Title: Visibility in point and polygonal obstacles

Problem:

The software implements the VisibleVertices algorithm for calculating visibility relationships between a starting point and a set of disjoint polygonal obstacles. This visibility relationship is critical for robotics and path planning applications, helping to determine which obstacle vertices are visible from the starting point and which are not. The software provides a user-friendly interface for creating disjoint polygonal obstacles, selecting starting points, and dynamically demonstrating visibility relationships.

The algorithm sorts obstacle vertices based on the clockwise angle between p and each vertex. It then builds a search tree and determines which obstacle edges to insert and delete that intersect the half-line from p to each vertex. The algorithm identifies visible vertices from p by iteratively processing the vertices in sorted order.

Approach:

This software tool provides several visualizations to help understand the algorithm:

1. Polygon Editor:

   Users can use the mouse to draw disjoint polygonal obstacles.
2. Animation Control:

   Enables users to adjust the animation speed of the algorithm.
3. Animation of the starting point and obstacle vertices:

   Animating the movement of the starting point p and the obstacle vertices, displaying visibility relationships, direction lines and edges saved in the search tree in real time.
4. Obstacle vertex sorting:

   Display the order of obstacle vertices according to the clockwise angle from the starting point.
5. Search tree animation:

   The animation demonstrates the insertion and deletion of obstacle edges in the search tree, providing insight into the inner workings of the algorithm.

6. Text description of search tree updates:

    Provides text descriptions of search tree updates, supplemented by animations for comprehensive understanding.

7. Explanation of Visibility:

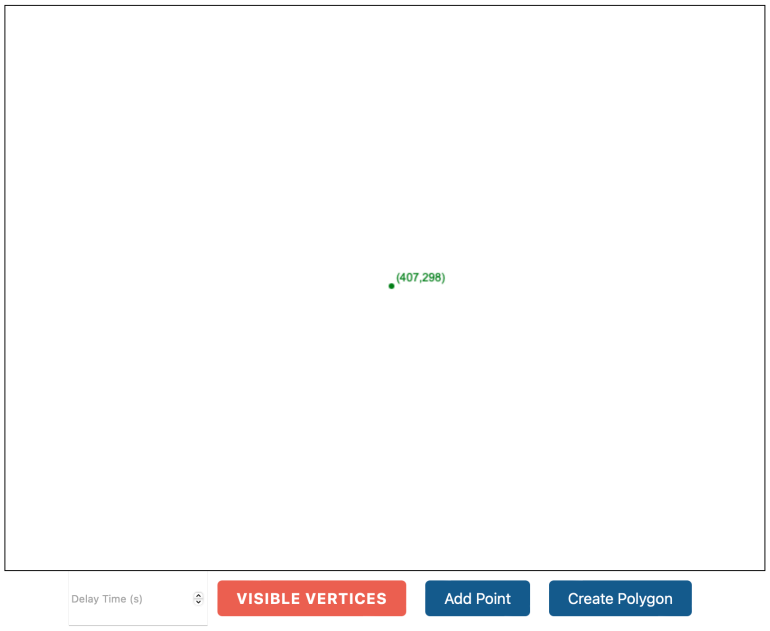
    A detailed explanation of why some obstacle vertices are visible from the starting point, while other obstacle vertices remain invisible, is helpful in understanding the Visible algorithm.

Compiling/Running:

The software is implemented using HTML, JavaScript and CSS and can be accessed from any modern web browser on any computer or system. To compile and run the software, simply open the HTML file in a web browser.

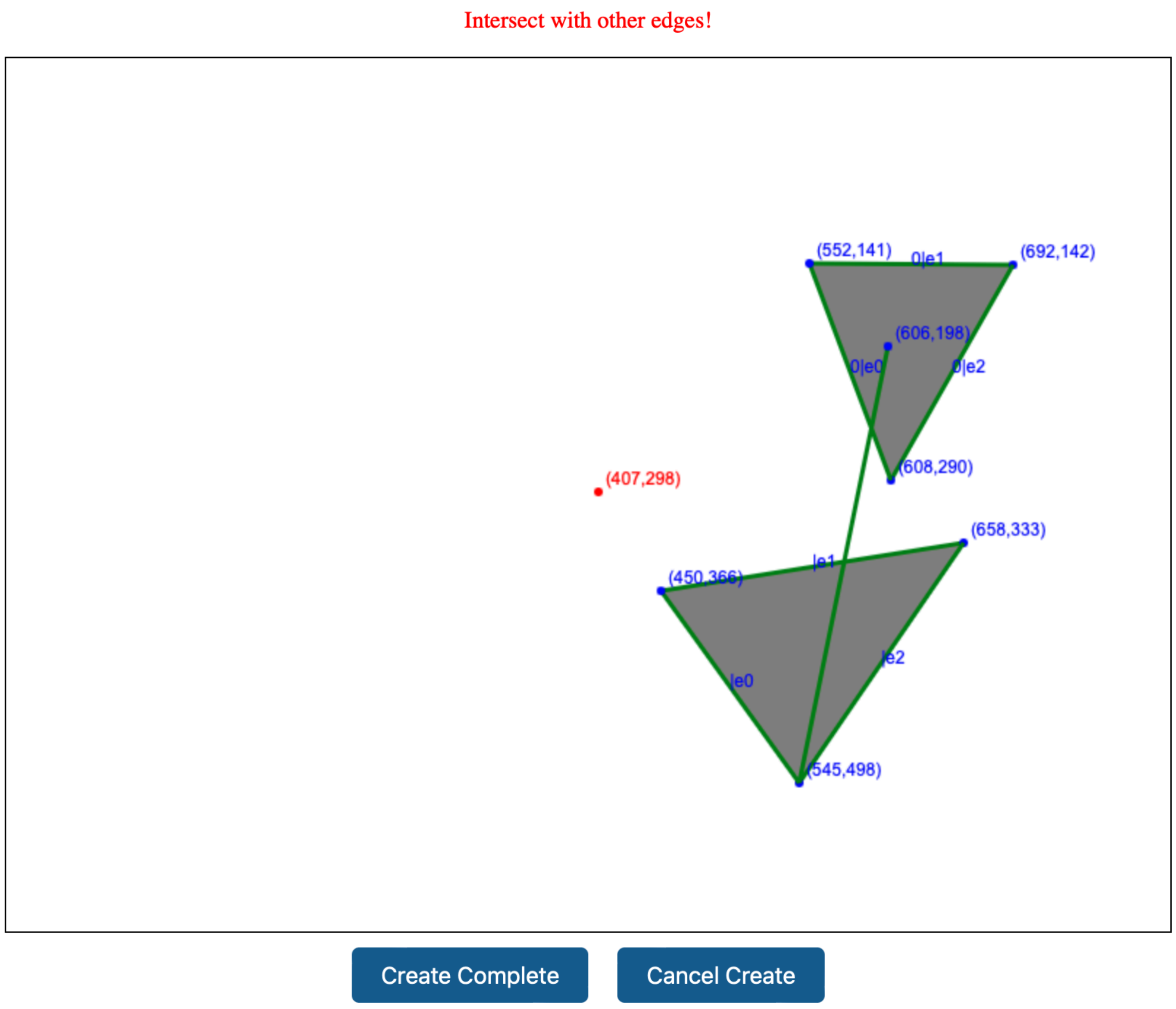
Specific instructions for using the software:

1. **Starting interface:**



Clicking "Add Point" will set the green point clicked by the mouse as the starting point p.
Clicking "Create Polygon" will open the polygon editor.
Clicking "VISIBLE VERTICES" will open the algorithm animation interface.
Setting "Delay Time" can control the speed of algorithm animation, the default is 1s.

2. Polygon editing interface:

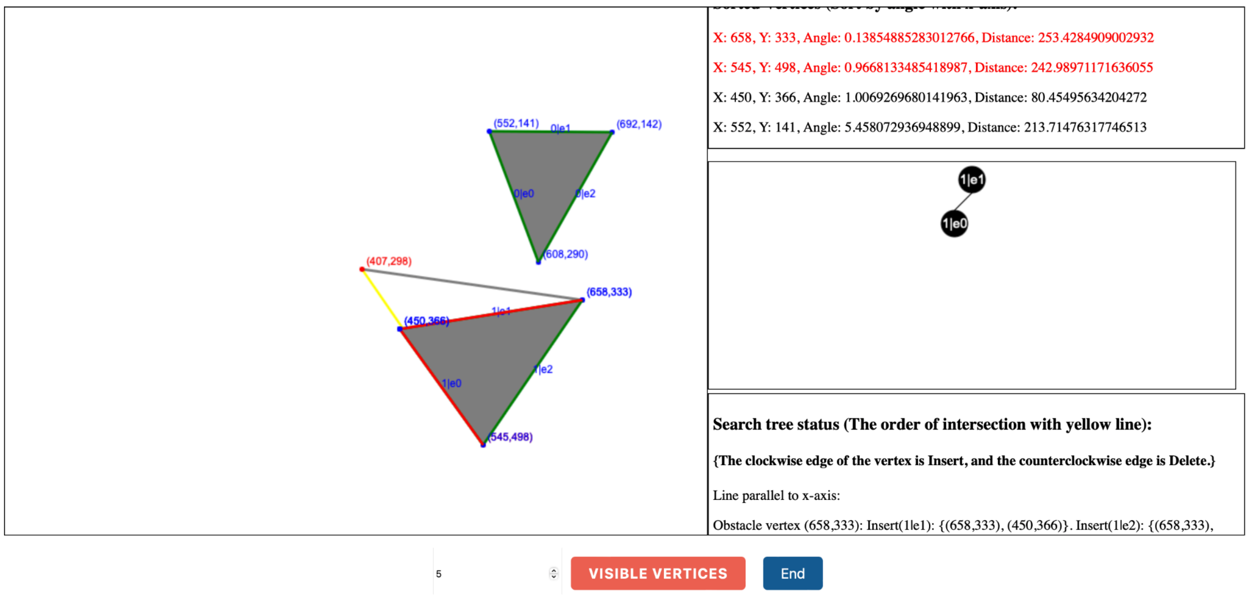


The polygon editor will automatically detect whether the polygons you draw intersect and prompt in red text.

Clicking "Create Complete" will create a polygon.

Clicking "Cancel Create" will cancel the creation of the polygon.

3. Algorithm animation interface:



X: 658, Y: 333, Angle: 0.13854885283012766, Distance: 253.4284909002932

X: 545, Y: 498, Angle: 0.9668133485418987, Distance: 242.98971171636055

X: 450, Y: 366, Angle: 1.0069269680141963, Distance: 80.45495634204272

X: 552, Y: 141, Angle: 5.458072936948899, Distance: 213.71476317746513

Search tree status (The order of intersection with yellow line):

{The clockwise edge of the vertex is Insert, and the counterclockwise edge is Delete.}

Line parallel to x-axis:

Obstacle vertex (658,333): Insert(1|e1): {(658,333), (450,366)}. Insert(1|e2): {(658,333),

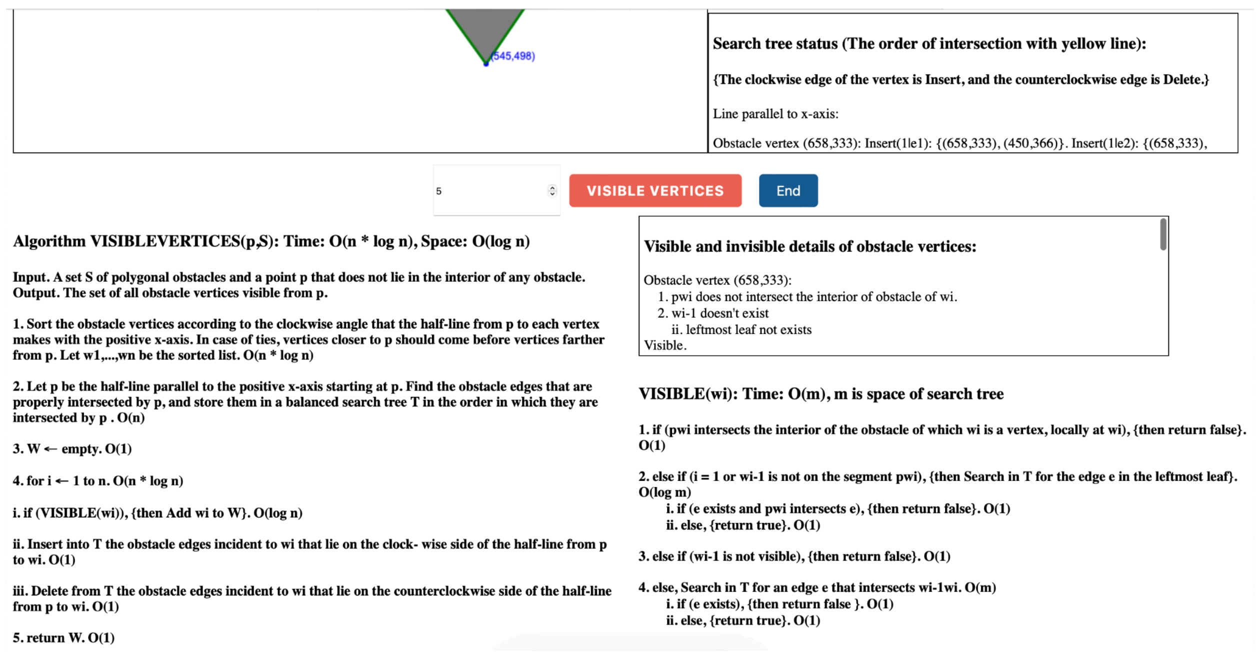
The box on the left is an animation demonstration of the algorithm. The yellow line represents the direction in which the starting point p is looking. The gray line represents the visible vertices of the starting point p. The red edges of the polygon represent edges saved in the search tree.

The first box on the right is the order of the obstacle vertices. The order is sorted by the clockwise angle between the starting point and the obstacle. This sorting can use duality to make sorting time O(n*logn). Each time a vertex is completed while the algorithm is running, the vertex will turn red.

The second box on the right is the update animation of the search tree. This animation shows the insertion and deletion of obstacle edges in the search tree.

The third box on the right is a text description of the search tree update. It describes at which obstacle vertices the search tree inserts and deletes obstacle edges. It can be understood that it is the text description of the update animation of the search tree in the second box on the right.

4. Algorithm text explanation:



Search tree status (The order of intersection with yellow line):

{The clockwise edge of the vertex is Insert, and the counterclockwise edge is Delete.}

Line parallel to x-axis:

Obstacle vertex (658,333): Insert(1|e1): {(658,333), (450,366)}. Insert(1|e2): {(658,333),

Algorithm VISIBLEVERTICES(p,S): Time: O(n * log n), Space: O(log n)

```
Input. A set S of polygonal obstacles and a point p that does not lie in the interior of any obstacle.
Output. The set of all obstacle vertices visible from p.

1. Sort the obstacle vertices according to the clockwise angle that the half-line from p to each vertex
   makes with the positive x-axis. In case of ties, vertices closer to p should come before vertices farther
   from p. Let w1,...,wn be the sorted list. O(n * log n)

2. Let p be the half-line parallel to the positive x-axis starting at p. Find the obstacle edges that are
   properly intersected by p, and store them in a balanced search tree T in the order in which they are
   intersected by p . O(n)

3. W ← empty. O(1)

4. for i ← 1 to n. O(n * log n)

i. if (VISIBLE(wi)), {then Add wi to W}. O(log n)

ii. Insert into T the obstacle edges incident to wi that lie on the clock- wise side of the half-line from p
    to wi. O(1)

iii. Delete from T the obstacle edges incident to wi that lie on the counterclockwise side of the half-line
     from p to wi. O(1)

5. return W. O(1)
```

Visible and invisible details of obstacle vertices:

Obstacle vertex (658,333):
1. pwi does not intersect the interior of obstacle of wi.
2. wi-1 doesn't exist
   ii. leftmost leaf not exists

Visible.

VISIBLE(wi): Time: O(m), m is space of search tree

```
1. if (pwi intersects the interior of the obstacle of which wi is a vertex, locally at wi), {then return false}.
   O(1)

2. else if (i = 1 or wi-1 is not on the segment pwi), {then Search in T for the edge e in the leftmost leaf}.
   O(log m)
     i. if (e exists and pwi intersects e), {then return false}. O(1)
     ii. else, {return true}. O(1)

3. else if (wi-1 is not visible), {then return false}. O(1)

4. else, Search in T for an edge e that intersects wi-1wi. O(m)
     i. if (e exists), {then return false }. O(1)
     ii. else, {return true}. O(1)
```

VisibleVertices and Visible algorithms are written at the bottom of the page, and big O is written after each step.

The box on the right details why some obstacle vertices are visible from p and other obstacle vertices remain invisible. This box can help understand the Visible algorithm.