Individual Project 2
Constructing Minimum Spanning Trees
Software Design Document

Zhang Huimeng, 2015011280

June 6, 2016

Contents

Part I

Introduction

Chapter 1

Purpose

Chapter 2

Scope

Chapter 3

Overview

(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. http://www.cs.princeton.edu/courses/archive/spr07/cos226/lectures/mst.pdf

(3) Use the CGAL Library (http://www.cgal.org/) 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 4

Reference Material

Chapter 5

Definitions and Acronyms

Part II

System Overview

Chapter 6

System Architecture

Chapter 7

Architectural Design

Chapter 8

Decomposition Description

Chapter 9

Design Rationale

Part III

Data Design

Chapter 10

Data Description

Chapter 11

Data Dictionary

Part IV

Human Interface Design

Chapter 12

Overview of Human Interface

Chapter 13

Screen Images

Chapter 14

Screen Objects and Actions

Part V

Design Patterns

Part VI

Component Design

Chapter 15

Namespace Index

15.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 16

Hierarchical Index

16.1 Class Hierarchy

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

Chapter 17

Class Index

17.1 Class List

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

Chapter 18

Namespace Documentation

18.1 cmst Namespace Reference

Classes

- class Edge2D
- class Graph2D
- class IndexEdge2D
- class Point2D
- class Stat
- class Timer
- class Window

Enumerations

- enum Menu {
  NEW, NEW_4_10, NEW_11_100, NEW_101_1000,
  NEW_1001_5000, NEW_5001_10000, SHOW, SHOW_VORONOI,
  SHOW_DELAUNAY, TEST, TEST_5, TEST_20,
  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)

18.1.1 Enumeration Type Documentation

enum cmst::Menu

Return values for GLUT menus.

Enumerator

NEW

NEW_4_10

NEW_11_100

NEW_101_1000

NEW_1001_5000

NEW_5001_10000

SHOW

SHOW_VORONOI

SHOW_DELAUNAY

TEST

TEST_5

TEST_20

QUIT

18.1.2 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`)

To-do: output the testcase file to ../testcase; Add erase and unique
Here is the call graph for this function:
Here is the caller graph for this function:

Chapter 19

Class Documentation

19.1 Delaunay Class Reference

`#include <delaunay.h>`

Collaboration diagram for Delaunay:

Public Member Functions

- std::vector< Triangle > triangulate (std::vector< Vec2f > &vertices)
- std::vector< Triangle > getTriangles ()
- std::vector< Edge > getEdges ()

Private Attributes

- std::vector< Triangle > _triangles
- std::vector< Edge > _edges

19.1.1 Member Function Documentation

std::vector<Edge> Delaunay::getEdges ( ) `[inline]`

std::vector<Triangle> Delaunay::getTriangles ( ) `[inline]`

std::vector< Triangle > Delaunay::triangulate (std::vector< Vec2f > & *vertices*)

Here is the call graph for this function:

19.1.2 Member Data Documentation

std::vector<Edge> Delaunay::_edges `[private]`

std::vector<Triangle> Delaunay::_triangles `[private]`

19.2 Edge Class Reference

`#include <edge.h>`

Collaboration diagram for Edge:

Public Member Functions

- Edge (const Vec2f &p1, const Vec2f &p2)
- Edge (const Edge &e)
- bool operator== (const Edge &e2) const

Public Attributes

- Vec2f p1
- Vec2f p2

Friends

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

19.2.1 Constructor & Destructor Documentation

Edge::Edge (const Vec2f & *p1*, const Vec2f & *p2*) `[inline]`

Edge::Edge (const Edge & *e*) `[inline]`

19.2.2 Member Function Documentation

bool Edge::operator== (const Edge & *e2*) const `[inline]`

19.2.3 Friends And Related Function Documentation

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

19.2.4 Member Data Documentation

Vec2f Edge::p1

Vec2f Edge::p2

19.3 cmst::Edge2D Class Reference

`#include <Edge2D.h>`

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

Public Member Functions

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

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

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)

19.3.1 Detailed Description

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

19.3.2 Constructor & Destructor Documentation

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

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

19.3.3 Member Function Documentation

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

Returns the end point.

Returns

end point

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

Returns the length of the edge.

Returns

length of the edge

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

Compares edges by length.

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

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

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

19.3.4 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.

19.3.5 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.

19.4 cmst::Graph2D Class Reference

`#include <Graph2D.h>`

Collaboration diagram for cmst::Graph2D:

Public Member Functions

- Graph2D (std::vector< Point2D > &points)
- double Kruskal (bool naive=false)
- void drawPoint ()
- void drawDelaunay ()
- void drawMST ()
- double mstLength ()
- int delaunayTime () const
- int mstTime ()
- int graphConstructTime () const
- int pointNum () const
- int edgeNum () const

Protected Member Functions

- int findFather (int x)
- void initFather ()

Protected Attributes

- std::vector< int > father
- std::vector< Point2D > m_points
- std::vector< IndexEdge2D > m_delaunayEdge
- std::vector< IndexEdge2D > m_MSTEdge
- std::vector< CGAL::Object > m_voronoiEdge
- std::vector< std::vector< int > > m_graph
- Delaunay m_delaunay

Private Attributes

- bool m_mstDone
- double m_mstLength
- int m_delaunayTime
- int m_mstTime
- int m_graphConstructTime
- int m_kMSTTime

19.4.1 Constructor & Destructor Documentation

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

Constructor which does everything.

- Compute naiveEdge which contains all edges
- Compute Delaunay graph

Here is the call graph for this function:

19.4.2 Member Function Documentation

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

Here is the caller graph for this function:

void cmst::Graph2D::drawDelaunay ( )

Here is the caller graph for this function:

void cmst::Graph2D::drawMST ( )

Here is the caller graph for this function:

void cmst::Graph2D::drawPoint ( )

Here is the caller graph for this function:

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

Here is the caller graph for this function:

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

Here is the caller graph for this function:

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

Here is the caller graph for this function:

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

Here is the caller graph for this function:

double cmst::Graph2D::Kruskal (bool *naive = false*)

The Kruskal algorithm for finding the minimal spanning tree.

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]

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

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

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

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

Here is the caller graph for this function:

19.4.3 Member Data Documentation

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

Delaunay cmst::Graph2D::m_delaunay [protected]

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

int cmst::Graph2D::m_delaunayTime [private]

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

int cmst::Graph2D::m_graphConstructTime [private]

int cmst::Graph2D::m_kMSTTime [private]

bool cmst::Graph2D::m_mstDone [private]

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

double cmst::Graph2D::m_mstLength [private]

int cmst::Graph2D::m_mstTime [private]

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

std::vector<CGAL::Object> cmst::Graph2D::m_voronoiEdge [protected]

19.5 cmst::IndexEdge2D Class Reference

`#include <IndexEdge2D.h>`

Inheritance diagram for cmst::IndexEdge2D:

Collaboration diagram for cmst::IndexEdge2D:

Public Member Functions

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

Private Attributes

- int m_index [2]

Friends

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

Additional Inherited Members

19.5.1 Constructor & Destructor Documentation

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

Here is the call graph for this function:

19.5.2 Member Function Documentation

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

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

19.5.3 Friends And Related Function Documentation

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

19.5.4 Member Data Documentation

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

19.6 cmst::Point2D Class Reference

`#include <Point2D.h>`

Collaboration diagram for cmst::Point2D:

Public Member Functions

- Point2D (double x=0.0, double y=0.0)
- Point2D (const Point2D &other)
- double x () const
- double y () const
- bool operator< (const Point2D &right)
- bool operator== (const Point2D &right)

Private Attributes

- double m_x
- double m_y

Friends

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

19.6.1 Constructor & Destructor Documentation

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

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

19.6.2 Member Function Documentation

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

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

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

Here is the caller graph for this function:

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

Here is the caller graph for this function:

19.6.3 Friends And Related Function Documentation

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

19.6.4 Member Data Documentation

double cmst::Point2D::m_x [private]

double cmst::Point2D::m_y [private]

19.7 cmst::Stat Class Reference

`#include <Stat.h>`

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
- double m_max
- double m_mean
- double m_standardDeviation
- std::vector< double > m_data

19.7.1 Detailed Description

Simple statistics.
Including:

- Minimum
- Maximum
- Mean
- Standard Deviation

19.7.2 Constructor & Destructor Documentation

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

Constructor
Set m_max to DOUBLE_MIN and m_min to DOUBLE_MAX

19.7.3 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]

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:

19.7.4 Member Data Documentation

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

double cmst::Stat::m_max [private]

double cmst::Stat::m_mean [private]

double cmst::Stat::m_min [private]

double cmst::Stat::m_standardDeviation [private]

19.8 cmst::Window::Test Struct Reference

`#include <Window.h>`
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.

19.8.1 Detailed Description

Stores information of a test.

Including the generated graphs and statistics of times.

19.8.2 Constructor & Destructor Documentation

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

Constructor

No test is generated in initialization.

19.8.3 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.

19.9 cmst::Timer Class Reference

`#include <Timer.h>`

Collaboration diagram for cmst::Timer:

Public Member Functions

- Timer ()
- int time ()
- void reset ()

Private Attributes

- int m_begin

19.9.1 Constructor & Destructor Documentation

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

19.9.2 Member Function Documentation

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

Here is the caller graph for this function:

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

Here is the caller graph for this function:

19.9.3 Member Data Documentation

int cmst::Timer::m_begin [private]

19.10 Triangle Class Reference

`#include <triangle.h>`

Collaboration diagram for Triangle:

Public Member Functions

- Triangle (const Vec2f &_p1, const Vec2f &_p2, const Vec2f &_p3)
- bool containsVertex (const Vec2f &v)
- bool circumCircleContains (const Vec2f &v)
- bool operator== (const Triangle &t2) const

Public Attributes

- Vec2f p1
- Vec2f p2
- Vec2f p3
- Edge e1
- Edge e2
- Edge e3

Friends

- std::ostream & operator<< (std::ostream &str, const Triangle &t)

19.10.1 Constructor & Destructor Documentation

Triangle::Triangle (const Vec2f & *_p1,* const Vec2f & *_p2,* const Vec2f & *_p3*)

19.10.2 Member Function Documentation

bool Triangle::circumCircleContains (const Vec2f & *v*)

bool Triangle::containsVertex (const Vec2f & *v*)

Here is the caller graph for this function:

bool Triangle::operator== (const Triangle & *t2*) const `[inline]`

19.10.3 Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *str,* const Triangle & *t*) `[friend]`

This has to be a friend.

19.10.4 Member Data Documentation

Edge Triangle::e1

Edge Triangle::e2

Edge Triangle::e3

Vec2f Triangle::p1

Vec2f Triangle::p2

Vec2f Triangle::p3

19.11 Vec2f Class Reference

`#include <vector2.h>`

Collaboration diagram for Vec2f:

Public Member Functions

- Vec2f ()
- Vec2f (double _x, double _y)
- Vec2f (const Vec2f &v)
- void set (const Vec2f &v)

- double dist2 (const Vec2f &v) const
- double dist (const Vec2f &v) const
- bool operator== (const Vec2f &right) const
- bool operator< (const Vec2f &right) const

Public Attributes

- double x
- double y

Friends

- std::ostream & operator<< (std::ostream &str, Vec2f p)

19.11.1 Constructor & Destructor Documentation

Vec2f::Vec2f ( ) [inline]

Vec2f::Vec2f (double *_x*, double *_y*) [inline]

Vec2f::Vec2f (const Vec2f & *v*) [inline]

19.11.2 Member Function Documentation

double Vec2f::dist (const Vec2f & *v*) const [inline]

Here is the call graph for this function:

double Vec2f::dist2 (const Vec2f & *v*) const [inline]

Here is the caller graph for this function:

bool Vec2f::operator< (const Vec2f & *right*) const [inline]

bool Vec2f::operator== (const Vec2f & *right*) const [inline]

void Vec2f::set (const Vec2f & *v*) [inline]

19.11.3 Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *str*, Vec2f *p*) [friend]

19.11.4 Member Data Documentation

double Vec2f::x

double Vec2f::y

19.12 cmst::Window Class Reference

`#include <Window.h>`

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)
- void resetShowDelaunay ()

  Change whether the Delaunay diagram is 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)

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 Diagram is 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.

19.12.1 Detailed Description

Manipulates the window.
Uses Singleton pattern.

19.12.2 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.

19.12.3 Member Function Documentation

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:

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::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:

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::resetWidth (int *width*) [inline]

Record the width of current GLUT window.

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:

19.12.4 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 Diagram 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.

Bibliography

[1] This is an example.