Streamline Query

The C++ implmentation for distance-based Query

Generated by Doxygen 1.8.11

Contents

Chapter 1

Query

The program is focused on streamline/pathline query given a candidate line. It is similar to line matching and pattern search, and will output the required number of lines that have minimal dissimilarity measure towards a given candidate (either by spatial proxity or shape similarity).

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 files with brief descriptions:

Chapter 4

Class Documentation

4.1 Query Class Reference

`#include <Query.h>`

Public Member Functions

- Query ()
- Query (const Eigen::MatrixXf &data, const int &Row, const int &Column)
- ~Query ()
- void getClosestInteresting (const int &normOption, std::vector< StringQuery > &searchResult)
- void getClosestCurve (const int &normOption, std::vector< StringQuery > &searchResult)
- bool interestedEmpty ()

Private Member Functions

- void getInteresting (const Eigen::MatrixXf &data, const int &Row, const int &Column)
- void searchClosest (const int &target, const int &closestNumber, const int &normOption, std::vector< int > &neighbor)

Private Attributes

- int similarityOption
- std::vector< int > interestedCurve
- std::vector< float > rotation
- Eigen::MatrixXf storage

4.1.1 Detailed Description

Definition at line 48 of file Query.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Query::Query ( )

Definition at line 65 of file Query.cpp.

```
66 {
67     //default constructor
68 }
```

4.1.2.2 Query::Query (const Eigen::MatrixXf & *data*, const int & *Row*, const int & *Column*)

Definition at line 78 of file Query.cpp.

```
81 {
82     storage = data;
83     getInteresting(data, Row, Column);
84 }
```

4.1.2.3 Query::~Query ( )

Definition at line 90 of file Query.cpp.

```
91 {
92     interestedCurve.clear();
93 }
```

4.1.3 Member Function Documentation

4.1.3.1 void Query::getClosestCurve (const int & *normOption*, std::vector< StringQuery > & *searchResult*)

Definition at line 149 of file Query.cpp.

```
151 {
152     assert(!interestedCurve.empty());
153     StringQuery tempObject;
154     string option;
155     int closestNumber;
156     int index;
157     do
158     {
159         std::cout << "Input index of curves? "
160                   << "Range from [0, " << (storage.rows()-1) << "]" << std::endl;
161         std::cin >> index;
162         assert(index >=0 && index < storage.rows());
163         std::cin.ignore();
164
165         std::cout << "Input number of closest strings? :" << std::endl;
166         std::cin >> closestNumber;
167         std::cin.ignore();
168
169         searchClosest(index, closestNumber,
170                       normOption, tempObject.neighbor);
171
172         tempObject.index = index;
173         searchResult.push_back(tempObject);
174
175         std::cout << "Want to have more string query? Y:Yes, N:No." << std::endl;
176         getline(cin, option);
177     }while(option.compare(string("Y"))==0 || option.compare(string("y"))==0);
178 }
```

4.1.3.2 void Query::getClosestInteresting (const int & *normOption,* std::vector< StringQuery > & *searchResult*)

Definition at line 111 of file Query.cpp.

```
113 {
114     assert(!interestedCurve.empty());
115     StringQuery tempObject;
116     string option;
117     int closestNumber;
118     int index;
119     do
120     {
121         std::cout << "Input index of interesting curves? "
122                   << "Range from [0, " << (interestedCurve.size()-1) << "]" << std::endl;
123         std::cin >> index;
124         assert(index >=0 && index < interestedCurve.size());
125         std::cin.ignore();
126
127         std::cout << "Input number of closest strings? :" << std::endl;
128         std::cin >> closestNumber;
129         std::cin.ignore();
130
131         searchClosest(interestedCurve[index], closestNumber,
132                       normOption, tempObject.neighbor);
133
134         tempObject.index = interestedCurve[index];
135         searchResult.push_back(tempObject);
136
137         std::cout << "Want to have more string query? Y:Yes, N:No." << std::endl;
138         getline(cin, option);
139     }while(option.compare(string("Y"))==0 || option.compare(string("y"))==0);
140 }
```

4.1.3.3 void Query::getInteresting (const Eigen::MatrixXf & *data,* const int & *Row,* const int & *Column*) [private]

Definition at line 17 of file Query.cpp.

```
20 {
21     rotation = std::vector<float>(Row);
22 #pragma omp parallel for schedule(static) num_threads(8)
23     for (int i = 0; i < Row; ++i)
24     {
25         float accumulation = 0.0, leftNorm, rightNorm, dotValue, result;
26         const Eigen::VectorXf& array = data.row(i);
27         const int& size = Column/3-2;
28         Vector3f left, right;
29         for (int j = 0; j < size; ++j)
30         {
31             //std::cout << array[j*3+3] << " " << array[j*3+4] << " " << array[j*3+5] << std::endl;
32             left << array(j*3+3)-array(j*3),
33                     array(j*3+4)-array(j*3+1),
34                     array(j*3+5)-array(j*3+2);
35             right << array(j*3+6)-array(j*3+3),
36                      array(j*3+7)-array(j*3+4),
37                      array(j*3+8)-array(j*3+5);
38             dotValue = left.dot(right);
39             leftNorm = left.norm();
40             rightNorm = right.norm();
41             if(leftNorm >= 1.0e-8 && rightNorm >=1.0e-8)
42             {
43                 result = dotValue/leftNorm/rightNorm;
44                 result = min(1.0,(double)result);
45                 result = max(-1.0,(double)result);
46                 accumulation += acos(result);
47             }
48         }
49         rotation[i] = accumulation;
50     }
51
52     for (int i = 0; i < Row; ++i)
53     {
54         if(rotation[i]>4.0*M_PI)
55             interestedCurve.push_back(i);
56     }
57
58     std::cout << "Found " << interestedCurve.size() << " interesting streamlines!" <<
      std::endl;
59 }
```

4.1.3.4 bool Query::interestedEmpty ( )

Definition at line 99 of file Query.cpp.

```
100 {
101     return interestedCurve.empty();
102 }
```

4.1.3.5 void Query::searchClosest (const int & *target*, const int & *closestNumber*, const int & *normOption*, std::vector< int > & *neighbor*) [private]

Definition at line 189 of file Query.cpp.

```
193 {
194     const int& row = storage.rows();
195     const int& lineNumber = storage.cols()/3-2;
196     std::vector<QueryDistance> distance(row);
197     neighbor = std::vector<int>(closestNumber);
198 #pragma omp parallel for schedule(static) num_threads(8)
199     for (int i = 0; i < row; ++i)
200     {
201         float dist;
202         if(i==target)
203         {
204             distance[i].distance = 0;
205             distance[i].index = target;
206             continue;
207         }
208 
209         MetricPreparation object;
210         object.preprocessing(storage, storage.rows(), storage.cols(), normOption);
211 
212         dist = getDisimilarity(storage.row(target),storage.row(i),target,i,normOption,object)
    ;
213 
214         distance[i].distance = -dist;
215         distance[i].index = i;
216     }
217 
218     std::make_heap(distance.begin(), distance.end());
219     std::cout << "Target: " << target << ", closest is: "
220               << distance.front().index << std::endl;
221     assert(distance.front().index==target);
222     std::pop_heap(distance.begin(), distance.end());
223     distance.pop_back();
224     for (int i = 0; i < closestNumber; ++i)
225     {
226         neighbor[i] = distance.front().index;
227         std::cout << distance.front().index << " " << distance.front().distance << std::endl;
228         std::pop_heap(distance.begin(), distance.end());
229         distance.pop_back();
230     }
231 
232     for (int i = 0; i < neighbor.size(); ++i)
233     {
234         std::cout << neighbor[i] << std::endl;
235     }
236 }
```

4.1.4 Member Data Documentation

4.1.4.1 std::vector<int> Query::interestedCurve [private]

Definition at line 60 of file Query.h.

4.1.4.2 std::vector<float> Query::rotation [private]

Definition at line 65 of file Query.h.

4.1.4.3 int Query::similarityOption [private]

Definition at line 55 of file Query.h.

4.1.4.4 Eigen::MatrixXf Query::storage [private]

Definition at line 70 of file Query.h.

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

- Query.h
- Query.cpp

4.2 QueryDistance Struct Reference

`#include <Query.h>`

Public Member Functions

- QueryDistance ()
- QueryDistance (const float &distance, const int &index)
- bool operator< (const QueryDistance &others)

Public Attributes

- float distance
- int index

4.2.1 Detailed Description

Definition at line 27 of file Query.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 QueryDistance::QueryDistance ( ) [inline]

Definition at line 31 of file Query.h.

```
32     {}
```

4.2.2.2 QueryDistance::QueryDistance (const float & *distance,* const int & *index*) [inline]

Definition at line 33 of file Query.h.

```
33                                                  :
34                  distance(distance), index(index){}
```

4.2.3 Member Function Documentation

4.2.3.1 bool QueryDistance::operator< (const QueryDistance & *others*) [inline]

Definition at line 35 of file Query.h.

```
36     {
37         return distance < others.distance;
38     }
```

4.2.4 Member Data Documentation

4.2.4.1 float QueryDistance::distance

Definition at line 29 of file Query.h.

4.2.4.2 int QueryDistance::index

Definition at line 30 of file Query.h.

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

- Query.h

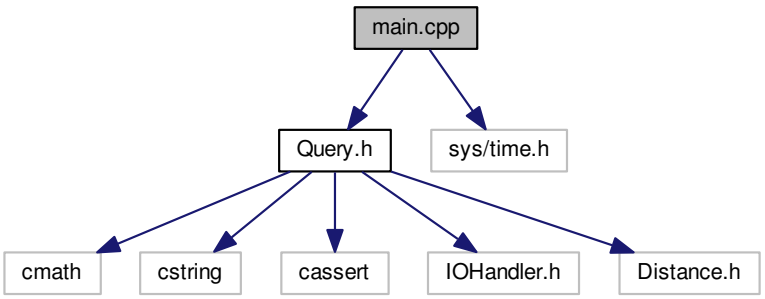
Chapter 5

File Documentation

5.1 main.cpp File Reference

```
#include "Query.h"
#include <sys/time.h>
```

Include dependency graph for main.cpp:



Functions

- void streamlineQuery (const int &argc, char **argv)
- int main (int argc, char *argv[ ])

Variables

- int initializationOption
- bool isPathlines

5.1.1 Function Documentation

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

Definition at line 40 of file main.cpp.

```
41 {
42     streamlineQuery(argc, argv);
43     return 0;
44 }
```

5.1.1.2 void streamlineQuery (const int & *argc*, char ** *argv*)

Definition at line 53 of file main.cpp.

```
55 {
56     while(argc!=3)
57     {
58         std::cout << "Input argument should have 3!" << endl
59                   << "./cluster inputFile data_dimension" << endl;
60         exit(1);
61     }
62     const string& strName = string("../dataset/")+string(argv[1]);
63     const int& dimension = atoi(argv[2]);
64     std::vector< std::vector<float> > dataVec;
65     int vertexCount, maxElements;
66     IOHandler::readFile(strName, dataVec, vertexCount, dimension, maxElements);
67
68     /*stringstream ss;
69     ss << strName << "_differentNorm_full.vtk";
70     const string& fullName = ss.str();
71     IOHandler::printVTK(ss.str(), dataVec, vertexCount, dimension);
72     ss.str("");*/
73
74     Eigen::MatrixXf data;
75     int userInput;
76     std::cout << "It is pathlines? 1.Yes, 0.No" << std::endl;
77     std::cin >> userInput;
78     assert(userInput==1||userInput==0);
79     isPathlines = (isPathlines==1);
80
81     int samplingMethod;
82
83     if(isPathlines)
84         samplingMethod = 1;
85     else
86     {
87         /* select sampling strategy, and 2 is often for geometric clustering */
88         std::cout << "Please choose sampling strategy: " << std::endl
89                   << "1.directly filling, 2.uniformly sampling" << std::endl;
90         int samplingMethod;
91         std::cin >> samplingMethod;
92     }
93
94     assert(samplingMethod==1 || samplingMethod==2);
95     if(samplingMethod==1)
96         IOHandler::expandArray(data, dataVec, dimension, maxElements); //directly filling
97     else if(samplingMethod==2)
98         IOHandler::sampleArray(data, dataVec, dimension, maxElements); //uniform sampling
99
100     Query q = Query(data, dataVec.size(), maxElements);
101     std::vector<StringQuery> searchResult;
102
103     int searchInteresting;
104     char isContinued;
105
106     /*  0: Euclidean Norm
107         1: Fraction Distance Metric
108         2: piece-wise angle average
109         3: Bhattacharyya metric for rotation
110         4: average rotation
111         5: signed-angle intersection
112         6: normal-direction multivariate distribution
113         7: Bhattacharyya metric with angle to a fixed direction
114         8: Piece-wise angle average \times standard deviation
```

```
115         9: normal-direction multivariate un-normalized distribution
116         10: x*y/|x||y| borrowed from machine learning
117         11: cosine similarity
118         12: Mean-of-closest point distance (MCP)
119         13: Hausdorff distance min_max(x_i,y_i)
120         14: Signature-based measure from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6231627
121         15: Procrustes distance take from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6787131
122     */
123
124     if(isPathlines)
125     {
126         for (int i = 2; i < 17; ++i)
127         {
128             //std::cout << "Wanted to continue the query?" << std::endl
129             //        << "Y. Yes; N. No" << std::endl;
130             //std::cin >> isContinued;
131             //if(isContinued=='N'||isContinued=='n')
132             //  break;
133             std::cin.ignore();
134             std::cout << "-------------------------------------------------" << std::endl
135                       << "---------------------- Norm " << i << " string query"
136                       << "------------------" << std::endl;
137
138             std::cout << "Want to search among interesting curves or not?"
139                       << " 0.No, 1.Yes!" << std::endl;
140             std::cin >> searchInteresting;
141             std::cin.ignore();
142
143             if(searchInteresting==0)
144                 q.getClosestCurve(i,searchResult);
145             else if(searchInteresting==1&&!q.interestedEmpty())
146                 q.getClosestInteresting(i,searchResult);
147
148             for (int j = 0; j < searchResult.size(); ++j)
149             {
150                 IOHandler::printQuery(i,j,searchResult[j], dataVec);
151             }
152             searchResult.clear();
153             std::cout << "-------------------------------------------------" << std::endl;
154         }
155     }
156     else
157     {
158         for (int i = 2; i < 15; ++i)
159         {
160             if(i!=12)
161                 continue;
162             //std::cout << "Wanted to continue the query?" << std::endl
163             //        << "Y. Yes; N. No" << std::endl;
164             //std::cin >> isContinued;
165             //if(isContinued=='N'||isContinued=='n')
166             //  break;
167             std::cin.ignore();
168             std::cout << "-------------------------------------------------" << std::endl
169                       << "---------------------- Norm " << i << " string query"
170                       << "------------------" << std::endl;
171
172             std::cout << "Want to search among interesting curves or not?"
173                       << " 0.No, 1.Yes!" << std::endl;
174             std::cin >> searchInteresting;
175             std::cin.ignore();
176
177             if(searchInteresting==0)
178                 q.getClosestCurve(i,searchResult);
179             else if(searchInteresting==1&&!q.interestedEmpty())
180                 q.getClosestInteresting(i,searchResult);
181
182             for (int j = 0; j < searchResult.size(); ++j)
183             {
184                 IOHandler::printQuery(i,j,searchResult[j], dataVec);
185             }
186             searchResult.clear();
187             std::cout << "-------------------------------------------------" << std::endl;
188         }
189     }
190 }
```

5.1.2 Variable Documentation

5.1.2.1 int initializationOption

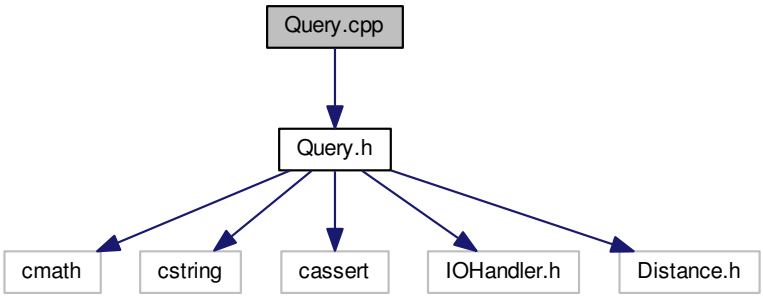
Definition at line 25 of file main.cpp.

5.1.2.2 bool isPathlines

Definition at line 30 of file main.cpp.

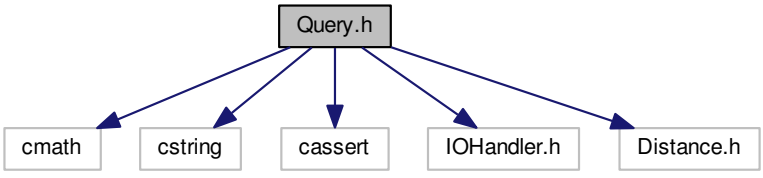
5.2 Query.cpp File Reference

```
#include "Query.h"
```
Include dependency graph for Query.cpp:

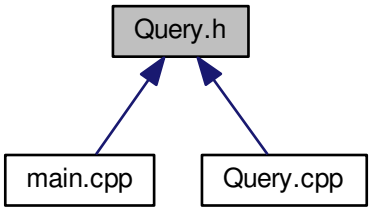


5.3 Query.h File Reference

```
#include <cmath>
#include <cstring>
#include <cassert>
#include "IOHandler.h"
#include "Distance.h"
```
Include dependency graph for Query.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct QueryDistance
- class Query

Macros

- #define M_PI 3.141592653

5.3.1 Macro Definition Documentation

5.3.1.1 #define M_PI 3.141592653

Definition at line 18 of file Query.h.

5.4 README.md File Reference

Index