Polygon Triangulation

Generated by Doxygen 1.8.18

Chapter 1

Main Page

1.1 Overview

This program deals with triangulation of polygons.
How it works
A simple overview of the triangulation process:
Step 1: Subdivide the input polygon into monotones by drawing diagonals in the polygon.
Step 2: For each monotone, perform Monotone triangulation which can be easily done.
Step 3: Combine the output of all monotone triangulations to get the final triangulation for the polygon

The ear clipping algorithm is even more simpler, but is not efficient w.r.t time complexity:
Step 1: Find the "ear" of the polygon. which is the vertex whose internal angle is less than PI
Step 2: Check if any other Reflex Vertex (whose internal angel is greater than PI) comes under the triangle to be formed. If so, then repeat Step 1 until a valid "ear" can be found. It is proved that there are always atleast 2 ears for any vertex hence, it is sure that we will find it.
Step 3: Triangulate this ear by drawing a diagonal on the adjacent vertices to this vertex, store this operation/triangle and remove the ear vertex from the polygon. loop from step 1 until we triangulate completely.

Steps to Compile and Run :
1) *cd* into the src directory
2) Run *g++ main.cpp* which generates an executable called *a.out* in the same directory
3) Run the executable using *./a.out* (on linux)
3.1) The executable takes a dataset from command line argument. For example, to use an existing dataset, run *./a.out ../datasets/1sq.txt*
3.2) If no command-line argument is given, it takes input from the shell directly (stdin)

Performance of the algorithm is documented in the report

1.1.1 Input

Input is given as follows:

- Input can be given both from file (via command-line args) or stdin
- Input **must** contain the clockwise ordering of input vertices
- First line must contain the no of Points to be taken as input by the program.

- Each of next line must contain 2 integers, space seperated denoting the (x, y) coordinates of each point.
- Each coordinate must be of integer type in the range -10^8 to 10^8.
- Number of coordinates must be less than 1 Billion.

1.1.2 Output

- The output contains the coordinates of all the triangles formed after triangulation of the polygon.

1.1.3 Author

The algorithm is implemented and documented by **Rikil Gajarla (2017A7PS0202H)**.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Edge Class Reference

Basic geometric Edge class to store endpoints of edges.
`#include <Geometry.h>`

Public Member Functions

- Edge ()

  Construct a new empty Edge object.
- Edge (Vertex &source, Vertex &destination)

  Construct a new Edge object from given vertices.
- Edge (float x1, float y1, float x2, float y2)

  Construct a new Edge object from vertex coordinates.
- bool operator== (const Edge &e)

  To overload == operator on Edge type objects.
- bool operator!= (const Edge &e)

  To overload != operator on Edge type objects.

Public Attributes

- Vertex **src**
- Vertex **dst**

4.1.1 Detailed Description

Basic geometric Edge class to store endpoints of edges.
This class stores the 2 Vertex type objects to represent an edge. Directionality is NOT assumed.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Edge() `[1/3]`

`Edge::Edge ( )  [inline]`
Construct a new empty Edge object.

4.1.2.2 Edge() [2/3]

```
Edge::Edge (
            Vertex & source,
            Vertex & destination )  [inline]
```

Construct a new Edge object from given vertices.
To construct a Edge type object from 2 Vertex object. The parameter names might be misleading, but it is guarenteed that no directionality is assumed

Parameters

source	First Vertex object
destination	Second Vertex object

4.1.2.3 Edge() [3/3]

```
Edge::Edge (
            float x1,
            float y1,
            float x2,
            float y2 )  [inline]
```

Construct a new Edge object from vertex coordinates.

Parameters

x1	x coordinate of first vertex
y1	y coordinate of first vertex
x2	x coordinate of second vertex
y2	y coordinate of second vertex

4.1.3 Member Function Documentation

4.1.3.1 operator"!=()

```
bool Edge::operator!= (
            const Edge & e )  [inline]
```

To overload != operator on Edge type objects.
This returns NOT of the == overloaded operator

Parameters

e	Edge object to be compared with

Returns

true if the edge does not match

false if the edge matches

4.1.3.2 operator==()

```
bool Edge::operator== (
                const Edge & e )  [inline]
```

To overload == operator on Edge type objects.
This returns true only if both endpoints match between the objects being compared. The order is considered to be important as we want to know if its the same edge or not.

Parameters

e	Edge object to be compared with

Returns

true if the edge matches

false otherwise

The documentation for this class was generated from the following file:

- Geometry.h

4.2 Face Class Reference

Basic geometric Face class to store a Face structure.
`#include <Geometry.h>`

Public Member Functions

- Face ()

  Construct a new empty Face object.
- Face (long long Id)

  Construct a new Face object.

Public Attributes

- long long **id**

4.2.1 Detailed Description

Basic geometric Face class to store a Face structure.
This Face class has an id and a representative HalfEdge which can be traversed to get the surrounding vertices and edges.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Face() [1/2]

```
Face::Face ( )  [inline]
```

Construct a new empty Face object.

4.2.2.2 Face() [2/2]

```
Face::Face (
                long long Id )  [inline]
```

Construct a new Face object.

Parameters

Id	id to be given to this face object (must be unique)
representative	A HalfEdge to represent this face (can be any of the incident half edges)

The documentation for this class was generated from the following file:

- Geometry.h

4.3 Graph Class Reference

This class stores the graph representation of the input polygon.
`#include <monotoneSubdivision.h>`

Public Member Functions

- Graph (vector< Vertex > &input)

  Construct a new Graph object from the clockwise ordering of vertices.
- void connect (const Vertex &u, const Vertex &v)

  connect or draw diagonal to the given Vertices
- Vertex prev (Vertex &v)

  The previous vertex to the given vertex in the clockwise ordering.
- Vertex next (Vertex &v)

  The next vertex to the given vertex in the clockwise ordering.
- list< vector< Vertex > > getSubPolygons ()

  Get the Sub Polygons of the Graph object.

Public Attributes

- vector< Vertex > **original**

4.3.1 Detailed Description

This class stores the graph representation of the input polygon.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Graph()

```
Graph::Graph (
              vector< Vertex > & input )  [inline]
```

Construct a new Graph object from the clockwise ordering of vertices.

Parameters

input	input vertices of polygon as given in clockwise order

4.3.3 Member Function Documentation

4.3.3.1 connect()

```
void Graph::connect (
             const Vertex & u,
             const Vertex & v )  [inline]
```

connect or draw diagonal to the given Vertices

Parameters

u	First Vertex
v	Second Vertex

4.3.3.2 getSubPolygons()

```
list<vector<Vertex> > Graph::getSubPolygons ( )  [inline]
```

Get the Sub Polygons of the Graph object.

This function returns all the sub polygons which resulted after drawing diagonals excluding the original polygon itself. In case no diagonals have been drawn, it returns an empty list.

Returns

list<vector<Vertex>> List of sub polygons (cloclwise ordering of points in each polygon)

4.3.3.3 next()

```
Vertex Graph::next (
             Vertex & v )  [inline]
```

The next vertex to the given vertex in the clockwise ordering.

Parameters

v	Vertex to which we need next vertex

Returns

Vertex The next Vertex to the given vertex

4.3.3.4 prev()

```
Vertex Graph::prev (
             Vertex & v )  [inline]
```

The previous vertex to the given vertex in the clockwise ordering.

Parameters

v	Vertex to which we need previous vertex

Returns

Vertex The previous Vertex to the given vertex

The documentation for this class was generated from the following file:

- monotoneSubdivision.h

4.4 SweepStatus Class Reference

This class represents the sweep line status for the plane sweep algorithm.
`#include <monotoneSubdivision.h>`

Public Member Functions

- SweepStatus ()

    Construct a new Sweep Line Status object.
- Vertex helper (Edge &e)

    get the helper Vertex of the given Edge
- void setHelper (Edge e, Vertex v)

    Set the helper Vertex for given Edge.
- void insert (const Edge &e, const Vertex &helper)

    Insert a new Edge and its helper into the Sweep Line Status.
- Edge getUpper (const Vertex &v)

    Get the Edge which is immediately above the given Vertex in the polygon.
- void remove (const Edge &e)

    Remove the given Edge from the Sweep Line Status.

4.4.1 Detailed Description

This class represents the sweep line status for the plane sweep algorithm.
This class stores the edges which are intersecting with the plane at the current position. the Edges are ordered based on increasing y coordinate. Provides methods to insert, delete and fetch the intersecting edges

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SweepStatus()

```
SweepStatus::SweepStatus ( )  [inline]
```
Construct a new Sweep Line Status object.

4.4.3 Member Function Documentation

4.4.3.1 getUpper()

```
Edge SweepStatus::getUpper (
            const Vertex & v )  [inline]
```
Get the Edge which is immediately above the given Vertex in the polygon.

Parameters

v	Vertex whose upper Edge is required

Returns

Edge The edge immediately above Vertex v

4.4.3.2 helper()

```
Vertex SweepStatus::helper (
            Edge & e )   [inline]
```

get the helper Vertex of the given Edge

Parameters

e	Edge to which we need the helper for

Returns

Vertex The helper vertex of the given edge: helper(e)

4.4.3.3 insert()

```
void SweepStatus::insert (
            const Edge & e,
            const Vertex & helper )   [inline]
```

Insert a new Edge and its helper into the Sweep Line Status.

Parameters

e	Edge to insert into status
helper	helper Vertex of given Edge.

4.4.3.4 remove()

```
void SweepStatus::remove (
            const Edge & e )   [inline]
```

Remove the given Edge from the Sweep Line Status.
This operation can be done preferably when the sweep line/plane is no longer intersecting with the Edge.

Parameters

e	Edge to be removed from the status

4.4.3.5 setHelper()

```
void SweepStatus::setHelper (
            Edge e,
            Vertex v )   [inline]
```

Set the helper Vertex for given Edge.
setHelper sets helper(e) = v.

Parameters

e	Edge for which helper must be assigned
v	Vertex to be assigned as helper

The documentation for this class was generated from the following file:

- monotoneSubdivision.h

4.5 Triangle Class Reference

This object is to represent the triangle object with Vertices.

```
#include <Geometry.h>
```

Public Member Functions

- Triangle (vector< Vertex > &vertices)

  Construct a new Triangle object.
- Triangle (Vertex &a, Vertex &b, Vertex &c)

  Construct a new Triangle object.

Public Attributes

- vector< Vertex > **v**

4.5.1 Detailed Description

This object is to represent the triangle object with Vertices.
This class is just to hold the 3 vertices of the triangle. The purpose of this class is to provide abstraction in code to convey that we are dealing with triangles and not just list of vertices

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Triangle() [1/2]

```
Triangle::Triangle (
             vector< Vertex > & vertices )  [inline]
```

Construct a new Triangle object.

Parameters

vertices	List of vertices which constitute this triangle

4.5.2.2 Triangle() [2/2]

```
Triangle::Triangle (
             Vertex & a,
             Vertex & b,
             Vertex & c )  [inline]
```

Construct a new Triangle object.

Parameters

a	First Vertex
b	Second Vertex
c	Third Vertex

The documentation for this class was generated from the following file:

- Geometry.h

4.6 Vertex Class Reference

Basic geometric Vertex class to store coordinates of vertex.

```
#include <Geometry.h>
```

Public Member Functions

- Vertex ()

    Construct a new empty Vertex object.
- Vertex (float X, float Y)

    Construct a new Vertex object with the given coordinates.
- Vertex (float X, float Y, unsigned long Tag)

    Construct a new Vertex object with the given coordinates and extra tag.
- Vertex (const Vertex &p)

    Construct a new Vertex object from another Vertex object.
- bool operator== (const Vertex &v)

    To overload == operator on Vertex type objects.
- bool operator!= (const Vertex &v)

    To overload != operator on Vertex type objects.
- bool operator< (const Vertex &v) const

    To overload < operator on Vertex type objects.
- Vertex operator- (const Vertex &v)

    To overlaod - operator on Vertex type objects.

Public Attributes

- float **x**
- float **y**
- unsigned long **tag**
- unsigned long **parent**

4.6.1 Detailed Description

Basic geometric Vertex class to store coordinates of vertex.
This class stores the (x, y) coordinate of a vertex as taken from the user input.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 Vertex() [1/4]

```
Vertex::Vertex ( )  [inline]
```

Construct a new empty Vertex object.

4.6.2.2 Vertex() [2/4]

```
Vertex::Vertex (
              float X,
              float Y )  [inline]
```

Construct a new Vertex object with the given coordinates.
A vertex type object will be constructed with the given coordinates.

Parameters

X	x coordinate of the input vertex
Y	y coordinate of the input vertex

4.6.2.3 Vertex() [3/4]

```
Vertex::Vertex (
             float X,
             float Y,
             unsigned long Tag )  [inline]
```

Construct a new Vertex object with the given coordinates and extra tag.
A vertex type object will be constructed with the given coordinates and tag.

Parameters

X	x coordinate of the input vertex
Y	y coordinate of the input vertex
Tag	integer tag to be assigned to this vertex

4.6.2.4 Vertex() [4/4]

```
Vertex::Vertex (
             const Vertex & p )  [inline]
```

Construct a new Vertex object from another Vertex object.
This is effectively a copy constructor for the new Vertex object

Parameters

p	Vertex type object whose all parameters will be copied

4.6.3 Member Function Documentation

4.6.3.1 operator"!=()

```
bool Vertex::operator!= (
             const Vertex & v )  [inline]
```

To overload != operator on Vertex type objects.
This returns NOT of the == overloaded operator

Parameters

v	Vertex object to be compared with

Returns

true if at least one coordinate differs

false if both coordinates match

4.6.3.2 operator-()

```
Vertex Vertex::operator- (
             const Vertex & v )  [inline]
```

To overlaod - operator on Vertex type objects.

This can be mainly used for applications like shifting the point with reference to other point. Another application is subtract operation when the Vertex class is used to represent vectors(since even vectors attributes are similar to points)

Parameters

v	Vertex object to be subtracted

Returns

Vertex Vertex with the final values after subtraction

4.6.3.3 operator<()

```
bool Vertex::operator< (
             const Vertex & v ) const  [inline]
```

To overload $<$ operator on Vertex type objects.
Vertex a is less than Vertex b ($a < b$) if x coordinate of a is lesser than b. In case x coordinate is equal, their y coordinates are compared.

Parameters

v	Vertex object to be compared with

Returns

true if current x coordinate is less than the other (y is used if x is same)

false otherwise

4.6.3.4 operator==()

```
bool Vertex::operator== (
             const Vertex & v )  [inline]
```

To overload == operator on Vertex type objects.
This returns true only if both x and y coordinates are equal between the objects being compared.

Parameters

v	Vertex object to be compared with

Returns

true if both coordinates match

false if at least one coordinate differs

The documentation for this class was generated from the following file:

- Geometry.h

Chapter 5

File Documentation

5.1 earClippingTriangulation.h File Reference

This file contains the implementation of the ear clipping triangulation algorithm.

```
#include <vector>
#include "Geometry.h"
#include "Tools.h"
```

Functions

- vector< Triangle > earClippingTriangulate (vector< Vertex > points)

  This function implements the basic ear clipping triangulation.

5.1.1 Detailed Description

This file contains the implementation of the ear clipping triangulation algorithm.

Author

Rikil Gajarla (`f20170202@hyderabad.bits-pilani.ac.in`)

5.1.2 Function Documentation

5.1.2.1 earClippingTriangulate()

```
vector<Triangle> earClippingTriangulate (
            vector< Vertex > points )
```

This function implements the basic ear clipping triangulation.

The main approach behind this algorithm is that we try to find "ears" which are basically the vertices of the polygon whose interior angle is less than PI. We then triangulate these ears and remove them from the Polygon. we perform the same operation on the polygon until it is completely triangulated

Parameters

points	The input vertices of polygon in clockwise order

Returns

vector<Triangle> List of Triangle objects which are the triangulations of the input polygon

5.2 Geometry.h File Reference

This file contains the implementation of the basic geometric objects such as Vertex, Edge, Face.

```
#include <iostream>
#include <vector>
```

Classes

- class Vertex

  Basic geometric Vertex class to store coordinates of vertex.
- class Edge

  Basic geometric Edge class to store endpoints of edges.
- class Face

  Basic geometric Face class to store a Face structure.
- class Triangle

  This object is to represent the triangle object with Vertices.

Functions

- ostream & **operator**<< (ostream &os, const Vertex &v)
- ostream & **operator**<< (ostream &os, const Edge &e)
- ostream & **operator**<< (ostream &os, const Face &f)
- ostream & **operator**<< (ostream &os, const Triangle &t)

Variables

- const float **epsillion** = 0.001f

5.2.1 Detailed Description

This file contains the implementation of the basic geometric objects such as Vertex, Edge, Face.

Author

Rikil Gajarla (`f20170202@hyderabad.bits-pilani.ac.in`)

5.3 main.cpp File Reference

The main runner program to take input, triangulate and provide output.

```
#include <iostream>
#include <vector>
#include <chrono>
#include "Geometry.h"
#include "Tools.h"
#include "earClippingTriangulation.h"
#include "planeSweepTriangulation.h"
```

Functions

- int main (int argc, char *argv[ ])

  main function which starts the execution of the triangulation program

5.3.1 Detailed Description

The main runner program to take input, triangulate and provide output.

Author

Rikil Gajarla (`f20170202@hyderabad.bits-pilani.ac.in`)

5.3.2 Function Documentation

5.3.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

main function which starts the execution of the triangulation program

Parameters

argc	No of command line arguments proveided by the os
argv	Command line arguments given by the user, supplied by the os

Returns

int 0 if program exits successfully

5.4 monotoneSubdivision.h File Reference

This file contains the algorithm to subdivide the polygon into monotones.

```
#include <list>
#include <vector>
#include <map>
#include <algorithm>
#include "Geometry.h"
#include "Tools.h"
```

Classes

- class Graph

    This class stores the graph representation of the input polygon.
- class SweepStatus

    This class represents the sweep line status for the plane sweep algorithm.

Functions

- void fixup (Vertex &v, Edge &e, Graph &graph, SweepStatus &status)

    try to fix and connect the Vertex v to the helper of Edge e
- void **checkSort** (vector< Vertex > &events, vector< Vertex > &original)
- list< vector< Vertex > > getMonotones (vector< Vertex > input)

    Returns the monotones present in the input polygon.

5.4.1 Detailed Description

This file contains the algorithm to subdivide the polygon into monotones.

Author

Rikil Gajarla (f20170202@hyderabad.bits-pilani.ac.in)

5.4.2 Function Documentation

5.4.2.1 fixup()

```
void fixup (
            Vertex & v,
            Edge & e,
            Graph & graph,
            SweepStatus & status )
```

try to fix and connect the Vertex v to the helper of Edge e
This method only connects v to helper(e) only if v is a merge vertex

Parameters

v	Vertex who needs to be connected to a helper
e	The upper Edge whose helper us used to draw diagonal
graph	The current Graph object on which the operations are performed
status	The current SweepStatus object which contains all the intersecting edges

5.4.2.2 getMonotones()

```
list<vector<Vertex> > getMonotones (
            vector< Vertex > input )
```

Returns the monotones present in the input polygon.
This is the main function which performs plane sweep algorithm to detect and return the monotones present in the polygon after adding required diagonals

Parameters

input	Clockwise ordering of vertices of input polygon

Returns

list<vector<Vertex>> List of polygons which are monotone (vertices of each monotone are clockwise ordered)

5.5 monotoneTriangulation.h File Reference

This file contains the method to triangulate a monotone polygon.

```
#include <list>
#include <vector>
#include <algorithm>
#include "Geometry.h"
#include "Tools.h"
```

Functions

- vector< Vertex > getTopChain (vector< Vertex > &sorted, vector< Vertex > &original)

Get the Top Chain of the monotone polygon.

- vector< Vertex > getBottomChain (vector< Vertex > &sorted, vector< Vertex > &original)

  Get the Bottom Chain of the monotone polygon.

- vector< Vertex > preprocess (vector< Vertex > input, Vertex &left, Vertex &right)

  Assign top or bottom to each vertex and sort them w.r.t x-axis.

- bool validTriangle (Vertex &a, Vertex &b, Vertex &c, bool midSide)

  Check if the triangle formed by the vertices is valid.

- Vertex findMin (const vector< Vertex > &input)

  Get the minimum Vertex (based on the x-axis)

- Vertex findMax (const vector< Vertex > &input)

  Get the maximum Vertex (based on the x-axis)

- vector< Triangle > monotoneTriangulate (vector< Vertex > input)

  This method performs the triangulation of the given monotone.

5.5.1 Detailed Description

This file contains the method to triangulate a monotone polygon.

Author

Rikil Gajarla (`f20170202@hyderabad.bits-pilani.ac.in`)

5.5.2 Function Documentation

5.5.2.1 findMax()

```
Vertex findMax (
            const vector< Vertex > & input )
```

Get the maximum Vertex (based on the x-axis)

Parameters

input	List of vertices of polygon

Returns

Vertex The maximum vertex

5.5.2.2 findMin()

```
Vertex findMin (
            const vector< Vertex > & input )
```

Get the minimum Vertex (based on the x-axis)

Parameters

input	List of vertices of polygon

Returns

Vertex The minimum vertex

5.5.2.3 getBottomChain()

```
vector<Vertex> getBottomChain (
             vector< Vertex > & sorted,
             vector< Vertex > & original )
```

Get the Bottom Chain of the monotone polygon.

Parameters

sorted	Vertices of polygon sorted w.r.t x-axis
original	unmodified clockwise ordering of vertices of polygon as given by the user

Returns

vector<Vertex> List of vertices present on the bottom chain of monotone

5.5.2.4 getTopChain()

```
vector<Vertex> getTopChain (
             vector< Vertex > & sorted,
             vector< Vertex > & original )
```

Get the Top Chain of the monotone polygon.

Parameters

sorted	Vertices of polygon sorted w.r.t x-axis
original	unmodified clockwise ordering of vertices of polygon as given by the user

Returns

vector<Vertex> List of vertices present on the top chain of monotone

5.5.2.5 monotoneTriangulate()

```
vector<Triangle> monotoneTriangulate (
             vector< Vertex > input )
```

This method performs the triangulation of the given monotone.

Parameters

input	List of vertices present on the monotone in clockwise order

Returns

vector<Triangle> List of Triangle objects produced after triangulating the monotone

5.5.2.6 preprocess()

```
vector<Vertex> preprocess (
             vector< Vertex > input,
             Vertex & left,
             Vertex & right )
```

Assign top or bottom to each vertex and sort them w.r.t x-axis.

Parameters

input	List of vertices representing the clockwise ordering of polygon vertices
left	The left most Vertex present on the polygon
right	The right most Vertex present on the polygon

Returns

vector<Vertex> x-axis soreted list of Vertices of the polygon

5.5.2.7 validTriangle()

```
bool validTriangle (
            Vertex & a,
            Vertex & b,
            Vertex & c,
            bool midSide )
```

Check if the triangle formed by the vertices is valid.
A triangle is said to be valid only if it completely exists inside the polygon

Parameters

a	First Vertex
b	Second Vertex
c	Third Vertex
midSide	Side of the middle vertex (top or bottom as set in the tag of Vertex)

Returns

true If the triangle formed is valid

false otherwise

5.6 planeSweepTriangulation.h File Reference

This file provides the wrapper function to monotone triangulate a polygon.

```
#include <list>
#include <vector>
#include "Geometry.h"
#include "monotoneSubdivision.h"
#include "monotoneTriangulation.h"
```

Functions

- vector< Triangle > planeSweepTriangulate (vector< Vertex > input)

  The wrapper method to monotone triangulate a polygon.

5.6.1 Detailed Description

This file provides the wrapper function to monotone triangulate a polygon.

Author

Rikil Gajarla (`f20170202@hyderabad.bits-pilani.ac.in`)

5.6.2 Function Documentation

5.6.2.1 planeSweepTriangulate()

```
vector<Triangle> planeSweepTriangulate (
            vector< Vertex > input )
```

The wrapper method to monotone triangulate a polygon.
This function internally calls getMonotones() and monotoneTriangulate() functions to get the triangulations

Parameters

input	List of Vertices of Polygons present in clockwise order

Returns

vector<Triangle> List of Triangle objects representing the triangulation of polygon

5.7 Tools.h File Reference

The main purpose of this file is to provide a data input function.

```
#include <cmath>
#include <vector>
#include <fstream>
#include <sstream>
#include "Geometry.h"
```

Functions

- double det (const Vertex &u, const Vertex &v)

    Compute the determinant of the given triangle or the cross product to the given vectors.
- double dist (Vertex const &u, Vertex const &v)

    Compute the distance between given points.
- int orient (const Vertex &p, const Vertex &q, const Vertex &r)

    To check the orientation of the given 3 points.
- bool pointInTriangle (Vertex a, Vertex b, Vertex c, Vertex p)

    Check if a point is present in the triangle.
- void **log** (string s)
- vector< Vertex > readvertices (int argc, char *argv[ ])

    Read input vertices from stdin or from file (if provided in args)

5.7.1 Detailed Description

The main purpose of this file is to provide a data input function.
This file provides the readPoints() function which helps to read file and take input from it. In case any file is not provided, It tries to take input from the user via stdin. some other functions like det(), dist(), and other triangle related helper functions are also present

5.7.2 Function Documentation

5.7.2.1 det()

```
double det (
            const Vertex & u,
            const Vertex & v )
```

Compute the determinant of the given triangle or the cross product to the given vectors.

Parameters

u	First Vertex type object
v	Second Vertex type object

Returns

double determinant or cross product

5.7.2.2 dist()

```
double dist (
            Vertex const & u,
            Vertex const & v )
```

Compute the distance between given points.

Parameters

u	First Vertex
v	Second Vertex

Returns

double distance between the Vertex objects

5.7.2.3 orient()

```
int orient (
            const Vertex & p,
            const Vertex & q,
            const Vertex & r )
```

To check the orientation of the given 3 points.

Parameters

p	Point object
q	Point object
r	Point object

Returns

Orientation: -1 if clockwise, +1 if counter-clockwise and 0 if colinear

5.7.2.4 pointInTriangle()

```
bool pointInTriangle (
             Vertex a,
             Vertex b,
             Vertex c,
             Vertex p )
```

Check if a point is present in the triangle.

Parameters

a	First Vertex of the triangle
b	Second Vertex of the triangle
c	Third Vertex of the triangle
p	Vertex/Point to be checked if it lies inside the triangle

Returns

true if the point lies inside the triangle

false if the point does not lie inside the triangle

5.7.2.5 readvertices()

```
vector<Vertex> readvertices (
             int argc,
             char * argv[ ] )
```

Read input vertices from stdin or from file (if provided in args)

Parameters

argc	no of command-line arguments
argv	command-line arguments

Returns

vector<Vertex> object containing the input points in the given order

Index