Individual Project 2
Constructing Minimum Spanning Trees
Software Design Document

Zhang Huimeng, 2015011280

February 3, 2021

Contents

I Introduction 4

1 Purpose 5

2 Scope 5

II System Overview 6

III System Architecture 8

3 Architectural Design 9

4 Decomposition Description 10
4.1 Basics . . . 10
4.2 Computational . . . 11
4.3 Display . . . 12

IV Data Design 14

5 Data Description 15

V Human Interface Design 16

6 Overview of Human Interface 17
6.1 Main Menu . . . 18
6.2 'Generate' sub-Menu . . . 18
6.3 'Show' sub-Menu . . . 19
6.4 'Run Test' sub-Menu . . . 19

7 Screen Images 20

VI Design Patterns 22

8 Singleton 23

VII Component Design 24

9 Namespace Index 25
9.1 Namespace List . . . 25

10 Hierarchical Index **25**
10.1 Class Hierarchy . . . 25

11 Class Index **26**
11.1 Class List . . . 26

12 Namespace Documentation **26**
12.1 cmst Namespace Reference . . . 26
12.1.1 Enumeration Type Documentation . . . 27
12.1.2 Function Documentation . . . 28

13 Class Documentation **28**
13.1 cmst::Edge2D Class Reference . . . 28
13.1.1 Detailed Description . . . 29
13.1.2 Constructor & Destructor Documentation . . . 29
13.1.3 Member Function Documentation . . . 29
13.1.4 Friends And Related Function Documentation . . . 30
13.1.5 Member Data Documentation . . . 30
13.2 cmst::Graph2D Class Reference . . . 30
13.2.1 Constructor & Destructor Documentation . . . 32
13.2.2 Member Function Documentation . . . 32
13.2.3 Member Data Documentation . . . 34
13.3 cmst::IndexEdge2D Class Reference . . . 35
13.3.1 Detailed Description . . . 36
13.3.2 Constructor & Destructor Documentation . . . 36
13.3.3 Member Function Documentation . . . 36
13.3.4 Friends And Related Function Documentation . . . 36
13.3.5 Member Data Documentation . . . 36
13.4 cmst::Point2D Class Reference . . . 37
13.4.1 Detailed Description . . . 37
13.4.2 Constructor & Destructor Documentation . . . 37
13.4.3 Member Function Documentation . . . 37
13.4.4 Friends And Related Function Documentation . . . 38
13.4.5 Member Data Documentation . . . 38
13.5 cmst::Graph2D::ST Struct Reference . . . 38
13.5.1 Detailed Description . . . 38
13.5.2 Constructor & Destructor Documentation . . . 39
13.5.3 Member Data Documentation . . . 39
13.6 cmst::Stat Class Reference . . . 39
13.6.1 Detailed Description . . . 40
13.6.2 Constructor & Destructor Documentation . . . 40
13.6.3 Member Function Documentation . . . 40
13.6.4 Member Data Documentation . . . 41
13.7 cmst::Window::Test Struct Reference . . . 42
13.7.1 Detailed Description . . . 42
13.7.2 Constructor & Destructor Documentation . . . 42
13.7.3 Member Data Documentation . . . 43
13.8 cmst::Timer Class Reference . . . 43
13.8.1 Detailed Description . . . 43
13.8.2 Constructor & Destructor Documentation . . . 43
13.8.3 Member Function Documentation . . . 44
13.8.4 Member Data Documentation . . . 44
13.9 cmst::Window Class Reference . . . 44
13.9.1 Detailed Description . . . 46

13.9.2 Constructor & Destructor Documentation 46
13.9.3 Member Function Documentation 46
13.9.4 Member Data Documentation 49

Part I

Introduction

Chapter 1

Purpose

This software design document describes the architecture and system design of project ComputeMST. This is a homework of Fundamentals of Object-Oriented Programming class, Spring 2016. The homework requirement is as follows:

(II) Project: On Constructing Minimum Spanning Trees (Difficulty: 1.1)

Requirements and TIPS:

(1) Implement the MST algorithm based on Voronoi diagram for computing Euclidean minimum spanning trees in 2D plane.

(2) Please refer to the following link for MST construction algorithms. Either Prim's or Kruskal's algorithm is ok. Please refer to Page 39 for the idea of using Voronoi for MST computation.

(3) Use the CGAL Library for constructing the Voronoi diagram and Delaunay triangulation.

(4) Randomly generate 5 different testcases with more than 5000 points without duplicates to test the implemented method.

(5) It is suggested that a validity checking function be implemented to verify the experimental results are correct. For example, you may directly apply Prim's or Kruskal's MST algorithm on the testcase to verify that the MST trees are correct.

(6) Report the statistics of the experimental results, e.g., total runtime, total number of points, total length of the MST edges, etc. Figures and tables on the experimental results are welcome.

(7) [This is not mandatory to finish, but it is a challenging topic] Again, can you compute the top K ($1 <= K <= 20$) minimum spanning trees?

Chapter 2

Scope

I implemented the first six requirements of the homework, mainly using CGAL and Kruskal's MST algorithm. The program can load input from file or randomly generate input for itself; it has a GUI interface and can print results of the MST computation to file.

Part II

System Overview

The program can compute Delaunay triangulation (via CGAL library) and MST for up to 10000 points in a 2-dimensional plane in 1 or 2 seconds. It can output the result to file. Also, it has a interface written with Freeglut.

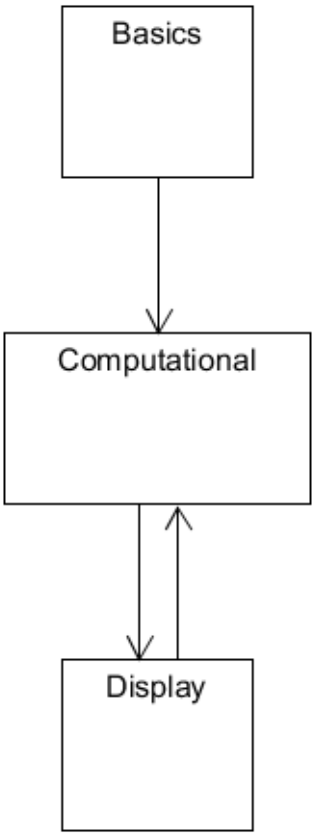
Part III

System Architecture

Chapter 3

Architectural Design

The program can be divided into three parts: the Basics part, the Computational part and the Display part.



The basics part provides some useful tools for the Computational part, like statistics and timer.

The computational part does the computation, and interacts with the Display part to provide a workable interface.

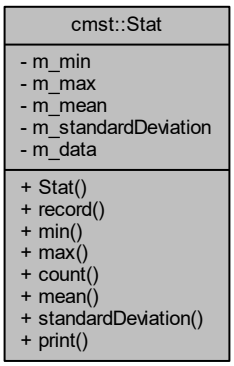
The display part deals with the interface and user input.

Chapter 4

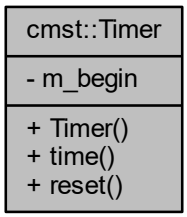
Decomposition Description

4.1 Basics

The Stat class deals with statistics. It receives data in the form of floating-point number and can work continuously. It calculates the mean, maximum, minimum and standard deviation of the data.

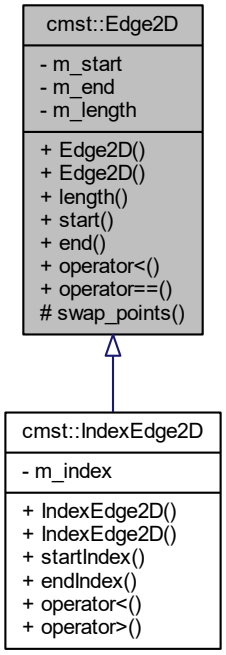


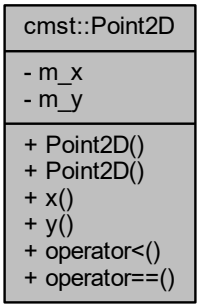
The Timer class deals with timing. It uses the clock() function.



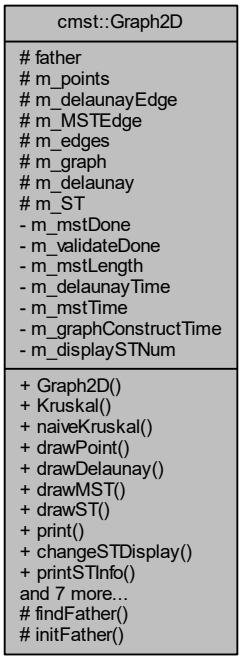
4.2 Computational

The Point2D, Edge2D and IndexEdge2D classes are the basic 2-dimensional computational geometry classes.





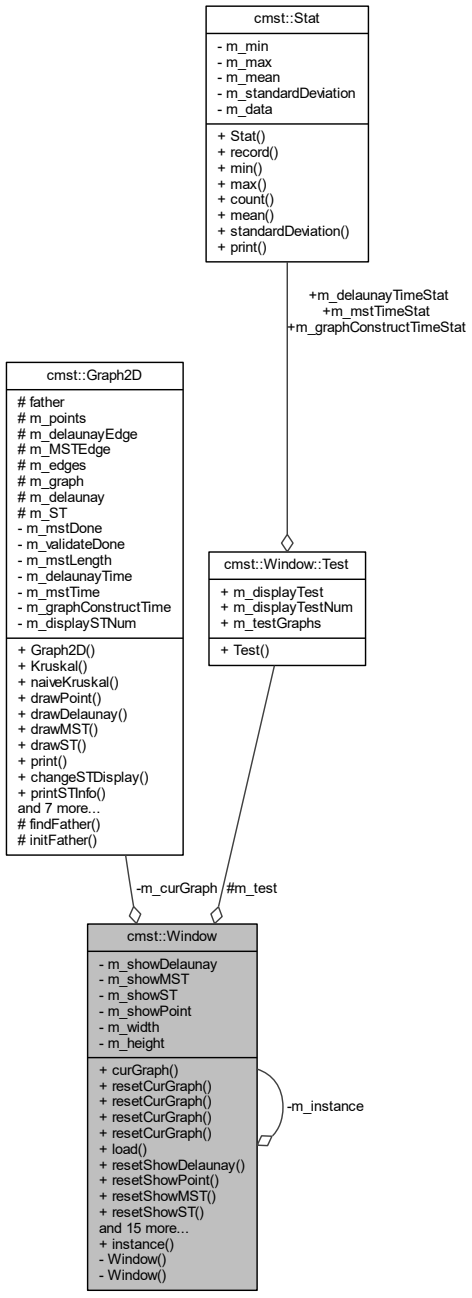
The Graph2D class deals with all the computations. The Delaunay triangulation and MST are computed in the constructor. This class also provides drawing functions that draw the graph by Freeglut.



It also contains a nested struct, Graph2D::ST that stores spanning trees of the graph. If I had time, I would have implemented the k-top spanning tree calculation.

4.3 Display

The Display part contains the Freeglut window and pop-up menu management functions, as well as the Window class that manages the window. Here is the UML diagram of this class:


cmst::Stat
- m_min
- m_max
- m_mean
- m_standardDeviation
- m_data
+ Stat()
+ record()
+ min()
+ max()
+ count()
+ mean()
+ standardDeviation()
+ print()
+m_delaunayTimeStat
+m_mstTimeStat
+m_graphConstructTimeStat
cmst::Graph2D
father
m_points
m_delaunayEdge
m_MSTEdge
m_edges
m_graph
m_delaunay
m_ST
- m_mstDone
- m_validateDone
- m_mstLength
- m_delaunayTime
- m_mstTime
- m_graphConstructTime
- m_displaySTNum
+ Graph2D()
+ Kruskal()
+ naiveKruskal()
+ drawPoint()
+ drawDelaunay()
+ drawMST()
+ drawST()
+ print()
+ changeSTDisplay()
+ printSTInfo()
and 7 more...
findFather()
initFather()
cmst::Window::Test
+ m_displayTest
+ m_displayTestNum
+ m_testGraphs
+ Test()
-m_curGraph
#m_test
cmst::Window
- m_showDelaunay
- m_showMST
- m_showST
- m_showPoint
- m_width
- m_height
+ curGraph()
+ resetCurGraph()
+ resetCurGraph()
+ resetCurGraph()
+ resetCurGraph()
+ load()
+ resetShowDelaunay()
+ resetShowPoint()
+ resetShowMST()
+ resetShowST()
and 15 more...
+ instance()
- Window()
- Window()
-m_instance

Part IV

Data Design

Chapter 5

Data Description

The program uses very simple ways of managing data: most data is stored in vectors and arrays.

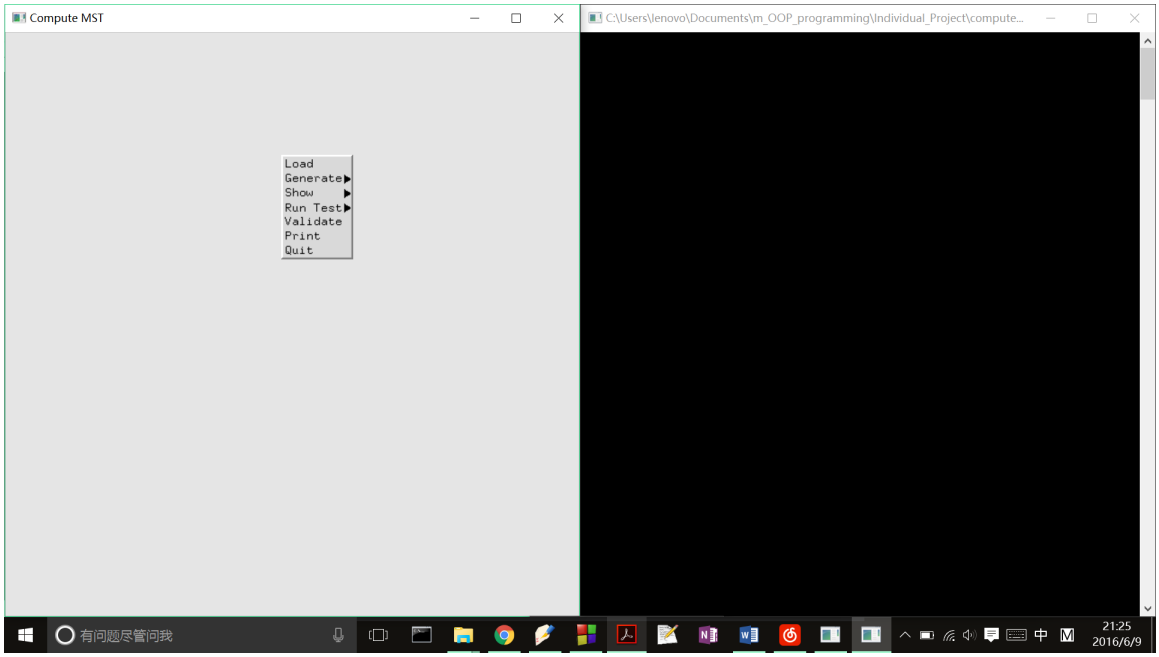
Part V

Human Interface Design

Chapter 6

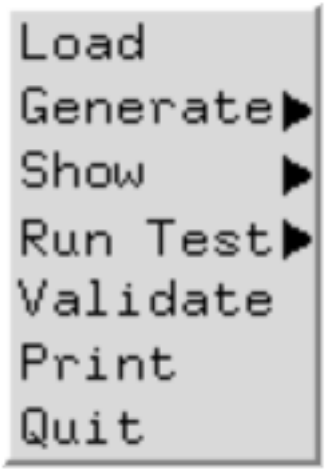
Overview of Human Interface

Here is a screen shot of the main screen:



On the left is the Freeglut window, and the console is on the right. If you right-click on the glut window, a pop-up window will appear, as in the screenshot above. The console prints information about graphs and MSTs being displayed in the window.

6.1 Main Menu



As can be seen from the screenshot above, main menu has 7 menu entries:

Load This option loads point information from a file to construct the graph. There are some requirements of the input. You have to provide the full path to the input file.

Generate See 6.2.

Show See 6.3.

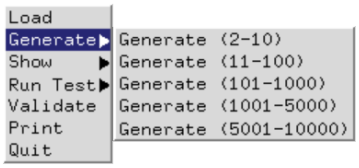
Run Test See 6.4.

Validate Run the validating naive MST algorithm for the graph.

Print Print information to graph.txt.

Quit Exit the program.

6.2 'Generate' sub-Menu



This sub-menu has 5 menu entries:

Generate(2-10) Generate 2-10 points.

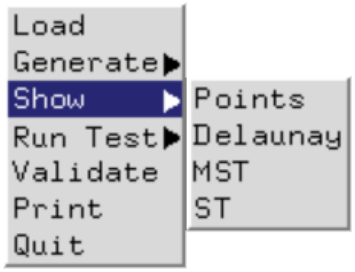
Generate(11-100) Generate 11-100 points.

Generate(101-1000) Generate 101-1000 points.

Generate(1001-5000) Generate 1001-5000 points.

Generate(5001-10000) Generate 5001-10000 points.

6.3 'Show' sub-Menu



This sub-menu has 4 menu entries:

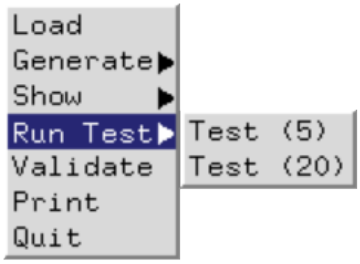
Points Whether the points are drawn.

Delaunay Whether the Delaunay triangulation is drawn.

MST Whether the MST is draw.

ST Whether the STs are drawn. These include the MST calculated in validating and other spanning trees.

6.4 'Run Test' sub-Menu



This sub-menu has 2 menu entries:

Test(5) Run a test of 5 graphs, each having 10000 points.

Test(20) Run a test of 20 graphs, each having 10000 points.

After a test is run, you can use left and right arrow keys to move through test graphs.

Chapter 7

Screen Images

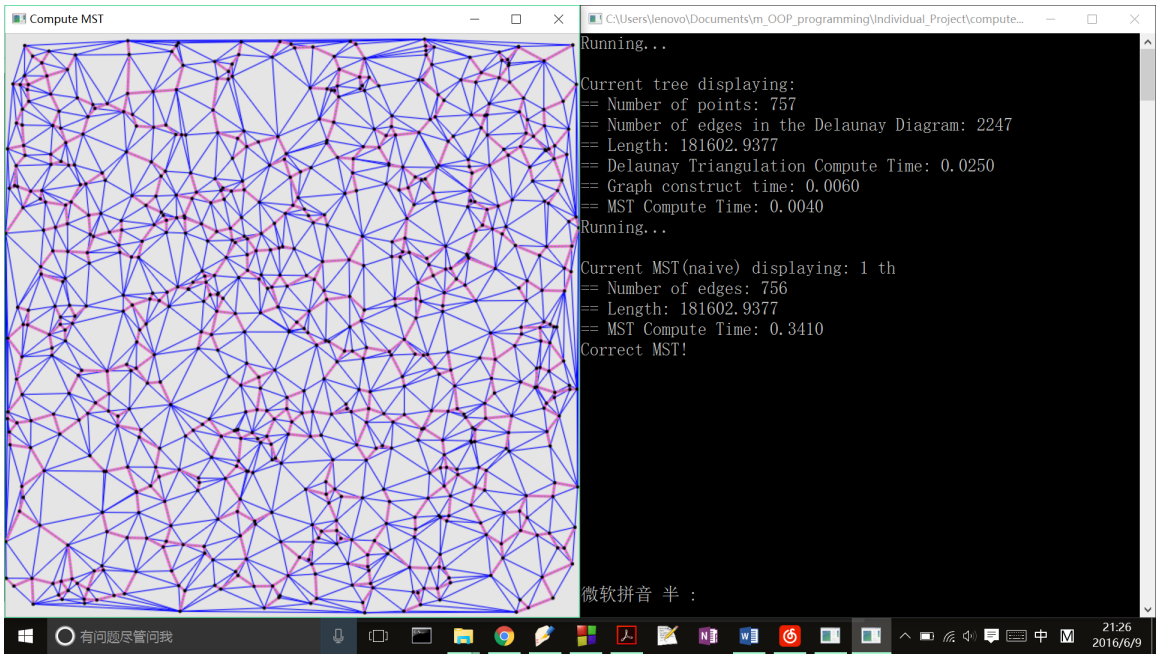


Figure 7.1 After a graph is generated

Delaunay triangulation is drawn in thin blue lines, and MST is drawn in thick pink lines. Spanning trees are drawn in dotted lines.

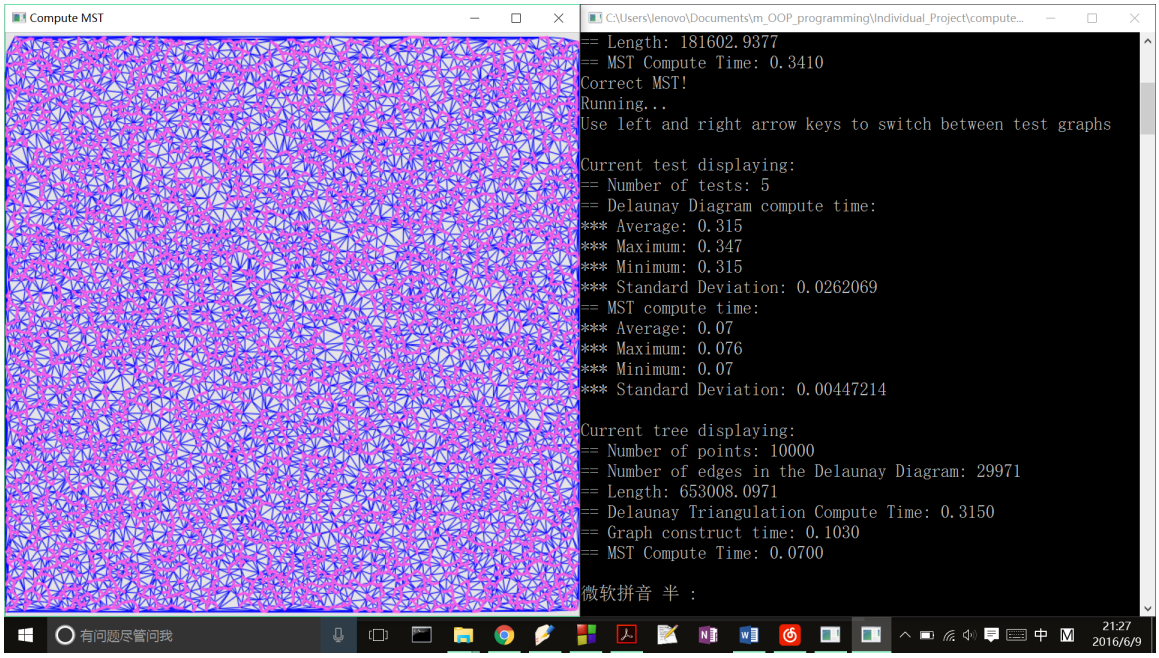


Figure 7.2 After running a test

In the above screenshot, the points and spanning trees are not drawn.

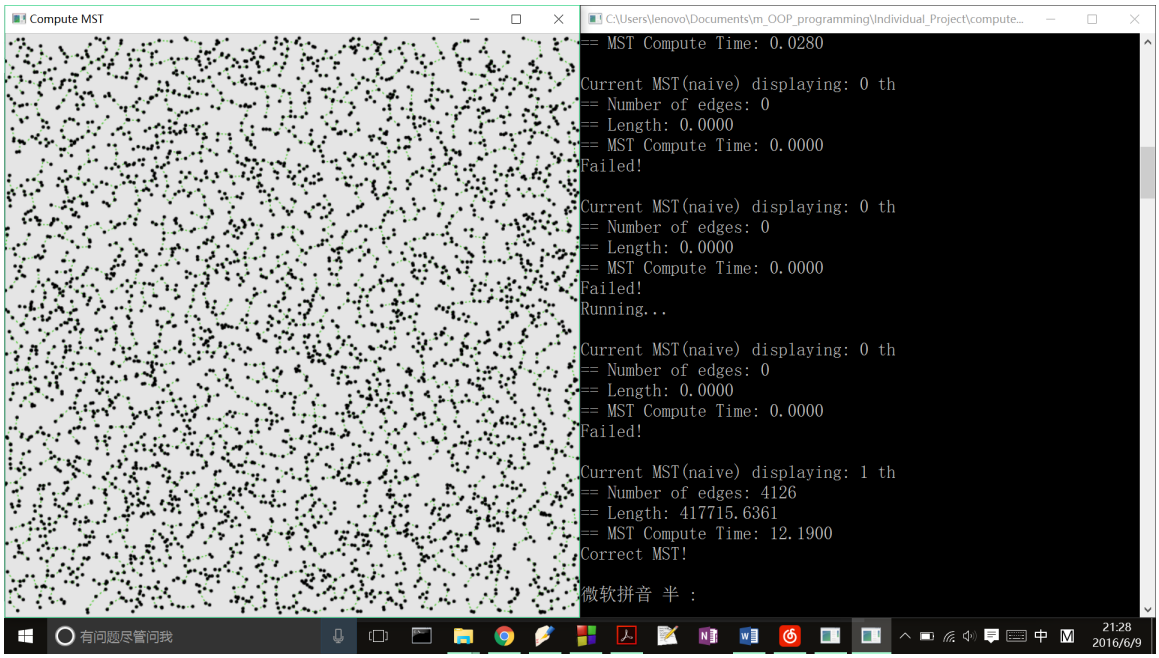


Figure 7.3 After running a validate

In the above screenshot, the Delaunay triangulation and MST are not drawn.

Part VI

Design Patterns

Chapter 8

Singleton

This program uses Singleton pattern in Window. There can only be one main window in one program.

Part VII

Component Design

Chapter 9

Namespace Index

9.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cmst 26

Chapter 10

Hierarchical Index

10.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

cmst::Edge2D 28
cmst::IndexEdge2D 35
cmst::Graph2D 30
cmst::Point2D 37
cmst::Graph2D::ST 38
cmst::Stat 39
cmst::Window::Test 42
cmst::Timer 43
cmst::Window 44

Chapter 11

Class Index

11.1 Class List

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

cmst::Edge2D 28
cmst::Graph2D 30
cmst::IndexEdge2D
    Edge with start and end point indices in an array 35
cmst::Point2D
    Points in a 2D plane 37
cmst::Graph2D::ST
    Store a spanning tree of the graph 38
cmst::Stat 39
cmst::Window::Test 42
cmst::Timer 43
cmst::Window 44

Chapter 12

Namespace Documentation

12.1 cmst Namespace Reference

Classes

- class Edge2D
- class Graph2D
- class IndexEdge2D

  Edge with start and end point indices in an array.
- class Point2D

Points in a 2D plane.

- class Stat
- class Timer
- class Window

Enumerations

- enum Menu {
  LOAD, NEW, NEW_4_10, NEW_11_100,
  NEW_101_1000, NEW_1001_5000, NEW_5001_10000, SHOW,
  SHOW_VORONOI, SHOW_DELAUNAY, SHOW_POINT, SHOW_MST,
  SHOW_ST, TEST, TEST_5, TEST_20,
  VALIDATOR, PRINT, QUIT }

  Return values for GLUT menus.

Functions

- int randomInt (int a, int b)
- double randomDouble (double a, double b)
- std::vector< Point2D > TestcaseGenerator (int num_lower_bound=100, int num_upper_bound=500, double x_upper_bound=MAX_X, double y_upper_bound=MAX_Y)

12.1.1 Enumeration Type Documentation

enum cmst::Menu

Return values for GLUT menus.

Enumerator

LOAD

NEW

NEW_4_10

NEW_11_100

NEW_101_1000

NEW_1001_5000

NEW_5001_10000

SHOW

SHOW_VORONOI

SHOW_DELAUNAY

SHOW_POINT

SHOW_MST

SHOW_ST

TEST

TEST_5

TEST_20

VALIDATOR

PRINT

QUIT

12.1.2 Function Documentation

double cmst::randomDouble (double *a*, double *b*)

Generate a floating-point number in the range [a, b]
Needs to be improved using other random classes
Here is the caller graph for this function:

int cmst::randomInt (int *a*, int *b*)

Generate an integer in the range [a, b]
Needs to be improved using other random classes
Here is the caller graph for this function:

std::vector< cmst::Point2D > cmst::TestcaseGenerator (int *num_lower_bound* = `100`, int *num_upper_bound* = `500`, double *x_upper_bound* = `MAX_X`, double *y_upper_bound* = `MAX_Y`)

Generate some random points.
The number of points is generated randomly in the range [num_lower_bound, num_upper_bound], and the x, y coordinates of the points are respectively in the range [0, x_upper_bound] and [0, y_upper_bound].
Here is the call graph for this function:
Here is the caller graph for this function:

Chapter 13

Class Documentation

13.1 cmst::Edge2D Class Reference

Inheritance diagram for cmst::Edge2D:
Collaboration diagram for cmst::Edge2D:

Public Member Functions

- Edge2D ()
- Edge2D (const Point2D &start, const Point2D &end)
- double length () const
- Point2D start () const
- Point2D end () const
- bool operator< (const Edge2D &right) const

  Compares edges by length.
- bool operator== (const Edge2D &right) const

Protected Member Functions

- void swap_points ()

  Swaps the start and end point.

Private Attributes

- Point2D m_start

  Start point.
- Point2D m_end

  End point.
- double m_length

  Length.

Friends

- std::ostream & operator<< (std::ostream &out, const Edge2D &e)

13.1.1 Detailed Description

Stores edges in 2D plane.
The start and end points are stored in the edge.

13.1.2 Constructor & Destructor Documentation

cmst::Edge2D::Edge2D ( ) [inline]

cmst::Edge2D::Edge2D (const Point2D & *start,* const Point2D & *end*) [inline]

Constructor
Calculates the length.
Here is the call graph for this function:

13.1.3 Member Function Documentation

Point2D cmst::Edge2D::end ( ) const [inline]

Returns the end point.

Returns

end point

Here is the caller graph for this function:

double cmst::Edge2D::length ( ) const [inline]

Returns the length of the edge.

Returns

length of the edge

Here is the caller graph for this function:

bool cmst::Edge2D::operator< (const Edge2D & *right*) const [inline]

Compares edges by length.

bool cmst::Edge2D::operator== (const Edge2D & *right*) const `[inline]`

Compares cmst::Edge2D by start point and end point.
Take the cmst::Edge2D as undirected.

Point2D cmst::Edge2D::start ( ) const `[inline]`

Returns the start point.

Returns

start point

Here is the caller graph for this function:

void cmst::Edge2D::swap_points ( ) `[inline]`, `[protected]`

Swaps the start and end point.
Here is the caller graph for this function:

13.1.4 Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *out,* const Edge2D & *e*) `[friend]`

Prints information about the edge.
Prints the length, start point and end point.

13.1.5 Member Data Documentation

Point2D cmst::Edge2D::m_end `[private]`

End point.

double cmst::Edge2D::m_length `[private]`

Length.

Point2D cmst::Edge2D::m_start `[private]`

Start point.

13.2 cmst::Graph2D Class Reference

Collaboration diagram for cmst::Graph2D:

Classes

- struct ST

  Store a spanning tree of the graph.

Public Member Functions

- Graph2D (std::vector< Point2D > &points)
- double Kruskal ()
- double naiveKruskal ()
- void drawPoint ()

  Use GLUT to draw the points in the graph.
- void drawDelaunay ()

  Use GLUT to draw the Delaunay Diagram of the graph.
- void drawMST ()

  Use GLUT to draw the MST computed by Kruskal().
- void drawST ()

  Use GLUT to draw the ST computed by naiveKruskal().
- bool print (std::string file="graph.txt")

  Print the graph information to file.
- void changeSTDisplay (int direc)
- void printSTInfo ()

  Print the information of the current spanning tree displayed.
- double mstLength ()
- int delaunayTime () const

  Return the time used for computing Delaunay diagram.
- int mstTime ()
- int graphConstructTime () const
- int pointNum () const

  Return the number of points in this graph.
- int edgeNum () const

  Return the number of edges in the Delaunay diagram.
- bool validateDone () const

  Return if the MST has been validated.

Protected Member Functions

- int findFather (int x)

  Find the father of x in the Union-find Sets structure.
- void initFather ()

  Initializes the father array for Union-find Sets structure.

Protected Attributes

- std::vector< int > father

  Father array for Union-find Sets structure.
- std::vector< Point2D > m_points

  Points in the graph.
- std::vector< IndexEdge2D > m_delaunayEdge

  Delaunay edges of the graph.
- std::vector< IndexEdge2D > m_MSTEdge

  MST edges of the graph.
- std::vector< IndexEdge2D > m_edges

  All possible edges in the graph.

- std::vector< std::vector< int > > m_graph

  Adjacency list of the Delaunay diagram of the graph.
- Delaunay m_delaunay

  CGAL data structure for storing and computing a Delaunay diagram.
- std::vector< ST > m_ST

  Spanning trees of the graph.

Private Attributes

- bool m_mstDone

  If Kruskal() has been called.
- bool m_validateDone

  If naiveKruskal() has been called.
- double m_mstLength

  Length of the MST.
- int m_delaunayTime

  Time used for computing the Delaunay diagram.
- int m_mstTime

  Time used for computing the MST.
- int m_graphConstructTime

  Time used for reconstructing the graph.
- int m_displaySTNum

13.2.1 Constructor & Destructor Documentation

cmst::Graph2D::Graph2D (std::vector< Point2D > & *points*)

Constructor which does everything.

- Compute Delaunay graph
- Reconstruct the graph

Here is the call graph for this function:

13.2.2 Member Function Documentation

void cmst::Graph2D::changeSTDisplay (int *direc*) [inline]

Change the displaying spanning tree
To-do: calculate the top k spanning trees
Here is the caller graph for this function:

int cmst::Graph2D::delaunayTime ( ) const [inline]

Return the time used for computing Delaunay diagram.
Here is the caller graph for this function:

void cmst::Graph2D::drawDelaunay ( )

Use GLUT to draw the Delaunay Diagram of the graph.
Here is the caller graph for this function:

void cmst::Graph2D::drawMST ( )

Use GLUT to draw the MST computed by Kruskal().
Here is the caller graph for this function:

void cmst::Graph2D::drawPoint ( )

Use GLUT to draw the points in the graph.
Here is the caller graph for this function:

void cmst::Graph2D::drawST ( )

Use GLUT to draw the ST computed by naiveKruskal().
Here is the call graph for this function:
Here is the caller graph for this function:

int cmst::Graph2D::edgeNum ( ) const `[inline]`

Return the number of edges in the Delaunay diagram.
Here is the caller graph for this function:

int cmst::Graph2D::findFather (int *x*) `[protected]`

Find the father of x in the Union-find Sets structure.
Here is the caller graph for this function:

int cmst::Graph2D::graphConstructTime ( ) const `[inline]`

Return the time used for reconstructing the graph.
When using CGAL library, the internal data structure is different from the one used in this program. So you need some conversion.
Here is the caller graph for this function:

void cmst::Graph2D::initFather ( ) `[protected]`

Initializes the father array for Union-find Sets structure.
Here is the caller graph for this function:

double cmst::Graph2D::Kruskal ( )

The Kruskal algorithm for finding the minimal spanning tree.
Use the CGAL computed Delaunay Diagram.

Returns

The length of the MST.

Here is the call graph for this function:
Here is the caller graph for this function:

double cmst::Graph2D::mstLength ( ) `[inline]`

Return the length of the MST

Returns

Length of MST

Here is the call graph for this function:
Here is the caller graph for this function:

int cmst::Graph2D::mstTime ( ) [inline]

Return the time used for computing MST, using Kruskal's algorithm
Here is the call graph for this function:
Here is the caller graph for this function:

double cmst::Graph2D::naiveKruskal ( )

The naive Kruskal algorithm.
Construct all the edges in the graph, then run Kruskal.

Returns

The length of the MST.

Here is the call graph for this function:
Here is the caller graph for this function:

int cmst::Graph2D::pointNum ( ) const [inline]

Return the number of points in this graph.
Here is the caller graph for this function:

bool cmst::Graph2D::print (std::string *file* = *"graph.txt"*)

Print the graph information to file.
Here is the caller graph for this function:

void cmst::Graph2D::printSTInfo ( ) [inline]

Print the information of the current spanning tree displayed.
Here is the caller graph for this function:

bool cmst::Graph2D::validateDone ( ) const [inline]

Return if the MST has been validated.
Here is the caller graph for this function:

13.2.3 Member Data Documentation

std::vector<int> cmst::Graph2D::father [protected]

Father array for Union-find Sets structure.

Delaunay cmst::Graph2D::m_delaunay [protected]

CGAL data structure for storing and computing a Delaunay diagram.

std::vector<IndexEdge2D> cmst::Graph2D::m_delaunayEdge [protected]

Delaunay edges of the graph.

int cmst::Graph2D::m_delaunayTime [private]

Time used for computing the Delaunay diagram.

int cmst::Graph2D::m_displaySTNum [private]

std::vector<IndexEdge2D> cmst::Graph2D::m_edges [protected]

All possible edges in the graph.

std::vector<std::vector<int> > cmst::Graph2D::m_graph [protected]

Adjacency list of the Delaunay diagram of the graph.

int cmst::Graph2D::m_graphConstructTime [private]

Time used for reconstructing the graph.

bool cmst::Graph2D::m_mstDone [private]

If Kruskal() has been called.

std::vector<IndexEdge2D> cmst::Graph2D::m_MSTEdge [protected]

MST edges of the graph.

double cmst::Graph2D::m_mstLength [private]

Length of the MST.

int cmst::Graph2D::m_mstTime [private]

Time used for computing the MST.

std::vector<Point2D> cmst::Graph2D::m_points [protected]

Points in the graph.

std::vector<ST> cmst::Graph2D::m_ST [protected]

Spanning trees of the graph.

bool cmst::Graph2D::m_validateDone [private]

If naiveKruskal() has been called.

13.3 cmst::IndexEdge2D Class Reference

Edge with start and end point indices in an array.
Inheritance diagram for cmst::IndexEdge2D:
Collaboration diagram for cmst::IndexEdge2D:

Public Member Functions

- IndexEdge2D ()
- IndexEdge2D (Point2D p1, Point2D p2, int index1, int index2)
- int startIndex () const

  The index of the starting point.
- int endIndex () const

  The index of the end point.

- bool operator< (const IndexEdge2D &right) const
  Compares edges by length.
- bool operator> (const IndexEdge2D &right) const
  Compares edges by length.

Private Attributes

- int m_index [2]
  Indices of the end points.

Friends

- std::ostream & operator<< (std::ostream &str, const IndexEdge2D &e)

Additional Inherited Members

13.3.1 Detailed Description

Edge with start and end point indices in an array.

13.3.2 Constructor & Destructor Documentation

cmst::IndexEdge2D::IndexEdge2D ( ) [inline]

cmst::IndexEdge2D::IndexEdge2D (Point2D *p1*, Point2D *p2*, int *index1*, int *index2*) [inline]

Store the edge as an undirected one. The two end points will be sorted according to indices.
Here is the call graph for this function:

13.3.3 Member Function Documentation

int cmst::IndexEdge2D::endIndex ( ) const [inline]

The index of the end point.

bool cmst::IndexEdge2D::operator< (const IndexEdge2D & *right*) const [inline]

Compares edges by length.
Here is the call graph for this function:

bool cmst::IndexEdge2D::operator> (const IndexEdge2D & *right*) const [inline]

Compares edges by length.
Here is the call graph for this function:

int cmst::IndexEdge2D::startIndex ( ) const [inline]

The index of the starting point.

13.3.4 Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *str*, const IndexEdge2D & *e*) [friend]

13.3.5 Member Data Documentation

int cmst::IndexEdge2D::m_index[2] [private]

Indices of the end points.

13.4 cmst::Point2D Class Reference

Points in a 2D plane.

Collaboration diagram for cmst::Point2D:

Public Member Functions

- Point2D (double x=0.0, double y=0.0)

  Constructor.
- Point2D (const Point2D &other)

  Copy-constructor.
- double x () const
- double y () const
- bool operator< (const Point2D &right) const

  Compare points by x coordinates and y coordinates.
- bool operator== (const Point2D &right) const

Private Attributes

- double m_x

  x coordinate
- double m_y

  y coordinate

Friends

- std::ostream & operator<< (std::ostream &out, const Point2D &p)

13.4.1 Detailed Description

Points in a 2D plane.

13.4.2 Constructor & Destructor Documentation

cmst::Point2D::Point2D (double *x* = *0.0*, double *y* = *0.0*) [inline]

Constructor.

cmst::Point2D::Point2D (const Point2D & *other*) [inline]

Copy-constructor.

13.4.3 Member Function Documentation

bool cmst::Point2D::operator< (const Point2D & *right*) const [inline]

Compare points by x coordinates and y coordinates.

bool cmst::Point2D::operator== (const Point2D & *right*) const [inline]

Compare if two points are the same.

Some epsilon loss is allowed.

double cmst::Point2D::x ( ) const `[inline]`

Returns

x

Here is the caller graph for this function:

double cmst::Point2D::y ( ) const `[inline]`

Returns

y

Here is the caller graph for this function:

13.4.4 Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *out*, const Point2D & *p*) `[friend]`

13.4.5 Member Data Documentation

double cmst::Point2D::m_x `[private]`

x coordinate

double cmst::Point2D::m_y `[private]`

y coordinate

13.5 cmst::Graph2D::ST Struct Reference

Store a spanning tree of the graph.

Collaboration diagram for cmst::Graph2D::ST:

Public Member Functions

- ST (std::vector< IndexEdge2D > edges=std::vector< IndexEdge2D >(), int stTime=0, double length=0.0)

  Constructor.

Public Attributes

- std::vector< IndexEdge2D > m_edges

  Edges of the spanning tree.
- int m_stTime

  Time used to compute the spanning tree.
- double m_length

  Length of the spanning tree.

13.5.1 Detailed Description

Store a spanning tree of the graph.

13.5.2 Constructor & Destructor Documentation

cmst::Graph2D::ST::ST (std::vector< IndexEdge2D > *edges* = *std::vector*<IndexEdge2D>*()*, int *stTime* = *0*, double *length* = *0.0*) [inline]

Constructor.

13.5.3 Member Data Documentation

std::vector<IndexEdge2D> cmst::Graph2D::ST::m_edges

Edges of the spanning tree.

double cmst::Graph2D::ST::m_length

Length of the spanning tree.

int cmst::Graph2D::ST::m_stTime

Time used to compute the spanning tree.

13.6 cmst::Stat Class Reference

Collaboration diagram for cmst::Stat:

Public Member Functions

- Stat ()
- void record (double data)

  Record a datum and update m_min, m_max.
- double min () const
- double max () const
- int count () const
- double mean ()
- double standardDeviation ()
- std::string print ()

Private Attributes

- double m_min

  Minimum of the data.
- double m_max

  Maximum of the data.
- double m_mean

  Average of the data.
- double m_standardDeviation

  Standard deviation of the data.
- std::vector< double > m_data

  Data.

13.6.1 Detailed Description

Simple statistics.
Including:

- Minimum
- Maximum
- Mean
- Standard Deviation

13.6.2 Constructor & Destructor Documentation

cmst::Stat::Stat ( ) [inline]

Constructor
Set m_max to DOUBLE_MIN and m_min to DOUBLE_MAX

13.6.3 Member Function Documentation

int cmst::Stat::count ( ) const [inline]

Return the number of recorded data.

Returns

The number of recorded data

Return values

0	If no data has been recorded.

double cmst::Stat::max ( ) const [inline]

Return the maximum of recorded data.

Returns

Maximum of recorded data

Return values

0.0	If no data has been recorded

double cmst::Stat::mean ( ) [inline]

Return the mean of all data.

Returns

Mean of all data

Return values

0.0	If no data has been recorded.

Here is the caller graph for this function:

double cmst::Stat::min ( ) const `[inline]`

Return the minimum of recorded data.

Returns

Minimum of recorded data

Return values

0.0	If no data has been recorded

std::string cmst::Stat::print ( ) `[inline]`

Print the information of the statistic.

- Average
- Maximum
- Minimum
- Standard deviation

Here is the call graph for this function:
Here is the caller graph for this function:

void cmst::Stat::record (double *data*) `[inline]`

Record a datum and update m_min, m_max.
Here is the caller graph for this function:

double cmst::Stat::standardDeviation ( ) `[inline]`

Return the standard deviation of all data.

Returns

Standard deviation of all data

Return values

0.0	If no data has been recorded.

Here is the call graph for this function:
Here is the caller graph for this function:

13.6.4 Member Data Documentation

std::vector<double> cmst::Stat::m_data `[private]`

Data.

double cmst::Stat::m_max `[private]`

Maximum of the data.

double cmst::Stat::m_mean `[private]`

Average of the data.

double cmst::Stat::m_min `[private]`

Minimum of the data.

double cmst::Stat::m_standardDeviation `[private]`

Standard deviation of the data.

13.7 cmst::Window::Test Struct Reference

Collaboration diagram for cmst::Window::Test:

Public Member Functions

- Test ()

Public Attributes

- bool m_displayTest

  Whether a test has been generated and displayed.
- int m_displayTestNum

  The number of graphs in the test.
- std::vector< Graph2D > m_testGraphs

  The graphs generated in the test.
- Stat m_delaunayTimeStat

  Statistics of Delaunay Diagram computational time.
- Stat m_graphConstructTimeStat

  Statistics of graph re-construction time.
- Stat m_mstTimeStat

  Statistics of MST computational time.

13.7.1 Detailed Description

Stores information of a test.
Including the generated graphs and statistics of times.

13.7.2 Constructor & Destructor Documentation

cmst::Window::Test::Test ( ) `[inline]`

Constructor
No test is generated in initialization.

13.7.3 Member Data Documentation

Stat cmst::Window::Test::m_delaunayTimeStat

Statistics of Delaunay Diagram computational time.

bool cmst::Window::Test::m_displayTest

Whether a test has been generated and displayed.

int cmst::Window::Test::m_displayTestNum

The number of graphs in the test.

Stat cmst::Window::Test::m_graphConstructTimeStat

Statistics of graph re-construction time.

Stat cmst::Window::Test::m_mstTimeStat

Statistics of MST computational time.

std::vector<Graph2D> cmst::Window::Test::m_testGraphs

The graphs generated in the test.

13.8 cmst::Timer Class Reference

Collaboration diagram for cmst::Timer:

Public Member Functions

- Timer ()
  Constructor. Begin the timer.
- int time ()
- void reset ()
  Reset the timer.

Private Attributes

- int m_begin
  The time at construction or reset.

13.8.1 Detailed Description

A class for timing.
Uses simple clock() function.

13.8.2 Constructor & Destructor Documentation

cmst::Timer::Timer ( ) [inline]

Constructor. Begin the timer.

13.8.3 Member Function Documentation

void cmst::Timer::reset ( ) [inline]

Reset the timer.

Here is the caller graph for this function:

int cmst::Timer::time ( ) [inline]

Return the time since construction or reset.

The time unit is ms.

Here is the caller graph for this function:

13.8.4 Member Data Documentation

int cmst::Timer::m_begin [private]

The time at construction or reset.

13.9 cmst::Window Class Reference

Collaboration diagram for cmst::Window:

Classes

- struct Test

Public Member Functions

- Graph2D * curGraph ()

  Returns a pointer to the graph in display currently.
- void resetCurGraph (std::vector< Point2D > &points)
- void resetCurGraph ()
- void resetCurGraph (int n)
- void resetCurGraph (int low, int hi)
- bool load ()
- void resetShowDelaunay ()

  Change whether the Delaunay diagram is to be drawn to the GLUT window.
- void resetShowPoint ()

  Change whether the points are to be drawn to the GLUT window.
- void resetShowMST ()

  Change whether the MST is to be drawn to the GLUT window.
- void resetShowST ()

  Change whether the STs are to be drawn to the GLUT window.
- void resetWidth (int width)

  Record the width of current GLUT window.
- void resetHeight (int height)

  Record the height of current GLUT window.
- int width () const
- int height () const
- void draw ()
- void printCurInfo ()
- bool displayTest () const

- void generateTest (int n)
- void printTestInfo ()
- int testDisplayNum () const
- void changeTestDisplay (int direc)
- bool printToFile ()

  Print the information of the current graph to file graph.txt.
- void changeMSTDisplay (int direc)

  Change the MST that is being displayed.
- void printSTInfo ()

  Print information of the current ST to console.
- void runValidate ()

  Run the validator for small graphs.

Static Public Member Functions

- static Window * instance ()

Protected Attributes

- struct cmst::Window::Test m_test

  The test.

Private Member Functions

- Window ()

  Constructor.
- Window (const Window &)

  Private copy-constructor.

Private Attributes

- Graph2D * m_curGraph

  The pointer to the graph that is being displayed.
- bool m_showDelaunay

  Whether the Delaunay iagram is to be drawn.
- bool m_showMST

  Whether the MST is to be drawn.
- bool m_showST

  Whether the MST is to be drawn.
- bool m_showPoint

  Whether the points are to be drawn.
- int m_width

  The width of current GLUT window.
- int m_height

  The height of current GLUT window.

Static Private Attributes

- static Window * m_instance = NULL

  The pointer to an instance of cmst::Window.

13.9.1 Detailed Description

Manipulates the window.
Uses Singleton pattern.

13.9.2 Constructor & Destructor Documentation

cmst::Window::Window ( ) [inline], [private]

Constructor.
Here is the caller graph for this function:

cmst::Window::Window (const Window &) [private]

Private copy-constructor.

13.9.3 Member Function Documentation

void cmst::Window::changeMSTDisplay (int *direc*) [inline]

Change the MST that is being displayed.
Here is the call graph for this function:

void cmst::Window::changeTestDisplay (int *direc*) [inline]

If a test is being displayed, then changes the graph in the test that is being displayed.
If no test has been generated, does nothing.

Parameters

direc	If negative, display the last graph (if there is one); if positive, display the next graph (if there is one).

Graph2D* cmst::Window::curGraph ( ) [inline]

Returns a pointer to the graph in display currently.
Here is the call graph for this function:

bool cmst::Window::displayTest ( ) const [inline]

Returns if a test has been generated

Returns

If a test has been generated

Here is the call graph for this function:

void cmst::Window::draw ( )

Draws the current graph

- Points: definitely
- Delaunay Diagram: change whether to draw it by Window::resetShowDelaunay()
- MST: definitely

- Other spanning trees: draws one of them

Here is the call graph for this function:
Here is the caller graph for this function:

void cmst::Window::generateTest (int *n*)

Generates a test of n graphs and display the first one.

Parameters

n	The number of graphs in the test to be generated

Here is the call graph for this function:
Here is the caller graph for this function:

int cmst::Window::height ( ) const `[inline]`

Return the height of current GLUT window.

Returns

The height of current GLUT window

Here is the call graph for this function:
Here is the caller graph for this function:

static Window* cmst::Window::instance ( ) `[inline]`, `[static]`

Return the pointer to the instance of cmst::Window class.

Returns

the pointer to the instance

Here is the call graph for this function:

bool cmst::Window::load ( )

Here is the call graph for this function:
Here is the caller graph for this function:

void cmst::Window::printCurInfo ( )

Prints information about the current displayed graph to console
Information including numbers and computational time
Here is the call graph for this function:
Here is the caller graph for this function:

void cmst::Window::printSTInfo ( ) `[inline]`

Print information of the current ST to console.
Here is the call graph for this function:

void cmst::Window::printTestInfo ( )

Prints information about the test that has been generated to console.
If no test has been generated, then nothing is printed.
Here is the call graph for this function:
Here is the caller graph for this function:

bool cmst::Window::printToFile ( ) [inline]

Print the information of the current graph to file graph.txt.
Here is the call graph for this function:

void cmst::Window::resetCurGraph (std::vector< Point2D > & *points*)

Reset the current graph with a vector of points.

Parameters

points	A vector of points.

void cmst::Window::resetCurGraph ( )

Reset the current graph with cmst::TestcaseGenerator
The size of the graph is defaulted.
Here is the call graph for this function:
Here is the caller graph for this function:

void cmst::Window::resetCurGraph (int *n*)

Reset the current graph with n random generated points.

Parameters

n	The size of the graph to be generated.

Here is the call graph for this function:

void cmst::Window::resetCurGraph (int *low*, int *hi*)

Reset the current graph with random generated points.
The size of the graph to be generated is randomly selected between low and hi.

Parameters

low	The least number of points to be generated.
hi	The most number of points to be generated.

Here is the call graph for this function:

void cmst::Window::resetHeight (int *height*) [inline]

Record the height of current GLUT window.
Here is the call graph for this function:

void cmst::Window::resetShowDelaunay ( ) [inline]

Change whether the Delaunay diagram is to be drawn to the GLUT window.

void cmst::Window::resetShowMST ( ) [inline]

Change whether the MST is to be drawn to the GLUT window.

void cmst::Window::resetShowPoint ( ) [inline]

Change whether the points are to be drawn to the GLUT window.

void cmst::Window::resetShowST ( ) [inline]

Change whether the STs are to be drawn to the GLUT window.

void cmst::Window::resetWidth (int *width*) [inline]

Record the width of current GLUT window.
Here is the call graph for this function:

void cmst::Window::runValidate ( ) [inline]

Run the validator for small graphs.
Here is the call graph for this function:

int cmst::Window::testDisplayNum ( ) const [inline]

Returns the number of graphs in the test that has been generated.

Returns

the number of graphs in the test that has been generated.

Return values

0	If no test has been generated.

int cmst::Window::width ( ) const [inline]

Return the width of current GLUT window.

Returns

The width of current GLUT window

Here is the caller graph for this function:

13.9.4 Member Data Documentation

Graph2D* cmst::Window::m_curGraph [private]

The pointer to the graph that is being displayed.

int cmst::Window::m_height [private]

The height of current GLUT window.

cmst::Window * cmst::Window::m_instance = NULL [static], [private]

The pointer to an instance of cmst::Window.

bool cmst::Window::m_showDelaunay [private]

Whether the Delaunay iagram is to be drawn.

bool cmst::Window::m_showMST [private]

Whether the MST is to be drawn.

bool cmst::Window::m_showPoint [private]

Whether the points are to be drawn.

bool cmst::Window::m_showST [private]

Whether the MST is to be drawn.

struct cmst::Window::Test cmst::Window::m_test [protected]

The test.

int cmst::Window::m_width [private]

The width of current GLUT window.