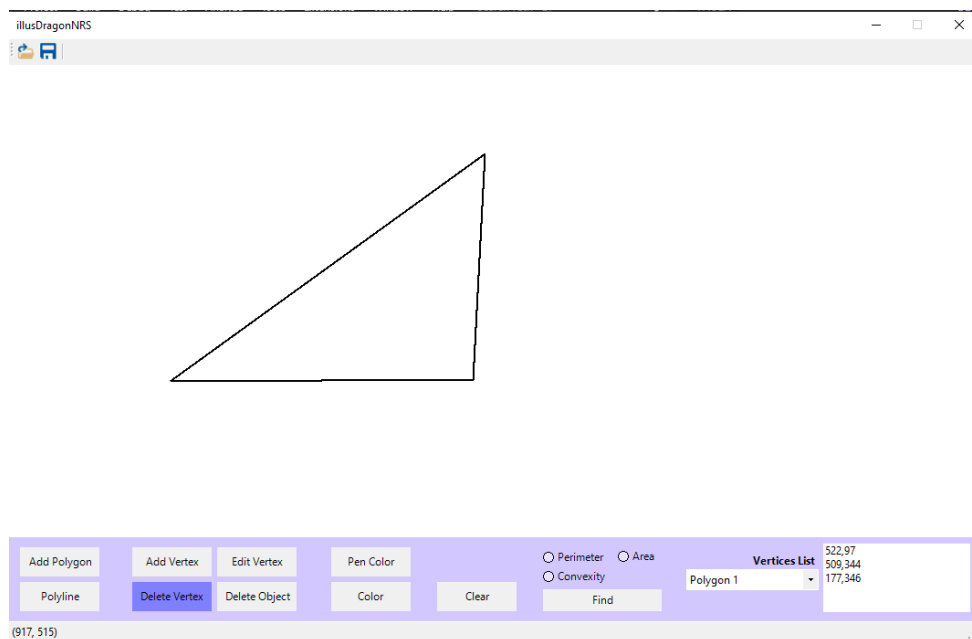
- Test Case 5a: Delete the first vertex

This test case successfully deletes the first vertex, 172,99, and changes the first vertex into 522,97.

Initial Polygon



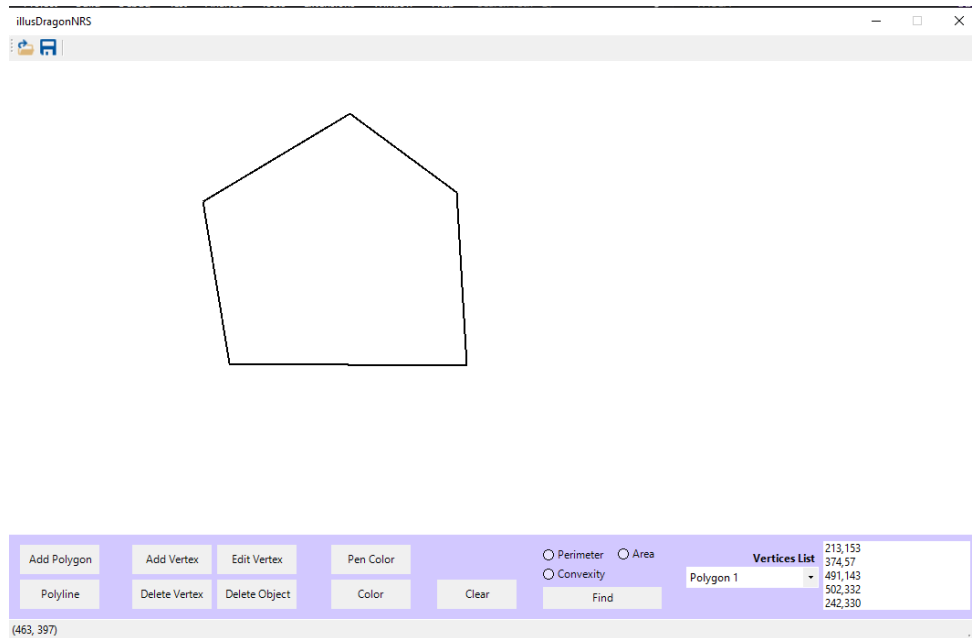
The result after deleting the first vertex



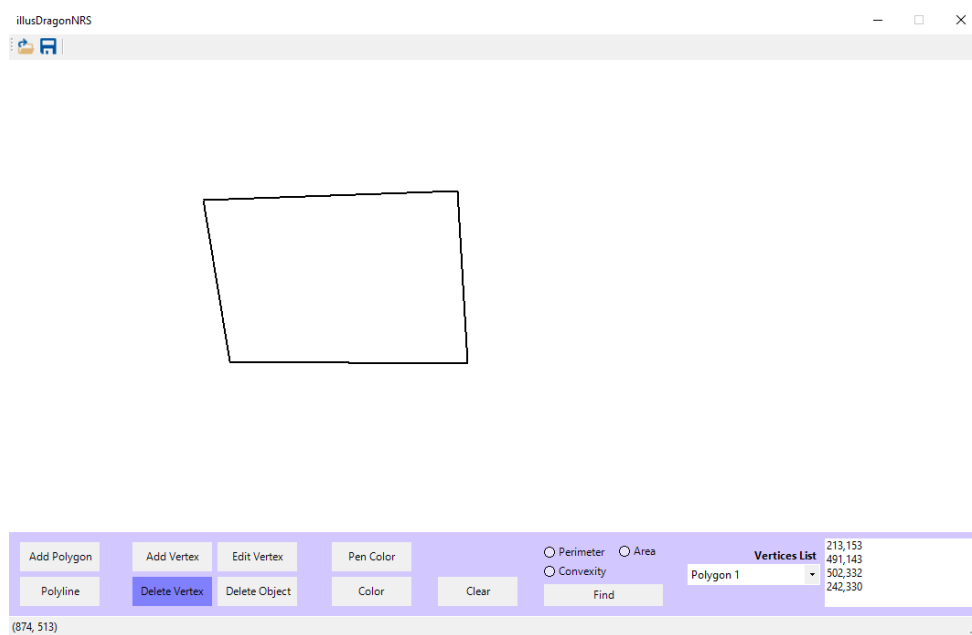
- Test Case 5b: Delete a “middle” vertex

This test case successfully deletes a vertex in the middle. In the figure shown below, we would like to delete the second vertex, 374,157.

Initial Polygon



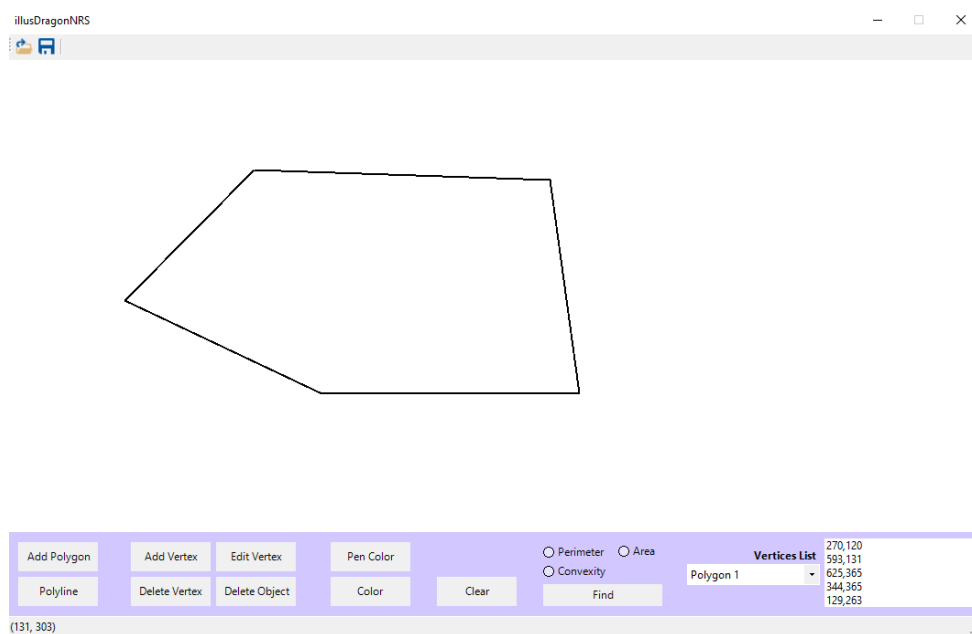
The result after deleting a “middle” vertex



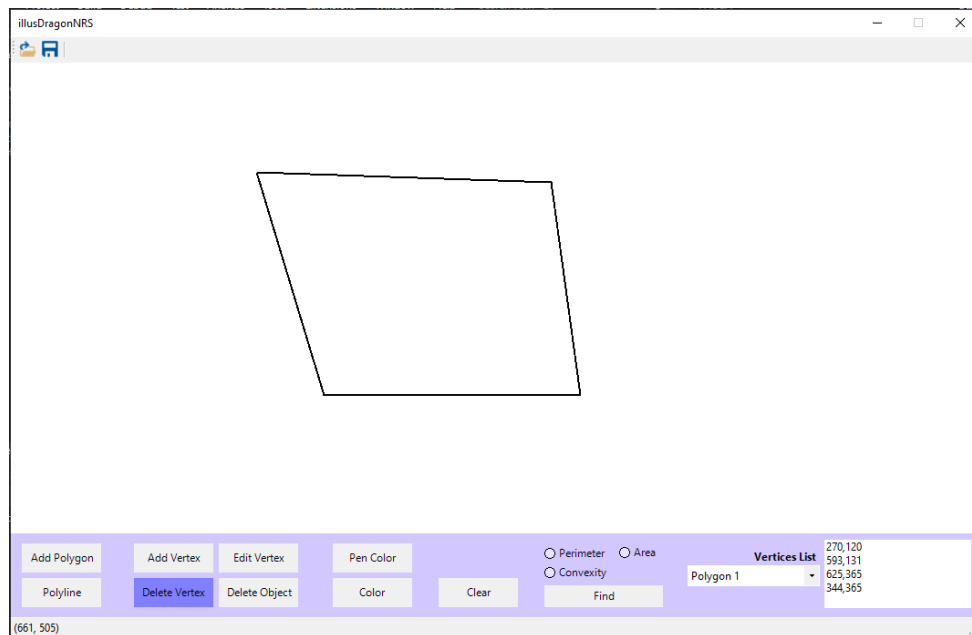
- Test Case 5c: Delete the last vertex

This test case successfully deletes the last vertex. In the figure shown below, we are deleting the last vertex, 129,263..

Initial Polygon:



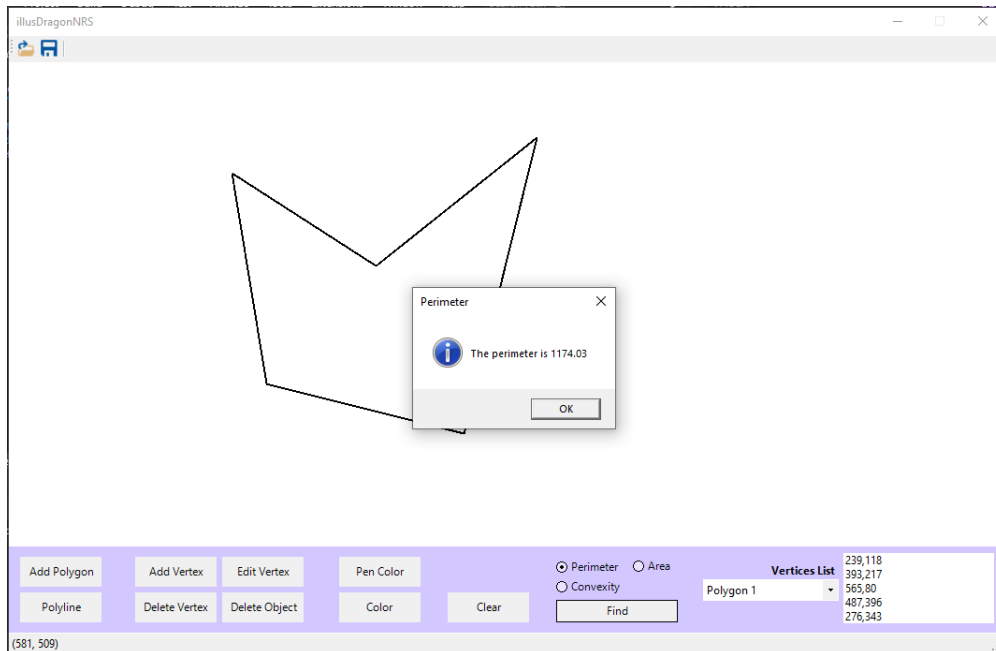
The result after deleting the last vertex



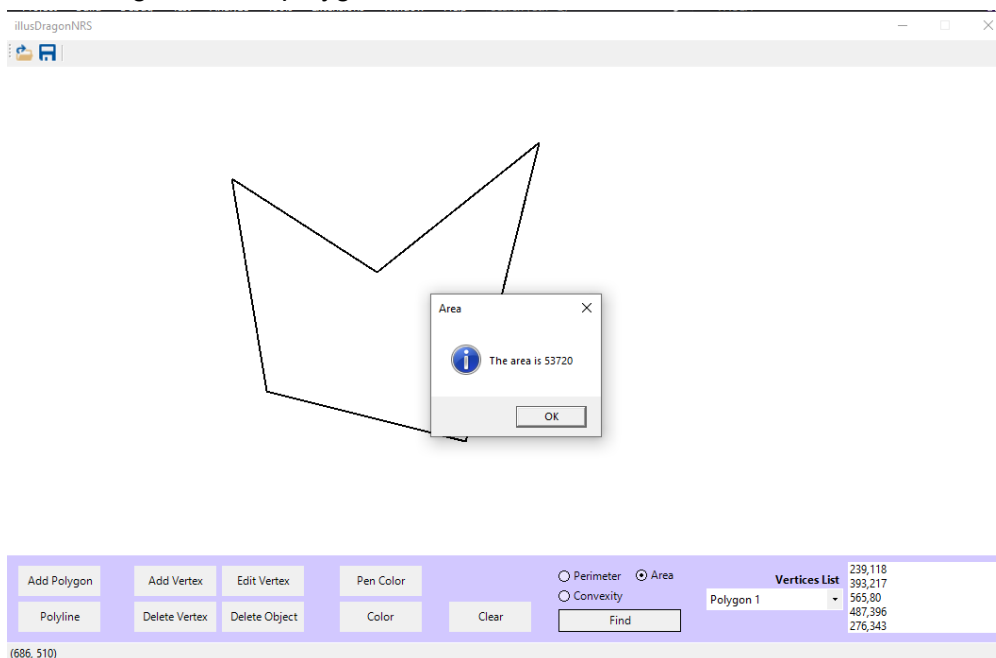
Bonus Test Case

In the bonus case, the application successfully calculates the perimeter of a polygon, the area of a polygon, also determines the convexity of a polygon.

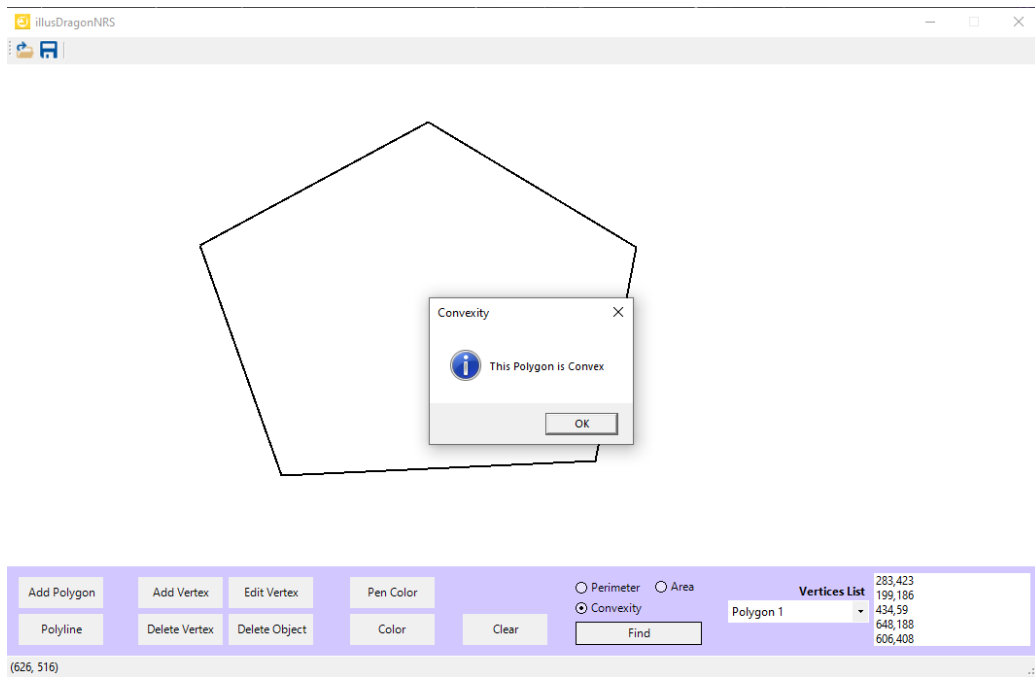
- Calculating perimeter of a polygon



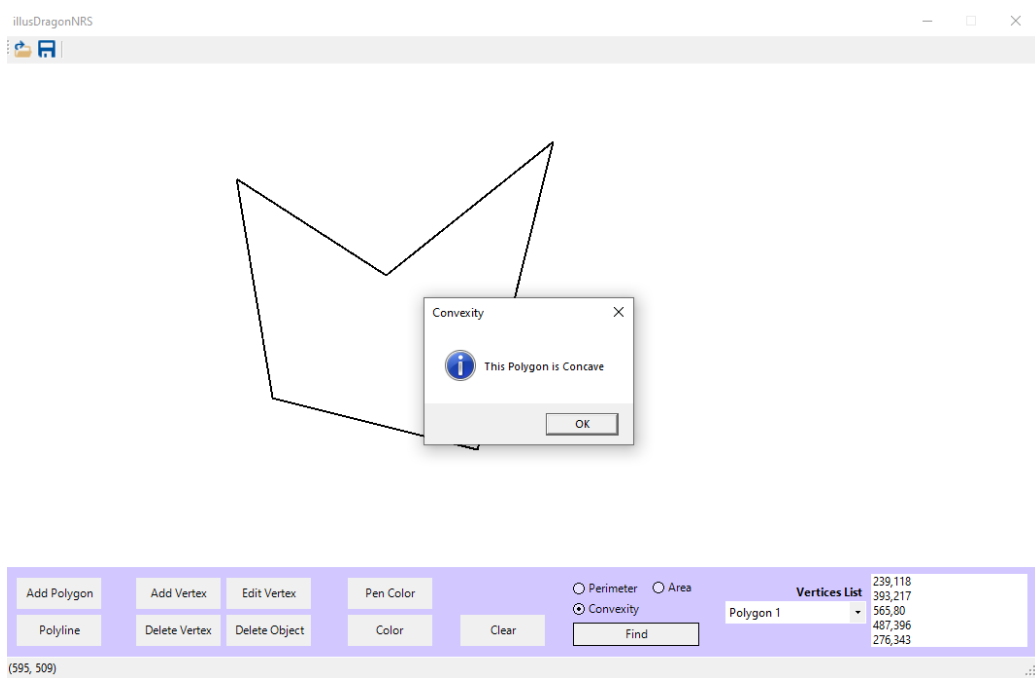
- Calculating Area of a polygon



- Determining the convexity of a polygon
Resulting convex polygon



Resulting concave polygon



VI. Work Log

Date	Activity / Progress	Personnel Involved
Friday, March 19 th , 2021	1. Briefing 2. Basic research	Natasya, Renaldo F, Samuel
Monday, March 22 nd , 2021	1. Creating basic GUI 2. Creating a basic linked list	Natasya, Renaldo F, Samuel
Tuesday, March 23 rd , 2021	1. Drawing polygon and polyline 2. Implementing linked list for points	Natasya, Renaldo F, Samuel
Wednesday, March 24 th , 2021	1. Optimizing linked list for points 2. Prototyping delete vertex	Natasya, Renaldo F, Samuel
Thursday, March 25 th , 2021	1. Coloring polygon 2. Bug fixing	Natasya, Renaldo F, Samuel
Friday, March 26 th , 2021	1. Implementing delete a vertex 2. Prototyping edit and add vertex	Natasya, Renaldo F, Samuel
Saturday, March 27 th , 2021	1. Implementing edit vertex 2. Implementing add vertex 3. Submitting week 1 progress	Natasya, Renaldo F, Samuel
Sunday, March 28 th , 2021	1. UI upgrade 2. Creating a report	Natasya, Renaldo F, Samuel
Monday, March 29 th , 2021	1. Calculating perimeter for a polygon 2. Calculating area for a polygon 3. Determine convexity of a polygon 4. Developing a linked list for multiple polygons/polylines	Natasya, Renaldo F, Samuel
Tuesday, March 30 th , 2021	1. Optimizing main form for multiple polygons/polylines	Natasya, Renaldo F, Samuel
Wednesday, March 31 st , 2021	1. Organizing report 2. Implementing linked list for multiple polygons/polylines	Natasya, Renaldo F, Samuel
Thursday, April 1 st , 2021	1. Organizing report 2. Bug fixing 3. Implementing delete the object 4. Implementing save data to a txt file 5. Implementing load data from a txt file	Natasya, Renaldo F, Samuel

Friday, April 2 nd , 2021	1. Finalize application 2. Finalize report	Natasya, Renaldo F, Samuel
-----------------------------------------	-----------------------------------------------	----------------------------

- Based on the requirements given for Programming Assignment 9 - Drawing Polygons - Linked List, below is the details of whether the requirement is fully implemented, partially implemented, or not implemented:
 1. Draw a polygon on the screen is **fully implemented**,
 2. Draw a polyline on the screen is **fully implemented**,
 3. Delete a polygon/polyline is **fully implemented**,
 4. Add a vertex to a polygon/polyline, the user able to select existing vertices will be adjacent to the added vertex, is **fully implemented**,
 5. Edit a location of a vertex on a polygon/polyline is **fully implemented**,
 6. Delete a vertex from a polygon/polyline, the user able to select a vertex to be deleted, is **fully implemented**,
 7. Change the color of the polygon is **fully implemented**,
 8. Clear the screen, also deleting all polygons and polylines, is **fully implemented**,
 9. Save/load data of polygons and polylines to/from a file is **fully implemented**,
 10. Determine the convexity of a polygon is **fully implemented**, and
 11. Calculating the perimeter and area of a polygon is **fully implemented**.

To sum up, all the requirements given are fully implemented in this application.

VII. Conclusion and Remarks

The application has been created according to the requirements. It works as our expectation, so with the bonus requirements. Using several types of data structures, we are able to store every data of the polygon. In this assignment, we are using a linked list as the data structure. We also try our best to give the best user interface so with the experience to make the application easy to use.

After 2 weeks have passed, we have learned a lot of new things. In the other course, we are using linked lists in the structured programming language. It is a clearly new challenge to create linked lists in object-oriented programming language. We are really happy to finish this programming assignment with new experiences that have never been experienced before.

References

“Polygons.” *Math Is Fun*, www.mathsisfun.com/geometry/polygons.html.

McMillan, Michael. *Data structures and algorithms using Visual Basic.NET*. E-book, Cambridge University Press, 2005.

Lutus, Paul. *Area of an Irregular Polygon*, arachnoid.com/area_irregular_polygon/index.html.

Perimeter and Area,

www.montereyinstitute.org/courses/DevelopmentalMath/COURSE_TEXT2_RESOURCE/U07_L2_T2_text_final.html.