Mesh Smoothing Tool

Generated by Doxygen 1.8.13

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Mesh Processing

Mesh denoising functions.

Data Structures

- struct MyTraits

  The MyTraits struct - OpenMesh custom traits.

Typedefs

- typedef OpenMesh::TriMesh_ArrayKernelT< MyTraits > TriMesh

  TriMesh - triangular mesh definition.
- typedef float num_t

  num_t - type definition for numbers used in mesh processing module

Enumerations

- enum FaceNeighborType { kVertexBased, kEdgeBased, kRadiusBased }

  The FaceNeighborType enum - determines the face neighborhood features.

Functions

- void updateVertexPositions (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, int iteration_↩ number, bool fixed_boundary)

  updateVertexPositions - computes new vertex positions adapted to an input normal field.
- num_t getSigmaC (TriMesh &mesh, vector< TriMesh::Point > &face_centroids, num_t sigma_c_scalar)

  getSigmaC - sigma_c computation based on the average of face centroid distances (only between adjacent faces), and multiplied by a scalar
- num_t getRadius (TriMesh &mesh, num_t scalar)

  getRadius - computation of radius for radius-based face neighborhood, based on the average of face centroid distances (only between adjacent faces) and multiplied by a scalar

- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale)

  Uniform Laplacian smoothing.
- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale, vector< size_t > &vertex_ids)

  uniformLaplacian - denoise a subset of vertices of an input mesh using Uniform Laplacian Smoothing Algorithm
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta)

  HC Laplacian smoothing (Vollmer et al.)
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta, vector< size_t > &vertex_ids)

  HCLaplacian - denoise a subset of vertices of an input mesh using HC Laplacian Smoothing Algorithm.
- void updateFilteredNormals (TriMesh &mesh, int normal_iteration_number, num_t sigma_c_scalar, num_t sigma_s, vector< TriMesh::Normal > &filtered_normals)

  Bilateral normal filtering for mesh denoising (Zheng et al.)
- TriMesh bilateralNormal (TriMesh &_mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s)

  bilateralNormal - Denoise the input mesh using Bilateral Normal Filtering Algorithm
- void getAllFaceNeighborsGMNF (TriMesh &mesh, FaceNeighborType face_neighbor_type, num_t radius, bool include_central_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  Guided mesh normal filtering (Zhang et al.)
- void getAllGuidedNeighborsGMNF (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors)

  getAllGuidedNeighborsGMNF - neighborhood computation for guidance signal
- void getFaceNeighborsInnerEdges (TriMesh &mesh, vector< TriMesh::FaceHandle > &face_neighbors, vector< TriMesh::EdgeHandle > &inner_edges)

  getFaceNeighborsInnerEdges - get all inner edges of a face neighborhood
- void getConsistenciesAndMeanNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > &consistencies_and_mean_normals)

  getConsistenciesAndMeanNormals - consistency and mean normal computation for all patches
- void getGuidedNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > consistencies_and_mean_normals, vector< TriMesh::Normal > &guided_normals)

  getGuidedNormals - guided normal computation for each face of the mesh
- void updateFilteredNormalsGuided (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, num_t radius_scalar, num_t sigma_c_scalar, int normal_iteration_number, num_t sigma_s, int vertex_iteration_number)

  updateFilteredNormalsGuided - guided bilateral filtering of a normal field (face based) of a mesh
- TriMesh guided (TriMesh _mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s, num_t radius_scalar)

  guided - Denoise the input mesh using Guided Mesh Normal Filtering Algorithm
- bool importMesh (TriMesh &mesh, string filename)

  importMesh - read mesh file (.obj, .off, .ply, .stl)
- bool exportMesh (TriMesh &mesh, string filename)

  exportMesh - write mesh file (.obj, .off, .ply, .stl)
- void getVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, int k, vector< TriMesh::VertexHandle > &vertex_neighbors)

  Vertex neighborhood.
- void getAdaptiveVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, num_t radius, vector< TriMesh::VertexHandle > &vertex_neighbors)

  getAdaptiveVertexNeighbors: get all neighboring vertices of a given vertex regarding a radius
- void getAllVertexNeighbors (TriMesh &mesh, int k, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllVertexNeighbors: get all neighboring vertices of all vertices regarding a depth k (k-ring)

- void getAllAdaptiveVertexNeighbors (TriMesh &mesh, num_t radius, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllAdaptiveVertexNeighbors: get all neighboring vertices of all vertices regarding a radius
- void getFaceNeighbors_EdgeBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  Face neighborhood.
- void getFaceNeighbors_VertexBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_VertexBased: get neighboring faces based on face vertices
- void getFaceNeighbors_RadiusBased (TriMesh &mesh, TriMesh::FaceHandle fh, num_t radius, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_RadiusBased: get neighboring faces regarding a radius
- void getAllFaceNeighbors_EdgeBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  getAllFaceNeighbors_EdgeBased: get all neighboring faces of all faces (edge based)
- void getAllFaceNeighbors_VertexBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  getAllFaceNeighbors_VertexBased: get all neighboring faces of all faces (vertex based)
- num_t getArea (TriMesh &mesh)

  getArea - area computation of a mesh (sum of triangle areas)
- num_t getVolume (TriMesh &mesh)

  getVolume - volume computation of a mesh
- num_t getAverageEdgeLength (TriMesh &mesh)

  getAverageEdgeLength - average edge length computation of a mesh
- void getAllFaceAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllFaceAreas - area computation for all faces
- void getAllFaceCentroids (TriMesh &mesh, vector< TriMesh::Point > &centroids)

  getAllFaceCentroids - centroid computation for all faces
- void getAllFaceNormals (TriMesh &mesh, vector< TriMesh::Normal > &normals)

  getAllFaceNormals - normal computation for all faces
- void getFaceVertexAngle (TriMesh &mesh, TriMesh::FaceHandle fh, TriMesh::VertexHandle vh, num_t &angle)

  getFaceVertexAngle - angle computation for a given vertex included in a face
- num_t getVertexArea (TriMesh &mesh, TriMesh::VertexHandle vh, vector< num_t > &areas)

  getVertexArea - vertex area computation (barycentric area)
- void getAllVertexAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllVertexAreas - vertex area computation for all vertices (barycentric areas)
- void getAllPoints (TriMesh &mesh, vector< TriMesh::Point > &points)

  getAllPoints - get all vertex coordinates
- num_t GaussianWeight (num_t distance, num_t sigma)

  GaussianWeight - gaussian function computation used for bilateral filtering.
- num_t NormalDistance (const TriMesh::Normal &n1, const TriMesh::Normal &n2)

  NormalDistance - computation of normal distance: $|n1 - n2|$.

5.1.1 Detailed Description

Mesh denoising functions.

Mesh processing auxiliar functions.

Mesh processing neighborhood functions.

Mesh processing half-edge data structure definition (OpenMesh library).

Mesh processing I/O functions (OpenMesh library).

Module containing mesh processing tools useful for the application

5.1.2 Typedef Documentation

5.1.2.1 num_t

```
typedef float num_t
```

num_t - type definition for numbers used in mesh processing module

Definition at line 49 of file mesh.h.

5.1.2.2 TriMesh

```
typedef OpenMesh::TriMesh_ArrayKernelT<MyTraits> TriMesh
```

TriMesh - triangular mesh definition.

Definition at line 44 of file mesh.h.

5.1.3 Enumeration Type Documentation

5.1.3.1 FaceNeighborType

```
enum FaceNeighborType
```

The FaceNeighborType enum - determines the face neighborhood features.

Enumerator

kVertexBased	vertex based face neighborhood
kEdgeBased	edge based face neighborhood
kRadiusBased	radius based face neighborhood

Definition at line 18 of file denoising.h.

```
18                      {
19     kVertexBased,
20     kEdgeBased,
21     kRadiusBased
22 };
```

5.1.4 Function Documentation

5.1.4.1 bilateralNormal()

```
TriMesh bilateralNormal (
            TriMesh & _mesh,
            int normal_iteration_number,
            int vertex_iteration_number,
            num_t sigma_c_scalar,
            num_t sigma_s )
```

bilateralNormal - Denoise the input mesh using Bilateral Normal Filtering Algorithm

Parameters

_mesh	input mesh
normal_iteration_number	number of iterations for normal field filtering
vertex_iteration_number	number of iterations for vertex updating
sigma_c_scalar	bilateral normal field filtering influence regarding a $sigma_c$ based on average edge length (spatial distance influence)
sigma_s	bilateral normal field filtering parameter $sigma_s$

Returns

denoised mesh

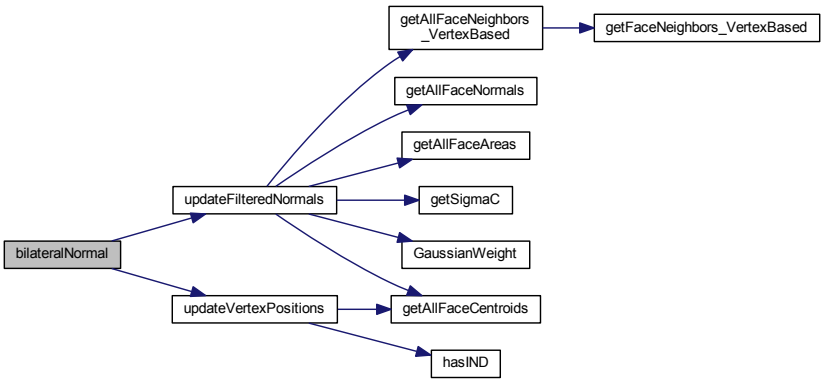
Definition at line 327 of file denoising.cpp.

References updateFilteredNormals(), and updateVertexPositions().

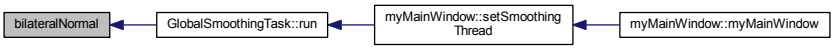
Referenced by GlobalSmoothingTask::run().

```
328 {
329     TriMesh mesh = _mesh;
330     vector<TriMesh::Normal> filtered_normals;
331     updateFilteredNormals(mesh, normal_iteration_number, sigma_c_scalar, sigma_s,
      filtered_normals);
332     updateVertexPositions(mesh, filtered_normals, vertex_iteration_number, true);
333     mesh.request_face_normals();
334     mesh.request_vertex_normals();
335     mesh.update_normals();
336     return mesh;
337 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.2 exportMesh()

```
bool exportMesh (
            TriMesh & mesh,
            string filename )
```

exportMesh - write mesh file (.obj, .off, .ply, .stl)

Parameters

mesh	- input mesh
filename	- output file name

Returns

success

Definition at line 12 of file iomesh.cpp.

Referenced by myDataManager::saveOutputMesh().

```
13 {
14     OpenMesh::IO::Options opt;
15     //opt += OpenMesh::IO::Options::VertexColor;
16     return OpenMesh::IO::write_mesh(mesh, filename,opt);
17 }
```

Here is the caller graph for this function:

5.1.4.3 GaussianWeight()

```
num_t GaussianWeight (
            num_t distance,
            num_t sigma )  [inline]
```

GaussianWeight - gaussian function computation used for bilateral filtering.

Parameters

distance	input distance
sigma	input sigma

Returns

function value for inputs (distance and sigma)

Definition at line 98 of file util.h.

Referenced by updateFilteredNormals(), and updateFilteredNormalsGuided().

```
99 {
100     return static_cast<num_t>(exp( -0.5 * distance * distance / (sigma * sigma)));
101 }
```

Here is the caller graph for this function:

5.1.4.4 getAdaptiveVertexNeighbors()

```
void getAdaptiveVertexNeighbors (
            TriMesh & mesh,
            TriMesh::VertexHandle vh,
            num_t radius,
            vector< TriMesh::VertexHandle > & vertex_neighbors )
```

getAdaptiveVertexNeighbors: get all neighboring vertices of a given vertex regarding a radius

Parameters

mesh	input mesh
vh	evaluated vertex
radius	radius
vertex_neighbors	vector containing neighbors (output)

Definition at line 42 of file neighborhood.cpp.

Referenced by getAllAdaptiveVertexNeighbors().

```
44 {
45     vertex_neighbors.clear();
```

```
46      vector<bool> mark(mesh.n_vertices(),false);
47      queue<TriMesh::VertexHandle> queue_vertex_handle;
48      mark[vh.idx()] = true;
49      queue_vertex_handle.push(vh);
50      TriMesh::Point ci = mesh.point(vh);
51
52      while(!queue_vertex_handle.empty())
53      {
54          TriMesh::VertexHandle vh = queue_vertex_handle.front();
55          vertex_neighbors.push_back(vh);
56          queue_vertex_handle.pop();
57          for(TriMesh::VertexVertexIter vv_it = mesh.vv_iter(vh); vv_it.is_valid(); ++vv_it)
58          {
59              TriMesh::VertexHandle vh_neighbor = *vv_it;
60              if(mark[vh_neighbor.idx()] == false)
61              {
62                  TriMesh::Point cj = mesh.point(vh_neighbor);
63                  num_t length = (cj - ci).length();
64                  if(length <= radius)
65                      queue_vertex_handle.push(vh_neighbor);
66                  mark[vh_neighbor.idx()] = true;
67              }
68          }
69      }
70  }
```

Here is the caller graph for this function:

5.1.4.5 getAllAdaptiveVertexNeighbors()

```
void getAllAdaptiveVertexNeighbors (
              TriMesh & mesh,
              num_t radius,
              vector< vector< TriMesh::VertexHandle > > & all_vertex_neighbors )
```

getAllAdaptiveVertexNeighbors: get all neighboring vertices of all vertices regarding a radius

Parameters

mesh	input mesh
radius	radius
all_vertex_neighbors	vector containing all neighbor vectors (output)

Definition at line 83 of file neighborhood.cpp.

References getAdaptiveVertexNeighbors().

```
85  {
86      all_vertex_neighbors.resize(mesh.n_vertices());
87      for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
```

```
88     {
89         vector<TriMesh::VertexHandle> vertex_neighbor;
90         getAdaptiveVertexNeighbors(mesh, *v_it, radius, vertex_neighbor);
91         all_vertex_neighbors[v_it->idx()] = vertex_neighbor;
92     }
93 }
```

Here is the call graph for this function:

5.1.4.6 getAllFaceAreas()

```
void getAllFaceAreas (
             TriMesh & mesh,
             vector< num_t > & areas )
```

getAllFaceAreas - area computation for all faces

Parameters

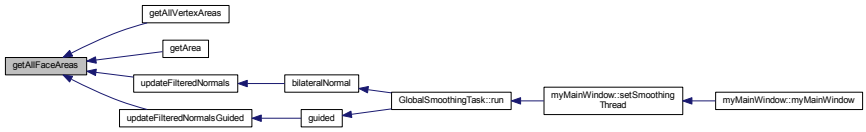
mesh	input mesh
areas	output vector containing areas for all faces

Definition at line 54 of file util.cpp.

Referenced by getAllVertexAreas(), getArea(), updateFilteredNormals(), and updateFilteredNormalsGuided().

```
55 {
56     areas.resize(mesh.n_faces());
57 
58     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
59     {
60         vector<TriMesh::Point> point;
61         point.resize(3); int index = 0;
62         for(TriMesh::FaceVertexIter fv_it = mesh.fv_iter(*f_it); fv_it.is_valid(); fv_it++)
63         {
64             point[index] = mesh.point(*fv_it);
65             index++;
66         }
67         TriMesh::Point edge1 = point[1] - point[0];
68         TriMesh::Point edge2 = point[1] - point[2];
69         num_t S = 0.5f * (edge1 % edge2).length();
70         areas[(*f_it).idx()] = S;
71     }
72 }
```

Here is the caller graph for this function:



5.1.4.7 getAllFaceCentroids()

```
void getAllFaceCentroids (
            TriMesh & mesh,
            vector< TriMesh::Point > & centroids )
```

getAllFaceCentroids - centroid computation for all faces

Parameters

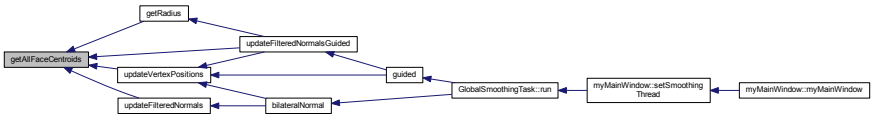
mesh	input mesh
centroids	output vector containing centroids for all faces

Definition at line 74 of file util.cpp.

Referenced by getRadius(), updateFilteredNormals(), updateFilteredNormalsGuided(), and updateVertex↩ Positions().

```
75 {
76     centroids.resize(mesh.n_faces(), TriMesh::Point(0.0, 0.0, 0.0));
77     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
78     {
79         TriMesh::Point pt = mesh.calc_face_centroid(*f_it);
80         centroids[(*f_it).idx()] = pt;
81     }
82 }
```

Here is the caller graph for this function:



5.1.4.8 getAllFaceNeighbors_EdgeBased()

```
void getAllFaceNeighbors_EdgeBased (
            TriMesh & mesh,
            bool include_target_face,
            vector< vector< TriMesh::FaceHandle > > & all_face_neighbors )
```

getAllFaceNeighbors_EdgeBased: get all neighboring faces of all faces (edge based)

Parameters

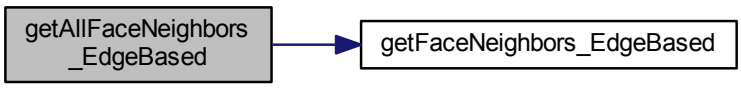
mesh	input mesh
include_target_face	include fh as neighbor
all_face_neighbors	vector containing neighbors (output)

Definition at line 157 of file neighborhood.cpp.

References getFaceNeighbors_EdgeBased().

```
158 {
159     all_face_neighbors.resize(mesh.n_faces());
160     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
161     {
162         vector<TriMesh::FaceHandle> face_neighbors;
163         getFaceNeighbors_EdgeBased(mesh, *f_it, face_neighbors);
164         if(include_target_face) face_neighbors.push_back(*f_it); //include target face
165         all_face_neighbors[f_it->idx()] = face_neighbors;
166     }
167 }
```

Here is the call graph for this function:



5.1.4.9 getAllFaceNeighbors_VertexBased()

```
void getAllFaceNeighbors_VertexBased (
             TriMesh & mesh,
             bool include_target_face,
             vector< vector< TriMesh::FaceHandle > > & all_face_neighbors )
```

getAllFaceNeighbors_VertexBased: get all neighboring faces of all faces (vertex based)

Parameters

mesh	input mesh
include_target_face	include fh as neighbor
all_face_neighbors	vector containing neighbors (output)

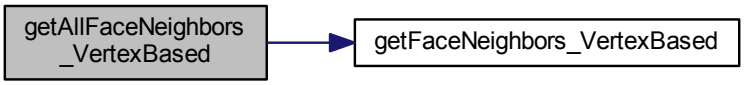
Definition at line 169 of file neighborhood.cpp.

References getFaceNeighbors_VertexBased().

Referenced by updateFilteredNormals().

```
170 {
171     all_face_neighbors.resize(mesh.n_faces());
172     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
173     {
174         vector<TriMesh::FaceHandle> face_neighbors;
175         getFaceNeighbors_VertexBased(mesh, *f_it, face_neighbors);
176         if(include_target_face) face_neighbors.push_back(*f_it); //include target face
177         all_face_neighbors[f_it->idx()] = face_neighbors;
178     }
179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.10 getAllFaceNeighborsGMNF()

```
void getAllFaceNeighborsGMNF (
            TriMesh & mesh,
            FaceNeighborType face_neighbor_type,
            num_t radius,
            bool include_central_face,
            vector< vector< TriMesh::FaceHandle > > & all_face_neighbors )
```

Guided mesh normal filtering (Zhang et al.)

getAllFaceNeighborsGMNF - neighborhood computation for all mesh faces (for bilateral filtering)

Parameters

mesh	input mesh
face_neighbor_type	type of neighborhood to be considered
radius	radius for radius based face neighborhood
include_central_face	if the same face is considered as a neighbor
all_face_neighbors	output vector containing neighboring faces IDs for each face of the mesh

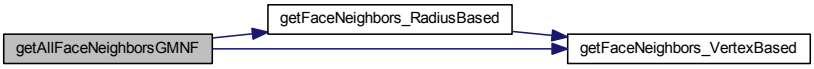
Definition at line 344 of file denoising.cpp.

References getFaceNeighbors_RadiusBased(), getFaceNeighbors_VertexBased(), kRadiusBased, and kVertex↩ Based.

Referenced by updateFilteredNormalsGuided().

```
346 {
347     vector<TriMesh::FaceHandle> face_neighbors;
348     for (TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
349     {
350         if (face_neighbor_type == kVertexBased)
351             getFaceNeighbors_VertexBased(mesh, *f_it, face_neighbors);
352         else if (face_neighbor_type == kRadiusBased)
353             getFaceNeighbors_RadiusBased(mesh, *f_it, radius, face_neighbors);
354 
355         if (include_central_face)
356             face_neighbors.push_back(*f_it);
357         all_face_neighbors[f_it->idx()] = face_neighbors;
358     }
359 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.11 getAllFaceNormals()

```
void getAllFaceNormals (
            TriMesh & mesh,
            vector< TriMesh::Normal > & normals )
```

getAllFaceNormals - normal computation for all faces

Parameters

mesh	input mesh
normals	vector containing normals for all faces

Definition at line 84 of file util.cpp.

Referenced by updateFilteredNormals(), and updateFilteredNormalsGuided().

```
85 {
86     mesh.request_face_normals();
87     mesh.update_face_normals();
88 
89     normals.resize(mesh.n_faces());
90     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
91     {
92         TriMesh::Normal n = mesh.normal(*f_it);
93         normals[f_it->idx()] = n;
94     }
95 }
```

Here is the caller graph for this function:

5.1.4.12 getAllGuidedNeighborsGMNF()

```
void getAllGuidedNeighborsGMNF (
            TriMesh & mesh,
            vector< vector< TriMesh::FaceHandle > > & all_guided_neighbors )
```

getAllGuidedNeighborsGMNF - neighborhood computation for guidance signal

Parameters

mesh	input mesh
all_guided_neighbors	output vector containing neighboring faces IDs for each face of the mesh

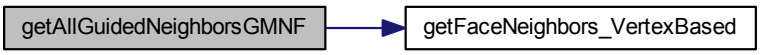
Definition at line 361 of file denoising.cpp.

References getFaceNeighbors_VertexBased().

Referenced by updateFilteredNormalsGuided().

```
362 {
363     vector<TriMesh::FaceHandle> face_neighbors;
364     for (TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
365     {
366         getFaceNeighbors_VertexBased(mesh, *f_it, face_neighbors);
367         face_neighbors.push_back(*f_it);
368         all_guided_neighbors[f_it->idx()] = face_neighbors;
369     }
370 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.13 getAllPoints()

```
void getAllPoints (
            TriMesh & mesh,
            vector< TriMesh::Point > & points )
```

getAllPoints - get all vertex coordinates

Parameters

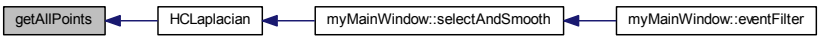
mesh	input mesh
points	all vertex coordinates

Definition at line 135 of file util.cpp.

Referenced by HCLaplacian().

```
136 {
137     points.resize(mesh.n_vertices());
138     for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
139     {
140         TriMesh::Point p = mesh.point(*v_it);
141         points[v_it->idx()] = p;
142     }
143 }
```

Here is the caller graph for this function:



5.1.4.14 getAllVertexAreas()

```
void getAllVertexAreas (
            TriMesh & mesh,
            vector< num_t > & areas )
```

getAllVertexAreas - vertex area computation for all vertices (barycentric areas)

Parameters

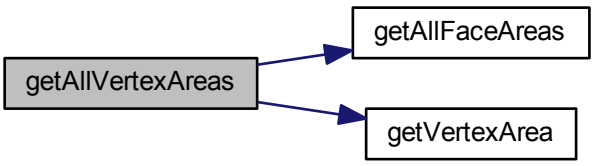
mesh	input mesh
areas	output vector containing all vertex areas

Definition at line 126 of file util.cpp.

References getAllFaceAreas(), and getVertexArea().

```
127 {
128     areas.resize(mesh.n_vertices());
129     vector<num_t> face_areas;
130     getAllFaceAreas(mesh,face_areas);
131     for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
132         areas[v_it->idx()]=getVertexArea(mesh,*v_it,face_areas);
133 }
```

Here is the call graph for this function:



5.1.4.15 getAllVertexNeighbors()

```
void getAllVertexNeighbors (
            TriMesh & mesh,
            int k,
            vector< vector< TriMesh::VertexHandle > > & all_vertex_neighbors )
```

getAllVertexNeighbors: get all neighboring vertices of all vertices regarding a depth k (k-ring)

Parameters

mesh	input mesh
k	depth
all_vertex_neighbors	vector containing all neighbor vectors (output)

Definition at line 72 of file neighborhood.cpp.

References getVertexNeighbors().

```
73 {
74     all_vertex_neighbors.resize(mesh.n_vertices());
75     for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
76     {
77         vector<TriMesh::VertexHandle> vertex_neighbor;
78         getVertexNeighbors(mesh, *v_it, k, vertex_neighbor);
79         all_vertex_neighbors[v_it->idx()] = vertex_neighbor;
80     }
81 }
```

Here is the call graph for this function:

5.1.4.16 getArea()

```
num_t getArea (
              TriMesh & mesh )
```

getArea - area computation of a mesh (sum of triangle areas)

Parameters

mesh	input mesh

Returns

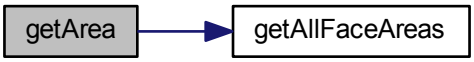
total area

Definition at line 3 of file util.cpp.

References getAllFaceAreas().

```
4 {
5     vector<num_t> areas;
6     getAllFaceAreas(mesh,areas);
7     num_t sum = 0;
8     for (size_t i=0;i<areas.size();i++)
9         sum+=areas[i];
10     return sum;
11 }
```

Here is the call graph for this function:



5.1.4.17 getAverageEdgeLength()

```
num_t getAverageEdgeLength (
            TriMesh & mesh )
```

getAverageEdgeLength - average edge length computation of a mesh

Parameters

mesh	input mesh

Returns

average edge length

Definition at line 43 of file util.cpp.

```
44 {
45     num_t average_edge_length = 0.0;
46     for(TriMesh::EdgeIter e_it = mesh.edges_begin(); e_it != mesh.edges_end(); e_it++)
47         average_edge_length += mesh.calc_edge_length(*e_it);
48     num_t edgeNum = (num_t)mesh.n_edges();
49     average_edge_length /= edgeNum;
50
51     return average_edge_length;
52 }
```

5.1.4.18 getConsistenciesAndMeanNormals()

```
void getConsistenciesAndMeanNormals (
            TriMesh & mesh,
            vector< vector< TriMesh::FaceHandle > > & all_guided_neighbors,
            vector< num_t > & face_areas,
            vector< TriMesh::Normal > & normals,
            vector< pair< num_t, TriMesh::Normal > > & consistencies_and_mean_normals )
```

getConsistenciesAndMeanNormals - consistency and mean normal computation for all patches

Parameters

mesh	input mesh
all_guided_neighbors	all neighborhoods defining the patches
face_areas	all face areas
normals	all face normals
consistencies_and_mean_normals	output vector containing the consistency and mean normal of all patches

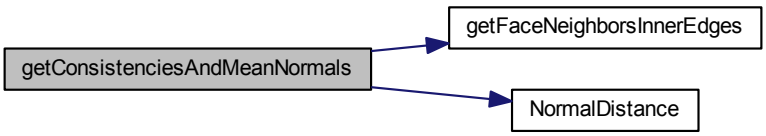
Definition at line 399 of file denoising.cpp.

References getFaceNeighborsInnerEdges(), and NormalDistance().

Referenced by getGuidedNormals().

```
402 {
403     const num_t epsilon = 1.0e-9f;
404 
405     for (TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
406     {
407         int index = f_it->idx();
408         vector<TriMesh::FaceHandle> face_neighbors = all_guided_neighbors[index];
409         num_t metric = 0.0f;
410         TriMesh::Normal average_normal(0.0f, 0.0f, 0.0f);
411         num_t maxdiff = -1.0f;
412 
413         for (int i = 0; i < (int)face_neighbors.size(); i++)
414         {
415             int index_i = face_neighbors[i].idx();
416             num_t area_weight = face_areas[index_i];
417             TriMesh::Normal ni = normals[index_i];
418             average_normal += ni * area_weight;
419 
420             for (int j = i + 1; j < (int)face_neighbors.size(); j++)
421             {
422                 int index_j = face_neighbors[j].idx();
423                 TriMesh::Normal nj = normals[index_j];
424                 num_t diff = NormalDistance(ni, nj);
425 
426                 if (diff > maxdiff)
427                 {
428                     maxdiff = diff;
429                 }
430             }
431         }
432 
433         vector<TriMesh::EdgeHandle> inner_edge_handles;
434         getFaceNeighborsInnerEdges(mesh, face_neighbors, inner_edge_handles);
435         num_t sum_tv = 0.0, max_tv = -1.0;
436         for (int i = 0; i < (int)inner_edge_handles.size(); i++)
437         {
438             TriMesh::HalfedgeHandle heh = mesh.halfedge_handle(inner_edge_handles[i], 0);
439             TriMesh::FaceHandle f = mesh.face_handle(heh);
440             TriMesh::Normal n1 = normals[f.idx()];
441             TriMesh::HalfedgeHandle heho = mesh.opposite_halfedge_handle(heh);
442             TriMesh::FaceHandle fo = mesh.face_handle(heho);
443             TriMesh::Normal n2 = normals[fo.idx()];
444             num_t current_tv = NormalDistance(n1, n2);
445             max_tv = (current_tv > max_tv) ? current_tv : max_tv;
446             sum_tv += current_tv;
447         }
448         average_normal.normalize_cond();
449         metric = maxdiff * max_tv / (sum_tv + epsilon);
450 
451         consistencies_and_mean_normals[index] = make_pair(metric, average_normal);
452     }
453 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.19 getFaceNeighbors_EdgeBased()

```
void getFaceNeighbors_EdgeBased (
            TriMesh & mesh,
            TriMesh::FaceHandle fh,
            vector< TriMesh::FaceHandle > & face_neighbors )
```

Face neighborhood.

getFaceNeighbors_EdgeBased: get neighboring faces based on face edges

Parameters

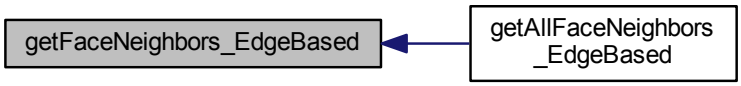
mesh	input mesh
fh	evaluated face
face_neighbors	vector containing neighbors (output)

Definition at line 99 of file neighborhood.cpp.

Referenced by getAllFaceNeighbors_EdgeBased().

```
100 {
101     face_neighbors.clear();
102     for(TriMesh::FaceFaceIter ff_it = mesh.ff_iter(fh); ff_it.is_valid(); ff_it++)
103         face_neighbors.push_back(*ff_it);
104 }
```

Here is the caller graph for this function:



5.1.4.20 getFaceNeighbors_RadiusBased()

```
void getFaceNeighbors_RadiusBased (
            TriMesh & mesh,
            TriMesh::FaceHandle fh,
            num_t radius,
            vector< TriMesh::FaceHandle > & face_neighbors )
```

getFaceNeighbors_RadiusBased: get neighboring faces regarding a radius

Parameters

mesh	input mesh
fh	evaluated face
radius	radius
face_neighbors	vector containing neighbors (output)

Definition at line 124 of file neighborhood.cpp.

References getFaceNeighbors_VertexBased().

Referenced by getAllFaceNeighborsGMNF().

```
125 {
126     TriMesh::Point ci = mesh.calc_face_centroid(fh);
127     vector<bool> flag((int)mesh.n_faces(), false);
128 
129     face_neighbors.clear();
130     flag[fh.idx()] = true;
131     queue<TriMesh::FaceHandle> queue_face_handle;
132     queue_face_handle.push(fh);
133 
134     vector<TriMesh::FaceHandle> temp_face_neighbors;
135     while(!queue_face_handle.empty())
136     {
137         TriMesh::FaceHandle temp_face_handle_queue = queue_face_handle.front();
138         if(temp_face_handle_queue != fh)
139             face_neighbors.push_back(temp_face_handle_queue);
140         queue_face_handle.pop();
141         getFaceNeighbors_VertexBased(mesh, temp_face_handle_queue,
    temp_face_neighbors);
142         for(int i = 0; i < (int)temp_face_neighbors.size(); i++)
143         {
144             TriMesh::FaceHandle temp_face_handle = temp_face_neighbors[i];
145             if(!flag[temp_face_handle.idx()])
146             {
```

```
147                 TriMesh::Point cj = mesh.calc_face_centroid(temp_face_handle);
148                 num_t distance = (ci - cj).length();
149                 if(distance <= radius)
150                     queue_face_handle.push(temp_face_handle);
151                 flag[temp_face_handle.idx()] = true;
152             }
153         }
154     }
155 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

5.1.4.21 getFaceNeighbors_VertexBased()

```
void getFaceNeighbors_VertexBased (
            TriMesh & mesh,
            TriMesh::FaceHandle fh,
            vector< TriMesh::FaceHandle > & face_neighbors )
```

getFaceNeighbors_VertexBased: get neighboring faces based on face vertices

Parameters

mesh	input mesh
fh	evaluated face
face_neighbors	vector containing neighbors (output)

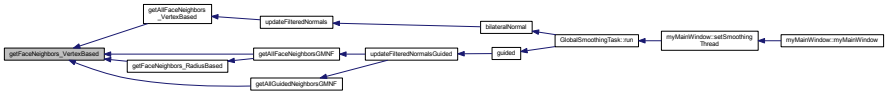
Definition at line 106 of file neighborhood.cpp.

Referenced by getAllFaceNeighbors_VertexBased(), getAllFaceNeighborsGMNF(), getAllGuidedNeighborsGMN↩F(), and getFaceNeighbors_RadiusBased().

```
107 {
108     face_neighbors.clear();
109     set<int> neighbor_face_index; neighbor_face_index.clear();
110     for(TriMesh::FaceVertexIter fv_it = mesh.fv_begin(fh); fv_it.is_valid(); fv_it++)
111     {
112         for(TriMesh::VertexFaceIter vf_it = mesh.vf_iter(*fv_it); vf_it.is_valid(); vf_it++)
```

```
113         {
114             if((*vf_it) != fh)
115                 neighbor_face_index.insert(vf_it->idx());
116         }
117     }
118     for(set<int>::iterator iter = neighbor_face_index.begin(); iter != neighbor_face_index.end(); ++ iter)
119     {
120         face_neighbors.push_back(TriMesh::FaceHandle(*iter));
121     }
122 }
```

Here is the caller graph for this function:



5.1.4.22 getFaceNeighborsInnerEdges()

```
void getFaceNeighborsInnerEdges (
            TriMesh & mesh,
            vector< TriMesh::FaceHandle > & face_neighbors,
            vector< TriMesh::EdgeHandle > & inner_edges )
```

getFaceNeighborsInnerEdges - get all inner edges of a face neighborhood

Parameters

mesh	input mesh
face_neighbors	face neighborhood
inner_edges	output vector containing all inner edges

Definition at line 372 of file denoising.cpp.

Referenced by getConsistenciesAndMeanNormals().

```
373 {
374     inner_edges.clear();
375     vector<bool> edge_flags((int)mesh.n_edges(), false);
376     vector<bool> face_flags((int)mesh.n_faces(), false);
377 
378     for (int i = 0; i < (int)face_neighbors.size(); i++)
379         face_flags[face_neighbors[i].idx()] = true;
380 
381     for (int i = 0; i < (int)face_neighbors.size(); i++)
382     {
383         for (TriMesh::FaceEdgeIter fe_it = mesh.fe_iter(face_neighbors[i]); fe_it.is_valid(); fe_it++)
384         {
385             if ((!edge_flags[fe_it->idx()]) && (!mesh.is_boundary(*fe_it)))
386             {
387                 edge_flags[fe_it->idx()] = true;
388                 TriMesh::HalfedgeHandle heh = mesh.halfedge_handle(*fe_it, 0);
389                 TriMesh::FaceHandle f = mesh.face_handle(heh);
390                 TriMesh::HalfedgeHandle heho = mesh.opposite_halfedge_handle(heh);
391                 TriMesh::FaceHandle fo = mesh.face_handle(heho);
392                 if (face_flags[f.idx()] && face_flags[fo.idx()])
393                     inner_edges.push_back(*fe_it);
394             }
395         }
396     }
397 }
```

Here is the caller graph for this function:

5.1.4.23 getFaceVertexAngle()

```
void getFaceVertexAngle (
            TriMesh & mesh,
            TriMesh::FaceHandle fh,
            TriMesh::VertexHandle vh,
            num_t & angle )
```

getFaceVertexAngle - angle computation for a given vertex included in a face

Parameters

mesh	input mesh
fh	corresponding face handle
vh	corresponding vertex handle
angle	angle

Definition at line 97 of file util.cpp.

```
98 {
99     TriMesh::FaceVertexIter verts = mesh.fv_iter(fh);
100     TriMesh::VertexHandle vh1,vh2;
101     if (vh == *verts)
102     {
103         vh1 = *(++verts);
104         vh2 = *(++verts);
105     }
106     else
107     {
108         vh1 = *verts;
109         vh2 = (vh == *(++verts)) ? *(++verts) : *verts;
110     }
111     TriMesh::Point v1 = mesh.point(vh) - mesh.point(vh1);
112     TriMesh::Point v2 = mesh.point(vh) - mesh.point(vh2);
113     TriMesh::Point v3 = (mesh.point(vh1) - mesh.point(vh2));
114     v1.normalize_cond(); v2.normalize_cond();
115     angle = acos(v1[0] * v2[0] + v1[1] * v2[1] + v1[2] * v2[2]);
116 }
```

5.1.4.24 getGuidedNormals()

```
void getGuidedNormals (
            TriMesh & mesh,
            vector< vector< TriMesh::FaceHandle > > & all_guided_neighbors,
            vector< num_t > & face_areas,
            vector< TriMesh::Normal > & normals,
            vector< pair< num_t, TriMesh::Normal > > consistencies_and_mean_normals,
            vector< TriMesh::Normal > & guided_normals )
```

getGuidedNormals - guided normal computation for each face of the mesh

Parameters

mesh	input mesh
all_guided_neighbors	all neighborhoods defining the patches
face_areas	all face areas
normals	all face normals
consistencies_and_mean_normals	consistencies and mean normals of all patches
guided_normals	output vector containing guided normals for all faces

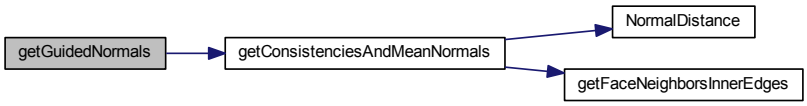
Definition at line 455 of file denoising.cpp.

References getConsistenciesAndMeanNormals().

Referenced by updateFilteredNormalsGuided().

```
459 {
460     getConsistenciesAndMeanNormals(mesh, all_guided_neighbors, face_areas,
      normals, consistencies_and_mean_normals);
461 
462     for (TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
463     {
464         vector<TriMesh::FaceHandle> face_neighbors = all_guided_neighbors[f_it->idx()];
465         num_t min_consistency = 1.0e8f;
466         int min_idx = 0;
467         for (int i = 0; i < (int)face_neighbors.size(); i++)
468         {
469             num_t current_consistency = consistencies_and_mean_normals[face_neighbors[i].idx()].first;
470             if (min_consistency > current_consistency){
471                 min_consistency = current_consistency;
472                 min_idx = i;
473             }
474         }
475         TriMesh::FaceHandle min_face_handle = face_neighbors[min_idx];
476         guided_normals[f_it->idx()] = consistencies_and_mean_normals[min_face_handle.idx()].second;
477     }
478 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.25 getRadius()

```
num_t getRadius (
            TriMesh & mesh,
            num_t scalar )
```

getRadius - computation of radius for radius-based face neighborhood, based on the average of face centroid distances (only between adjacent faces) and multiplied by a scalar

Parameters

mesh	input mesh
scalar	

Returns

radius value

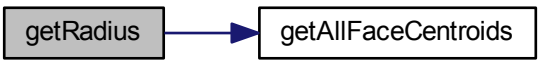
Definition at line 71 of file denoising.cpp.

References getAllFaceCentroids().

Referenced by updateFilteredNormalsGuided().

```
72 {
73     vector<TriMesh::Point> centroids;
74     getAllFaceCentroids(mesh, centroids);
75 
76     num_t radius = 0.0f;
77     num_t num = 0.0f;
78     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
79     {
80         TriMesh::Point fi = centroids[f_it->idx()];
81         for(TriMesh::FaceFaceIter ff_it = mesh.ff_iter(*f_it); ff_it.is_valid(); ff_it++)
82         {
83             TriMesh::Point fj = centroids[ff_it->idx()];
84             radius += (fj - fi).length();
85             num++;
86         }
87     }
88     return radius * scalar / num;
89 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.1.4.26 getSigmaC()

```
num_t getSigmaC (
            TriMesh & mesh,
            vector< TriMesh::Point > & face_centroids,
            num_t sigma_c_scalar )
```

getSigmaC - sigma_c computation based on the average of face centroid distances (only between adjacent faces), and multiplied by a scalar

Parameters

mesh	input mesh
face_centroids	precomputed face centroids
sigma_c_scalar	scalar

Returns

value of sigma_c

Definition at line 53 of file denoising.cpp.

Referenced by updateFilteredNormals(), and updateFilteredNormalsGuided().

```
54 {
55     num_t sigma_c = 0.0f;
56     num_t num = 0.0f;
57     for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
58     {
59         TriMesh::Point ci = face_centroids[f_it->idx()];
60         for(TriMesh::FaceFaceIter ff_it = mesh.ff_iter(*f_it); ff_it.is_valid(); ff_it++)
61         {
62             TriMesh::Point cj = face_centroids[ff_it->idx()];
63             sigma_c += (ci - cj).length();
64             num++;
65         }
66     }
67     sigma_c *= sigma_c_scalar / num;
68     return sigma_c;
69 }
```

Here is the caller graph for this function:

5.1.4.27 getVertexArea()

```
num_t getVertexArea (
            TriMesh & mesh,
            TriMesh::VertexHandle vh,
            vector< num_t > & areas )
```

getVertexArea - vertex area computation (barycentric area)

Parameters

mesh	input mesh
vh	input vertex (vertex handle)
areas	all precomputed triangle areas

Returns

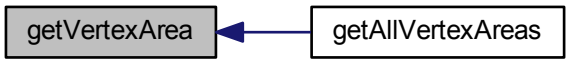
vertex area

Definition at line 118 of file util.cpp.

Referenced by getAllVertexAreas().

```
119 {
120     num_t area_sum = 0.0f;
121     for(TriMesh::VertexFaceIter vf_iter = mesh.vf_iter(vh);vf_iter.is_valid();vf_iter++)
122         area_sum+=areas[vf_iter->idx()];
123     return area_sum*(1.0f/3.0f);
124 }
```

Here is the caller graph for this function:



5.1.4.28 getVertexNeighbors()

```
void getVertexNeighbors (
              TriMesh & mesh,
              TriMesh::VertexHandle vh,
              int k,
              vector< TriMesh::VertexHandle > & vertex_neighbors )
```

Vertex neighborhood.

getVertexNeighbors: get all neighboring vertices of a given vertex regarding a depth k (k-ring)

Parameters

mesh	input mesh
vh	evaluated vertex
k	depth
vertex_neighbors	vector containing neighbors (output)

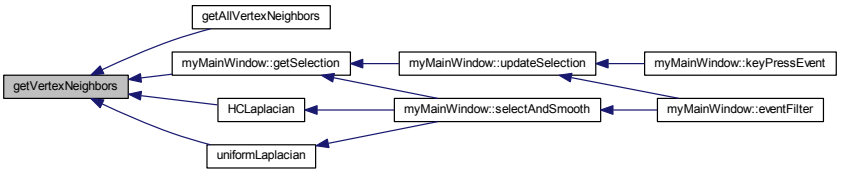
Definition at line 7 of file neighborhood.cpp.

Referenced by getAllVertexNeighbors(), myMainWindow::getSelection(), HCLaplacian(), and uniformLaplacian().

```
9 {
```

```
10      vertex_neighbors.clear();
11      vector<bool> mark(mesh.n_vertices(),false);
12      queue<TriMesh::VertexHandle> queue_vertex_handle;
13      queue<int> queue_depth_handle;
14      mark[vh.idx()] = true;
15      queue_vertex_handle.push(vh);
16      queue_depth_handle.push(0);
17      TriMesh::Point ci = mesh.point(vh);
18      while(!queue_vertex_handle.empty())
19      {
20          TriMesh::VertexHandle vh = queue_vertex_handle.front();
21          int current_depth = queue_depth_handle.front();
22          vertex_neighbors.push_back(vh);
23          queue_vertex_handle.pop();
24          queue_depth_handle.pop();
25          for(TriMesh::VertexVertexIter vv_it = mesh.vv_iter(vh); vv_it.is_valid(); ++vv_it)
26          {
27              TriMesh::VertexHandle vh_neighbor = *vv_it;
28              if(mark[vh_neighbor.idx()] == false)
29              {
30                  int new_depth = current_depth + 1;
31                  if(new_depth <= k)
32                  {
33                      queue_vertex_handle.push(vh_neighbor);
34                      queue_depth_handle.push(new_depth);
35                  }
36                  mark[vh_neighbor.idx()] = true;
37              }
38          }
39      }
40  }
```

Here is the caller graph for this function:



5.1.4.29 getVolume()

```
num_t getVolume (
              TriMesh & mesh )
```

getVolume - volume computation of a mesh

Parameters

mesh	input mesh

Returns

total area

Definition at line 13 of file util.cpp.

```
13                                  {
14
15      typedef TriMesh::HalfedgeHandle hh_t;
16      typedef TriMesh::VertexHandle vh_t;
17      typedef TriMesh::Point p_t;
18
19      num_t volume = 0.0f;
20      num_t three_inv = 1.0f / 3.0f;
21      TriMesh::FaceIter f_it;
22      for (f_it = mesh.faces_begin(); f_it != mesh.faces_end(); ++f_it)
23      {
24          hh_t hh = mesh.halfedge_handle(*f_it);
25          hh_t hhn = mesh.next_halfedge_handle(hh);
26
27          vh_t v0(mesh.from_vertex_handle(hh));
28          vh_t v1(mesh.to_vertex_handle(hh));
29          vh_t v2(mesh.to_vertex_handle(hhn));
30
31          p_t p0(mesh.point(v0));
32          p_t p1(mesh.point(v1));
33          p_t p2(mesh.point(v2));
34
35          p_t g = (p0 + p1 + p2) * three_inv;
36          p_t n = (p1 - p0) % (p2 - p0);
37          volume += (g|n);
38      }
39      volume *= 1.0f / 6.0f;
40      return volume;
41  }
```

5.1.4.30 guided()

```
TriMesh guided (
              TriMesh _mesh,
              int normal_iteration_number,
              int vertex_iteration_number,
              num_t sigma_c_scalar,
              num_t sigma_s,
              num_t radius_scalar )
```

guided - Denoise the input mesh using Guided Mesh Normal Filtering Algorithm

Parameters

_mesh	input mesh
normal_iteration_number	number of iterations for normal field filtering
vertex_iteration_number	number of iterations for vertex updating
sigma_c_scalar	bilateral normal field filtering influence regarding a $sigma_c$ based on average edge length (spatial distance influence)
sigma_s	bilateral normal field filtering parameter $sigma_s$
radius_scalar	average radius ratio for face neighborhood computation

Returns

denoised mesh

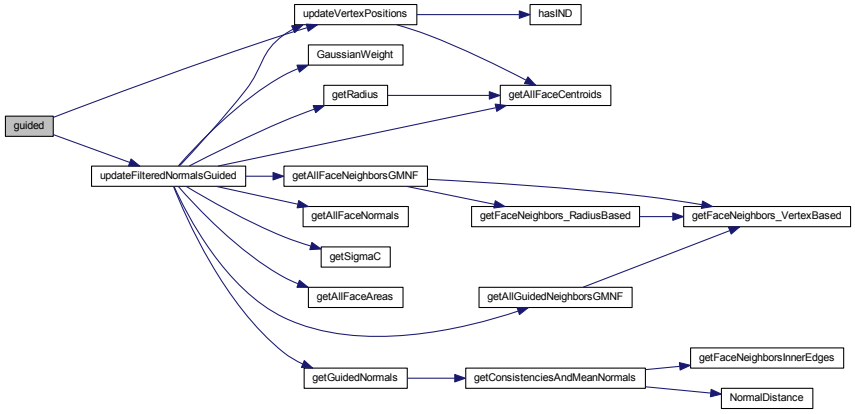
Definition at line 531 of file denoising.cpp.

References updateFilteredNormalsGuided(), and updateVertexPositions().

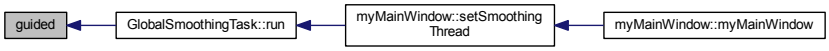
Referenced by GlobalSmoothingTask::run().

```
532 {
533     TriMesh mesh = _mesh;
534     if (mesh.n_vertices() == 0)
535         return mesh;
536     vector<TriMesh::Normal> filtered_normals;
537     updateFilteredNormalsGuided(mesh, filtered_normals, radius_scalar,
      sigma_c_scalar, normal_iteration_number, sigma_s, vertex_iteration_number);
538     updateVertexPositions(mesh, filtered_normals, vertex_iteration_number, true);
539     mesh.request_face_normals();
540     mesh.request_vertex_normals();
541     mesh.update_normals();
542     return mesh;
543 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.31 HCLaplacian() [1/2]

```
TriMesh HCLaplacian (
            TriMesh & _mesh,
            int iteration_number,
            num_t alpha,
            num_t beta )
```

HC Laplacian smoothing (Vollmer et al.)

HCLaplacian - denoise the input mesh using HC Laplacian Smoothing Algorithm

Parameters

_mesh	input mesh
iteration_number	number of iterations
alpha	alpha defined in HC Laplacian Smoothing paper
beta	beta defined in HC Laplacian Smoothing paper

Returns

denoised mesh

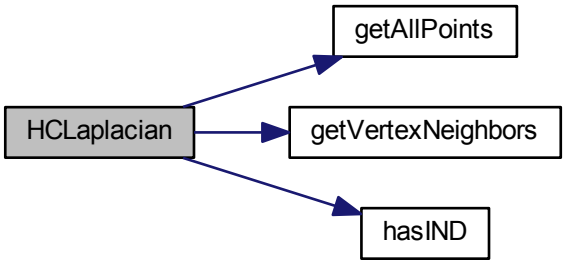
Definition at line 161 of file denoising.cpp.

References getAllPoints(), getVertexNeighbors(), and hasIND().

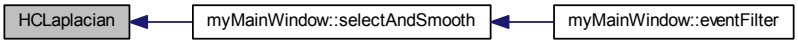
Referenced by myMainWindow::selectAndSmooth().

```
162 {
163     TriMesh mesh = _mesh;
164     vector<TriMesh::Point> original_points;
165     getAllPoints(mesh,original_points);
166     TriMesh temp_mesh_p;
167     vector<TriMesh::Point> temp_points_p(original_points);
168     vector<TriMesh::Point> uniform_displacement_points;
169     uniform_displacement_points.resize(mesh.n_vertices());
170     for(int iter = 0; iter < iteration_number; iter++)
171     {
172         vector<TriMesh::Point> temp_points_q(temp_points_p);
173         for (TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
174         {
175             vector<TriMesh::VertexHandle> vertex_neighbors;
176             getVertexNeighbors(_mesh, *v_it, 1, vertex_neighbors);
177             if (vertex_neighbors.size()>0)
178             {
179                 num_t weight = 1.0f / ((num_t)vertex_neighbors.size());
180                 TriMesh::Point sum(0.0f, 0.0f, 0.0f);
181                 for (int i = 0; i<(int)(vertex_neighbors.size()); i++)
182                     sum = sum + ((mesh.point(vertex_neighbors[i]) - mesh.point(*v_it))*weight);
183                 uniform_displacement_points[v_it->idx()] = sum;
184                 temp_points_p[v_it->idx()] += sum;
185             }
186             else
187                 uniform_displacement_points[v_it->idx()] = TriMesh::Point(0,0,0);
188         }
189         for (TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
190             temp_points_p[v_it->idx()]=uniform_displacement_points[v_it->idx()] + mesh.point(*v_it);
191         vector<TriMesh::Point> temp_points_b(temp_points_p.size());
192         for(size_t i = 0;i<temp_points_p.size();i++)
193             temp_points_b[i]=temp_points_p[i]-(original_points[i]*alpha+(1.0f-alpha)*temp_points_q[i]);
194         for(TriMesh::VertexIter v_it = mesh.vertices_begin();v_it!=mesh.vertices_end();v_it++)
195         {
196             TriMesh::Point t_sum(0,0,0);
197             num_t num = 0;
198             for(TriMesh::VertexVertexIter vv_it = mesh.vv_iter(*v_it);vv_it.is_valid();vv_it++,num+=1.0f)
199                 t_sum+=temp_points_b[vv_it->idx()];
200
201             TriMesh::Point disp = (beta*temp_points_b[v_it->idx()] + ((1.0f - beta) / num)*t_sum);
202             if (!hasIND(disp))
203                 temp_points_p[v_it->idx()]=temp_points_p[v_it->idx()]-disp;
204         }
205     }
206     for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
207         mesh.set_point(*v_it, temp_points_p[v_it->idx()]);
208     mesh.request_face_normals();
209     mesh.request_vertex_normals();
210     mesh.update_normals();
211     return mesh;
212 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.32 HCLaplacian() [2/2]

```
TriMesh HCLaplacian (
            TriMesh & _mesh,
            int iteration_number,
            num_t alpha,
            num_t beta,
            vector< size_t > & vertex_ids )
```

HCLaplacian - denoise a subset of vertices of an input mesh using HC Laplacian Smoothing Algorithm.

Parameters

_mesh	input mesh
iteration_number	number of iterations
alpha	alpha defined in HC Laplacian Smoothing paper
beta	beta defined in HC Laplacian Smoothing paper
vertex_ids	subset of vertex IDs to be denoised

Returns

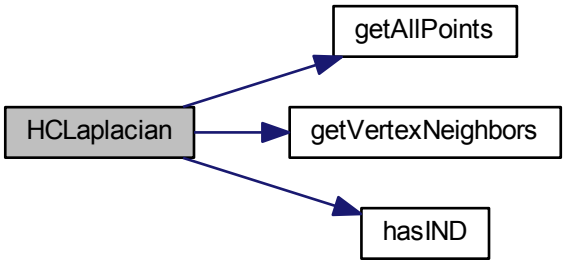
denoised mesh (taking into account only the subset of IDs)

Definition at line 214 of file denoising.cpp.

References getAllPoints(), getVertexNeighbors(), and hasIND().

```
215 {
216     TriMesh mesh = _mesh;
217     vector<TriMesh::Point> original_points;
218     getAllPoints(mesh, original_points);
219     TriMesh temp_mesh_p;
220     vector<TriMesh::Point> temp_points_p(original_points);
221     vector<TriMesh::Point> uniform_displacement_points;
222     uniform_displacement_points.resize(mesh.n_vertices());
223     for (int iter = 0; iter < iteration_number; iter++)
224     {
225         vector<TriMesh::Point> temp_points_q(temp_points_p);
226         for (size_t i = 0; i < vertex_ids.size(); i++)
227         {
228             TriMesh::VertexHandle vh((int)vertex_ids[i]);
229             vector<TriMesh::VertexHandle> vertex_neighbors;
230             getVertexNeighbors(_mesh, vh, 1, vertex_neighbors);
231             if (vertex_neighbors.size()>0)
232             {
233                 num_t weight = 1.0f / ((num_t)vertex_neighbors.size());
234                 TriMesh::Point sum(0.0f, 0.0f, 0.0f);
235                 for (int i = 0; i<(int)(vertex_neighbors.size()); i++)
236                     sum = sum + ((mesh.point(vertex_neighbors[i]) - mesh.point(vh))*weight);
237                 uniform_displacement_points[vh.idx()] = sum;
238                 temp_points_p[vh.idx()] += sum;
239             }
240             else
241                 uniform_displacement_points[vh.idx()] = TriMesh::Point(0, 0, 0);
242         }
243         for (size_t i = 0; i < vertex_ids.size(); i++)
244         {
245             TriMesh::VertexHandle vh((int)vertex_ids[i]);
246             temp_points_p[vh.idx()] = uniform_displacement_points[vh.idx()] + mesh.point(vh);
247         }
248         vector<TriMesh::Point> temp_points_b(temp_points_p.size());
249         for (size_t i = 0; i<temp_points_p.size(); i++)
250             temp_points_b[i] = temp_points_p[i] - (original_points[i] * alpha + (1.0f - alpha)*
    temp_points_q[i]);
251         for (size_t i = 0; i < vertex_ids.size(); i++)
252         {
253             TriMesh::VertexHandle vh((int)vertex_ids[i]);
254             TriMesh::Point t_sum(0, 0, 0);
255             num_t num = 0;
256             for (TriMesh::VertexVertexIter vv_it = mesh.vv_iter(vh); vv_it.is_valid(); vv_it++, num += 1.0f
    )
257                 t_sum += temp_points_b[vv_it->idx()];
258
259             TriMesh::Point disp = (beta*temp_points_b[vh.idx()] + ((1.0f - beta) / num)*t_sum);
260             if (!hasIND(disp))
261                 temp_points_p[vh.idx()] = temp_points_p[vh.idx()] - disp;
262         }
263     }
264     for (size_t i = 0; i < vertex_ids.size(); i++)
265     {
266         TriMesh::VertexHandle vh((int)vertex_ids[i]);
267         mesh.set_point(vh, temp_points_p[vh.idx()]);
268     }
269     mesh.request_face_normals();
270     mesh.request_vertex_normals();
271     mesh.update_normals();
272     return mesh;
273 }
```

Here is the call graph for this function:



5.1.4.33 importMesh()

```
bool importMesh (
            TriMesh & mesh,
            string filename )
```

importMesh - read mesh file (.obj, .off, .ply, .stl)

Parameters

mesh	- output mesh
filename	- input file name

Returns

success

Definition at line 3 of file iomesh.cpp.

Referenced by ShapeData::loadFromFile(), and myDataManager::loadInputMesh().

```
4 {
5     OpenMesh::IO::Options opt;
6     //opt += OpenMesh::IO::Options::VertexColor;
7     if(!OpenMesh::IO::read_mesh(mesh, filename, opt))
8         return false;
9     return true;
10 }
```

Here is the caller graph for this function:

5.1.4.34 NormalDistance()

```
num_t NormalDistance (
            const TriMesh::Normal & n1,
            const TriMesh::Normal & n2 )  [inline]
```

NormalDistance - computation of normal distance: $|n1 - n2|$.

Parameters

n1	- input normal 1
n2	- input normal 2

Returns

normal distance

Definition at line 109 of file util.h.

Referenced by getConsistenciesAndMeanNormals().

```
110 {
111     return (n1 - n2).length();
112 }
```

Here is the caller graph for this function:

5.1.4.35 uniformLaplacian() [1/2]

```
TriMesh uniformLaplacian (
            TriMesh & _mesh,
            int iteration_number,
            num_t scale )
```

Uniform Laplacian smoothing.

uniformLaplacian - denoise the input mesh using Uniform Laplacian Smoothing Algorithm

Parameters

_mesh	input mesh
iteration_number	number of iterations
scale	displacement influence for each iteration

Returns

denoised mesh

Isotropic mesh denoising algorithms

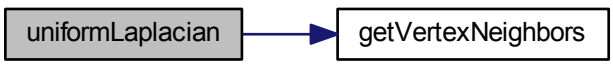
Definition at line 101 of file denoising.cpp.

References getVertexNeighbors().

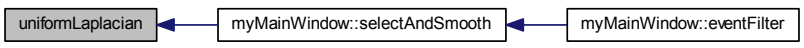
Referenced by myMainWindow::selectAndSmooth().

```
102 {
103     TriMesh mesh = _mesh;
104     vector<TriMesh::Point> displacement_points;
105     displacement_points.resize(mesh.n_vertices());
106     for (int iter = 0; iter < iteration_number; iter++)
107     {
108         for (TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
109         {
110             vector<TriMesh::VertexHandle> vertex_neighbors;
111             getVertexNeighbors(_mesh, *v_it, 1, vertex_neighbors);
112             num_t weight = 1.0f / ((num_t)vertex_neighbors.size());
113             TriMesh::Point sum(0.0f, 0.0f, 0.0f);
114             for (int i = 0; i<(int)(vertex_neighbors.size()); i++)
115                 sum = sum + ((mesh.point(vertex_neighbors[i]) - mesh.point(*v_it))*weight);
116             displacement_points[v_it->idx()] = sum;
117         }
118         for (TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
119             mesh.set_point(*v_it, displacement_points[v_it->idx()] * scale + mesh.point(*v_it));
120     }
121     mesh.request_face_normals();
122     mesh.request_vertex_normals();
123     mesh.update_normals();
124     return mesh;
125 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.36 uniformLaplacian() [2/2]

```
TriMesh uniformLaplacian (
            TriMesh & _mesh,
            int iteration_number,
            num_t scale,
            vector< size_t > & vertex_ids )
```

uniformLaplacian - denoise a subset of vertices of an input mesh using Uniform Laplacian Smoothing Algorithm

Parameters

_mesh	input mesh
iteration_number	number of iterations
scale	displacement influence for each iteration
vertex_ids	subset of vertex IDs to be denoised

Returns

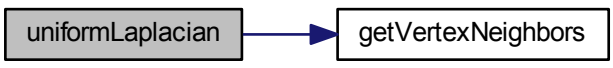
denoised mesh (taking into account only the subset of IDs)

Definition at line 127 of file denoising.cpp.

References getVertexNeighbors().

```
128 {
129     TriMesh mesh = _mesh;
130     vector<TriMesh::Point> displacement_points;
131     displacement_points.resize(mesh.n_vertices());
132     for (int iter = 0; iter < iteration_number; iter++)
133     {
134         for (size_t i = 0; i < vertex_ids.size(); i++)
135         {
136             TriMesh::VertexHandle vh((int)vertex_ids[i]);
137             vector<TriMesh::VertexHandle> vertex_neighbors;
138             getVertexNeighbors(_mesh, vh, 1, vertex_neighbors);
139             num_t weight = 1.0f / ((num_t)vertex_neighbors.size());
140             TriMesh::Point sum(0.0f, 0.0f, 0.0f);
141             for (int i = 0; i<(int)(vertex_neighbors.size()); i++)
142                 sum = sum + ((mesh.point(vertex_neighbors[i]) - mesh.point(vh))*weight);
143             displacement_points[vh.idx()] = sum;
144         }
145         for (size_t i = 0; i < vertex_ids.size(); i++)
146         {
147             TriMesh::VertexHandle vh((int)vertex_ids[i]);
148             mesh.set_point(vh, displacement_points[vh.idx()] * scale + mesh.point(vh));
149         }
150     }
151     mesh.request_face_normals();
152     mesh.request_vertex_normals();
153     mesh.update_normals();
154     return mesh;
155 }
```

Here is the call graph for this function:



5.1.4.37 updateFilteredNormals()

```
void updateFilteredNormals (
            TriMesh & mesh,
            int normal_iteration_number,
            num_t sigma_c_scalar,
            num_t sigma_s,
            vector< TriMesh::Normal > & filtered_normals )
```

Bilateral normal filtering for mesh denoising (Zheng et al.)

updateFilteredNormals - bilateral filtering of the normal field (face based) of a mesh

Parameters

mesh	input mesh
normal_iteration_number	number of iterations
sigma_c_scalar	bilateral filtering influence regarding a $sigma_c$ based on average edge length (spatial distance influence)
sigma_s	bilateral filtering parameter $sigma_s$
filtered_normals	output vector of filtered face normals

Definition at line 279 of file denoising.cpp.

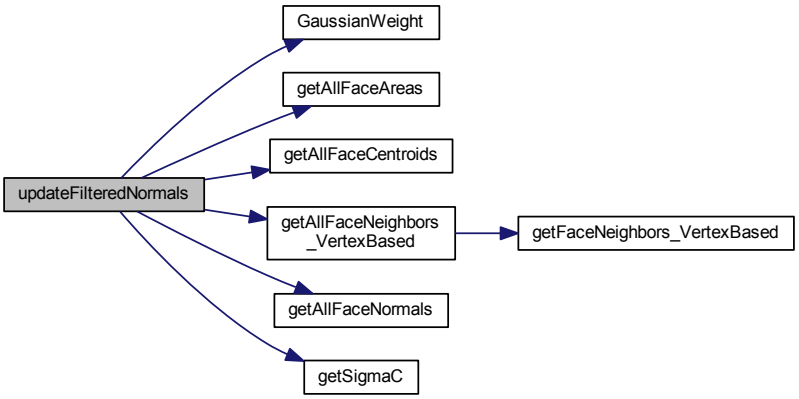
References GaussianWeight(), getAllFaceAreas(), getAllFaceCentroids(), getAllFaceNeighbors_VertexBased(), getAllFaceNormals(), and getSigmaC().

Referenced by bilateralNormal().

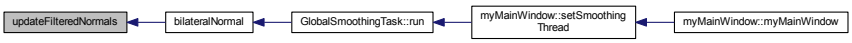
```
280 {
281     filtered_normals.resize(mesh.n_faces());
282 
283     vector< vector<TriMesh::FaceHandle> > all_face_neighbors;
284     getAllFaceNeighbors_VertexBased(mesh,0,all_face_neighbors);
285     vector<TriMesh::Normal> previous_normals;
286     getAllFaceNormals(mesh, previous_normals);
287     vector<num_t> face_areas;
288     getAllFaceAreas(mesh, face_areas);
289     vector<TriMesh::Point> face_centroids;
290     getAllFaceCentroids(mesh, face_centroids);
291 
292     num_t sigma_c = getSigmaC(mesh, face_centroids, sigma_c_scalar);
293 
294     for(int it = 0; it < normal_iteration_number; it++)
295     {
296         for(TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
297         {
298             int face_idx = f_it->idx();
299             TriMesh::Normal ni = previous_normals[face_idx];
300             TriMesh::Point ci = face_centroids[face_idx];
301             vector<TriMesh::FaceHandle> face_neighbors = all_face_neighbors[face_idx];
302             TriMesh::Normal temp_normal(0.0, 0.0, 0.0);
303             num_t weight_sum = 0.0;
304             for (int i = 0; i < (int)face_neighbors.size(); i++)
305             {
306                 int neighbor_idx = face_neighbors[i].idx();
307                 TriMesh::Normal nj = previous_normals[neighbor_idx];
308                 TriMesh::Point cj = face_centroids[neighbor_idx];
309 
310                 num_t spatial_distance = (ci - cj).length();
311                 num_t spatial_weight = GaussianWeight(spatial_distance,sigma_c);
312                 num_t range_distance = (ni - nj).length();
313                 num_t range_weight = GaussianWeight(range_distance,sigma_s);
314 
315                 num_t weight = face_areas[neighbor_idx] * spatial_weight * range_weight;
316                 weight_sum += weight;
317                 temp_normal += nj * weight;
318             }
```

```
319             temp_normal /= weight_sum;
320             temp_normal.normalize_cond();
321             filtered_normals[face_idx] = temp_normal;
322         }
323         previous_normals = filtered_normals;
324     }
325 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.38 updateFilteredNormalsGuided()

```
void updateFilteredNormalsGuided (
            TriMesh & mesh,
            vector< TriMesh::Normal > & filtered_normals,
            num_t radius_scalar,
            num_t sigma_c_scalar,
            int normal_iteration_number,
            num_t sigma_s,
            int vertex_iteration_number )
```

updateFilteredNormalsGuided - guided bilateral filtering of a normal field (face based) of a mesh

Parameters

mesh	input mesh
filtered_normals	guided normals for each face

Parameters

radius_scalar	average radius ratio for face neighborhood computation
sigma_c_scalar	bilateral filtering influence regarding a $sigma_c$ based on average edge length (spatial distance influence)
normal_iteration_number	number of iterations for normal field filtering
sigma_s	bilateral filtering parameter $sigma_s$
vertex_iteration_number	number of iterations for vertex updating

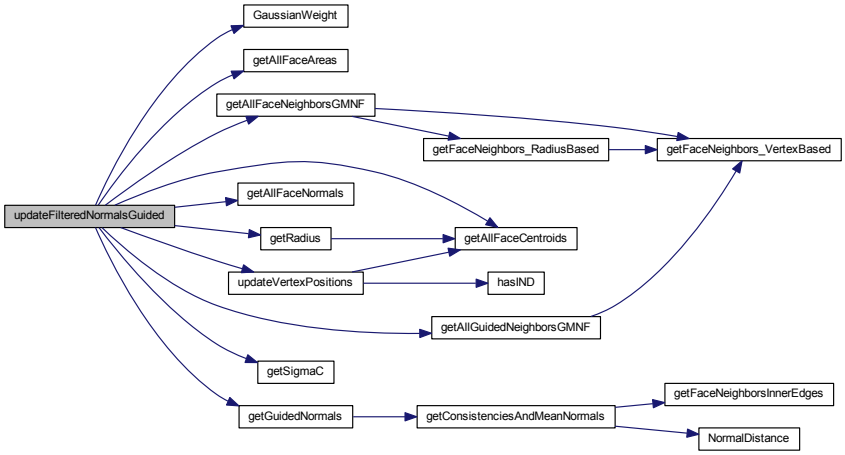
Definition at line 480 of file denoising.cpp.

References GaussianWeight(), getAllFaceAreas(), getAllFaceCentroids(), getAllFaceNeighborsGMNF(), getAll↩FaceNormals(), getAllGuidedNeighborsGMNF(), getGuidedNormals(), getRadius(), getSigmaC(), kRadiusBased, and updateVertexPositions().

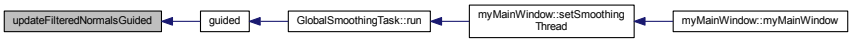
Referenced by guided().

```
482 {
483     filtered_normals.resize((int)mesh.n_faces());
484     bool include_central_face = 1;
485     FaceNeighborType face_neighbor_type = kRadiusBased;
486     num_t radius;
487     radius = getRadius(mesh, radius_scalar);
488     vector<vector<TriMesh::FaceHandle> > all_face_neighbors((int)mesh.n_faces());
489     getAllFaceNeighborsGMNF(mesh, face_neighbor_type, radius, include_central_face,
    all_face_neighbors);
490     vector<vector<TriMesh::FaceHandle> > all_guided_neighbors((int)mesh.n_faces());
491     getAllGuidedNeighborsGMNF(mesh, all_guided_neighbors);
492     getAllFaceNormals(mesh, filtered_normals);
493 
494     vector<num_t> face_areas((int)mesh.n_faces());
495     vector<TriMesh::Point> face_centroids((int)mesh.n_faces());
496     vector<TriMesh::Normal> previous_normals((int)mesh.n_faces());
497     vector<TriMesh::Normal> guided_normals((int)mesh.n_faces());
498     vector<pair<num_t, TriMesh::Normal> > consistencies_and_mean_normals((int)mesh.n_faces());
499     for (int iter = 0; iter < normal_iteration_number; iter++)
500     {
501         getAllFaceCentroids(mesh, face_centroids);
502         num_t sigma_c = getSigmaC(mesh, face_centroids, sigma_c_scalar);
503         getAllFaceAreas(mesh, face_areas);
504         getAllFaceNormals(mesh, previous_normals);
505 
506         getGuidedNormals(mesh, all_guided_neighbors, face_areas, previous_normals,
    consistencies_and_mean_normals, guided_normals);
507 
508         for (TriMesh::FaceIter f_it = mesh.faces_begin(); f_it != mesh.faces_end(); f_it++)
509         {
510             int index = f_it->idx();
511             const vector<TriMesh::FaceHandle> face_neighbors = all_face_neighbors[index];
512             TriMesh::Normal filtered_normal(0.0f, 0.0f, 0.0f);
513             for (int j = 0; j < (int)face_neighbors.size(); j++)
514             {
515                 int current_face_index = face_neighbors[j].idx();
516 
517                 num_t spatial_dis = (face_centroids[index] - face_centroids[current_face_index]).
    length();
518                 num_t spatial_weight = GaussianWeight(spatial_dis, sigma_c);
519                 num_t range_dis = (guided_normals[index] - guided_normals[current_face_index]).length(
    );
520                 num_t range_weight = GaussianWeight(range_dis, sigma_s);
521 
522                 filtered_normal += previous_normals[current_face_index] * (face_areas[current_face_index] *
     spatial_weight * range_weight);
523             }
524             if (face_neighbors.size())
525                 filtered_normals[index] = filtered_normal.normalize_cond();
526         }
527         updateVertexPositions(mesh, filtered_normals, vertex_iteration_number, false);
528     }
529 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.4.39 updateVertexPositions()

```
void updateVertexPositions (
            TriMesh & mesh,
            vector< TriMesh::Normal > & filtered_normals,
            int iteration_number,
            bool fixed_boundary )
```

updateVertexPositions - computes new vertex positions adapted to an input normal field.

Parameters

mesh	input mesh.
filtered_normals	input normal field.
iteration_number	number of iterations of the optimization algorithm (gradient descent approach).
fixed_boundary	determines if a boundary vertex will be updated

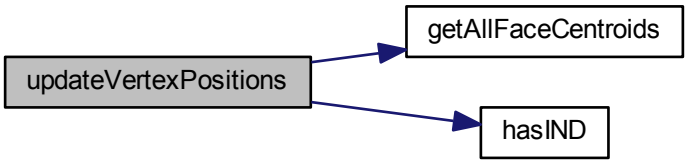
Definition at line 14 of file denoising.cpp.

References getAllFaceCentroids(), and hasIND().

Referenced by bilateralNormal(), guided(), and updateFilteredNormalsGuided().

```
15 {
16     vector<TriMesh::Point> new_points(mesh.n_vertices());
17     vector<TriMesh::Point> centroids;
18 
19     for(int iter = 0; iter < iteration_number; iter++)
20     {
21         getAllFaceCentroids(mesh, centroids);
22         for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
23         {
24             TriMesh::Point p = mesh.point(*v_it);
25             if(fixed_boundary && mesh.is_boundary(*v_it))
26             {
27                 new_points.at(v_it->idx()) = p;
28             }
29             else
30             {
31                 num_t face_num = 0.0f;
32                 TriMesh::Point temp_point(0.0f, 0.0f, 0.0f);
33                 for(TriMesh::VertexFaceIter vf_it = mesh.vf_iter(*v_it); vf_it.is_valid(); vf_it++)
34                 {
35                     TriMesh::Normal temp_normal = filtered_normals[vf_it->idx()];
36                     TriMesh::Point temp_centroid = centroids[vf_it->idx()];
37                     temp_point += temp_normal * (temp_normal | (temp_centroid - p));
38                     face_num++;
39                 }
40                 p += temp_point / face_num;
41                 if (!hasIND(p))
42                 {
43                     new_points.at(v_it->idx()) = p;
44                 }
45                 else new_points.at(v_it->idx()) = p;
46             }
47         }
48         for(TriMesh::VertexIter v_it = mesh.vertices_begin(); v_it != mesh.vertices_end(); v_it++)
49             mesh.set_point(*v_it, new_points[v_it->idx()]);
50     }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.2 Visualization based on OpenGL

Camera.

Data Structures

- class myCamera

  The myCamera class - camera representation in a typical rendering pipeline (i.e. OpenGL)
- class myRenderer

  The myRenderer class - Renderer for multiple shapes (triangular meshes) visualization.
- class myShader

  The myShader class - shader data for OpenGL.
- struct Vertex

  The Vertex struct - single vertex with its corresponding attributes.
- struct ShapeData

  The ShapeData struct - triangular mesh for OpenGL buffers manipulation.

Enumerations

- enum myDrawFlags { e_draw_faces, e_draw_wireframe, e_draw_points, e_draw_selection }

  The myDrawFlags enum - Rendering mode.

5.2.1 Detailed Description

Camera.

Shape data.

Shader.

Renderer.

Module containing OpenGL abstractions for visualization of triangular meshes

5.2.2 Enumeration Type Documentation

5.2.2.1 myDrawFlags

```
enum myDrawFlags
```

The myDrawFlags enum - Rendering mode.

Enumerator

e_draw_faces	draw mesh faces
e_draw_wireframe	draw mesh edges
e_draw_points	draw mesh points (vertices)
e_draw_selection	draw as a selection (red points / customizable)

Definition at line 24 of file myRenderer.h.

```
24                {
25     e_draw_faces,
26     e_draw_wireframe,
27     e_draw_points,
28     e_draw_selection
29 };
```

Chapter 6

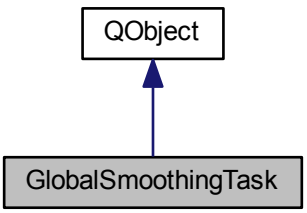
Data Structure Documentation

6.1 GlobalSmoothingTask Class Reference

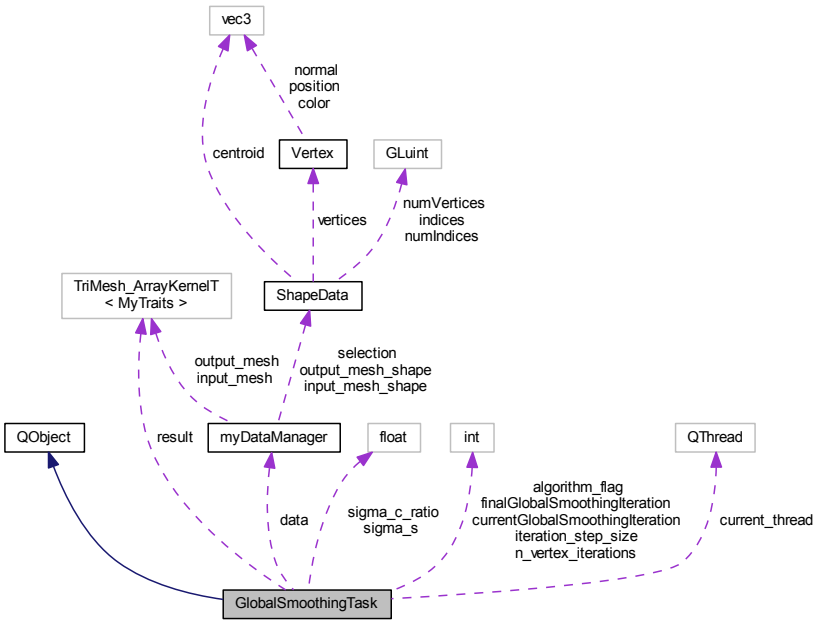
The GlobalSmoothingTask class - global smoothing task.

```
#include <smoothing.h>
```

Inheritance diagram for GlobalSmoothingTask:



Collaboration diagram for GlobalSmoothingTask:



Public Member Functions

- void run ()

  run - run current algorithm
- void finishThread ()

  finishThread - finalize the current thread
- void updateData ()

  updateData - set the current partial result as output mesh

Data Fields

- int currentGlobalSmoothingIteration

  currentGlobalSmoothingIteration - current global smoothing external iteration
- int finalGlobalSmoothingIteration

  finalGlobalSmoothingIteration - final global smoothing external iteration
- int n_vertex_iterations

  n_vertex_iterations - number of vertex iterations for vertex updating step
- int iteration_step_size

  iteration_step_size - number of normal filtering iterations for each step (internal iterations)
- float sigma_c_ratio

  sigma_c_ratio - current $sigma_c ratio$
- float sigma_s

  sigma_s - current $sigma_s$
- TriMesh result

  result - current partial result
- QThread * current_thread

current_thread - thread where this task is running

- myDataManager * data

  data - data manager pointer

- int algorithm_flag

  algorithm_flag - current algotihm flag

6.1.1 Detailed Description

The GlobalSmoothingTask class - global smoothing task.

Definition at line 11 of file smoothing.h.

6.1.2 Member Function Documentation

6.1.2.1 finishThread()

```
void GlobalSmoothingTask::finishThread ( )  [inline]
```

finishThread - finalize the current thread

Definition at line 70 of file smoothing.h.

Referenced by run().

```
71     {
72         current_thread->quit();
73     }
```

Here is the caller graph for this function:

6.1.2.2 run()

```
void GlobalSmoothingTask::run ( )  [inline]
```

run - run current algorithm

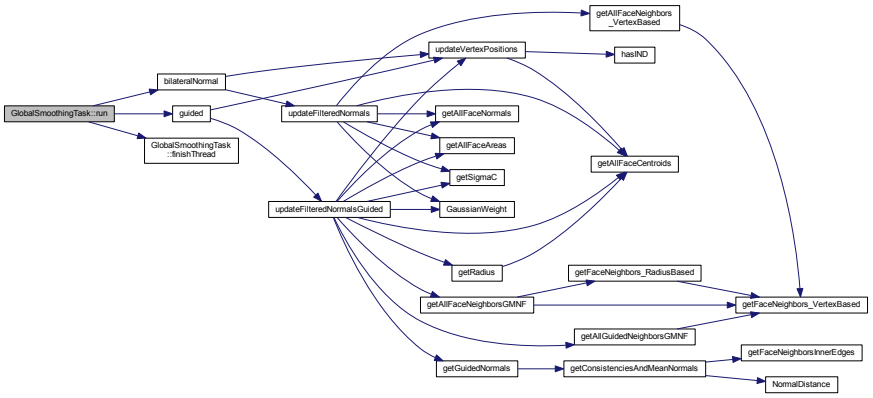
Definition at line 58 of file smoothing.h.

References bilateralNormal(), finishThread(), guided(), and myDataManager::output_mesh.

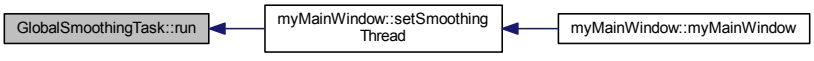
Referenced by myMainWindow::setSmoothingThread().

```
58               {
59         if (algorithm_flag == 0)
60             result = bilateralNormal(data->output_mesh,
      iteration_step_size, n_vertex_iterations,
      sigma_c_ratio, sigma_s);
61         else if (algorithm_flag == 1)
62             result = guided(data->output_mesh,
      iteration_step_size, n_vertex_iterations,
      sigma_c_ratio, sigma_s, 1.5f);
63         else
64             result = bilateralNormal(data->output_mesh,
      iteration_step_size, n_vertex_iterations,
      sigma_c_ratio, sigma_s);
65         finishThread();
66     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.3 updateData()

```
void GlobalSmoothingTask::updateData ( )  [inline]
```

updateData - set the current partial result as output mesh

Definition at line 77 of file smoothing.h.

References myDataManager::output_mesh, and result.

Referenced by myMainWindow::updateGlobalSmoothing().

```
78     {
79         data->output_mesh = result;
80     }
```

Here is the caller graph for this function:

6.1.3 Field Documentation

6.1.3.1 algorithm_flag

```
int GlobalSmoothingTask::algorithm_flag
```

algorithm_flag - current algotihm flag

Definition at line 53 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing().

6.1.3.2 current_thread

```
QThread* GlobalSmoothingTask::current_thread
```

current_thread - thread where this task is running

Definition at line 45 of file smoothing.h.

Referenced by myMainWindow::setSmoothingThread().

6.1.3.3 currentGlobalSmoothingIteration

```
int GlobalSmoothingTask::currentGlobalSmoothingIteration
```

currentGlobalSmoothingIteration - current global smoothing external iteration

Definition at line 17 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing(), and myMainWindow::updateGlobalSmoothing().

6.1.3.4 data

```
myDataManager* GlobalSmoothingTask::data
```

data - data manager pointer

Definition at line 49 of file smoothing.h.

Referenced by myMainWindow::setSmoothingThread().

6.1.3.5 finalGlobalSmoothingIteration

```
int GlobalSmoothingTask::finalGlobalSmoothingIteration
```

finalGlobalSmoothingIteration - final global smoothing external iteration

Definition at line 21 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing(), and myMainWindow::updateGlobalSmoothing().

6.1.3.6 iteration_step_size

```
int GlobalSmoothingTask::iteration_step_size
```

iteration_step_size - number of normal filtering iterations for each step (internal iterations)

Definition at line 29 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing().

6.1.3.7 n_vertex_iterations

```
int GlobalSmoothingTask::n_vertex_iterations
```

n_vertex_iterations - number of vertex iterations for vertex updating step

Definition at line 25 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing().

6.1.3.8 result

```
TriMesh GlobalSmoothingTask::result
```

result - current partial result

Definition at line 41 of file smoothing.h.

Referenced by updateData().

6.1.3.9 sigma_c_ratio

```
float GlobalSmoothingTask::sigma_c_ratio
```

sigma_c_ratio - current $sigma_c ratio$

Definition at line 33 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing().

6.1.3.10 sigma_s

```
float GlobalSmoothingTask::sigma_s
```

sigma_s - current $sigma_s$

Definition at line 37 of file smoothing.h.

Referenced by myMainWindow::runGlobalSmoothing().

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

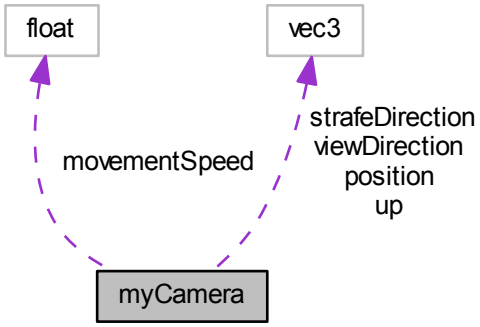
- smoothing.h

6.2 myCamera Class Reference

The myCamera class - camera representation in a typical rendering pipeline (i.e. OpenGL)

`#include <myCamera.h>`

Collaboration diagram for myCamera:



Public Member Functions

- myCamera ()

  myCamera - default constructor
- void updateStrafeDirection ()

  updateStrafeDirection - updates strafe direction vector given view direction and up vector
- glm::mat4 getWorldToViewMatrix () const

  getWorldToViewMatrix: generates world to view matrix
- void moveForward ()

  moveForward - moves camera position following view direction (regarding movement speed)
- void moveBackward ()

  moveBackward - moves camera position following the opposite direction of view direction (regarding movement speed)
- void strafeLeft ()

  strafeLeft - moves camera position following the opposite direction of strafe direction (regarding movement speed)
- void strafeRight ()

  strafeRight - moves camera position following the strafe direction (regarding movement speed)
- void moveUp ()

  moveUp - moves camera position following up direction (regarding movement speed)
- void moveDown ()

  moveDown - moves camera position following the opposite direction of up direction (regarding movement speed)
- glm::vec3 getPosition () const

  getPosition - get camera position
- glm::vec3 getViewDirection () const

  getViewDirection - get view direction
- glm::vec3 getUP () const

getUP - get up direction

- glm::vec3 getStrafeDirection () const

  getStrafeDirection - get strafe direction

- float getMovementSpeed () const

  getMovementSpeed - get camera movement speed

- void setPosition (glm::vec3 &_pos)

  setPosition - set camera position

- void setViewDirection (glm::vec3 &_viewDirection)

  setViewDirection - set camera view direction

- void setUp (glm::vec3 &_up)

  setUp - set camera up direction

- void setStrafeDirection (glm::vec3 &_strafeDirection)

  setStrafeDirection - set strafe direction

- void setMovementSpeed (float _movementSpeed)

  setMovementSpeed - set camera movement speed

Private Attributes

- glm::vec3 position

  position - camera position in world space (eye)

- glm::vec3 viewDirection

  viewDirection - camera view direction (should be unit vector)

- glm::vec3 up

  up - camera up direction (should be unit vector)

- glm::vec3 strafeDirection

  strafeDirection - camera strafe direction (should be unit vector)

- float movementSpeed

  movementSpeed - movement speed for camera translation (forward, backward, left and right)

6.2.1 Detailed Description

The myCamera class - camera representation in a typical rendering pipeline (i.e. OpenGL)

Definition at line 19 of file myCamera.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 myCamera()

```
myCamera::myCamera ( )
```

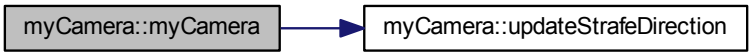
myCamera - default constructor

Definition at line 3 of file myCamera.cpp.

References movementSpeed, and updateStrafeDirection().

```
3                   :
4 viewDirection(0.0f, 0.0f, -1.0f),
5 position(0.f,0.f, 1.0f),
6 up(0.0f, 1.0f, 0.0f)
7 {
8     updateStrafeDirection();
9     movementSpeed = 1.0f;
10 }
```

Here is the call graph for this function:



6.2.3 Member Function Documentation

6.2.3.1 getMovementSpeed()

```
float myCamera::getMovementSpeed ( ) const  [inline]
```

getMovementSpeed - get camera movement speed

Returns

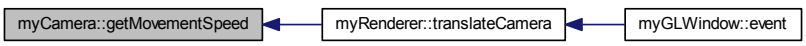
camera movement speed value

Definition at line 106 of file myCamera.h.

Referenced by myRenderer::translateCamera().

```
106 { return movementSpeed; }
```

Here is the caller graph for this function:



6.2.3.2 getPosition()

```
glm::vec3 myCamera::getPosition ( ) const  [inline]
```

getPosition - get camera position

Returns

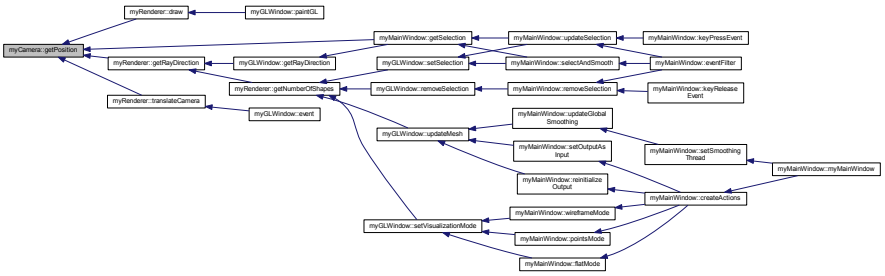
camera position vector

Definition at line 86 of file myCamera.h.

Referenced by myRenderer::draw(), myRenderer::getRayDirection(), myMainWindow::getSelection(), and myRenderer::translateCamera().

```
86 { return position; }
```

Here is the caller graph for this function:



6.2.3.3 getStrafeDirection()

```
glm::vec3 myCamera::getStrafeDirection ( ) const  [inline]
```

getStrafeDirection - get strafe direction

Returns

camera strafe direction vector

Definition at line 101 of file myCamera.h.

Referenced by myRenderer::translateCamera().

```
101 { return strafeDirection; }
```

Here is the caller graph for this function:

6.2.3.4 getUP()

```
glm::vec3 myCamera::getUP ( ) const  [inline]
```

getUP - get up direction

Returns

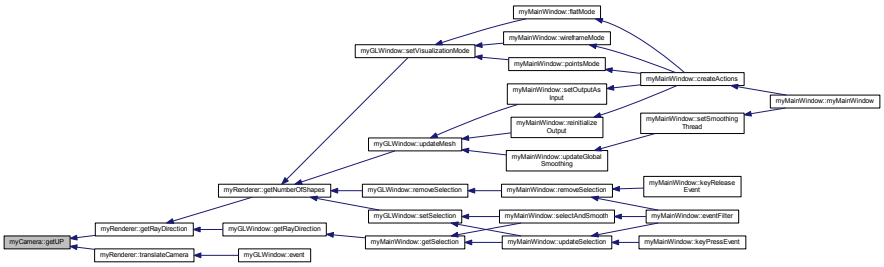
camera up direction vector

Definition at line 96 of file myCamera.h.

Referenced by myRenderer::getRayDirection(), and myRenderer::translateCamera().

```
96 { return up; }
```

Here is the caller graph for this function:



6.2.3.5 getViewDirection()

```
glm::vec3 myCamera::getViewDirection ( ) const  [inline]
```

getViewDirection - get view direction

Returns

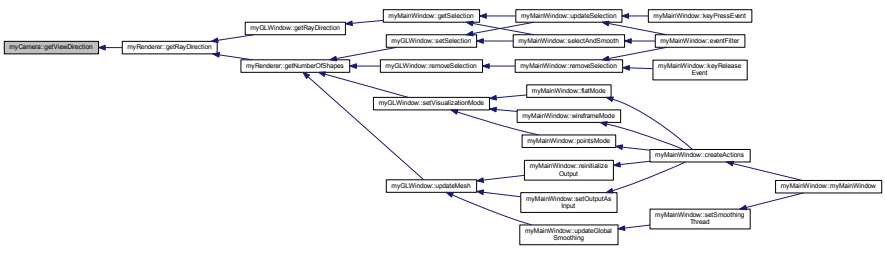
camera view direction vector

Definition at line 91 of file myCamera.h.

Referenced by myRenderer::getRayDirection().

```
91 { return viewDirection; }
```

Here is the caller graph for this function:



6.2.3.6 getWorldToViewMatrix()

```
glm::mat4 myCamera::getWorldToViewMatrix ( ) const
```

getWorldToViewMatrix: generates world to view matrix

Returns

4x4 world to view matrix

Definition at line 18 of file myCamera.cpp.

References position, up, and viewDirection.

Referenced by myRenderer::draw().

```
19 {
20     return glm::lookAt(position, position + viewDirection,
       up);
21 }
```

Here is the caller graph for this function:

6.2.3.7 moveBackward()

```
void myCamera::moveBackward ( )
```

moveBackward - moves camera position following the opposite direction of view direction (regarding movement speed)

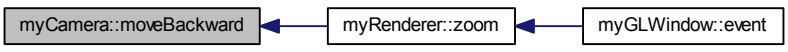
Definition at line 28 of file myCamera.cpp.

References movementSpeed, position, and viewDirection.

Referenced by myRenderer::zoom().

```
29 {
30     position += -movementSpeed * viewDirection;
31 }
```

Here is the caller graph for this function:



6.2.3.8 moveDown()

```
void myCamera::moveDown ( )
```

moveDown - moves camera position following the opposite direction of up direction (regarding movement speed)

Definition at line 48 of file myCamera.cpp.

References movementSpeed, position, and up.

```
49 {
50     position += -movementSpeed * up;
51 }
```

6.2.3.9 moveForward()

```
void myCamera::moveForward ( )
```

moveForward - moves camera position following view direction (regarding movement speed)

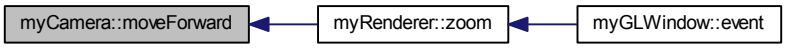
Definition at line 23 of file myCamera.cpp.

References movementSpeed, position, and viewDirection.

Referenced by myRenderer::zoom().

```
24 {
25     position += movementSpeed * viewDirection;
26 }
```

Here is the caller graph for this function:



6.2.3.10 moveUp()

```
void myCamera::moveUp ( )
```

moveUp - moves camera position following up direction (regarding movement speed)

Definition at line 43 of file myCamera.cpp.

References movementSpeed, position, and up.

```
44 {
45     position += movementSpeed * up;
46 }
```

6.2.3.11 setMovementSpeed()

```
void myCamera::setMovementSpeed (
            float _movementSpeed )  [inline]
```

setMovementSpeed - set camera movement speed

Parameters

_movementSpeed	new camera movement speed

Definition at line 132 of file myCamera.h.

Referenced by myRenderer::initializeInteractor().

```
132 { movementSpeed = _movementSpeed; }
```

Here is the caller graph for this function:

6.2.3.12 setPosition()

```
void myCamera::setPosition (
            glm::vec3 & _pos )  [inline]
```

setPosition - set camera position

Parameters

_pos	new camera position

Definition at line 112 of file myCamera.h.

Referenced by myRenderer::initializeInteractor(), and myRenderer::translateCamera().

```
112 { position = _pos; }
```

Here is the caller graph for this function:

6.2.3.13 setStrafeDirection()

```
void myCamera::setStrafeDirection (
            glm::vec3 & _strafeDirection )  [inline]
```

setStrafeDirection - set strafe direction

Parameters

_strafeDirection	new camera strafe direction

Definition at line 127 of file myCamera.h.

```
127 { strafeDirection = _strafeDirection; }
```

6.2.3.14 setUp()

```
void myCamera::setUp (
            glm::vec3 & _up )  [inline]
```

setUp - set camera up direction

Parameters

_up	new camera up direction

Definition at line 122 of file myCamera.h.

```
122 { up = _up; }
```

6.2.3.15 setViewDirection()

```
void myCamera::setViewDirection (
            glm::vec3 & _viewDirection )  [inline]
```

setViewDirection - set camera view direction

Parameters

_viewDirection	new camera view direction

Definition at line 117 of file myCamera.h.

```
117 { viewDirection = _viewDirection; }
```

6.2.3.16 strafeLeft()

```
void myCamera::strafeLeft ( )
```

strafeLeft - moves camera position following the opposite direction of strafe direction (regarding movement speed)

Definition at line 33 of file myCamera.cpp.

References movementSpeed, position, and strafeDirection.

```
34 {
35     position += -movementSpeed * strafeDirection;
36 }
```

6.2.3.17 strafeRight()

```
void myCamera::strafeRight ( )
```

strafeRight - moves camera position following the strafe direction (regarding movement speed)

Definition at line 38 of file myCamera.cpp.

References movementSpeed, position, and strafeDirection.

```
39 {
40     position += movementSpeed * strafeDirection;
41 }
```

6.2.3.18 updateStrafeDirection()

```
void myCamera::updateStrafeDirection ( )
```

updateStrafeDirection - updates strafe direction vector given view direction and up vector

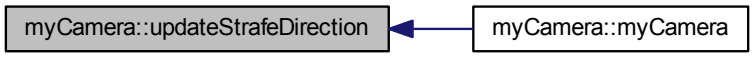
Definition at line 12 of file myCamera.cpp.

References strafeDirection, up, and viewDirection.

Referenced by myCamera().

```
13 {
14     strafeDirection = glm::cross(viewDirection, up);
15     strafeDirection = glm::normalize(strafeDirection);
16 }
```

Here is the caller graph for this function:



6.2.4 Field Documentation

6.2.4.1 movementSpeed

```
float myCamera::movementSpeed  [private]
```

movementSpeed - movement speed for camera translation (forward, backward, left and right)

Definition at line 41 of file myCamera.h.

Referenced by moveBackward(), moveDown(), moveForward(), moveUp(), myCamera(), strafeLeft(), and strafe↩ Right().

6.2.4.2 position

```
glm::vec3 myCamera::position  [private]
```

position - camera position in world space (eye)

Definition at line 24 of file myCamera.h.

Referenced by getWorldToViewMatrix(), moveBackward(), moveDown(), moveForward(), moveUp(), strafeLeft(), and strafeRight().

6.2.4.3 strafeDirection

```
glm::vec3 myCamera::strafeDirection  [private]
```

strafeDirection - camera strafe direction (should be unit vector)

Definition at line 36 of file myCamera.h.

Referenced by strafeLeft(), strafeRight(), and updateStrafeDirection().

6.2.4.4 up

```
glm::vec3 myCamera::up  [private]
```

up - camera up direction (should be unit vector)

Definition at line 32 of file myCamera.h.

Referenced by getWorldToViewMatrix(), moveDown(), moveUp(), and updateStrafeDirection().

6.2.4.5 viewDirection

```
glm::vec3 myCamera::viewDirection  [private]
```

viewDirection - camera view direction (should be unit vector)

Definition at line 28 of file myCamera.h.

Referenced by getWorldToViewMatrix(), moveBackward(), moveForward(), and updateStrafeDirection().

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

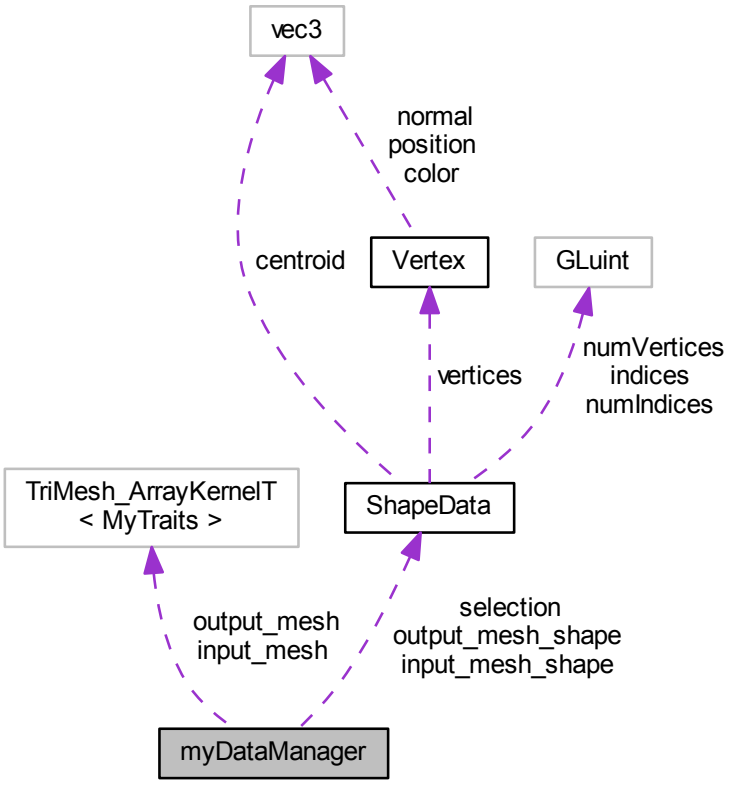
- visualization/myCamera.h
- visualization/myCamera.cpp

6.3 myDataManager Class Reference

The myDataManager class - data manager for mesh smoothing application.

```
#include <myDataManager.h>
```

Collaboration diagram for myDataManager:



Public Member Functions

- myDataManager ()

  myDataManager - default constructor
- ~myDataManager ()

  myDataManager - destructor
- void loadInputMesh (string &fileName)

  loadInputMesh - load a mesh file supported by OpenMesh library
- void saveOutputMesh (string &fileName)

  saveOutputMesh - save the output mesh in a file supported bu OpenMesh library
- void updateInputShape ()

  updateInputShape - generate the corresponding shape data of the input mesh
- void updateOutputShape ()

  updateOutputShape - generate the corresponding shape data of the output mesh
- void updateShapes ()

  updateShapes - generate the corresponding shape data for both meshes (input mesh and output mesh)
- void updateOutputSelection (vector< size_t > &indices)

  updateOutputSelection - generate the selection shape for a set of given vertex indices
- void setOutputAsInput ()

  setOutputAsInput - set output mesh as input mesh (save all changes)
- void reinitialize ()

  reinitialize - set input mesh as output mesh (undone all changes)

Data Fields

- TriMesh input_mesh

  input_mesh - input mesh (left side in the interface)
- TriMesh output_mesh

  output_mesh - output mesh/resulting mesh/edited mesh (right side in the interface)
- ShapeData input_mesh_shape

  input_mesh_shape - input mesh shape data (for visualization)
- ShapeData output_mesh_shape

  output_mesh_shape - output mesh shape data (for visualization)
- ShapeData selection

  selection - shape data for current selection (set of vertices)

6.3.1 Detailed Description

The myDataManager class - data manager for mesh smoothing application.

Definition at line 10 of file myDataManager.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 myDataManager()

```
myDataManager::myDataManager ( )
```

myDataManager - default constructor

Definition at line 4 of file myDataManager.cpp.

```
5 {
6 }
```

6.3.2.2 ~myDataManager()

```
myDataManager::~myDataManager ( )
```

myDataManager - destructor

Definition at line 9 of file myDataManager.cpp.

References ShapeData::clear(), input_mesh, input_mesh_shape, output_mesh, output_mesh_shape, and selection.

```
10 {
11     input_mesh.clear();
12     output_mesh.clear();
13     input_mesh_shape.clear();
14     output_mesh_shape.clear();
15     selection.clear();
16 }
```

Here is the call graph for this function:

6.3.3 Member Function Documentation

6.3.3.1 loadInputMesh()

```
void myDataManager::loadInputMesh (
            string & fileName )
```

loadInputMesh - load a mesh file supported by OpenMesh library

Parameters

fileName	mesh file name

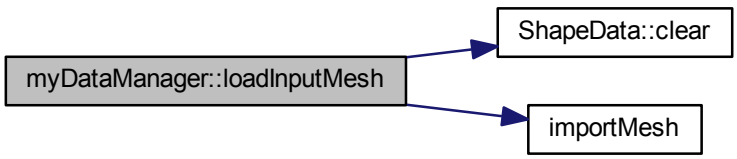
Definition at line 19 of file myDataManager.cpp.

References ShapeData::clear(), importMesh(), input_mesh, input_mesh_shape, output_mesh, and output_mesh_shape.

Referenced by myMainWindow::loadMesh().

```
20 {
21     input_mesh.clear();
22     output_mesh.clear();
23     input_mesh_shape.clear();
24     output_mesh_shape.clear();
25
26     importMesh(input_mesh, fileName);
27     input_mesh.request_face_normals();
28     input_mesh.request_vertex_normals();
29     input_mesh.update_normals();
30
31     output_mesh = input_mesh;
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.3.3.2 reinitialize()

```
void myDataManager::reinitialize ( )
```

reinitialize - set input mesh as output mesh (undone all changes)

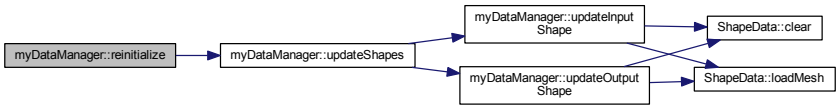
Definition at line 72 of file myDataManager.cpp.

References input_mesh, output_mesh, and updateShapes().

Referenced by myMainWindow::reinitializeOutput().

```
73 {
74     output_mesh = input_mesh;
75     updateShapes();
76 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.3.3.3 saveOutputMesh()

```
void myDataManager::saveOutputMesh (
            string & fileName )
```

saveOutputMesh - save the output mesh in a file supported bu OpenMesh library

Parameters

fileName	mesh file name

Definition at line 34 of file myDataManager.cpp.

References exportMesh(), and output_mesh.

Referenced by myMainWindow::saveMesh().

```
35 {
36     output_mesh.request_face_normals();
37     output_mesh.request_vertex_normals();
38     output_mesh.update_normals();
39     exportMesh(output_mesh, fileName);
40 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.3.3.4 setOutputAsInput()

```
void myDataManager::setOutputAsInput ( )
```

setOutputAsInput - set output mesh as input mesh (save all changes)

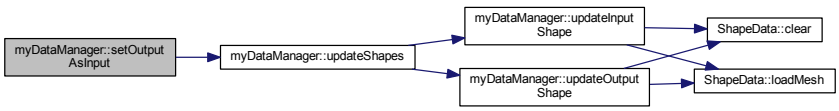
Definition at line 66 of file myDataManager.cpp.

References input_mesh, output_mesh, and updateShapes().

Referenced by myMainWindow::setOutputAsInput().

```
67 {
68     input_mesh = output_mesh;
69     updateShapes();
70 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.3.3.5 updateInputShape()

```
void myDataManager::updateInputShape ( )
```

updateInputShape - generate the corresponding shape data of the input mesh

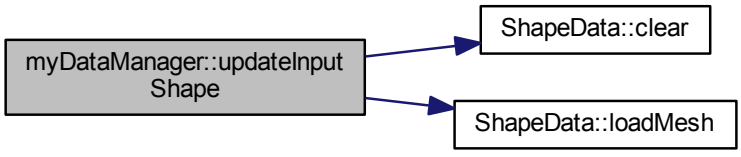
Definition at line 42 of file myDataManager.cpp.

References ShapeData::clear(), input_mesh, input_mesh_shape, and ShapeData::loadMesh().

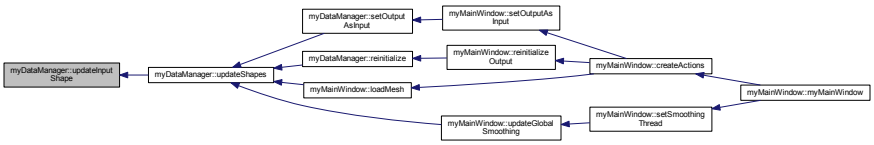
Referenced by updateShapes().

```
43 {
44     input_mesh_shape.clear();
45     input_mesh_shape.loadMesh(input_mesh);
46 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.6 updateOutputSelection()

```
void myDataManager::updateOutputSelection (
            vector< size_t > & indices )
```

updateOutputSelection - generate the selection shape for a set of given vertex indices

Parameters

indices	target set of vertex indices

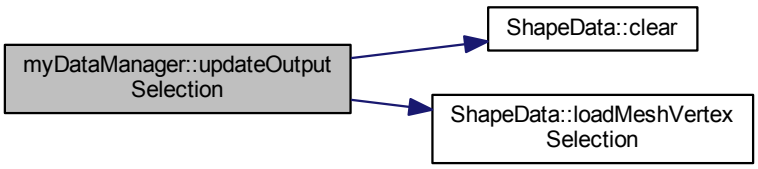
Definition at line 60 of file myDataManager.cpp.

References ShapeData::clear(), ShapeData::loadMeshVertexSelection(), output_mesh, and selection.

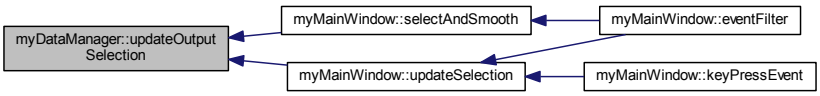
Referenced by myMainWindow::selectAndSmooth(), and myMainWindow::updateSelection().

```
61 {
62     selection.clear();
63     selection.loadMeshVertexSelection(
      output_mesh, indices);
64 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.7 updateOutputShape()

```
void myDataManager::updateOutputShape ( )
```

updateOutputShape - generate the corresponding shape data of the output mesh

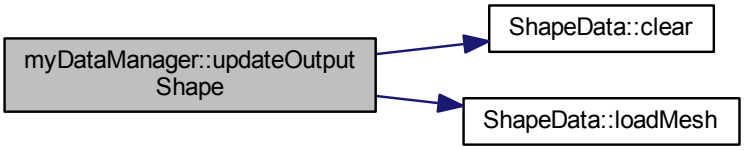
Definition at line 48 of file myDataManager.cpp.

References ShapeData::clear(), ShapeData::loadMesh(), output_mesh, and output_mesh_shape.

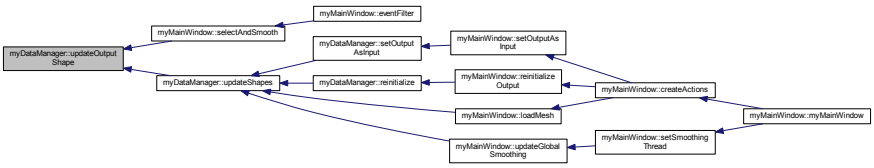
Referenced by myMainWindow::selectAndSmooth(), and updateShapes().

```
49 {
50     output_mesh_shape.clear();
51     output_mesh_shape.loadMesh(output_mesh);
52 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.8 updateShapes()

```
void myDataManager::updateShapes ( )
```

updateShapes - generate the corresponding shape data for both meshes (input mesh and output mesh)

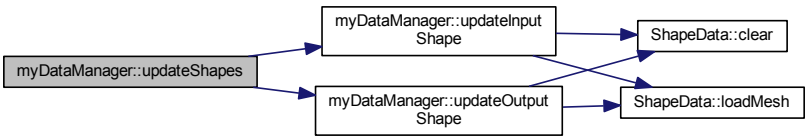
Definition at line 54 of file myDataManager.cpp.

References updateInputShape(), and updateOutputShape().

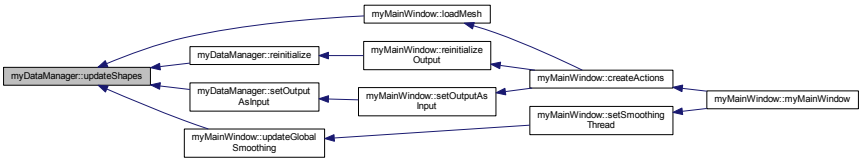
Referenced by myMainWindow::loadMesh(), reinitialize(), setOutputAsInput(), and myMainWindow::updateGlobalSmoothing().

```
55 {
56     updateInputShape();
57     updateOutputShape();
58 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.4 Field Documentation

6.3.4.1 input_mesh

```
TriMesh myDataManager::input_mesh
```

input_mesh - input mesh (left side in the interface)

Definition at line 16 of file myDataManager.h.

Referenced by loadInputMesh(), reinitialize(), setOutputAsInput(), updateInputShape(), and ~myDataManager().

6.3.4.2 input_mesh_shape

```
ShapeData myDataManager::input_mesh_shape
```

input_mesh_shape - input mesh shape data (for visualization)

Definition at line 25 of file myDataManager.h.

Referenced by loadInputMesh(), myMainWindow::loadMesh(), updateInputShape(), and ~myDataManager().

6.3.4.3 output_mesh

```
TriMesh myDataManager::output_mesh
```

output_mesh - output mesh/resulting mesh/edited mesh (right side in the interface)

Definition at line 20 of file myDataManager.h.

Referenced by myMainWindow::getSelection(), loadInputMesh(), reinitialize(), GlobalSmoothingTask::run(), saveOutputMesh(), myMainWindow::selectAndSmooth(), setOutputAsInput(), GlobalSmoothingTask::updateData(), updateOutputSelection(), updateOutputShape(), and ~myDataManager().

6.3.4.4 output_mesh_shape

```
ShapeData myDataManager::output_mesh_shape
```

output_mesh_shape - output mesh shape data (for visualization)

Definition at line 29 of file myDataManager.h.

Referenced by loadInputMesh(), myMainWindow::loadMesh(), updateOutputShape(), and ~myDataManager().

6.3.4.5 selection

```
ShapeData myDataManager::selection
```

selection - shape data for current selection (set of vertices)

Definition at line 34 of file myDataManager.h.

Referenced by myMainWindow::selectAndSmooth(), updateOutputSelection(), myMainWindow::updateSelection(), and ~myDataManager().

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

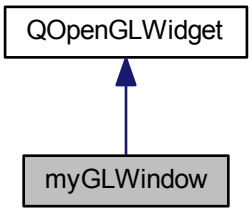
- myDataManager.h
- myDataManager.cpp

6.4 myGLWindow Class Reference

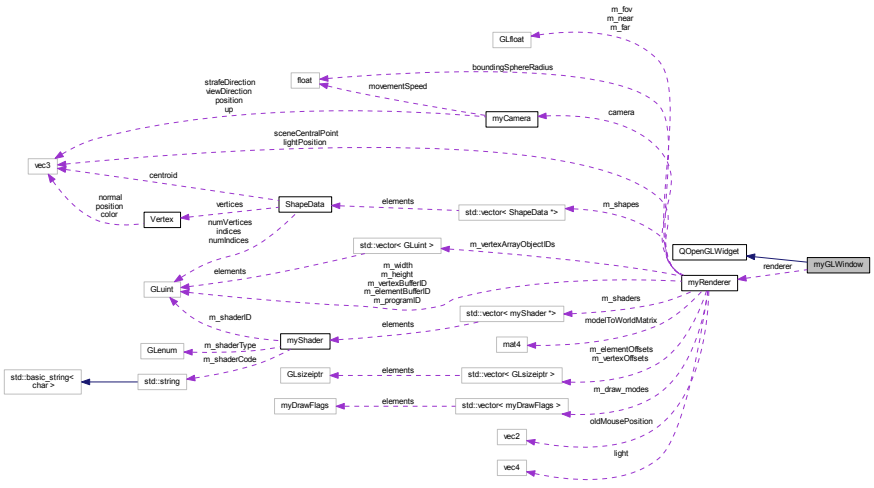
The myGLWindow class - Qt Widget for OpenGL Designed to support only one shape and one selection at time. So in the containers of the renderer there are two shapes at position 0 and 1. The first one for the current shape and the second one for the current selection. This object adapts a renderer to a QT OpenGL Widget which manages a single OpenGL context. Also, it simplifies the usage of the renderer for the mesh smoothing application.

`#include <myGLWindow.h>`

Inheritance diagram for myGLWindow:



Collaboration diagram for myGLWindow:



Public Member Functions

- myGLWindow (QWidget *parent=0)

  myGLWindow - default constructor
- ~myGLWindow ()

  ~myGLWindow - destructor
- void sendDataToOpenGL ()

sendDataToOpenGL - call sendDataToOpenGL of the renderer, in the current context

- void setVisualizationMode (int mode)

  setVisualizationMode - set rendering mode of the current shape
- void setShape (ShapeData *_shape)

  setShape - set a shape as current shape (removing previous shape)
- void setSelection (ShapeData *_selection)

  setSelection - set a selection as current selection (removing previous selection)
- void removeSelection ()

  removeSelection - remove current selection
- void updateMesh ()

  updateMesh - update vertex data of the current shape (resend vertex data)
- void addShader (GLenum _shaderType, const string &_fileName)

  addShader - add a shader to the renderer
- void clearAndDeleteShaders ()

  clearAndDeleteShaders - clear, deattach and delete shaders of the renderer
- void installShaders ()

  installShaders - call installShaders of the renderer, in the current context.
- glm::vec2 getCurrentMousePosition ()

  getCurrentMousePosition - get mouse position in the OpenGL Window (Qt Widget)
- glm::vec3 getRayDirection (glm::vec2 &pos)

  getRayDirection - call getRayDirection of the renderer for a given position
- myCamera * getCamera ()

  getCamera - get pointer to the camera of the renderer
- glm::mat4 getModelToWorldMatrix ()

  getModelToWorldMatrix - get model to world matrix applied to all objects in the renderer

Protected Member Functions

- void initializeGL ()

  initializeGL - initialize the renderer in the current OpenGL context
- void paintGL ()

  paintGL - call renderer draw functions
- bool event (QEvent *event)

  event - event manager for mouse interactions (in myGLWindow)

Private Attributes

- myRenderer * renderer

  renderer - renderer object for visualization in the widget

6.4.1 Detailed Description

The myGLWindow class - Qt Widget for OpenGL Designed to support only one shape and one selection at time. So in the containers of the renderer there are two shapes at position 0 and 1. The first one for the current shape and the second one for the current selection. This object adapts a renderer to a QT OpenGL Widget which manages a single OpenGL context. Also, it simplifies the usage of the renderer for the mesh smoothing application.

Definition at line 37 of file myGLWindow.h.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 myGLWindow()

```
myGLWindow::myGLWindow (
            QWidget * parent = 0 )
```

myGLWindow - default constructor

Parameters

parent	parent widget pointer

Definition at line 3 of file myGLWindow.cpp.

```
4 : QOpenGLWidget(parent)
5 {
6 }
```

6.4.2.2 ~myGLWindow()

```
myGLWindow::~myGLWindow ( )
```

~myGLWindow - destructor

Definition at line 9 of file myGLWindow.cpp.

References renderer.

```
10 {
11     delete renderer;
12 }
```

6.4.3 Member Function Documentation

6.4.3.1 addShader()

```
void myGLWindow::addShader (
            GLenum _shaderType,
            const string & _fileName )
```

addShader - add a shader to the renderer

Parameters

_shaderType	- shader type
_fileName	- shader code file name

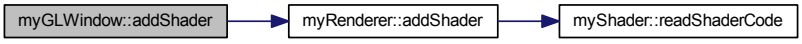
Definition at line 122 of file myGLWindow.cpp.

References myRenderer::addShader(), and renderer.

Referenced by myMainWindow::setShaders().

```
123 {
124     makeCurrent();
125     renderer->addShader(_shaderType,_fileName);
126     doneCurrent();
127 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.4.3.2 clearAndDeleteShaders()

```
void myGLWindow::clearAndDeleteShaders ( )
```

clearAndDeleteShaders - clear, deattach and delete shaders of the renderer

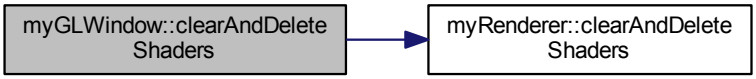
Definition at line 129 of file myGLWindow.cpp.

References myRenderer::clearAndDeleteShaders(), and renderer.

Referenced by myMainWindow::setShaders().

```
130 {
131     makeCurrent();
132     renderer->clearAndDeleteShaders();
133     doneCurrent();
134 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.4.3.3 event()

```
bool myGLWindow::event (
            QEvent * event )  [protected]
```

event - event manager for mouse interactions (in myGLWindow)

Parameters

event	- input event

Returns

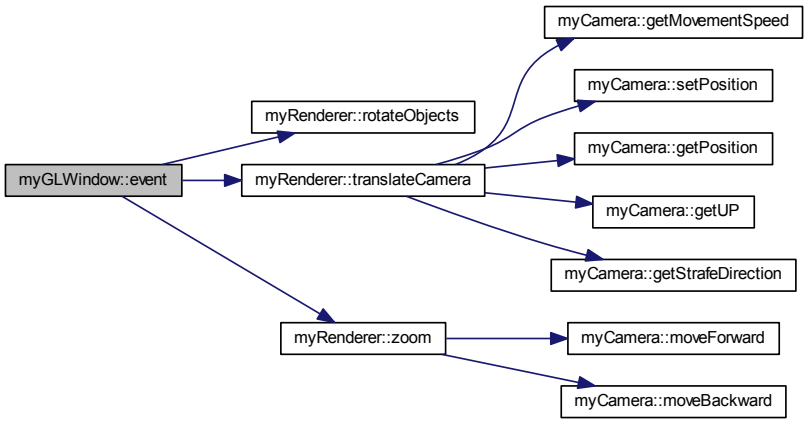
success

Definition at line 37 of file myGLWindow.cpp.

References renderer, myRenderer::rotateObjects(), myRenderer::translateCamera(), and myRenderer::zoom().

```
38 {
39     if (event->type() == QEvent::MouseMove) {
40         QMouseEvent * e = static_cast<QMouseEvent *>(event);
41         if (e->buttons() == Qt::LeftButton)
42             renderer->rotateObjects(glm::vec2(e->x(), e->y()));
43         if (e->buttons() == Qt::RightButton)
44             renderer->translateCamera(glm::vec2(e->x(), e->y()));
45         repaint();
46         return true;
47     }
48     else if (event->type() == QEvent::Wheel)
49     {
50         QWheelEvent * e = static_cast<QWheelEvent *>(event);
51         renderer->zoom(static_cast<float>(e->delta()));
52         repaint();
53         return true;
54     }
55     return QOpenGLWidget::event(event);
56 }
```

Here is the call graph for this function:



6.4.3.4 getCamera()

```
myCamera * myGLWindow::getCamera ( )
```

getCamera - get pointer to the camera of the renderer

Returns

camera pointer

Definition at line 153 of file myGLWindow.cpp.

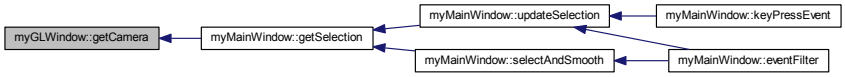
References myRenderer::getCamera(), and renderer.

Referenced by myMainWindow::getSelection().

```
154 {
155     return renderer->getCamera();
156 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.4.3.5 getCurrentMousePosition()

```
glm::vec2 myGLWindow::getCurrentMousePosition ( )
```

getCurrentMousePosition - get mouse position in the OpenGL Window (Qt Widget)

Returns

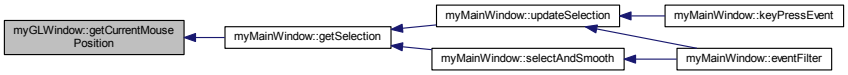
mouse position

Definition at line 143 of file myGLWindow.cpp.

Referenced by myMainWindow::getSelection().

```
144 {
145     QPoint p = mapFromGlobal(QCursor::pos()); return glm::vec2(p.x(), p.y());
146 }
```

Here is the caller graph for this function:



6.4.3.6 getModelToWorldMatrix()

```
glm::mat4 myGLWindow::getModelToWorldMatrix ( )
```

getModelToWorldMatrix - get model to world matrix applied to all objects in the renderer

Returns

4x4 model to world matrix

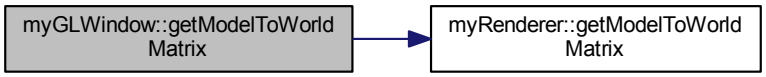
Definition at line 158 of file myGLWindow.cpp.

References myRenderer::getModelToWorldMatrix(), and renderer.

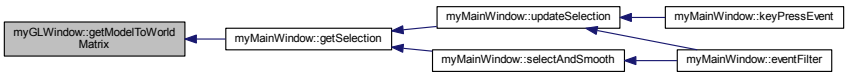
Referenced by myMainWindow::getSelection().

```
159 {
160     return renderer->getModelToWorldMatrix();
161 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.3.7 getRayDirection()

```
glm::vec3 myGLWindow::getRayDirection (
            glm::vec2 & pos )
```

getRayDirection - call getRayDirection of the renderer for a given position

Parameters

pos	- position

Returns

ray direction

Definition at line 148 of file myGLWindow.cpp.

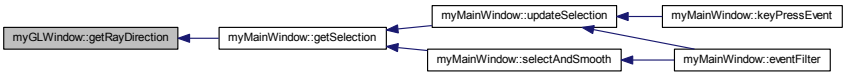
References myRenderer::getRayDirection(), and renderer.

Referenced by myMainWindow::getSelection().

```
149 {
150     return renderer->getRayDirection(pos);
151 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.4.3.8 initializeGL()

```
void myGLWindow::initializeGL ( )  [protected]
```

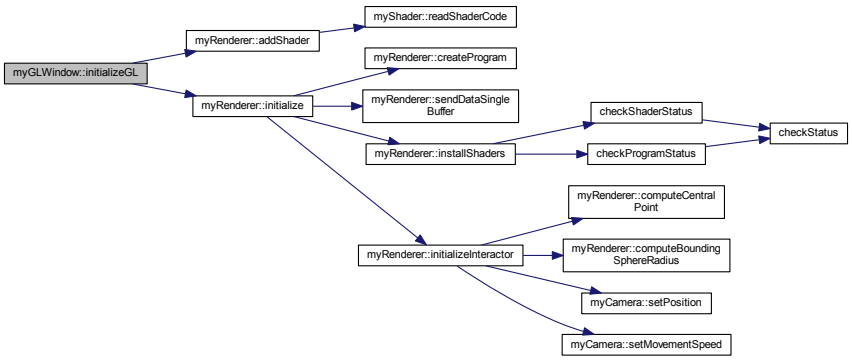
initializeGL - initialize the renderer in the current OpenGL context

Definition at line 21 of file myGLWindow.cpp.

References myRenderer::addShader(), myRenderer::initialize(), and renderer.

```
22 {
23     renderer = new myRenderer;
24     setMouseTracking(true);
25     renderer->addShader(GL_VERTEX_SHADER, "VertexShaderCodePhong.glsl");
26     renderer->addShader(GL_FRAGMENT_SHADER, "FragmentShaderCodePhong.glsl");
27     renderer->initialize();
28 }
```

Here is the call graph for this function:



6.4.3.9 installShaders()

```
void myGLWindow::installShaders ( )
```

installShaders - call installShaders of the renderer, in the current context.

Definition at line 136 of file myGLWindow.cpp.

References myRenderer::installShaders(), and renderer.

Referenced by myMainWindow::setShaders().

```
137 {
138     makeCurrent();
139     renderer->installShaders();
140     doneCurrent();
141 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.4.3.10 paintGL()

```
void myGLWindow::paintGL ( )  [protected]
```

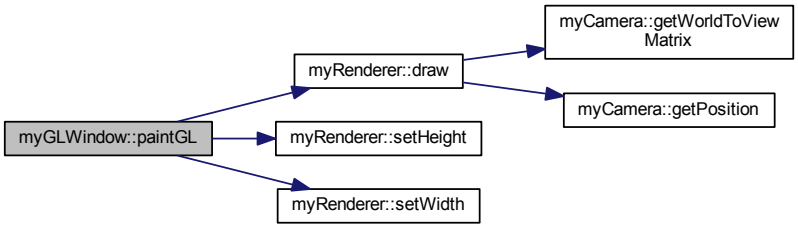
paintGL - call renderer draw functions

Definition at line 30 of file myGLWindow.cpp.

References myRenderer::draw(), renderer, myRenderer::setHeight(), and myRenderer::setWidth().

```
31 {
32     renderer->setWidth(width());
33     renderer->setHeight(height());
34     renderer->draw();
35 }
```

Here is the call graph for this function:



6.4.3.11 removeSelection()

```
void myGLWindow::removeSelection ( )
```

removeSelection - remove current selection

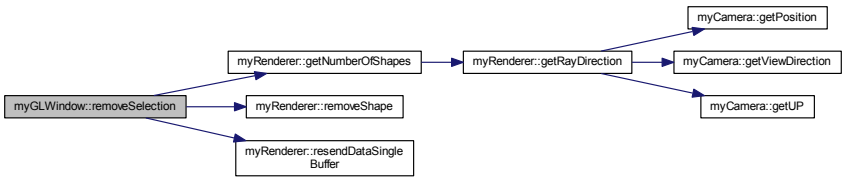
Definition at line 78 of file myGLWindow.cpp.

References myRenderer::getNumberOfShapes(), myRenderer::removeShape(), renderer, and myRenderer↩ ::resendDataSingleBuffer().

Referenced by myMainWindow::removeSelection().

```
79 {
80     if (renderer->getNumberOfShapes() > 1)
81     {
82         renderer->removeShape(1);
83         makeCurrent();
84         renderer->resendDataSingleBuffer();
85         doneCurrent();
86         repaint();
87     }
88     else
89     {
90         repaint();
91     }
92 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.4.3.12 sendDataToOpenGL()

```
void myGLWindow::sendDataToOpenGL ( )
```

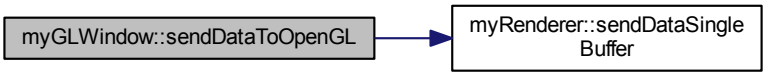
sendDataToOpenGL - call sendDataToOpenGL of the renderer, in the current context

Definition at line 14 of file myGLWindow.cpp.

References renderer, and myRenderer::sendDataSingleBuffer().

```
15 {
16     makeCurrent();
17     renderer->sendDataSingleBuffer();
18     doneCurrent();
19 }
```

Here is the call graph for this function:



6.4.3.13 setSelection()

```
void myGLWindow::setSelection (
            ShapeData * _selection )
```

setSelection - set a selection as current selection (removing previous selection)

Parameters

_selection	new selection

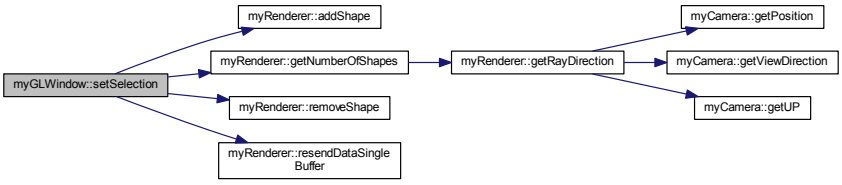
Definition at line 94 of file myGLWindow.cpp.

References myRenderer::addShape(), e_draw_selection, myRenderer::getNumberOfShapes(), myRenderer::removeShape(), renderer, and myRenderer::resendDataSingleBuffer().

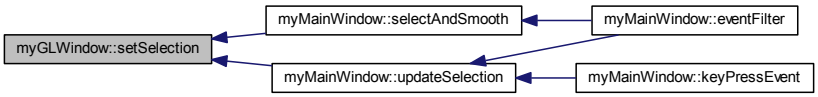
Referenced by myMainWindow::selectAndSmooth(), and myMainWindow::updateSelection().

```
95 {
96     if (renderer->getNumberOfShapes() <= 1)
97     {
98         renderer->addShape(_selection, e_draw_selection);
99     }
100     else
101     {
102         renderer->removeShape(1);
103         renderer->addShape(_selection,e_draw_selection);
104     }
105     makeCurrent();
106     renderer->resendDataSingleBuffer();
107     doneCurrent();
108     repaint();
109 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.3.14 setShape()

```
void myGLWindow::setShape (
            ShapeData * _shape )
```

setShape - set a shape as current shape (removing previous shape)

Parameters

_shape	new shape

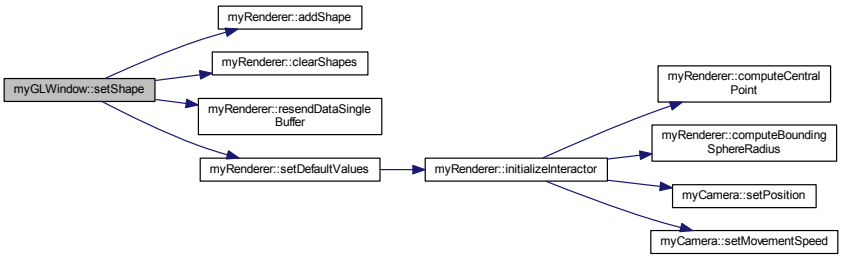
Definition at line 67 of file myGLWindow.cpp.

References myRenderer::addShape(), myRenderer::clearShapes(), renderer, myRenderer::resendDataSingle↩ Buffer(), and myRenderer::setDefaultValues().

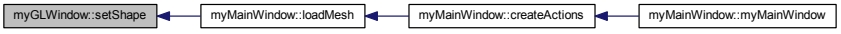
Referenced by myMainWindow::loadMesh().

```
68 {
69     renderer->clearShapes();
70     renderer->addShape(_shape);
71     makeCurrent();
72     renderer->resendDataSingleBuffer();
73     doneCurrent();
74     renderer->setDefaultValues();
75     repaint();
76 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.3.15 setVisualizationMode()

```
void myGLWindow::setVisualizationMode (
            int mode )
```

setVisualizationMode - set rendering mode of the current shape

Parameters

mode	- input rendering mode

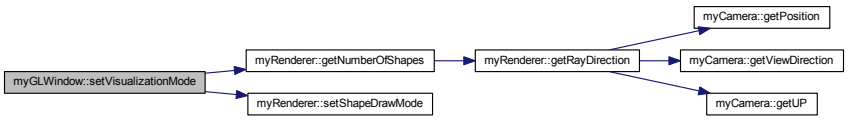
Definition at line 58 of file myGLWindow.cpp.

References myRenderer::getNumberOfShapes(), renderer, and myRenderer::setShapeDrawMode().

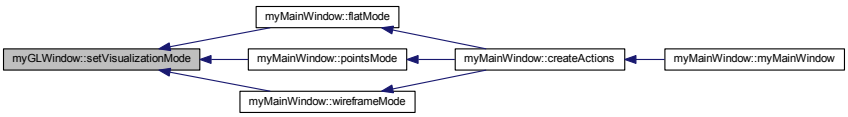
Referenced by myMainWindow::flatMode(), myMainWindow::pointsMode(), and myMainWindow::wireframeMode().

```
59 {
60     if (renderer->getNumberOfShapes() > 0)
61     {
62         renderer->setShapeDrawMode(0, (myDrawFlags)mode);
63         repaint();
64     }
65 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.3.16 updateMesh()

```
void myGLWindow::updateMesh ( )
```

updateMesh - update vertex data of the current shape (resend vertex data)

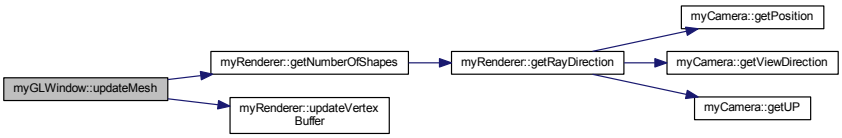
Definition at line 111 of file myGLWindow.cpp.

References myRenderer::getNumberOfShapes(), renderer, and myRenderer::updateVertexBuffer().

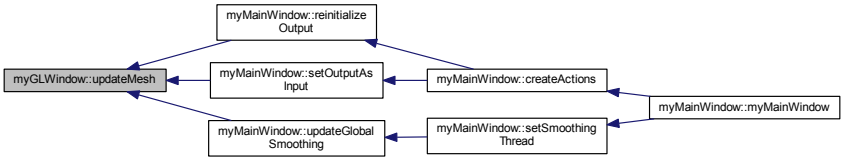
Referenced by myMainWindow::reinitializeOutput(), myMainWindow::setOutputAsInput(), and myMainWindow↩
::updateGlobalSmoothing().

```
112 {
113     if (renderer->getNumberOfShapes() > 0)
114     {
115         makeCurrent();
116         renderer->updateVertexBuffer(0);
117         doneCurrent();
118         repaint();
119     }
120 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.4 Field Documentation

6.4.4.1 renderer

```
myRenderer* myGLWindow::renderer  [private]
```

renderer - renderer object for visualization in the widget

Definition at line 42 of file myGLWindow.h.

Referenced by addShader(), clearAndDeleteShaders(), event(), getCamera(), getModelToWorldMatrix(), getRay↩ Direction(), initializeGL(), installShaders(), paintGL(), removeSelection(), sendDataToOpenGL(), setSelection(), setShape(), setVisualizationMode(), updateMesh(), and ∼myGLWindow().

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

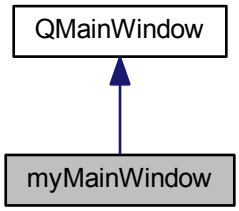
- myGLWindow.h
- myGLWindow.cpp

6.5 myMainWindow Class Reference

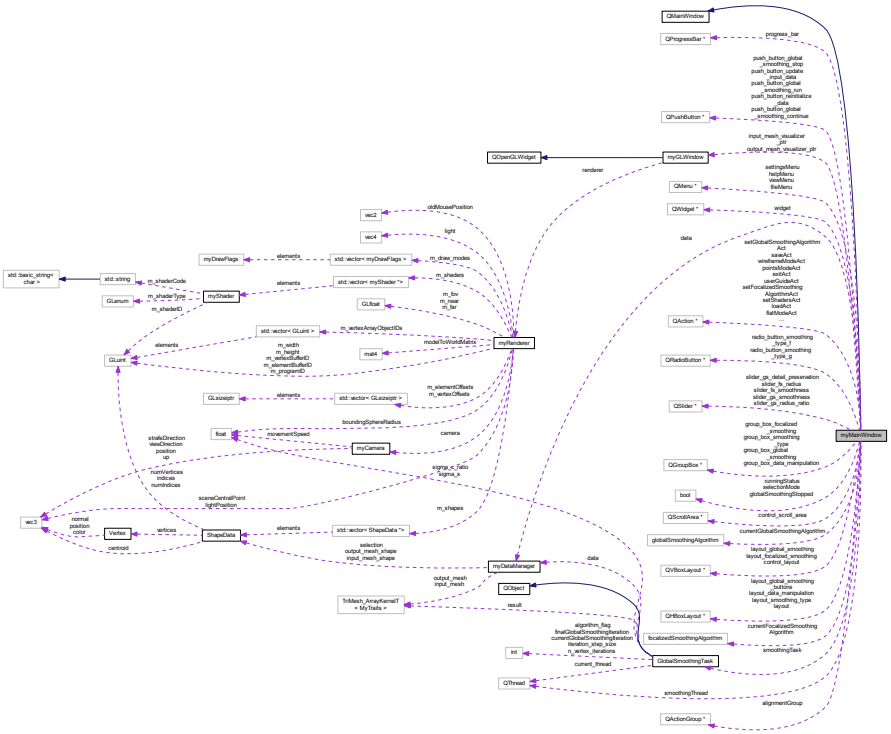
The myMainWindow class - Main Window for mesh smoothing application.

`#include <myMainWindow.h>`

Inheritance diagram for myMainWindow:



Collaboration diagram for myMainWindow:



Public Member Functions

- myMainWindow ()

  myMainWindow - default constructor: dynamic building of all the interface
- ~myMainWindow ()

  ~myMainWindow - destructor

Protected Member Functions

- void keyPressEvent (QKeyEvent ∗e)

  keyPressEvent - Qt key press event
- void keyReleaseEvent (QKeyEvent ∗e)

  keyReleaseEvent - Qt key release event
- bool eventFilter (QObject ∗object, QEvent ∗event)

  eventFilter - Qt event filter

Private Member Functions

- void createActions ()

  createActions - create menu actions and connect all signals and slots
- void createMenus ()

  createMenus - create menus
- void updateWidgetValues ()

  updateWidgetValues - set default slider widget values (min, max and current)
- void setSmoothingThread ()

  setSmoothingThread - set thread for global smoothing
- void loadMesh ()

  loadMesh - load input mesh from a file supported by OpenMesh library
- void saveMesh ()

  saveMesh - save output mesh to a file supported by OpenMesh library
- void exit ()

  exit - close the application
- void flatMode ()

  flatMode - set triangle rendering
- void wireframeMode ()

  wireframeMode - set wireframe rendering
- void pointsMode ()

  pointsMode - set point rendering
- void setShaders ()

  setShaders - set shaders profile (customizable)
- void setGlobalSmoothingAlgorithm ()

  setGlobalSmoothingAlgorithm - set global smoothing algorithm
- void setFocalizedSmoothingAlgorithm ()

  setFocalizedSmoothingAlgorithm - set focalized smoothing algorithm
- void about ()

  about - show about window
- void userGuide ()

  userGuide - show user guide
- void enableSmoothingType ()

  enableSmoothingType - enable widgets for current smoothing type
- void runGlobalSmoothing ()

  runGlobalSmoothing - run global smoothing algorithm
- void stopGlobalSmoothing ()

  stopGlobalSmoothing - stop global smoothing
- void continueGlobalSmoothing ()

  continueGlobalSmoothing - continue global smoothing
- void updateGlobalSmoothing ()

updateGlobalSmoothing - global smoothing management (step by step)

- void setGlobalSmoothingStatus (globalSmoothingStatus current_status)

  setGlobalSmoothingStatus - set current global smoothing status
- void reinitializeOutput ()

  reinitializeOutput - reinitialize output mesh (regarding input mesh)
- void setOutputAsInput ()

  setOutputAsInput - set output mesh as new input mesh
- void getSelection (vector< size_t > &selected_vertices_ids)

  getSelection - get selected vertices for the current mouse position in the output mesh visualizer
- void updateSelection ()

  updateSelection - update selection in data manager and visualization
- void removeSelection ()

  removeSelection - remove selection from data manager and visualization
- void selectAndSmooth ()

  selectAndSmooth - select and perform focalized smoothing regarding control widget values

Private Attributes

- myGLWindow * input_mesh_visualizer_ptr
- myGLWindow * output_mesh_visualizer_ptr
- QWidget * widget
- QHBoxLayout * layout
- QScrollArea * control_scroll_area
- QVBoxLayout * control_layout
- QGroupBox * group_box_smoothing_type
- QGroupBox * group_box_global_smoothing
- QGroupBox * group_box_focalized_smoothing
- QGroupBox * group_box_data_manipulation
- QHBoxLayout * layout_smoothing_type

  Auxiliar Layouts.
- QVBoxLayout * layout_global_smoothing
- QHBoxLayout * layout_global_smoothing_buttons
- QVBoxLayout * layout_focalized_smoothing
- QHBoxLayout * layout_data_manipulation
- QRadioButton * radio_button_smoothing_type_g
- QRadioButton * radio_button_smoothing_type_f
- QPushButton * push_button_global_smoothing_run
- QPushButton * push_button_global_smoothing_stop
- QPushButton * push_button_global_smoothing_continue
- QPushButton * push_button_reinitialize_data
- QPushButton * push_button_update_input_data
- QSlider * slider_gs_smoothness
- QSlider * slider_gs_radius_ratio
- QSlider * slider_gs_detail_preservation
- QSlider * slider_fs_smoothness
- QSlider * slider_fs_radius
- QProgressBar * progress_bar
- QMenu * fileMenu
- QMenu * viewMenu
- QMenu * settingsMenu
- QMenu * helpMenu
- QActionGroup * alignmentGroup

Menu Actions.

- QAction * loadAct
- QAction * saveAct
- QAction * exitAct
- QAction * flatModeAct
- QAction * wireframeModeAct
- QAction * pointsModeAct
- QAction * setShadersAct
- QAction * setGlobalSmoothingAlgorithmAct
- QAction * setFocalizedSmoothingAlgorithmAct
- QAction * aboutAct
- QAction * userGuideAct
- myDataManager data
- QThread smoothingThread
- GlobalSmoothingTask * smoothingTask
- globalSmoothingAlgorithm currentGlobalSmoothingAlgorithm
- focalizedSmoothingAlgorithm currentFocalizedSmoothingAlgorithm
- bool selectionMode
- bool runningStatus
- bool globalSmoothingStopped

6.5.1 Detailed Description

The myMainWindow class - Main Window for mesh smoothing application.

Definition at line 69 of file myMainWindow.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 myMainWindow()

```
myMainWindow::myMainWindow ( )
```

myMainWindow - default constructor: dynamic building of all the interface

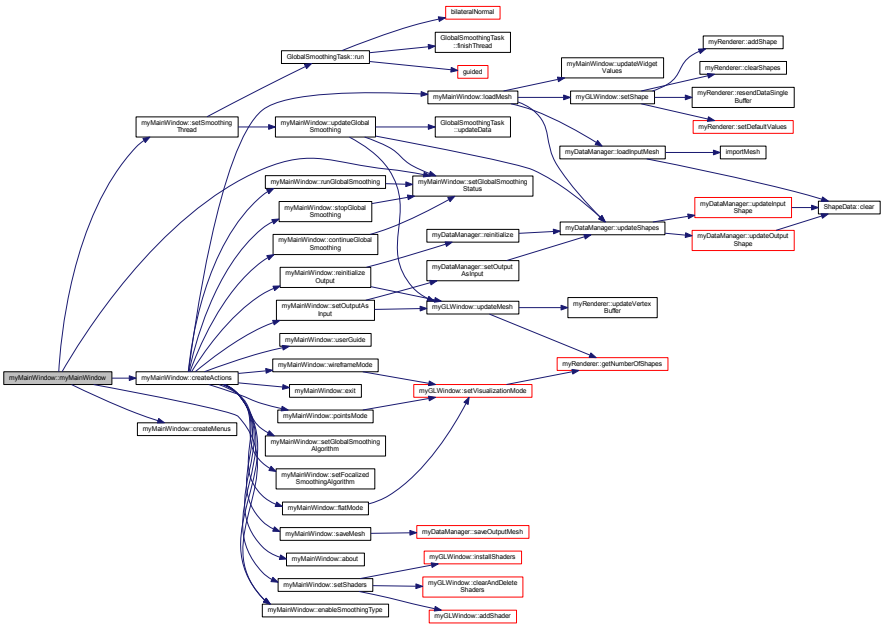
Definition at line 3 of file myMainWindow.cpp.

References control_layout, control_scroll_area, createActions(), createMenus(), currentFocalizedSmoothing↩Algorithm, currentGlobalSmoothingAlgorithm, enableSmoothingType(), fs_algorithm_uniform_laplacian, global↩SmoothingStopped, group_box_data_manipulation, group_box_focalized_smoothing, group_box_global_↩smoothing, group_box_smoothing_type, gs_algorithm_bilateral_normal, gs_status_init, input_mesh_visualizer↩_ptr, layout, layout_data_manipulation, layout_focalized_smoothing, layout_global_smoothing, layout_global_↩smoothing_buttons, layout_smoothing_type, output_mesh_visualizer_ptr, progress_bar, push_button_global_↩smoothing_continue, push_button_global_smoothing_run, push_button_global_smoothing_stop, push_button↩_reinitialize_data, push_button_update_input_data, radio_button_smoothing_type_f, radio_button_smoothing↩_type_g, runningStatus, selectionMode, setGlobalSmoothingStatus(), setSmoothingThread(), slider_fs_radius, slider_fs_smoothness, slider_gs_detail_preservation, slider_gs_radius_ratio, slider_gs_smoothness, and widget.

```
4 {
5     const int control_block_max_width = 250;
6
7     input_mesh_visualizer_ptr = new myGLWindow(this);
8     output_mesh_visualizer_ptr = new myGLWindow(this);
9
10     radio_button_smoothing_type_g = new QRadioButton("Global");
11     radio_button_smoothing_type_f = new QRadioButton("Focalized");
12     radio_button_smoothing_type_g->setChecked(true);
13
14     push_button_global_smoothing_run = new QPushButton("&Run", this);
15     push_button_global_smoothing_stop = new QPushButton("&Stop", this);
16     push_button_global_smoothing_continue = new QPushButton("&Continue
      ", this);
17
18     push_button_reinitialize_data = new QPushButton("&Restart", this);
19     push_button_update_input_data = new QPushButton("&Update", this);
20
21     slider_gs_smoothness = new QSlider(Qt::Horizontal, this);
22     slider_gs_radius_ratio = new QSlider(Qt::Horizontal, this);
23     slider_gs_detail_preservation = new QSlider(Qt::Horizontal, this);
24
25     progress_bar = new QProgressBar(this);
26
27     slider_fs_smoothness = new QSlider(Qt::Horizontal, this);
28     slider_fs_radius = new QSlider(Qt::Horizontal, this);
29
30     layout_smoothing_type = new QHBoxLayout;
31     layout_smoothing_type->addWidget(
      radio_button_smoothing_type_g);
32     layout_smoothing_type->addWidget(
      radio_button_smoothing_type_f);
33
34     layout_global_smoothing_buttons = new QHBoxLayout;
35     layout_global_smoothing_buttons->addWidget(
      push_button_global_smoothing_run);
36     layout_global_smoothing_buttons->addWidget(
      push_button_global_smoothing_stop);
37     layout_global_smoothing_buttons->addWidget(
      push_button_global_smoothing_continue);
38
39     layout_global_smoothing = new QVBoxLayout;
40     layout_global_smoothing->addWidget(new QLabel("Smoothness", this));
41     layout_global_smoothing->addWidget(
      slider_gs_smoothness);
42     layout_global_smoothing->addWidget(new QLabel("Influence radius ratio",this));
43     layout_global_smoothing->addWidget(
      slider_gs_radius_ratio);
44     layout_global_smoothing->addWidget(new QLabel("Detail preservation", this));
45     layout_global_smoothing->addWidget(
      slider_gs_detail_preservation);
46     layout_global_smoothing->addWidget(progress_bar);
47     layout_global_smoothing->addLayout(
      layout_global_smoothing_buttons);
48
49     layout_focalized_smoothing = new QVBoxLayout;
50     layout_focalized_smoothing->addWidget(new QLabel("Smoothness", this));
51     layout_focalized_smoothing->addWidget(
      slider_fs_smoothness);
52     layout_focalized_smoothing->addWidget(new QLabel("Radius", this));
53     layout_focalized_smoothing->addWidget(
      slider_fs_radius);
54
55     layout_data_manipulation = new QHBoxLayout;
56     layout_data_manipulation->addWidget(
      push_button_reinitialize_data);
57     layout_data_manipulation->addWidget(
      push_button_update_input_data);
58
59     group_box_smoothing_type = new QGroupBox(tr("Smoothing type"));
60     group_box_smoothing_type->setMaximumWidth(control_block_max_width);
61     group_box_smoothing_type->setLayout(
      layout_smoothing_type);
62
63     group_box_global_smoothing = new QGroupBox(tr("Global smoothing"));
64     group_box_global_smoothing->setMaximumWidth(control_block_max_width);
65     group_box_global_smoothing->setLayout(
      layout_global_smoothing);
66
67     group_box_focalized_smoothing = new QGroupBox(tr("Focalized smoothing"));
68     group_box_focalized_smoothing->setMaximumWidth(control_block_max_width);
69     group_box_focalized_smoothing->setLayout(
      layout_focalized_smoothing);
70
71     group_box_data_manipulation = new QGroupBox(tr("Data manipulation"));
72     group_box_data_manipulation->setMaximumWidth(control_block_max_width);
73     group_box_data_manipulation->setLayout(
```

```
    layout_data_manipulation);
74
75     control_layout = new QVBoxLayout;
76     control_layout->addWidget(group_box_smoothing_type);
77     control_layout->addWidget(group_box_global_smoothing);
78     control_layout->addWidget(group_box_focalized_smoothing);
79     control_layout->addWidget(group_box_data_manipulation);
80     control_layout->addStretch();
81
82     QWidget * t_widget = new QWidget(this);
83     t_widget->setLayout(control_layout);
84
85     control_scroll_area = new QScrollArea(this);
86     control_scroll_area->setWidget(t_widget);
87     control_scroll_area->setMaximumWidth(control_block_max_width+40);
88
89     layout = new QHBoxLayout;
90     layout->addWidget(input_mesh_visualizer_ptr);
91     layout->addWidget(output_mesh_visualizer_ptr);
92     layout->addWidget(control_scroll_area);
93
94     widget = new QWidget;
95     widget->setLayout(layout);
96     setCentralWidget(widget);
97
98     createActions();
99     createMenus();
100     setSmoothingThread();
101
102     QString message = tr("A context menu is available by right-clicking");
103
104     statusBar()->showMessage(message);
105
106     setWindowTitle(tr("Mesh Smoothing Tool"));
107     setMinimumSize(480, 320);
108     resize(1200, 600);
109
110     enableSmoothingType();
111
112     //setMouseTracking(true);
113     output_mesh_visualizer_ptr->installEventFilter(this);
114
115     selectionMode = 0;
116     runningStatus = 0;
117
118     globalSmoothingStopped = 0;
119
120     setGlobalSmoothingStatus(gs_status_init);
121
122     currentGlobalSmoothingAlgorithm =
    gs_algorithm_bilateral_normal;
123     currentFocalizedSmoothingAlgorithm =
    fs_algorithm_uniform_laplacian;
124 }
```

Here is the call graph for this function:



6.5.2.2 ~myMainWindow()

`myMainWindow::~myMainWindow ( )`

~myMainWindow - destructor

Definition at line 126 of file myMainWindow.cpp.

References smoothingThread.

```
127 {
128     smoothingThread.quit();
129     smoothingThread.wait();
130 }
```

6.5.3 Member Function Documentation

6.5.3.1 about()

```
void myMainWindow::about ( )  [private]
```

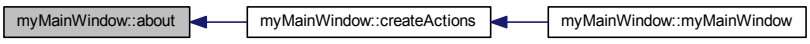
about - show about window

Definition at line 409 of file myMainWindow.cpp.

Referenced by createActions().

```
410 {
411     QMessageBox msgBox;
412     msgBox.setWindowTitle("About");
413     msgBox.setText("Mesh Smoothing Tool v1.0");
414     msgBox.setInformativeText("A mesh processing tool for surface smoothing");
415     int ret = msgBox.exec();
416 }
```

Here is the caller graph for this function:



6.5.3.2 continueGlobalSmoothing()

```
void myMainWindow::continueGlobalSmoothing ( )  [private]
```

continueGlobalSmoothing - continue global smoothing

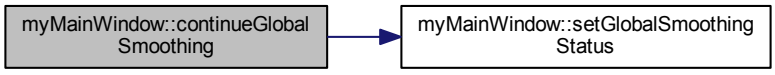
Definition at line 504 of file myMainWindow.cpp.

References gs_status_continuing, setGlobalSmoothingStatus(), and smoothingThread.

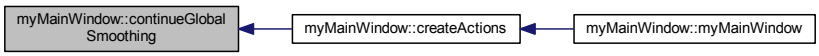
Referenced by createActions().

```
505 {
506     smoothingThread.start();
507     setGlobalSmoothingStatus(gs_status_continuing);
508 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.3 createActions()

```
void myMainWindow::createActions ( )  [private]
```

createActions - create menu actions and connect all signals and slots

Definition at line 132 of file myMainWindow.cpp.

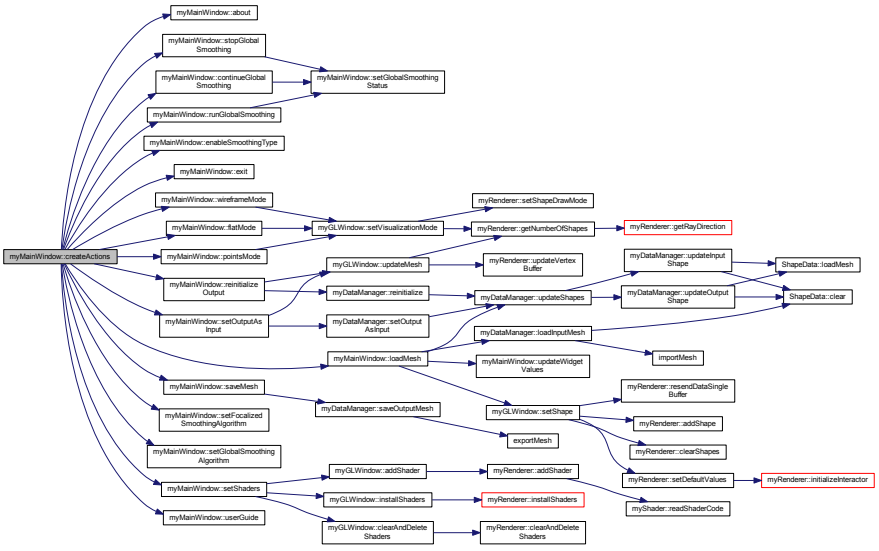
References about(), aboutAct, continueGlobalSmoothing(), enableSmoothingType(), exit(), exitAct, flatMode(), flatModeAct, loadAct, loadMesh(), pointsMode(), pointsModeAct, push_button_global_smoothing_continue, push_button_global_smoothing_run, push_button_global_smoothing_stop, push_button_reinitialize_data, push_button_update_input_data, radio_button_smoothing_type_f, radio_button_smoothing_type_g, reinitializeOutput(), runGlobalSmoothing(), saveAct, saveMesh(), setFocalizedSmoothingAlgorithm(), setFocalizedSmoothingAlgorithmAct, setGlobalSmoothingAlgorithm(), setGlobalSmoothingAlgorithmAct, setOutputAsInput(), setShaders(), setShadersAct, stopGlobalSmoothing(), userGuide(), userGuideAct, wireframeMode(), and wireframeModeAct.

Referenced by myMainWindow().

```
133 {
134     connect(push_button_global_smoothing_run, &QPushButton::released, this,
       &myMainWindow::runGlobalSmoothing);
135     connect(push_button_global_smoothing_stop, &QPushButton::released,
      this, &myMainWindow::stopGlobalSmoothing);
136     connect(push_button_global_smoothing_continue, &
      QPushButton::released, this, &myMainWindow::continueGlobalSmoothing);
137     connect(radio_button_smoothing_type_g, &QRadioButton::released, this, &
      myMainWindow::enableSmoothingType);
138     connect(radio_button_smoothing_type_f, &QRadioButton::released, this, &
      myMainWindow::enableSmoothingType);
139     connect(push_button_reinitialize_data, &QPushButton::released, this, &
      myMainWindow::reinitializeOutput);
140     connect(push_button_update_input_data, &QPushButton::released, this, &
      myMainWindow::setOutputAsInput);
141
142     loadAct = new QAction(tr("&Load"), this);
143     loadAct->setShortcuts(QKeySequence::Open);
144     loadAct->setStatusTip(tr("Load input mesh"));
145     connect(loadAct, &QAction::triggered, this, &myMainWindow::loadMesh);
146
147     saveAct = new QAction(tr("&Save"), this);
148     saveAct->setShortcuts(QKeySequence::Save);
149     saveAct->setStatusTip(tr("Save output mesh"));
150     connect(saveAct, &QAction::triggered, this, &myMainWindow::saveMesh);
151
152     exitAct = new QAction(tr("&Exit"), this);
153     exitAct->setShortcuts(QKeySequence::Close);
154     exitAct->setStatusTip(tr("Exit program"));
155     connect(exitAct, &QAction::triggered, this, &myMainWindow::exit);
156
157     flatModeAct = new QAction(tr("&Flat"), this);
158     flatModeAct->setStatusTip(tr("Flat rendering"));
159     connect(flatModeAct, &QAction::triggered, this, &
      myMainWindow::flatMode);
160
161     wireframeModeAct = new QAction(tr("&Wireframe"), this);
162     wireframeModeAct->setStatusTip(tr("Wireframe rendering"));
163     connect(wireframeModeAct, &QAction::triggered, this, &
```

```
    myMainWindow::wireframeMode);
164
165     pointsModeAct = new QAction(tr("&Points"), this);
166     pointsModeAct->setStatusTip(tr("Point rendering"));
167     connect(pointsModeAct, &QAction::triggered, this, &
    myMainWindow::pointsMode);
168
169     setShadersAct = new QAction(tr("&Set shaders"), this);
170     setShadersAct->setStatusTip(tr("Set shaders"));
171     connect(setShadersAct, &QAction::triggered, this, &
    myMainWindow::setShaders);
172
173     setGlobalSmoothingAlgorithmAct = new QAction(tr("&Set global smoothing
     algorithm"), this);
174     setGlobalSmoothingAlgorithmAct->setStatusTip(tr("Set global smoothing
     algorithm"));
175     connect(setGlobalSmoothingAlgorithmAct, &QAction::triggered, this, &
    myMainWindow::setGlobalSmoothingAlgorithm);
176
177     setFocalizedSmoothingAlgorithmAct = new QAction(tr("&Set focalized
     smoothing algorithm"), this);
178     setFocalizedSmoothingAlgorithmAct->setStatusTip(tr("Set focalized
     smoothing algorithm"));
179     connect(setFocalizedSmoothingAlgorithmAct, &QAction::triggered, this,
    &myMainWindow::setFocalizedSmoothingAlgorithm);
180
181     aboutAct = new QAction(tr("&About"), this);
182     aboutAct->setStatusTip(tr("About"));
183     connect(aboutAct, &QAction::triggered, this, &myMainWindow::about);
184
185     userGuideAct = new QAction(tr("&User Guide"), this);
186     userGuideAct->setStatusTip(tr("User Guide"));
187     connect(userGuideAct, &QAction::triggered, this, &
    myMainWindow::userGuide);
188 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.5.3.4 createMenus()

```
void myMainWindow::createMenus ( )  [private]
```

createMenus - create menus

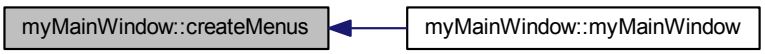
Definition at line 190 of file myMainWindow.cpp.

References aboutAct, exitAct, fileMenu, flatModeAct, helpMenu, loadAct, pointsModeAct, saveAct, setFocalized↩SmoothingAlgorithmAct, setGlobalSmoothingAlgorithmAct, setShadersAct, settingsMenu, userGuideAct, view↩Menu, and wireframeModeAct.

Referenced by myMainWindow().

```
191 {
192     fileMenu = menuBar()->addMenu(tr("&File"));
193     fileMenu->addAction(loadAct);
194     fileMenu->addAction(saveAct);
195     fileMenu->addAction(exitAct);
196 
197     viewMenu = menuBar()->addMenu(tr("&View"));
198     viewMenu->addAction(flatModeAct);
199     viewMenu->addAction(wireframeModeAct);
200     viewMenu->addAction(pointsModeAct);
201 
202     settingsMenu = menuBar()->addMenu(tr("&Settings"));
203     settingsMenu->addAction(setShadersAct);
204     settingsMenu->addAction(setGlobalSmoothingAlgorithmAct);
205     settingsMenu->addAction(setFocalizedSmoothingAlgorithmAct)
    ;
206 
207     helpMenu = menuBar()->addMenu(tr("&Help"));
208     helpMenu->addAction(aboutAct);
209     helpMenu->addAction(userGuideAct);
210 }
```

Here is the caller graph for this function:



6.5.3.5 enableSmoothingType()

```
void myMainWindow::enableSmoothingType ( )  [private]
```

enableSmoothingType - enable widgets for current smoothing type

Definition at line 560 of file myMainWindow.cpp.

References group_box_focalized_smoothing, group_box_global_smoothing, and radio_button_smoothing_type_g.

Referenced by createActions(), and myMainWindow().

```
561 {
562     if (radio_button_smoothing_type_g->isChecked())
563     {
564         group_box_global_smoothing->setEnabled(true);
565         group_box_focalized_smoothing->setEnabled(false);
566     }
567     else
568     {
569         group_box_global_smoothing->setEnabled(false);
570         group_box_focalized_smoothing->setEnabled(true);
571     }
572 }
```

Here is the caller graph for this function:

6.5.3.6 eventFilter()

```
bool myMainWindow::eventFilter (
            QObject * object,
            QEvent * event )  [protected]
```

eventFilter - Qt event filter

Parameters

object	current object
event	current event

Returns

success

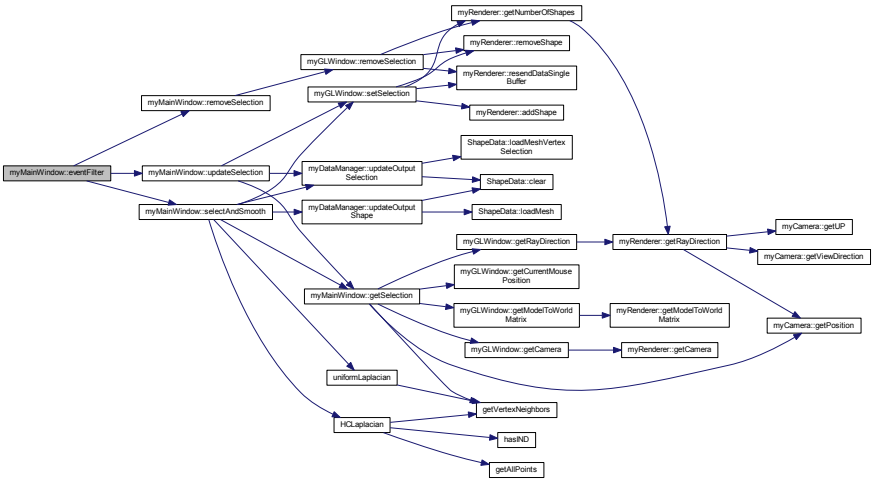
Definition at line 621 of file myMainWindow.cpp.

References output_mesh_visualizer_ptr, removeSelection(), selectAndSmooth(), selectionMode, and updateSelection().

```
622 {
623     if (object == output_mesh_visualizer_ptr)
624     {
625         if (event->type() == QEvent::MouseMove)
626         {
627             QMouseEvent * e = static_cast<QMouseEvent *>(event);
628             if (e->modifiers() & Qt::ShiftModifier)
629             {
630                 selectionMode = 1;
631                 updateSelection();
632                 return true;
633             }
634             else
635             {
```

```
636                 selectionMode = 0;
637                 removeSelection();
638                 return false;
639             }
640         }
641         if (event->type() == QEvent::MouseButtonRelease)
642         {
643             QMouseEvent * e = static_cast<QMouseEvent *>(event);
644             if (e->modifiers() & Qt::ShiftModifier)
645             {
646                 if (e->button() == Qt::LeftButton)
647                 {
648                     cout << "Focalized Smoothing ... ";
649                     selectionMode = 1;
650                     selectAndSmooth();
651                     return true;
652                 }
653             }
654         }
655     }
656     return QWidget::eventFilter(object, event);
657 }
```

Here is the call graph for this function:



6.5.3.7 exit()

```
void myMainWindow::exit ( )  [private]
```

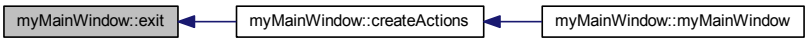
exit - close the application

Definition at line 267 of file myMainWindow.cpp.

Referenced by createActions().

```
268 {
269     close();
270 }
```

Here is the caller graph for this function:



6.5.3.8 flatMode()

```
void myMainWindow::flatMode ( )  [private]
```

flatMode - set triangle rendering

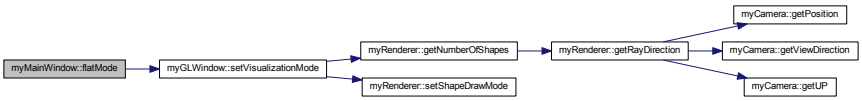
Definition at line 272 of file myMainWindow.cpp.

References e_draw_faces, input_mesh_visualizer_ptr, output_mesh_visualizer_ptr, and myGLWindow::set↩VisualizationMode().

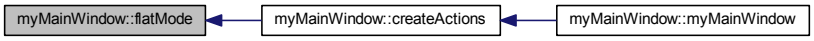
Referenced by createActions().

```
273 {
274     input_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_faces);
275     output_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_faces);
276 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.9 getSelection()

```
void myMainWindow::getSelection (
            vector< size_t > & selected_vertices_ids )  [private]
```

getSelection - get selected vertices for the current mouse position in the output mesh visualizer

Parameters

selected_vertices_ids	- output vector of selected vertices IDs

Definition at line 659 of file myMainWindow.cpp.

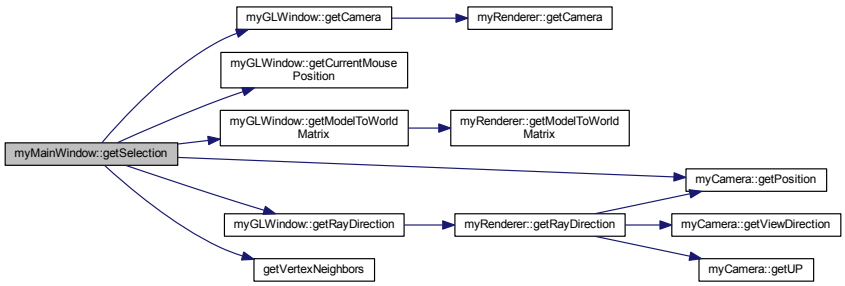
References data, myGLWindow::getCamera(), myGLWindow::getCurrentMousePosition(), myGLWindow::getModelToWorldMatrix(), myCamera::getPosition(), myGLWindow::getRayDirection(), getVertexNeighbors(), myDataManager::output_mesh, output_mesh_visualizer_ptr, and slider_fs_radius.

Referenced by selectAndSmooth(), and updateSelection().

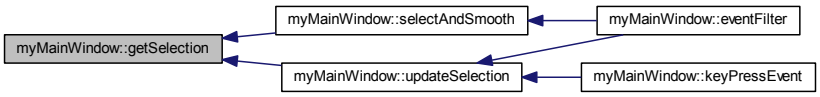
```
660 {
661     selected_vertices_ids.clear();
662     int k = slider_fs_radius->value();
663     glm::vec2 mousePos = output_mesh_visualizer_ptr->
    getCurrentMousePosition();
664     glm::vec3 dir = output_mesh_visualizer_ptr->
    getRayDirection(mousePos);
665     //cout << dir.x << " " << dir.y << " " << dir.z << endl;
666     vector<TriMesh::VertexHandle> selected_vertices;
667     glm::mat4 trMatrix = output_mesh_visualizer_ptr->
    getModelToWorldMatrix();
668     //compute neighborhood
669     float min_dist = 999999.0f;
670     TriMesh::VertexHandle nearest_vertex_handle;
671     for (TriMesh::FaceIter f_it = data.output_mesh.faces_begin(); f_it !=
    data.output_mesh.faces_end(); f_it++)
672     {
673         glm::vec3 a, b, c;
674         TriMesh::VertexHandle a_vh, b_vh, c_vh;
675
676         TriMesh::FaceVertexIter fv_it = data.output_mesh.fv_iter(*f_it);
677         a = glm::vec3(data.output_mesh.point(*fv_it)[0], data.
    output_mesh.point(*fv_it)[1], data.output_mesh.point(*fv_it)[2]);
678         a = glm::vec3(trMatrix*glm::vec4(a, 1.0));
679         a_vh = *fv_it;
680         fv_it++;
681         b = glm::vec3(data.output_mesh.point(*fv_it)[0], data.
    output_mesh.point(*fv_it)[1], data.output_mesh.point(*fv_it)[2]);
682         b = glm::vec3(trMatrix*glm::vec4(b, 1.0));
683         b_vh = *fv_it;
684         fv_it++;
685         c = glm::vec3(data.output_mesh.point(*fv_it)[0], data.
    output_mesh.point(*fv_it)[1], data.output_mesh.point(*fv_it)[2]);
686         c = glm::vec3(trMatrix*glm::vec4(c, 1.0));
687         c_vh = *fv_it;
688
689         glm::vec3 ab = b - a;
690         glm::vec3 ac = c - a;
691
692         glm::vec3 normal = glm::cross(ab, ac);
693
694         glm::vec3 o = output_mesh_visualizer_ptr->
    getCamera()->getPosition();
695         float val = glm::dot(dir, normal);
696         if (val == 0.0f) continue;
697         float ti = glm::dot((a - o), normal) / val;
698         glm::vec3 temp = dir*ti;
699         glm::vec3 pi = o + temp;
700         glm::vec3 v0 = b - a;
701         glm::vec3 v1 = c - a;
702         glm::vec3 v2 = pi - a;
703         float d00 = glm::dot(v0, v0);
704         float d01 = glm::dot(v0, v1);
705         float d11 = glm::dot(v1, v1);
706         float d20 = glm::dot(v2, v0);
707         float d21 = glm::dot(v2, v1);
708         double denom = glm::dot(d00, d11) - glm::dot(d01, d01);
709         float v = (d11 * d20 - d01 * d21) / denom;
710         float w = (d00 * d21 - d01 * d20) / denom;
711         float u = 1.0 - v - w;
712
713         bool liesOutside = v>1 || v<0 || w>1 || w<0 || u>1 || u<0;
714
715         if (!liesOutside)
716         {
717             //cout << "intersect" << endl;
718             if (ti < min_dist  && ti >= 0)
```

```
719             {
720                 if (u>v)
721                 {
722                     if (u > w)
723                         nearest_vertex_handle = a_vh;
724                     else
725                         nearest_vertex_handle = c_vh;
726                 }
727                 else if (v > w)
728                     nearest_vertex_handle = b_vh;
729                 else
730                     nearest_vertex_handle = c_vh;
731                 min_dist = ti;
732             }
733 
734         }
735     }
736     if (nearest_vertex_handle.is_valid())
737     {
738         getVertexNeighbors(data.output_mesh, nearest_vertex_handle, k,
    selected_vertices);
739         selected_vertices.push_back(nearest_vertex_handle);
740     }
741 
742     for (size_t i = 0; i < selected_vertices.size(); i++)
743         selected_vertices_ids.push_back(selected_vertices[i].idx());
744 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.10 keyPressEvent()

```
void myMainWindow::keyPressEvent (
            QKeyEvent * e )  [protected]
```

keyPressEvent - Qt key press event

Parameters

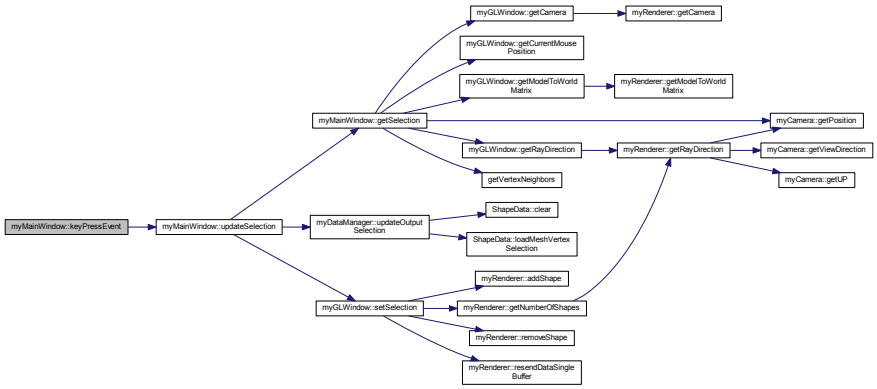
e	event

Definition at line 592 of file myMainWindow.cpp.

References output_mesh_visualizer_ptr, selectionMode, and updateSelection().

```
593 {
594     switch (e->key())
595     {
596     case Qt::Key::Key_Shift:
597         if (selectionMode == 0)
598         {
599             selectionMode = 1;
600             updateSelection();
601             output_mesh_visualizer_ptr->repaint();
602         }
603         break;
604     }
605 }
```

Here is the call graph for this function:



6.5.3.11 keyReleaseEvent()

```
void myMainWindow::keyReleaseEvent (
            QKeyEvent * e )  [protected]
```

keyReleaseEvent - Qt key release event

Parameters

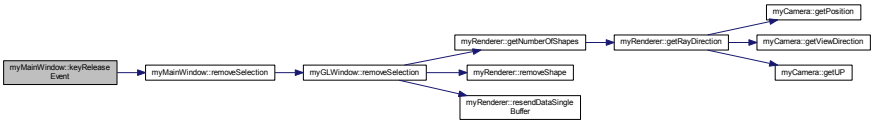
e	event

Definition at line 607 of file myMainWindow.cpp.

References removeSelection(), and selectionMode.

```
608 {
609     switch (e->key())
610     {
611     case Qt::Key::Key_Shift:
612         if (selectionMode == 1)
613         {
614             selectionMode = 0;
615             removeSelection();
616         }
617         break;
618     }
619 }
```

Here is the call graph for this function:



6.5.3.12 loadMesh()

```
void myMainWindow::loadMesh ( )  [private]
```

loadMesh - load input mesh from a file supported by OpenMesh library

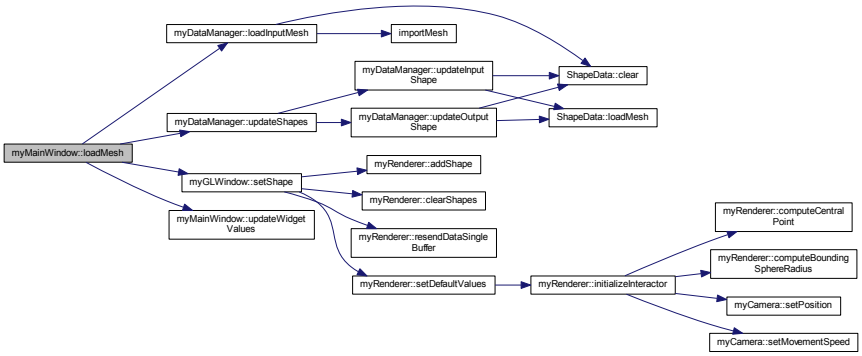
Definition at line 241 of file myMainWindow.cpp.

References data, myDataManager::input_mesh_shape, input_mesh_visualizer_ptr, myDataManager::loadInput↩ Mesh(), myDataManager::output_mesh_shape, output_mesh_visualizer_ptr, myGLWindow::setShape(), myData↩ Manager::updateShapes(), and updateWidgetValues().

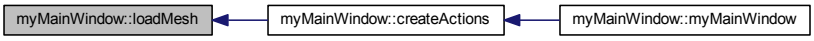
Referenced by createActions().

```
242 {
243     QString fileName = QFileDialog::getOpenFileName(this,
244         tr("Load Mesh"), "",
245         tr("mesh file format (*.obj *.off *.ply *.stl);;All Files (*)"));
246     if (fileName == "") return;
247     statusBar()->showMessage("Loading ... "+fileName);
248     data.loadInputMesh(fileName.toStdString());
249     data.updateShapes();
250     input_mesh_visualizer_ptr->setShape(&data.
    input_mesh_shape);
251     output_mesh_visualizer_ptr->setShape(&data.
    output_mesh_shape);
252     statusBar()->showMessage("Done");
253     updateWidgetValues();
254 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.13 pointsMode()

```
void myMainWindow::pointsMode ( )  [private]
```

pointsMode - set point rendering

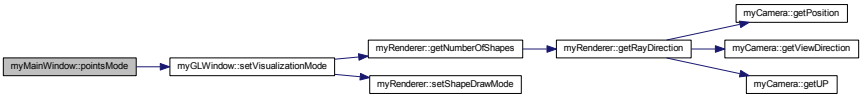
Definition at line 284 of file myMainWindow.cpp.

References e_draw_points, input_mesh_visualizer_ptr, output_mesh_visualizer_ptr, and myGLWindow::set↩ VisualizationMode().

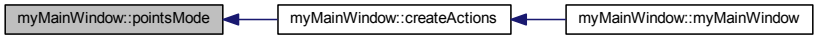
Referenced by createActions().

```
285 {
286     input_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_points);
287     output_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_points);
288 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.14 reinitializeOutput()

```
void myMainWindow::reinitializeOutput ( )  [private]
```

reinitializeOutput - reinitialize output mesh (regarding input mesh)

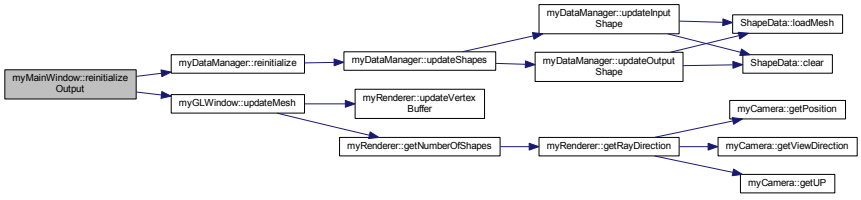
Definition at line 574 of file myMainWindow.cpp.

References data, input_mesh_visualizer_ptr, output_mesh_visualizer_ptr, myDataManager::reinitialize(), and myGLWindow::updateMesh().

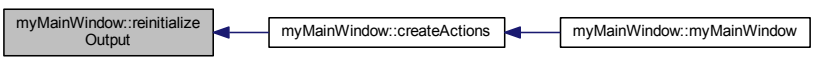
Referenced by createActions().

```
575 {
576     data.reinitialize();
577     input_mesh_visualizer_ptr->updateMesh();
578     input_mesh_visualizer_ptr->repaint();
579     output_mesh_visualizer_ptr->updateMesh();
580     output_mesh_visualizer_ptr->repaint();
581 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.15 removeSelection()

```
void myMainWindow::removeSelection ( )  [private]
```

removeSelection - remove selection from data manager and visualization

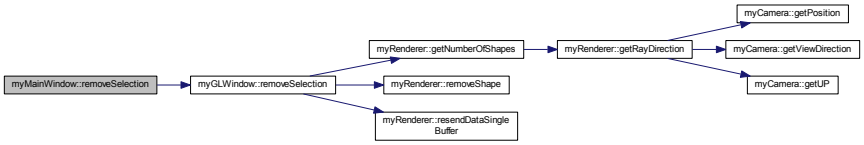
Definition at line 778 of file myMainWindow.cpp.

References output_mesh_visualizer_ptr, and myGLWindow::removeSelection().

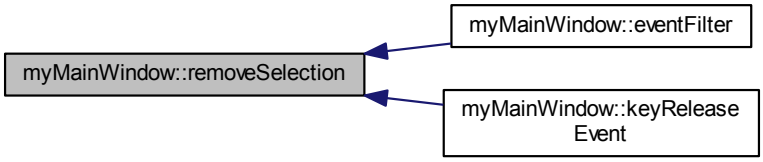
Referenced by eventFilter(), and keyReleaseEvent().

```
779 {
780     output_mesh_visualizer_ptr->removeSelection();
781     output_mesh_visualizer_ptr->repaint();
782 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.16 runGlobalSmoothing()

```
void myMainWindow::runGlobalSmoothing ( )  [private]
```

runGlobalSmoothing - run global smoothing algorithm

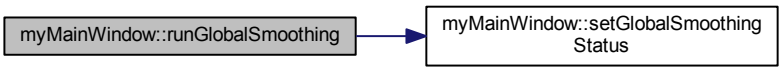
Definition at line 510 of file myMainWindow.cpp.

References GlobalSmoothingTask::algorithm_flag, currentGlobalSmoothingAlgorithm, GlobalSmoothingTask::currentGlobalSmoothingIteration, GlobalSmoothingTask::finalGlobalSmoothingIteration, gs_status_started, GlobalSmoothingTask::iteration_step_size, GlobalSmoothingTask::n_vertex_iterations, setGlobalSmoothingStatus(), GlobalSmoothingTask::sigma_c_ratio, GlobalSmoothingTask::sigma_s, slider_gs_detail_preservation, slider_gs_radius_ratio, slider_gs_smoothness, smoothingTask, and smoothingThread.

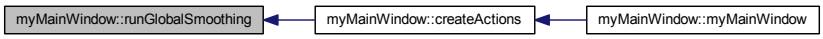
Referenced by createActions().

```
511 {
512     const int default_iteration_step_size = 1;
513     int smoothness_i = slider_gs_smoothness->value();
514     float smoothness = static_cast<float>(slider_gs_smoothness->value());
515     float radius_ratio = static_cast<float>(slider_gs_radius_ratio->value());
516     float detail_preservation = static_cast<float>(slider_gs_detail_preservation
    ->value());
517     smoothingTask->n_vertex_iterations = static_cast<int>(smoothness/30.0f
    + 7.0f);
518     smoothingTask->sigma_c_ratio = 0.5 + radius_ratio / 20.0f + smoothness/100.0f
    ;
519     smoothingTask->sigma_s = 0.8f-detail_preservation * 0.006f;
520     cout << smoothingTask->sigma_s << endl;
521     smoothingTask->iteration_step_size = default_iteration_step_size;
522     smoothingTask->currentGlobalSmoothingIteration = 0;
523     smoothingTask->finalGlobalSmoothingIteration = smoothness_i /
     default_iteration_step_size;
524     smoothingTask->algorithm_flag =
    currentGlobalSmoothingAlgorithm;
525     smoothingThread.start();
526     setGlobalSmoothingStatus(gs_status_started);
527 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.17 saveMesh()

```
void myMainWindow::saveMesh ( )  [private]
```

saveMesh - save output mesh to a file supported by OpenMesh library

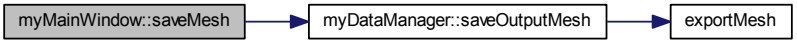
Definition at line 256 of file myMainWindow.cpp.

References data, and myDataManager::saveOutputMesh().

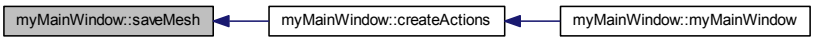
Referenced by createActions().

```
257 {
258     QString fileName = QFileDialog::getSaveFileName(this,
259         tr("Save Mesh"), "",
260         tr("mesh file format (*.obj *.off *.ply *.stl);;All Files (*)"));
261     if (fileName == "") return;
262     statusBar()->showMessage("Saving ... " + fileName);
263     data.saveOutputMesh(fileName.toStdString());
264     statusBar()->showMessage("Done");
265 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.18 selectAndSmooth()

```
void myMainWindow::selectAndSmooth ( )  [private]
```

selectAndSmooth - select and perform focalized smoothing regarding control widget values

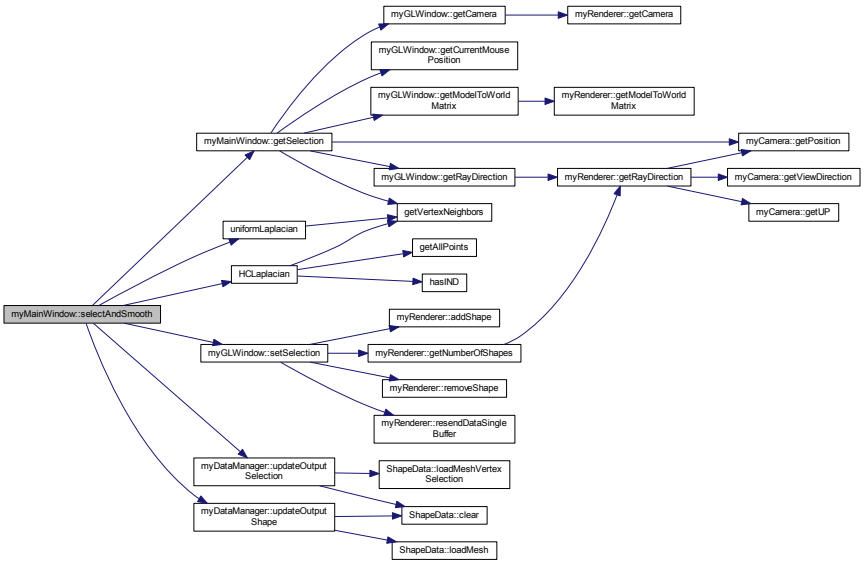
Definition at line 758 of file myMainWindow.cpp.

References currentFocalizedSmoothingAlgorithm, data, fs_algorithm_hc_laplacian, fs_algorithm_uniform_↩ laplacian, getSelection(), HCLaplacian(), myDataManager::output_mesh, output_mesh_visualizer_ptr, radio_↩ button_smoothing_type_f, runningStatus, myDataManager::selection, myGLWindow::setSelection(), slider_fs↩ _smoothness, uniformLaplacian(), myDataManager::updateOutputSelection(), and myDataManager::update↩ OutputShape().

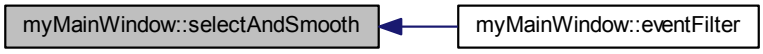
Referenced by eventFilter().

```
759 {
760     if (!runningStatus && radio_button_smoothing_type_f->
    isChecked())
761     {
762         vector<size_t> selected_vertices_ids;
763         getSelection(selected_vertices_ids);
764         int num_its = slider_fs_smoothness->value();
765         if (currentFocalizedSmoothingAlgorithm ==
    fs_algorithm_uniform_laplacian)
766             data.output_mesh = uniformLaplacian(
    data.output_mesh, num_its, 1.0f, selected_vertices_ids);
767         else if ((currentFocalizedSmoothingAlgorithm ==
    fs_algorithm_hc_laplacian))
768             data.output_mesh = HCLaplacian(data.
    output_mesh, num_its, 0.5f,0.5f,selected_vertices_ids);
769         else
770             data.output_mesh = uniformLaplacian(
    data.output_mesh, num_its, 1.0f, selected_vertices_ids);
771         data.updateOutputShape();
772         data.updateOutputSelection(selected_vertices_ids);
773         output_mesh_visualizer_ptr->setSelection(&
    data.selection);
774         output_mesh_visualizer_ptr->repaint();
775     }
776 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.19 setFocalizedSmoothingAlgorithm()

```
void myMainWindow::setFocalizedSmoothingAlgorithm ( )  [private]
```

setFocalizedSmoothingAlgorithm - set focalized smoothing algorithm

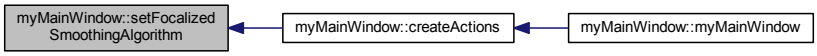
Definition at line 381 of file myMainWindow.cpp.

References currentFocalizedSmoothingAlgorithm, focalizedSmoothingAlgorithmLabels, fs_algorithm_hc_laplacian, and fs_algorithm_uniform_laplacian.

Referenced by createActions().

```
382 {
383     QInputDialog qDialog;
384 
385     QStringList items;
386     items << QString(focalizedSmoothingAlgorithmLabels[
    fs_algorithm_uniform_laplacian].c_str());
387     items << QString(focalizedSmoothingAlgorithmLabels[
    fs_algorithm_hc_laplacian].c_str());
388 
389     qDialog.setOptions(QInputDialog::UseListViewForComboBoxItems);
390     qDialog.setComboBoxItems(items);
391     qDialog.setWindowTitle("Focalized Smoothing Algorithm");
392     qDialog.setLabelText("Choose focalized smoothing algorithm: ");
393 
394     qDialog.setTextValue(focalizedSmoothingAlgorithmLabels[
    currentFocalizedSmoothingAlgorithm].c_str());
395 
396     if (qDialog.exec())
397     {
398         if (qDialog.textValue().toStdString() ==
    focalizedSmoothingAlgorithmLabels[
    fs_algorithm_uniform_laplacian])
399             currentFocalizedSmoothingAlgorithm =
    fs_algorithm_uniform_laplacian;
400         else if (qDialog.textValue().toStdString() ==
    focalizedSmoothingAlgorithmLabels[
    fs_algorithm_hc_laplacian])
401             currentFocalizedSmoothingAlgorithm =
    fs_algorithm_hc_laplacian;
402         else
403             currentFocalizedSmoothingAlgorithm =
    fs_algorithm_uniform_laplacian;
404         QString message = tr(focalizedSmoothingAlgorithmLabels[
    currentFocalizedSmoothingAlgorithm].c_str());
405         statusBar()->showMessage(message);
406     }
407 }
```

Here is the caller graph for this function:



6.5.3.20 setGlobalSmoothingAlgorithm()

```
void myMainWindow::setGlobalSmoothingAlgorithm ( )  [private]
```

setGlobalSmoothingAlgorithm - set global smoothing algorithm

Definition at line 353 of file myMainWindow.cpp.

References currentGlobalSmoothingAlgorithm, globalSmoothingAlgorithmLabels, gs_algorithm_bilateral_normal, and gs_algorithm_guided.

Referenced by createActions().

```
354 {
355     QInputDialog qDialog;
356 
357     QStringList items;
358     items << QString(globalSmoothingAlgorithmLabels[
    gs_algorithm_bilateral_normal].c_str());
359     items << QString(globalSmoothingAlgorithmLabels[
    gs_algorithm_guided].c_str());
360 
361     qDialog.setOptions(QInputDialog::UseListViewForComboBoxItems);
362     qDialog.setComboBoxItems(items);
363     qDialog.setWindowTitle("Global Smoothing Algorithm");
364     qDialog.setLabelText("Choose global smoothing algorithm: ");
365 
366     qDialog.setTextValue(globalSmoothingAlgorithmLabels[
    currentGlobalSmoothingAlgorithm].c_str());
367 
368     if (qDialog.exec())
369     {
370         if (qDialog.textValue().toStdString() == globalSmoothingAlgorithmLabels
    [gs_algorithm_bilateral_normal])
371             currentGlobalSmoothingAlgorithm =
    gs_algorithm_bilateral_normal;
372         else if (qDialog.textValue().toStdString() ==
    globalSmoothingAlgorithmLabels[gs_algorithm_guided])
373             currentGlobalSmoothingAlgorithm =
    gs_algorithm_guided;
374         else
375             currentGlobalSmoothingAlgorithm =
    gs_algorithm_bilateral_normal;
376         QString message = tr(globalSmoothingAlgorithmLabels[
    currentGlobalSmoothingAlgorithm].c_str());
377         statusBar()->showMessage(message);
378     }
379 }
```

Here is the caller graph for this function:

6.5.3.21 setGlobalSmoothingStatus()

```
void myMainWindow::setGlobalSmoothingStatus (
            globalSmoothingStatus current_status )  [private]
```

setGlobalSmoothingStatus - set current global smoothing status

Parameters

current_status	- global smoothing status

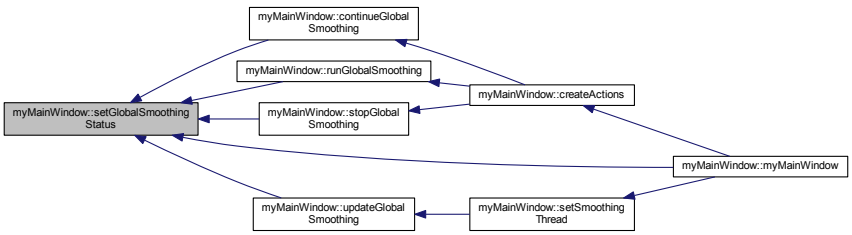
Definition at line 424 of file myMainWindow.cpp.

References globalSmoothingStopped, group_box_data_manipulation, group_box_smoothing_type, gs_status↩_continuing, gs_status_init, gs_status_started, gs_status_stopped, gs_status_stopping, progress_bar, push↩_button_global_smoothing_continue, push_button_global_smoothing_run, push_button_global_smoothing_stop, and runningStatus.

Referenced by continueGlobalSmoothing(), myMainWindow(), runGlobalSmoothing(), stopGlobalSmoothing(), and updateGlobalSmoothing().

```
425 {
426     if (current_status == gs_status_init)
427     {
428         push_button_global_smoothing_run->setEnabled(true);
429         push_button_global_smoothing_stop->setEnabled(false);
430         push_button_global_smoothing_continue->setEnabled(false);
431         group_box_smoothing_type->setEnabled(true);
432         group_box_data_manipulation->setEnabled(true);
433         menuBar()->setEnabled(true);
434         runningStatus = 0;
435         globalSmoothingStopped = 0;
436         QString message = tr("Ready ... ");
437         statusBar()->showMessage(message);
438     }
439     else if (current_status == gs_status_started)
440     {
441         runningStatus = 1;
442         globalSmoothingStopped = 0;
443         progress_bar->setValue(0);
444         push_button_global_smoothing_run->setEnabled(false);
445         push_button_global_smoothing_stop->setEnabled(true);
446         push_button_global_smoothing_continue->setEnabled(false);
447         group_box_smoothing_type->setEnabled(false);
448         group_box_data_manipulation->setEnabled(false);
449         menuBar()->setEnabled(false);
450         QString message = tr("Running ... ");
451         statusBar()->showMessage(message);
452     }
453     else if (current_status == gs_status_stopping)
454     {
455         globalSmoothingStopped = 1;
456         push_button_global_smoothing_stop->setEnabled(false);
457         push_button_global_smoothing_continue->setEnabled(false);
458         QString message = tr("Stopping ... ");
459         statusBar()->showMessage(message);
460     }
461     else if (current_status == gs_status_stopped)
462     {
463         push_button_global_smoothing_run->setEnabled(true);
464         group_box_smoothing_type->setEnabled(true);
465         group_box_data_manipulation->setEnabled(true);
466         menuBar()->setEnabled(true);
467         push_button_global_smoothing_stop->setEnabled(false);
468         push_button_global_smoothing_continue->setEnabled(true);
469         runningStatus = 0;
470         globalSmoothingStopped = 0;
471         QString message = tr("Stopped ... ");
472         statusBar()->showMessage(message);
473     }
474     else if (current_status == gs_status_continuing)
475     {
476         runningStatus = 1;
477         globalSmoothingStopped = 0;
478         push_button_global_smoothing_run->setEnabled(false);
479         push_button_global_smoothing_stop->setEnabled(true);
480         push_button_global_smoothing_continue->setEnabled(false);
481         group_box_smoothing_type->setEnabled(false);
482         group_box_data_manipulation->setEnabled(false);
483         menuBar()->setEnabled(false);
484         QString message = tr("Continuing ... ");
485         statusBar()->showMessage(message);
486     }
487 }
```

Here is the caller graph for this function:



6.5.3.22 setOutputAsInput()

```
void myMainWindow::setOutputAsInput ( )  [private]
```

setOutputAsInput - set output mesh as new input mesh

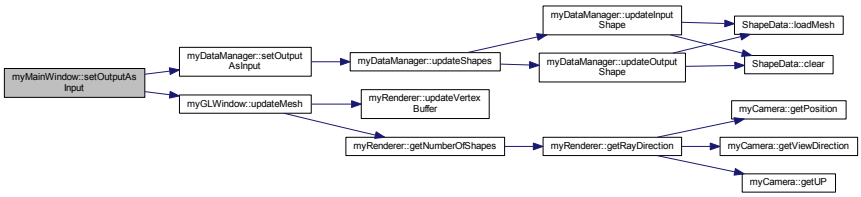
Definition at line 583 of file myMainWindow.cpp.

References data, input_mesh_visualizer_ptr, output_mesh_visualizer_ptr, myDataManager::setOutputAsInput(), and myGLWindow::updateMesh().

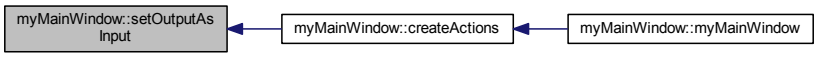
Referenced by createActions().

```
584 {
585     data.setOutputAsInput();
586     input_mesh_visualizer_ptr->updateMesh();
587     input_mesh_visualizer_ptr->repaint();
588     output_mesh_visualizer_ptr->updateMesh();
589     output_mesh_visualizer_ptr->repaint();
590 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.23 setShaders()

```
void myMainWindow::setShaders ( )  [private]
```

setShaders - set shaders profile (customizable)

Definition at line 290 of file myMainWindow.cpp.

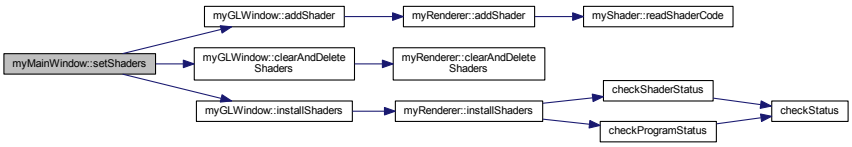
References myGLWindow::addShader(), myGLWindow::clearAndDeleteShaders(), input_mesh_visualizer_ptr, myGLWindow::installShaders(), and output_mesh_visualizer_ptr.

Referenced by createActions().

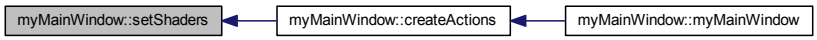
```
291 {
292     QInputDialog qDialog;
293 
294     QStringList items;
295     items << QString("Flat");
296     items << QString("Phong");
297     items << QString("Normal Map");
298 
299     qDialog.setOptions(QInputDialog::UseListViewForComboBoxItems);
300     qDialog.setComboBoxItems(items);
301     qDialog.setWindowTitle("Shaders");
302     qDialog.setLabelText("Choose shader options: ");
303 
304     if (!qDialog.exec()) return;
305 
306     if (qDialog.textValue().toStdString() == "Flat")
307     {
308         input_mesh_visualizer_ptr->
    clearAndDeleteShaders();
309         input_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodeFlat.glsl");
310         input_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodeFlat.glsl");
311         input_mesh_visualizer_ptr->installShaders();
312 
313         output_mesh_visualizer_ptr->
    clearAndDeleteShaders();
314         output_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodeFlat.glsl");
315         output_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodeFlat.glsl");
316         output_mesh_visualizer_ptr->installShaders();
317 
318         QString message = tr(qDialog.textValue().toStdString().c_str());
319         statusBar()->showMessage(message);
320     }
321     else if (qDialog.textValue().toStdString() == "Phong")
322     {
323         input_mesh_visualizer_ptr->
    clearAndDeleteShaders();
324         input_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodePhong.glsl");
325         input_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodePhong.glsl");
326         input_mesh_visualizer_ptr->installShaders();
327 
328         output_mesh_visualizer_ptr->
    clearAndDeleteShaders();
329         output_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodePhong.glsl");
330         output_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodePhong.glsl");
331         output_mesh_visualizer_ptr->installShaders();
332 
333         QString message = tr(qDialog.textValue().toStdString().c_str());
334         statusBar()->showMessage(message);
335     }
336     else if (qDialog.textValue().toStdString() == "Normal Map")
337     {
338         input_mesh_visualizer_ptr->
    clearAndDeleteShaders();
339         input_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodeNormalMap.glsl");
340         input_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodeNormalMap.glsl");
341         input_mesh_visualizer_ptr->installShaders();
```

```
342
343         output_mesh_visualizer_ptr->
    clearAndDeleteShaders();
344         output_mesh_visualizer_ptr->addShader(GL_VERTEX_SHADER, "
    VertexShaderCodeNormalMap.glsl");
345         output_mesh_visualizer_ptr->addShader(GL_FRAGMENT_SHADER, "
    FragmentShaderCodeNormalMap.glsl");
346         output_mesh_visualizer_ptr->installShaders();
347
348         QString message = tr(qDialog.textValue().toStdString().c_str());
349         statusBar()->showMessage(message);
350     }
351 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.24 setSmoothingThread()

```
void myMainWindow::setSmoothingThread ( )  [private]
```

setSmoothingThread - set thread for global smoothing

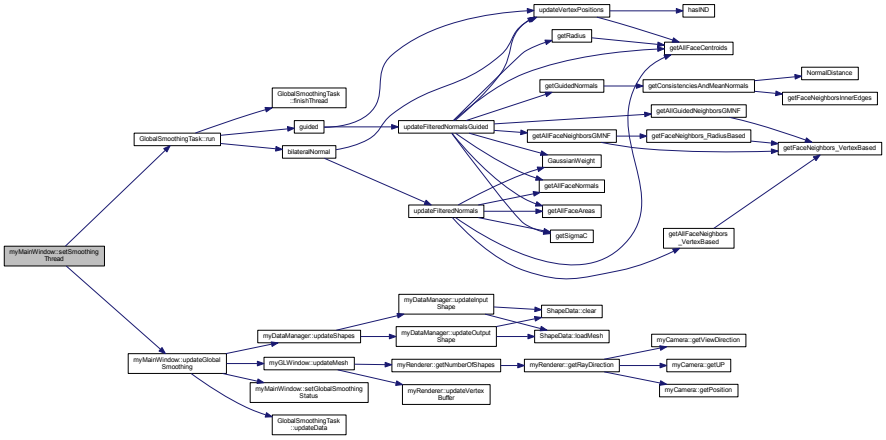
Definition at line 489 of file myMainWindow.cpp.

References GlobalSmoothingTask::current_thread, GlobalSmoothingTask::data, data, GlobalSmoothingTask::run(), smoothingTask, smoothingThread, and updateGlobalSmoothing().

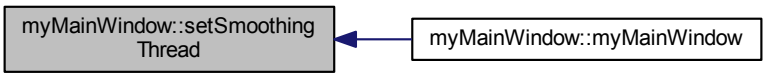
Referenced by myMainWindow().

```
490 {
491     smoothingTask = new GlobalSmoothingTask;
492     smoothingTask->moveToThread(&smoothingThread);
493     smoothingTask->current_thread = &smoothingThread;
494     connect(&smoothingThread, &QThread::started, smoothingTask, &
    GlobalSmoothingTask::run);
495     connect(&smoothingThread, &QThread::finished, this, &
    myMainWindow::updateGlobalSmoothing);
496     smoothingTask->data = &data;
497 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.25 stopGlobalSmoothing()

```
void myMainWindow::stopGlobalSmoothing ( )  [private]
```

stopGlobalSmoothing - stop global smoothing

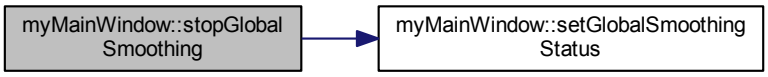
Definition at line 499 of file myMainWindow.cpp.

References gs_status_stopping, and setGlobalSmoothingStatus().

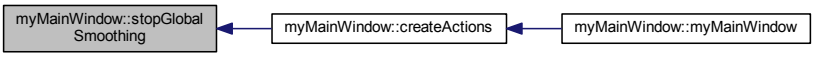
Referenced by createActions().

```
500 {
501     setGlobalSmoothingStatus(gs_status_stopping);
502 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.26 updateGlobalSmoothing()

```
void myMainWindow::updateGlobalSmoothing ( )  [private]
```

updateGlobalSmoothing - global smoothing management (step by step)

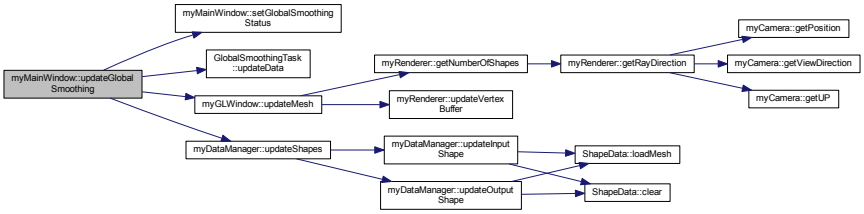
Definition at line 529 of file myMainWindow.cpp.

References GlobalSmoothingTask::currentGlobalSmoothingIteration, data, GlobalSmoothingTask::finalGlobal↩SmoothingIteration, globalSmoothingStopped, gs_status_init, gs_status_stopped, output_mesh_visualizer_ptr, progress_bar, setGlobalSmoothingStatus(), smoothingTask, smoothingThread, GlobalSmoothingTask::update↩Data(), myGLWindow::updateMesh(), and myDataManager::updateShapes().

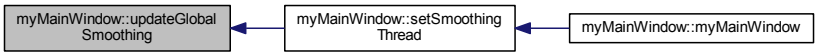
Referenced by setSmoothingThread().

```
530 {
531     int current_iteration = smoothingTask->
      currentGlobalSmoothingIteration;
532     int final_iteration = smoothingTask->
      finalGlobalSmoothingIteration;
533     if (globalSmoothingStopped)
534     {
535         if (current_iteration > 0)
536         {
537             smoothingTask->currentGlobalSmoothingIteration--;
538             current_iteration--;
539         }
540         setGlobalSmoothingStatus(gs_status_stopped);
541     }
542     else if (current_iteration >= final_iteration)
543     {
544         smoothingTask->updateData();
545         data.updateShapes();
546         setGlobalSmoothingStatus(gs_status_init);
547     }
548     else
549     {
550         smoothingTask->updateData();
551         data.updateShapes();
552         smoothingTask->currentGlobalSmoothingIteration ++;
553         smoothingThread.start();
554     }
555     output_mesh_visualizer_ptr->updateMesh();
556     output_mesh_visualizer_ptr->repaint();
557     progress_bar->setValue(static_cast<int>((static_cast<float>(current_iteration) /
      static_cast<float>(final_iteration))*100.0f));
558 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.27 updateSelection()

```
void myMainWindow::updateSelection ( )  [private]
```

updateSelection - update selection in data manager and visualization

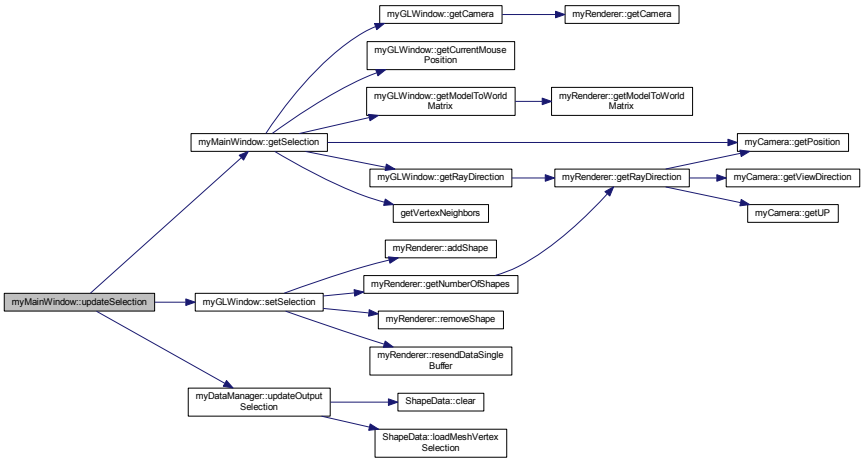
Definition at line 746 of file myMainWindow.cpp.

References data, getSelection(), output_mesh_visualizer_ptr, radio_button_smoothing_type_f, runningStatus, my↩DataManager::selection, myGLWindow::setSelection(), and myDataManager::updateOutputSelection().

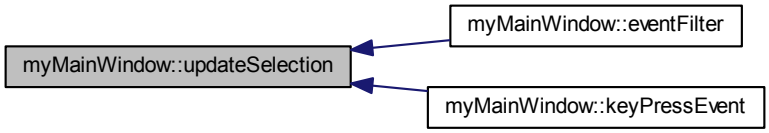
Referenced by eventFilter(), and keyPressEvent().

```
747 {
748     if (!runningStatus && radio_button_smoothing_type_f->
    isChecked())
749     {
750         vector<size_t> selected_vertices_ids;
751         getSelection(selected_vertices_ids);
752         data.updateOutputSelection(selected_vertices_ids);
753         output_mesh_visualizer_ptr->setSelection(&
    data.selection);
754         output_mesh_visualizer_ptr->repaint();
755     }
756 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3.28 updateWidgetValues()

```
void myMainWindow::updateWidgetValues ( )  [private]
```

updateWidgetValues - set default slider widget values (min, max and current)

Definition at line 212 of file myMainWindow.cpp.

References progress_bar, slider_fs_radius, slider_fs_smoothness, slider_gs_detail_preservation, slider_gs_↩ radius_ratio, and slider_gs_smoothness.

Referenced by loadMesh().

```
213 {
215 
216     progress_bar->setMaximum(100);
217     progress_bar->setMinimum(0);
218     progress_bar->setValue(0);
219 
220     slider_gs_smoothness->setMaximum(100);
221     slider_gs_smoothness->setMinimum(0);
```

```
222     slider_gs_smoothness->setValue(5);
223
224     slider_gs_radius_ratio->setMaximum(100);
225     slider_gs_radius_ratio->setMinimum(0);
226     slider_gs_radius_ratio->setValue(10);
227
228     slider_gs_detail_preservation->setMaximum(100);
229     slider_gs_detail_preservation->setMinimum(0);
230     slider_gs_detail_preservation->setValue(50);
231
232     slider_fs_smoothness->setMaximum(50);
233     slider_fs_smoothness->setMinimum(0);
234     slider_fs_smoothness->setValue(3);
235
236     slider_fs_radius->setMaximum(20);
237     slider_fs_radius->setMinimum(0);
238     slider_fs_radius->setValue(3);
239 }
```

Here is the caller graph for this function:

6.5.3.29 userGuide()

```
void myMainWindow::userGuide ( )  [private]
```

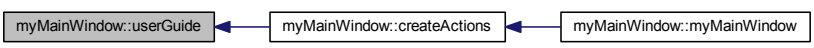
userGuide - show user guide

Definition at line 418 of file myMainWindow.cpp.

Referenced by createActions().

```
419 {
420     QString link = "http://www.google.com";
421     QDesktopServices::openUrl(QUrl(link));
422 }
```

Here is the caller graph for this function:



6.5.3.30 wireframeMode()

```
void myMainWindow::wireframeMode ( )  [private]
```

wireframeMode - set wireframe rendering

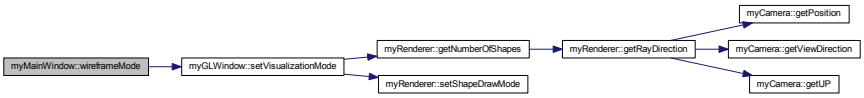
Definition at line 278 of file myMainWindow.cpp.

References e_draw_wireframe, input_mesh_visualizer_ptr, output_mesh_visualizer_ptr, and myGLWindow::setVisualizationMode().

Referenced by createActions().

```
279 {
280     input_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_wireframe);
281     output_mesh_visualizer_ptr->setVisualizationMode(
      e_draw_wireframe);
282 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.5.4 Field Documentation

6.5.4.1 aboutAct

```
QAction* myMainWindow::aboutAct  [private]
```

Definition at line 124 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.2 alignmentGroup

```
QActionGroup* myMainWindow::alignmentGroup  [private]
```

Menu Actions.

Definition at line 114 of file myMainWindow.h.

6.5.4.3 control_layout

```
QVBoxLayout* myMainWindow::control_layout  [private]
```

layout for control widgets

Definition at line 77 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.4 control_scroll_area

```
QScrollArea* myMainWindow::control_scroll_area  [private]
```

scroll area for control widgets

Definition at line 76 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.5 currentFocalizedSmoothingAlgorithm

```
focalizedSmoothingAlgorithm myMainWindow::currentFocalizedSmoothingAlgorithm  [private]
```

focalized smoothing algorithm flag

Definition at line 133 of file myMainWindow.h.

Referenced by myMainWindow(), selectAndSmooth(), and setFocalizedSmoothingAlgorithm().

6.5.4.6 currentGlobalSmoothingAlgorithm

```
globalSmoothingAlgorithm myMainWindow::currentGlobalSmoothingAlgorithm  [private]
```

global smoothing algorithm flag

Definition at line 132 of file myMainWindow.h.

Referenced by myMainWindow(), runGlobalSmoothing(), and setGlobalSmoothingAlgorithm().

6.5.4.7 data

```
myDataManager myMainWindow::data  [private]
```

Data manager for iunput and output meshes

Definition at line 127 of file myMainWindow.h.

Referenced by getSelection(), loadMesh(), reinitializeOutput(), saveMesh(), selectAndSmooth(), setOutputAs↩Input(), setSmoothingThread(), updateGlobalSmoothing(), and updateSelection().

6.5.4.8 exitAct

```
QAction* myMainWindow::exitAct  [private]
```

Definition at line 117 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.9 fileMenu

```
QMenu* myMainWindow::fileMenu  [private]
```

file menu

Definition at line 108 of file myMainWindow.h.

Referenced by createMenus().

6.5.4.10 flatModeAct

```
QAction* myMainWindow::flatModeAct  [private]
```

Definition at line 118 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.11 globalSmoothingStopped

```
bool myMainWindow::globalSmoothingStopped  [private]
```

if global smoothing is stopped

Definition at line 138 of file myMainWindow.h.

Referenced by myMainWindow(), setGlobalSmoothingStatus(), and updateGlobalSmoothing().

6.5.4.12 group_box_data_manipulation

```
QGroupBox* myMainWindow::group_box_data_manipulation  [private]
```

group box for data manipulation controls

Definition at line 82 of file myMainWindow.h.

Referenced by myMainWindow(), and setGlobalSmoothingStatus().

6.5.4.13 group_box_focalized_smoothing

```
QGroupBox* myMainWindow::group_box_focalized_smoothing  [private]
```

group box for focalized smoothing controls

Definition at line 81 of file myMainWindow.h.

Referenced by enableSmoothingType(), and myMainWindow().

6.5.4.14 group_box_global_smoothing

```
QGroupBox* myMainWindow::group_box_global_smoothing  [private]
```

group box for global smoothing controls

Definition at line 80 of file myMainWindow.h.

Referenced by enableSmoothingType(), and myMainWindow().

6.5.4.15 group_box_smoothing_type

```
QGroupBox* myMainWindow::group_box_smoothing_type  [private]
```

group box for smoothing type selection

Definition at line 79 of file myMainWindow.h.

Referenced by myMainWindow(), and setGlobalSmoothingStatus().

6.5.4.16 helpMenu

```
QMenu* myMainWindow::helpMenu  [private]
```

help menu

Definition at line 111 of file myMainWindow.h.

Referenced by createMenus().

6.5.4.17 input_mesh_visualizer_ptr

```
myGLWindow* myMainWindow::input_mesh_visualizer_ptr  [private]
```

visualization widget for input mesh

Definition at line 71 of file myMainWindow.h.

Referenced by flatMode(), loadMesh(), myMainWindow(), pointsMode(), reinitializeOutput(), setOutputAsInput(), setShaders(), and wireframeMode().

6.5.4.18 layout

```
QHBoxLayout* myMainWindow::layout  [private]
```

main layout contained in main window

Definition at line 75 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.19 layout_data_manipulation

```
QHBoxLayout* myMainWindow::layout_data_manipulation  [private]
```

Definition at line 89 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.20 layout_focalized_smoothing

```
QVBoxLayout* myMainWindow::layout_focalized_smoothing  [private]
```

Definition at line 88 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.21 layout_global_smoothing

```
QVBoxLayout* myMainWindow::layout_global_smoothing  [private]
```

Definition at line 86 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.22 layout_global_smoothing_buttons

```
QHBoxLayout* myMainWindow::layout_global_smoothing_buttons  [private]
```

Definition at line 87 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.23 layout_smoothing_type

```
QHBoxLayout* myMainWindow::layout_smoothing_type  [private]
```

Auxiliar Layouts.

Definition at line 85 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.24 loadAct

```
QAction* myMainWindow::loadAct  [private]
```

Definition at line 115 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.25 output_mesh_visualizer_ptr

```
myGLWindow* myMainWindow::output_mesh_visualizer_ptr  [private]
```

visualization widget for output mesh

Definition at line 72 of file myMainWindow.h.

Referenced by eventFilter(), flatMode(), getSelection(), keyPressEvent(), loadMesh(), myMainWindow(), pointsMode(), reinitializeOutput(), removeSelection(), selectAndSmooth(), setOutputAsInput(), setShaders(), updateGlobalSmoothing(), updateSelection(), and wireframeMode().

6.5.4.26 pointsModeAct

```
QAction* myMainWindow::pointsModeAct  [private]
```

Definition at line 120 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.27 progress_bar

```
QProgressBar* myMainWindow::progress_bar  [private]
```

global smoothing progress bar

Definition at line 106 of file myMainWindow.h.

Referenced by myMainWindow(), setGlobalSmoothingStatus(), updateGlobalSmoothing(), and updateWidget↩ Values().

6.5.4.28 push_button_global_smoothing_continue

```
QPushButton* myMainWindow::push_button_global_smoothing_continue  [private]
```

global smoothing continue button

Definition at line 96 of file myMainWindow.h.

Referenced by createActions(), myMainWindow(), and setGlobalSmoothingStatus().

6.5.4.29 push_button_global_smoothing_run

```
QPushButton* myMainWindow::push_button_global_smoothing_run  [private]
```

global smoothing run button

Definition at line 94 of file myMainWindow.h.

Referenced by createActions(), myMainWindow(), and setGlobalSmoothingStatus().

6.5.4.30 push_button_global_smoothing_stop

```
QPushButton* myMainWindow::push_button_global_smoothing_stop  [private]
```

global smoothing stop button

Definition at line 95 of file myMainWindow.h.

Referenced by createActions(), myMainWindow(), and setGlobalSmoothingStatus().

6.5.4.31 push_button_reinitialize_data

```
QPushButton* myMainWindow::push_button_reinitialize_data  [private]
```

reinitialize button

Definition at line 97 of file myMainWindow.h.

Referenced by createActions(), and myMainWindow().

6.5.4.32 push_button_update_input_data

```
QPushButton* myMainWindow::push_button_update_input_data  [private]
```

update button

Definition at line 98 of file myMainWindow.h.

Referenced by createActions(), and myMainWindow().

6.5.4.33 radio_button_smoothing_type_f

```
QRadioButton* myMainWindow::radio_button_smoothing_type_f  [private]
```

smoothing type selection radio button focalized smoothing

Definition at line 92 of file myMainWindow.h.

Referenced by createActions(), myMainWindow(), selectAndSmooth(), and updateSelection().

6.5.4.34 radio_button_smoothing_type_g

```
QRadioButton* myMainWindow::radio_button_smoothing_type_g  [private]
```

smoothing type selection radio button global smoothing

Definition at line 91 of file myMainWindow.h.

Referenced by createActions(), enableSmoothingType(), and myMainWindow().

6.5.4.35 runningStatus

```
bool myMainWindow::runningStatus  [private]
```

if global smoothing is running

Definition at line 136 of file myMainWindow.h.

Referenced by myMainWindow(), selectAndSmooth(), setGlobalSmoothingStatus(), and updateSelection().

6.5.4.36 saveAct

```
QAction* myMainWindow::saveAct  [private]
```

Definition at line 116 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.37 selectionMode

```
bool myMainWindow::selectionMode  [private]
```

if user is selecting something in output mesh visualizer

Definition at line 135 of file myMainWindow.h.

Referenced by eventFilter(), keyPressEvent(), keyReleaseEvent(), and myMainWindow().

6.5.4.38 setFocalizedSmoothingAlgorithmAct

```
QAction* myMainWindow::setFocalizedSmoothingAlgorithmAct  [private]
```

Definition at line 123 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.39 setGlobalSmoothingAlgorithmAct

```
QAction* myMainWindow::setGlobalSmoothingAlgorithmAct  [private]
```

Definition at line 122 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.40 setShadersAct

```
QAction* myMainWindow::setShadersAct  [private]
```

Definition at line 121 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.41 settingsMenu

```
QMenu* myMainWindow::settingsMenu  [private]
```

settings menu

Definition at line 110 of file myMainWindow.h.

Referenced by createMenus().

6.5.4.42 slider_fs_radius

```
QSlider* myMainWindow::slider_fs_radius  [private]
```

focalized smoothing radius slider

Definition at line 104 of file myMainWindow.h.

Referenced by getSelection(), myMainWindow(), and updateWidgetValues().

6.5.4.43 slider_fs_smoothness

```
QSlider* myMainWindow::slider_fs_smoothness  [private]
```

focalized smoothing smoothness slider

Definition at line 103 of file myMainWindow.h.

Referenced by myMainWindow(), selectAndSmooth(), and updateWidgetValues().

6.5.4.44 slider_gs_detail_preservation

```
QSlider* myMainWindow::slider_gs_detail_preservation  [private]
```

global smoothing detail preservation slider

Definition at line 102 of file myMainWindow.h.

Referenced by myMainWindow(), runGlobalSmoothing(), and updateWidgetValues().

6.5.4.45 slider_gs_radius_ratio

```
QSlider* myMainWindow::slider_gs_radius_ratio  [private]
```

global smoothing radius ratio slider

Definition at line 101 of file myMainWindow.h.

Referenced by myMainWindow(), runGlobalSmoothing(), and updateWidgetValues().

6.5.4.46 slider_gs_smoothness

```
QSlider* myMainWindow::slider_gs_smoothness  [private]
```

global smoothing smoothness slider

Definition at line 100 of file myMainWindow.h.

Referenced by myMainWindow(), runGlobalSmoothing(), and updateWidgetValues().

6.5.4.47 smoothingTask

```
GlobalSmoothingTask* myMainWindow::smoothingTask  [private]
```

global smoothing external step task

Definition at line 130 of file myMainWindow.h.

Referenced by runGlobalSmoothing(), setSmoothingThread(), and updateGlobalSmoothing().

6.5.4.48 smoothingThread

```
QThread myMainWindow::smoothingThread  [private]
```

thread for global smoothing

Definition at line 129 of file myMainWindow.h.

Referenced by continueGlobalSmoothing(), runGlobalSmoothing(), setSmoothingThread(), updateGlobal↩ Smoothing(), and ∼myMainWindow().

6.5.4.49 userGuideAct

```
QAction* myMainWindow::userGuideAct  [private]
```

Definition at line 125 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

6.5.4.50 viewMenu

```
QMenu* myMainWindow::viewMenu  [private]
```

view menu

Definition at line 109 of file myMainWindow.h.

Referenced by createMenus().

6.5.4.51 widget

```
QWidget* myMainWindow::widget  [private]
```

main widget contained in main window

Definition at line 74 of file myMainWindow.h.

Referenced by myMainWindow().

6.5.4.52 wireframeModeAct

```
QAction* myMainWindow::wireframeModeAct  [private]
```

Definition at line 119 of file myMainWindow.h.

Referenced by createActions(), and createMenus().

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

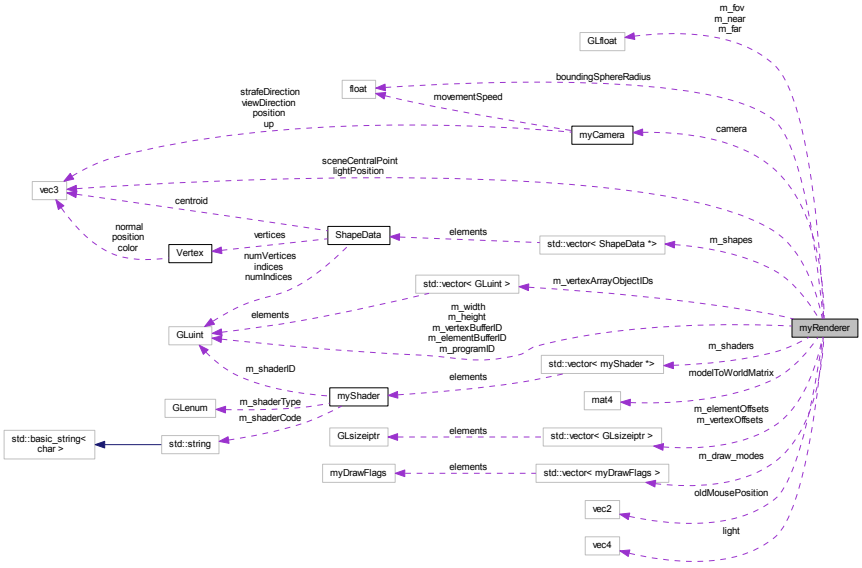
- myMainWindow.h
- myMainWindow.cpp

6.6 myRenderer Class Reference

The myRenderer class - Renderer for multiple shapes (triangular meshes) visualization.

```
#include <myRenderer.h>
```

Collaboration diagram for myRenderer:



Public Member Functions

- myRenderer ()

  myRenderer - default constructor
- virtual ~myRenderer ()

  ~myRenderer - destructor
- void addShape (ShapeData *_sd)

  addShape - add a shape (shape data) for rendering
- void addShape (ShapeData *_sd, myDrawFlags _draw_mode)

addShape - add a shape (shape data) for rendering specifying its rendering mode

- void setShapeDrawMode (size_t _shape_index, myDrawFlags _mode)

  setShapeDrawMode - set rendering mode for a specific shape
- void removeShape (size_t index)

  removeShape - remove a shape
- void addShader (GLenum _shaderType, const string &_fileName)

  addShader - create and add a shader
- void addShader (myShader *_shader)

  addShader - add a shader
- void clearShapes ()

  clearShapes - erase shape containers without deleting pointers
- void clearAndDeleteShapes ()

  clearAndDeleteShapes - erate shape containers and delete items
- void clearShaders ()

  clearShaders - erase shader containers without deleting pointers
- void clearAndDeleteShaders ()

  clearAndDeleteShaders - erase shader containers and delete items
- void createProgram ()

  createProgram - create OpenGl program in the current context
- bool installShaders ()

  installShaders - compile and attach shaders to the main program (m_programID) in the current context
- void sendDataSingleBuffer ()

  sendDataSingleBuffer - send shape data to the GPU using OpenGL
- void updateVertexBuffer (size_t index)

  updateVertexBuffer - update and send vertex data for a single shape
- void resendDataSingleBuffer ()

  resendDataSingleBuffer - resend shape data to the GPU using OpenGL
- void draw ()

  draw - draw al shapes considering their rendering modes
- void initialize ()

  initialize - renderer initialization
- void initializeInteractor ()

  initializeInteractor - default initialization of renderer parameters based on shape data
- void setDefaultValues ()

  setDefaultValues - set custom renderer parameters
- void computeCentralPoint ()

  computeCentralPoint - compute the centroid of all points of all shapes
- void computeBoundingSphereRadius ()

  computeBoundingSphereRadius - compute the bounding sphere radius centered on the centroid of all points of all shapes
- void rotateObjects (const glm::vec2 &newMousePosition)

  rotateObjects - rotate all shapes (model to world matrix manipulation) regarding a mouse movement based interaction
- void translateCamera (const glm::vec2 &newMousePosition)

  translateCamera - translate all shapes (model to world matrix manipulation) regarding a mouse movement interaction
- void zoom (float delta)

  zoom - translate camera position (forward/backward) regarding a step size
- GLuint getProgramID ()

  getProgramID - get OpenGL program ID of renderer
- GLsizeiptr getIndexOffsetAt (size_t pos)

  getIndexOffsetAt - get indices offset for a shape
- GLuint getVertexArrayObjectIDAt (size_t pos)

getVertexArrayObjectIDAt - get vertex array object ID (OpenGL) for a shape

- size_t getNumberOfShapes ()

  getNumberOfShapes - get the number of shapes contained in the renderer

- glm::vec3 getRayDirection (glm::vec2 &pos)

  getRayDirection - computation of the direction of a ray starting in camera position and ending in a point of the corresponding window contained in the near plane

- GLuint getWidth ()

  getters

- GLuint getHeight ()
- GLfloat getFOV ()
- GLfloat getNear ()
- GLfloat getFar ()
- glm::mat4 getModelToWorldMatrix ()
- glm::vec3 getSceneCentralPoint ()
- float getBoundingSphereRadius ()
- myCamera * getCamera ()
- void setWidth (GLuint _width)

  setters

- void setHeight (GLuint _height)
- void setFOV (GLfloat _fov)
- void setNear (GLfloat _near)
- void setFar (GLfloat _far)
- void setModelToWorldMatrix (glm::mat4 &_modelToWorldMatrix)
- void setSceneCentralPoint (glm::vec3 &_point)
- void setBoundingSphereRadius (float _radius)

Private Attributes

- GLuint m_width

  Rendering.

- GLuint m_height

  m_height - window height

- GLfloat m_fov

  m_fov - field of view (FOV)

- GLfloat m_near

  m_near - near plane distance

- GLfloat m_far

  m_far - far plane distance

- myCamera camera

  camera - main camera for renderer

- glm::vec4 light

  light - main light "color"

- glm::vec3 lightPosition

  lightPosition - main light position

- glm::mat4 modelToWorldMatrix

  Interaction.

- glm::vec2 oldMousePosition

  oldMousePosition - old mouse position for mouse move interactions

- glm::vec3 sceneCentralPoint

  sceneCentralPoint - scene central point

- float boundingSphereRadius

boundingSphereRadius - bounding sphere radius (regarding the central point)

- GLuint m_programID

  Data.
- GLuint m_vertexBufferID

  m_vertexBufferID - main vertex buffer ID
- GLuint m_elementBufferID

  m_elementBufferID - main element buffer ID
- vector< myShader * > m_shaders

  m_shaders - vector containing all shaders
- vector< ShapeData * > m_shapes

  m_shapes - vector containing all shapes
- vector< GLsizeiptr > m_vertexOffsets

  m_vertexOffsets - vector containing all shape vertices offsets
- vector< GLsizeiptr > m_elementOffsets

  m_elementOffsets - vector containing all shape indices offsets
- vector< GLuint > m_vertexArrayObjectIDs

  m_vertexArrayObjectIDs - vector containing all vertex array objects IDs
- vector< myDrawFlags > m_draw_modes

  m_draw_modes - vector containing all shape rendering modes

6.6.1 Detailed Description

The myRenderer class - Renderer for multiple shapes (triangular meshes) visualization.

Definition at line 34 of file myRenderer.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 myRenderer()

```
myRenderer::myRenderer ( )
```

myRenderer - default constructor

Definition at line 36 of file myRenderer.cpp.

References light, lightPosition, m_far, m_fov, m_height, m_near, m_width, modelToWorldMatrix, and sceneCentralPoint.

```
37 {
38     //currentDrawFlag = e_draw_faces;
39     light = glm::vec4(0.1f,0.1f,0.1f,1.0f);
40     lightPosition = glm::vec3(800.f, 800.f, 800.f);
41     sceneCentralPoint = glm::vec3(0.0f, 0.0f, 0.0f);
42     m_width = 1000;
43     m_height = 800;
44     m_fov = 45.0f;
45     m_near = 1.0f;
46     m_far = 1000.0f;
47     modelToWorldMatrix = glm::translate(glm::vec3(0.0f, 0.0f, 0.0f));
48 }
```

6.6.2.2 ~myRenderer()

```
myRenderer::~myRenderer ( )  [virtual]
```

~myRenderer - destructor

Definition at line 50 of file myRenderer.cpp.

References m_elementBufferID, m_programID, and m_vertexBufferID.

```
51 {
52     glDeleteBuffers(1, &m_vertexBufferID);
53     glDeleteBuffers(1, &m_elementBufferID);
54     glUseProgram(0);
55     glDeleteProgram(m_programID);
56 }
```

6.6.3 Member Function Documentation

6.6.3.1 addShader() [1/2]

```
void myRenderer::addShader (
            GLenum _shaderType,
            const string & _fileName )
```

addShader - create and add a shader

Parameters

_shaderType	shader type
_fileName	shader code file name

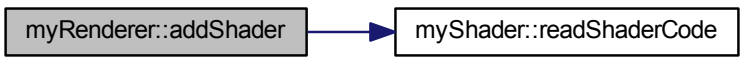
Definition at line 86 of file myRenderer.cpp.

References m_shaders, and myShader::readShaderCode().

Referenced by myGLWindow::addShader(), and myGLWindow::initializeGL().

```
87 {
88     myShader * t_ptr_shader = new myShader(_shaderType);
89     t_ptr_shader->readShaderCode(_fileName);
90     m_shaders.push_back(t_ptr_shader);
91 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.2 addShader() [2/2]

```
void myRenderer::addShader (
            myShader * _shader )
```

addShader - add a shader

Parameters

_shader	pointer to the shader

Definition at line 93 of file myRenderer.cpp.

References m_shaders.

```
94 {
95     m_shaders.push_back(_shader);
96 }
```

6.6.3.3 addShape() [1/2]

```
void myRenderer::addShape (
            ShapeData * _sd )
```

addShape - add a shape (shape data) for rendering

Parameters

_sd	shape data pointer to be added

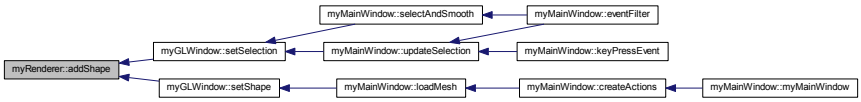
Definition at line 58 of file myRenderer.cpp.

References e_draw_faces, m_draw_modes, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, and m_↩ vertexOffsets.

Referenced by myGLWindow::setSelection(), and myGLWindow::setShape().

```
59 {
60     m_shapes.push_back(_sd);
61     m_vertexOffsets.push_back(0);
62     m_elementOffsets.push_back(0);
63     m_vertexArrayObjectIDs.push_back(0);
64     m_draw_modes.push_back(e_draw_faces);
65 }
```

Here is the caller graph for this function:



6.6.3.4 addShape() [2/2]

```
void myRenderer::addShape (
            ShapeData * _sd,
            myDrawFlags _draw_mode )
```

addShape - add a shape (shape data) for rendering specifying its rendering mode

Parameters

_sd	shape data pointer to be added
_draw_mode	rendering mode

Definition at line 67 of file myRenderer.cpp.

References m_draw_modes, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, and m_vertexOffsets.

```
68 {
69     m_shapes.push_back(_sd);
70     m_vertexOffsets.push_back(0);
71     m_elementOffsets.push_back(0);
72     m_vertexArrayObjectIDs.push_back(0);
73     m_draw_modes.push_back(_draw_mode);
74 }
```

6.6.3.5 clearAndDeleteShaders()

```
void myRenderer::clearAndDeleteShaders ( )
```

clearAndDeleteShaders - erase shader containers and delete items

Definition at line 135 of file myRenderer.cpp.

References m_programID, and m_shaders.

Referenced by myGLWindow::clearAndDeleteShaders().

```
136 {
137     for (size_t i = 0; i < m_shaders.size(); i++)
138     {
139         glDetachShader(m_programID,m_shaders[i]->getShaderID());
140         delete m_shaders[i];
141     }
142     m_shaders.clear();
143 }
```

Here is the caller graph for this function:

6.6.3.6 clearAndDeleteShapes()

```
void myRenderer::clearAndDeleteShapes ( )
```

clearAndDeleteShapes - erate shape containers and delete items

Definition at line 116 of file myRenderer.cpp.

References m_draw_modes, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, and m_vertexOffsets.

```
117 {
118     for (size_t i = 0; i < m_shapes.size(); i++)
119     {
120         glDeleteVertexArrays(1, &m_vertexArrayObjectIDs[i]);
121         delete m_shapes[i];
122     }
123     m_shapes.clear();
124     m_vertexOffsets.clear();
125     m_elementOffsets.clear();
126     m_vertexArrayObjectIDs.clear();
127     m_draw_modes.clear();
128 }
```

6.6.3.7 clearShaders()

```
void myRenderer::clearShaders ( )
```

clearShaders - erase shader containers without deleting pointers

Definition at line 130 of file myRenderer.cpp.

References m_shaders.

```
131 {
132     m_shaders.clear();
133 }
```

6.6.3.8 clearShapes()

```
void myRenderer::clearShapes ( )
```

clearShapes - erase shape containers without deleting pointers

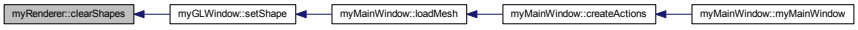
Definition at line 103 of file myRenderer.cpp.

References m_draw_modes, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, and m_vertexOffsets.

Referenced by myGLWindow::setShape().

```
104 {
105     for (size_t i = 0; i < m_vertexArrayObjectIDs.size(); i++)
106     {
107         glDeleteVertexArrays(1, &m_vertexArrayObjectIDs[i]);
108     }
109     m_shapes.clear();
110     m_vertexOffsets.clear();
111     m_elementOffsets.clear();
112     m_vertexArrayObjectIDs.clear();
113     m_draw_modes.clear();
114 }
```

Here is the caller graph for this function:



6.6.3.9 computeBoundingSphereRadius()

```
void myRenderer::computeBoundingSphereRadius ( )
```

computeBoundingSphereRadius - compute the bounding sphere radius centered on the centroid of all points of all shapes

Definition at line 391 of file myRenderer.cpp.

References boundingSphereRadius, m_shapes, and sceneCentralPoint.

Referenced by initializeInteractor().

```
392 {
393     float max = 0.0f;
394     for (size_t i = 0; i < m_shapes.size(); i++)
395     {
396         for (size_t j = 0; j < m_shapes[i]->numVertices; j++)
397         {
398             glm::vec3 current_point = glm::vec3(m_shapes[i]->vertices[j].position.x,
      m_shapes[i]->vertices[j].position.y, m_shapes[i]->vertices[j].position.z);
399             float dist = glm::distance(current_point,sceneCentralPoint);
400             if (dist>max)
401                 max = dist;
402         }
403     }
404     boundingSphereRadius = max;
405
406 }
```

Here is the caller graph for this function:

6.6.3.10 computeCentralPoint()

```
void myRenderer::computeCentralPoint ( )
```

computeCentralPoint - compute the centroid of all points of all shapes

Definition at line 376 of file myRenderer.cpp.

References m_shapes, and sceneCentralPoint.

Referenced by initializeInteractor().

```
377 {
378     float num = 0.0f;
379     glm::vec3 t_sceneCentralPoint = glm::vec3(0.0f, 0.0f, 0.0f);
380     for (size_t i = 0; i < m_shapes.size(); i++)
381     {
382         for (size_t j = 0; j < m_shapes[i]->numVertices; j++)
383         {
384             t_sceneCentralPoint += glm::vec3(m_shapes[i]->vertices[j].position.x,
      m_shapes[i]->vertices[j].position.y,m_shapes[i]->vertices[j].position.z);
385             num += 1.0f;
386         }
387     }
388     sceneCentralPoint = t_sceneCentralPoint/num;
389 }
```

Here is the caller graph for this function:

6.6.3.11 createProgram()

```
void myRenderer::createProgram ( )
```

createProgram - create OpenGl program in the current context

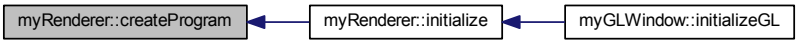
Definition at line 145 of file myRenderer.cpp.

References m_programID.

Referenced by initialize().

```
146 {
147     m_programID = glCreateProgram();
148 }
```

Here is the caller graph for this function:



6.6.3.12 draw()

```
void myRenderer::draw ( )
```

draw - draw al shapes considering their rendering modes

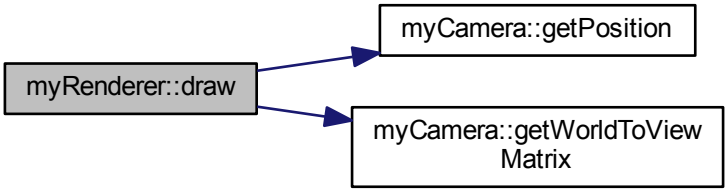
Definition at line 304 of file myRenderer.cpp.

References camera, e_draw_faces, e_draw_points, e_draw_selection, e_draw_wireframe, myCamera::getPosition(), myCamera::getWorldToViewMatrix(), light, lightPosition, m_draw_modes, m_elementOffsets, m_far, m_fov, m_height, m_near, m_programID, m_shapes, m_vertexArrayObjectIDs, m_width, and modelToWorldMatrix.

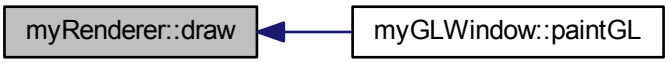
Referenced by myGLWindow::paintGL().

```
305 {
306
307     glViewport(0, 0, m_width, m_height);
308     //glClearColor(0, 0, 0, 1);
309     glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
310
311     glm::mat4 modelToProjectionMatrix;
312     glm::mat4 viewToProjectionMatrix = glm::perspective(m_fov, ((float)
      m_width) / ((float)(m_height)), m_near, m_far);
313     glm::mat4 worldToViewMatrix = camera.getWorldToViewMatrix();
314     glm::mat4 worldToProjectionMatrix = viewToProjectionMatrix * worldToViewMatrix;
315
316     // 0: pointSize            1: not defined            2: not defined
317     glm::vec3 aditionalProperties = glm::vec3(1.0f,0.0f,0.0f);
318
319     GLuint fullTransformationUniformLocation;
320     GLuint modelToWorldMatrixUniformLocation;
321     GLuint lightUniformLocation;
322     GLuint lightPositionUniformLocation;
323     GLuint cameraPositionUniformLocation;
324     GLuint aditionalPropertiesUniformLocation;
325
326     fullTransformationUniformLocation = glGetUniformLocation(m_programID, "
      modelToProjectionMatrix");
327     modelToWorldMatrixUniformLocation = glGetUniformLocation(m_programID, "modelToWorldMatrix");
328     lightUniformLocation = glGetUniformLocation(m_programID, "lightVector");
329     lightPositionUniformLocation = glGetUniformLocation(m_programID, "lightPositionVector");
330     cameraPositionUniformLocation = glGetUniformLocation(m_programID, "cameraPositionVector");
331     aditionalPropertiesUniformLocation = glGetUniformLocation(m_programID, "aditionalProperties"
      );
332
333     modelToProjectionMatrix = worldToProjectionMatrix * modelToWorldMatrix;
334
335     glUniformMatrix4fv(fullTransformationUniformLocation, 1, GL_FALSE, &modelToProjectionMatrix[0][0]);
336     glUniformMatrix4fv(modelToWorldMatrixUniformLocation, 1, GL_FALSE, &modelToWorldMatrix[0][0]);
337     glUniform4fv(lightUniformLocation, 1, &light[0]);
338     glUniform3fv(lightPositionUniformLocation, 1, &lightPosition[0]);
339     glUniform3fv(cameraPositionUniformLocation, 1, &(camera.getPosition())[0]);
340     glUniform3fv(aditionalPropertiesUniformLocation, 1, &aditionalProperties[0]);
341
342     for (size_t i = 0; i < m_shapes.size(); i++)
343     {
344         glBindVertexArray(m_vertexArrayObjectIDs[i]);
345         int currentDrawFlag = m_draw_modes[i];
346         switch (currentDrawFlag)
347         {
348         case e_draw_faces:
349             glDrawElements(GL_TRIANGLES, m_shapes[i]->numIndices, GL_UNSIGNED_INT, (void*)(
      m_elementOffsets[i]));
350             break;
351         case e_draw_wireframe:
352             glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
353             glDrawElements(GL_TRIANGLES, m_shapes[i]->numIndices, GL_UNSIGNED_INT, (void*)(
      m_elementOffsets[i]));
354             glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
355             break;
356         case e_draw_points:
357             glDrawArrays(GL_POINTS, 0, m_shapes[i]->numVertices);
358             break;
359         case e_draw_selection:
360             aditionalProperties[0] = 3.0f;
361             glUniform3fv(aditionalPropertiesUniformLocation, 1, &aditionalProperties[0]);
362             glDrawArrays(GL_POINTS, 0, m_shapes[i]->numVertices);
363             aditionalProperties[0] = 1.0f;
364             break;
365         }
366         glBindVertexArray(0);
367     }
368 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.13 getBoundingSphereRadius()

```
float myRenderer::getBoundingSphereRadius ( )  [inline]
```

Definition at line 196 of file myRenderer.h.

References boundingSphereRadius.

```
196 {return boundingSphereRadius;}
```

6.6.3.14 getCamera()

```
myCamera* myRenderer::getCamera ( )  [inline]
```

Definition at line 197 of file myRenderer.h.

References camera.

Referenced by myGLWindow::getCamera().

```
197 {return &camera;}
```

Here is the caller graph for this function:

6.6.3.15 getFar()

```
GLfloat myRenderer::getFar ( )  [inline]
```

Definition at line 193 of file myRenderer.h.

References m_far.

```
193 {return m_far;}
```

6.6.3.16 getFOV()

```
GLfloat myRenderer::getFOV ( )  [inline]
```

Definition at line 191 of file myRenderer.h.

References m_fov.

```
191 {return m_fov;}
```

6.6.3.17 getHeight()

```
GLuint myRenderer::getHeight ( )  [inline]
```

Definition at line 190 of file myRenderer.h.

References m_height.

```
190 {return m_height;}
```

6.6.3.18 getIndexOffsetAt()

```
GLsizeiptr myRenderer::getIndexOffsetAt (
            size_t pos )  [inline]
```

getIndexOffsetAt - get indices offset for a shape

Parameters

pos	index of the target shape

Returns

offset

Definition at line 167 of file myRenderer.h.

References m_elementOffsets.

```
167 { return m_elementOffsets[pos]; }
```

6.6.3.19 getModelToWorldMatrix()

```
glm::mat4 myRenderer::getModelToWorldMatrix ( )  [inline]
```

Definition at line 194 of file myRenderer.h.

References modelToWorldMatrix.

Referenced by myGLWindow::getModelToWorldMatrix().

```
194 {return modelToWorldMatrix;}
```

Here is the caller graph for this function:

6.6.3.20 getNear()

```
GLfloat myRenderer::getNear ( )  [inline]
```

Definition at line 192 of file myRenderer.h.

References m_near.

```
192 {return m_near;}
```

6.6.3.21 getNumberOfShapes()

```
size_t myRenderer::getNumberOfShapes ( )  [inline]
```

getNumberOfShapes - get the number of shapes contained in the renderer

Returns

number of shapes

Definition at line 178 of file myRenderer.h.

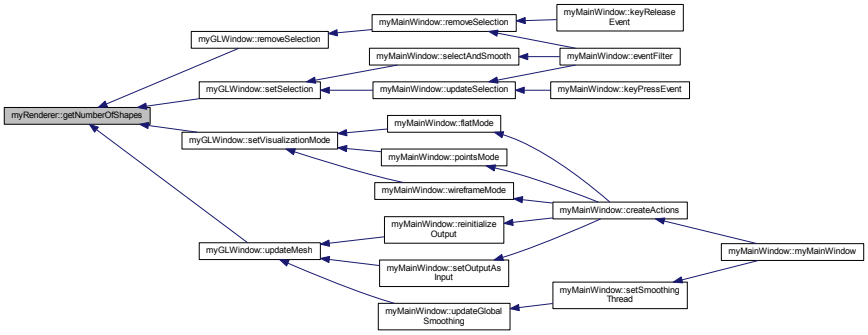
References getRayDirection(), and m_shapes.

Referenced by myGLWindow::removeSelection(), myGLWindow::setSelection(), myGLWindow::setVisualization↩Mode(), and myGLWindow::updateMesh().

```
178 { return m_shapes.size(); }
```

Here is the call graph for this function:

Here is the caller graph for this function:



6.6.3.22 getProgramID()

```
GLuint myRenderer::getProgramID ( )  [inline]
```

getProgramID - get OpenGL program ID of renderer

Returns

OpenGL porgram ID

Definition at line 161 of file myRenderer.h.

References m_programID.

```
161 { return m_programID; }
```

6.6.3.23 getRayDirection()

```
glm::vec3 myRenderer::getRayDirection (
            glm::vec2 & pos )
```

getRayDirection - computation of the direction of a ray starting in camera position and ending in a point of the corresponding window contained in the near plane

Parameters

pos	position in the window (contained in the near plane)

Returns

ray direction

Definition at line 460 of file myRenderer.cpp.

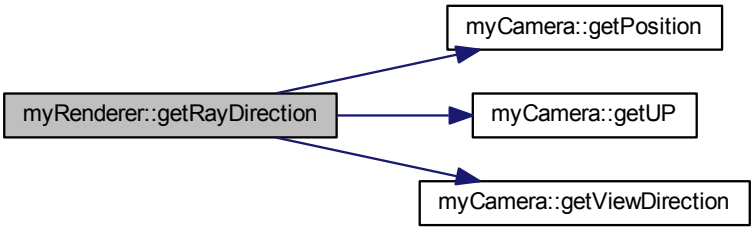
References camera, myCamera::getPosition(), myCamera::getUP(), myCamera::getViewDirection(), m_far, m_fov, m_height, m_near, and m_width.

Referenced by getNumberOfShapes(), and myGLWindow::getRayDirection().

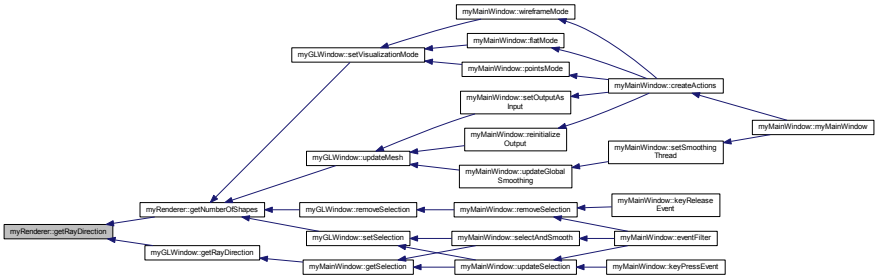
```
461 {
462     float x = (2.0f * pos.x) / m_width - 1.0f;
463     float y = 1.0f - (2.0f * pos.y) / m_height;
464     float z = 1.0f;
465     // normalised device space
466     glm::vec3 ray_nds = glm::vec3(x, y, z);
467     // clip space
468     glm::vec4 ray_clip = glm::vec4(ray_nds.x, ray_nds.y, -1.0, 1.0);
469     // eye space
470     glm::mat4 proj_mat = glm::perspective(m_fov, ((float)m_width) / ((float)(
      m_height)), m_near, m_far);
471     glm::mat4 view_mat = glm::lookAt(camera.getPosition(),
      camera.getPosition() + camera.getViewDirection(),
      camera.getUP());
```

```
472
473     glm::vec4 ray_eye = inverse(proj_mat) * ray_clip;
474     ray_eye = glm::vec4(ray_eye.x, ray_eye.y, -1.0, 0.0);
475     // world space
476     glm::vec3 ray_wor = glm::vec3(inverse(view_mat) * ray_eye);
477     // don't forget to normalise the vector at some point
478     ray_wor = glm::normalize(ray_wor);
479     return ray_wor;
480 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.24 getSceneCentralPoint()

```
glm::vec3 myRenderer::getSceneCentralPoint ( )  [inline]
```

Definition at line 195 of file myRenderer.h.

References sceneCentralPoint.

```
195 {return sceneCentralPoint;}
```

6.6.3.25 getVertexArrayObjectIDAt()

```
GLuint myRenderer::getVertexArrayObjectIDAt (
            size_t pos )  [inline]
```

getVertexArrayObjectIDAt - get vertex array object ID (OpenGL) for a shape

Parameters

pos	index of the target shape

Returns

vertex array object ID (OpenGL)

Definition at line 173 of file myRenderer.h.

References m_vertexArrayObjectIDs.

```
173 { return m_vertexArrayObjectIDs[pos]; }
```

6.6.3.26 getWidth()

```
GLuint myRenderer::getWidth ( )  [inline]
```

getters

Definition at line 189 of file myRenderer.h.

References m_width.

```
189 {return m_width;}
```

6.6.3.27 initialize()

```
void myRenderer::initialize ( )
```

initialize - renderer initialization

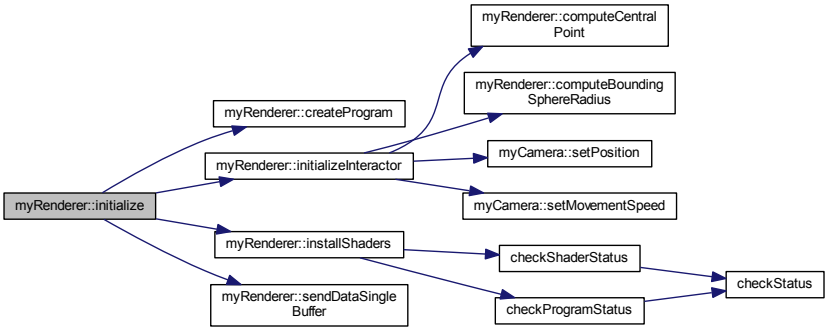
Definition at line 219 of file myRenderer.cpp.

References createProgram(), initializeInteractor(), installShaders(), and sendDataSingleBuffer().

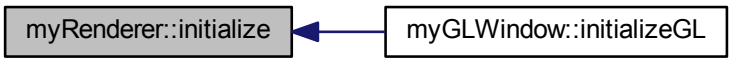
Referenced by myGLWindow::initializeGL().

```
220 {
221     glewInit();
222     glEnable(GL_DEPTH_TEST);
223     glEnable(GL_VERTEX_PROGRAM_POINT_SIZE);
224     createProgram();
225     sendDataSingleBuffer();
226     installShaders();
227     initializeInteractor();
228 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.28 initializeInteractor()

```
void myRenderer::initializeInteractor ( )
```

initializeInteractor - default initialization of renderer parameters based on shape data

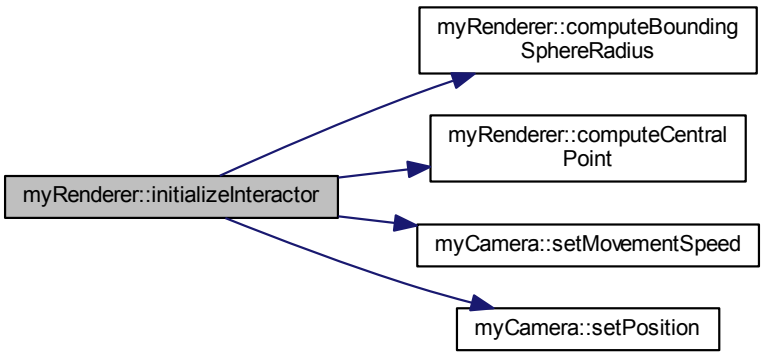
Definition at line 230 of file myRenderer.cpp.

References boundingSphereRadius, camera, computeBoundingSphereRadius(), computeCentralPoint(), sceneCentralPoint, myCamera::setMovementSpeed(), and myCamera::setPosition().

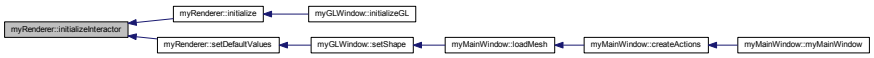
Referenced by initialize(), and setDefaultValues().

```
231 {
232     computeCentralPoint();
233     computeBoundingSphereRadius();
234     camera.setPosition(sceneCentralPoint+glm::vec3(0.0f,0.0f,2.0f*
      boundingSphereRadius));
235     camera.setMovementSpeed(boundingSphereRadius/50.0f);
236 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.29 installShaders()

```
bool myRenderer::installShaders ( )
```

installShaders - compile and attach shaders to the main program (m_programID) in the current context

Returns

success

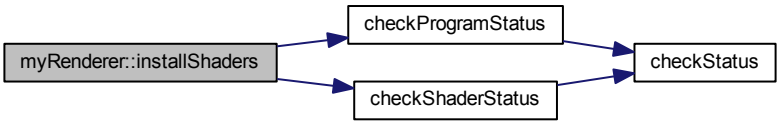
Definition at line 150 of file myRenderer.cpp.

References checkProgramStatus(), checkShaderStatus(), m_programID, and m_shaders.

Referenced by initialize(), and myGLWindow::installShaders().

```
151 {
152     for (size_t i = 0; i < m_shaders.size(); i++)
153     {
154         m_shaders[i]->createShader();
155         m_shaders[i]->compileShader();
156         if (!checkShaderStatus(m_shaders[i]->getShaderID())) return false;
157         glAttachShader(m_programID, m_shaders[i]->getShaderID());
158     }
159     glLinkProgram(m_programID);
160     if (!checkProgramStatus(m_programID)) return false;
161     glUseProgram(m_programID);
162     return true;
163 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.30 removeShape()

```
void myRenderer::removeShape (
            size_t index )
```

removeShape - remove a shape

Parameters

index	index of the target shape

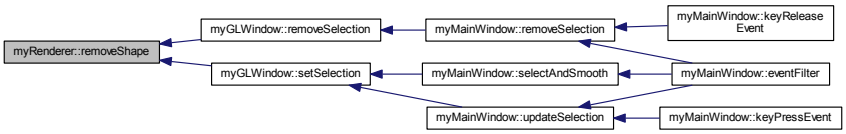
Definition at line 76 of file myRenderer.cpp.

References m_draw_modes, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, and m_vertexOffsets.

Referenced by myGLWindow::removeSelection(), and myGLWindow::setSelection().

```
77 {
78     glDeleteVertexArrays(1, &m_vertexArrayObjectIDs[index]);
79     m_shapes.erase(m_shapes.begin() + index);
80     m_vertexOffsets.erase(m_vertexOffsets.begin() + index);
81     m_elementOffsets.erase(m_elementOffsets.begin() + index);
82     m_vertexArrayObjectIDs.erase(m_vertexArrayObjectIDs.begin()
       + index);
83     m_draw_modes.erase(m_draw_modes.begin() + index);
84 }
```

Here is the caller graph for this function:



6.6.3.31 resendDataSingleBuffer()

```
void myRenderer::resendDataSingleBuffer ( )
```

resendDataSingleBuffer - resend shape data to the GPU using OpenGL

Definition at line 251 of file myRenderer.cpp.

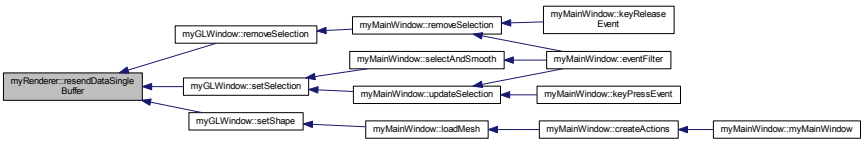
References m_elementBufferID, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, m_vertexBufferID, and m_vertexOffsets.

Referenced by myGLWindow::removeSelection(), myGLWindow::setSelection(), and myGLWindow::setShape().

```
252 {
253     const uint NUM_FLOATS_PER_VERTICE = 9;
254     const uint VERTEX_BYTE_SIZE = NUM_FLOATS_PER_VERTICE * sizeof(float);
255 
256     GLsizeiptr totalVertexBufferSize = 0;
257     GLsizeiptr totalElementBufferSize = 0;
258     for (size_t i = 0; i < m_shapes.size(); i++)
259     {
260         totalVertexBufferSize += m_shapes[i]->vertexBufferSize();
261         totalElementBufferSize += m_shapes[i]->indexBufferSize();
262     }
263 
264     //glGenBuffers(1, &m_vertexBufferID);
265     glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferID);
266     glBufferData(GL_ARRAY_BUFFER, totalVertexBufferSize, 0, GL_DYNAMIC_DRAW);
267     GLsizeiptr currentVertexOffset = 0;
268     for (size_t i = 0; i < m_shapes.size(); i++)
269     {
270         m_vertexOffsets[i] = currentVertexOffset;
271         glBufferSubData(GL_ARRAY_BUFFER, currentVertexOffset, m_shapes[i]->vertexBufferSize(),
    m_shapes[i]->vertices);
272         currentVertexOffset += m_shapes[i]->vertexBufferSize();
273     }
274 
275     //glGenBuffers(1, &m_elementBufferID);
276     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_elementBufferID);
277     glBufferData(GL_ELEMENT_ARRAY_BUFFER, totalElementBufferSize, 0, GL_STATIC_DRAW);
278     GLsizeiptr currentElementOffset = 0;
279     for (size_t i = 0; i < m_shapes.size(); i++)
280     {
281         m_elementOffsets[i] = currentElementOffset;
282         glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, currentElementOffset, m_shapes[i]->indexBufferSize
    (), m_shapes[i]->indices);
283         currentElementOffset += m_shapes[i]->indexBufferSize();
284     }
285     //cout << "vertex buffer : "<< m_vertexBufferID << endl;
286     for (size_t i = 0; i < m_shapes.size(); i++)
287     {
288         glGenVertexArrays(1, &m_vertexArrayObjectIDs[i]);
289 
290         glBindVertexArray(m_vertexArrayObjectIDs[i]);
291 
292         //cout << m_vertexArrayObjectIDs[i] << endl;
293         glEnableVertexAttribArray(0);
```

```
294         glEnableVertexAttribArray(1);
295         glEnableVertexAttribArray(2);
296         glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferID);
297         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
    m_vertexOffsets[i]));
298         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
    m_vertexOffsets[i] + sizeof(float)* 3));
299         glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
    m_vertexOffsets[i] + sizeof(float)* 6));
300         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_elementBufferID);
301     }
302 }
```

Here is the caller graph for this function:



6.6.3.32 rotateObjects()

```
void myRenderer::rotateObjects (
            const glm::vec2 & newMousePosition )
```

rotateObjects - rotate all shapes (model to world matrix manipulation) regarding a mouse movement based interaction

Parameters

newMousePosition	current mouse position

Definition at line 408 of file myRenderer.cpp.

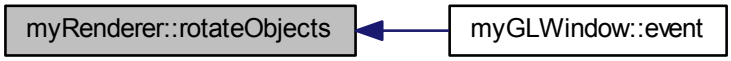
References modelToWorldMatrix, oldMousePosition, and sceneCentralPoint.

Referenced by myGLWindow::event().

```
409 {
410     glm::vec2 mouseDelta = newMousePosition - oldMousePosition;
411     if (glm::length(mouseDelta) > 20.0f || glm::length(mouseDelta) < 1.0f)
412     {
413         oldMousePosition = newMousePosition;
414         return;
415     }
416     const float ROTATIONAL_SPEED = 0.1f;
417
418     float valx = mouseDelta.x * ROTATIONAL_SPEED;
419     float valy = mouseDelta.y * ROTATIONAL_SPEED;
420
421     glm::mat4 transform = glm::translate(glm::vec3(-sceneCentralPoint.x, -
    sceneCentralPoint.y, -sceneCentralPoint.z));
422     glm::mat4 rotator = glm::rotate(valx, glm::vec3(0.0f,1.0f,0.0f)) * glm::rotate(valy, glm::vec3(1.0f,0.0
    f,0.0f));
423     transform = rotator*transform;
424     glm::mat4 temp = translate(glm::vec3(sceneCentralPoint.x,
    sceneCentralPoint.y, sceneCentralPoint.z));
```

```
425     transform = temp * transform;
426
427     modelToWorldMatrix = transform * modelToWorldMatrix;
428
429     oldMousePosition = newMousePosition;
430     return;
431 }
```

Here is the caller graph for this function:



6.6.3.33 sendDataSingleBuffer()

```
void myRenderer::sendDataSingleBuffer ( )
```

sendDataSingleBuffer - send shape data to the GPU using OpenGL

Definition at line 165 of file myRenderer.cpp.

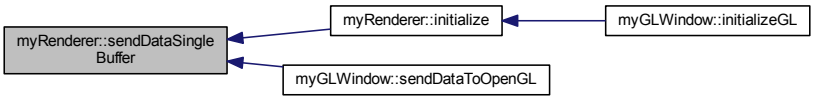
References m_elementBufferID, m_elementOffsets, m_shapes, m_vertexArrayObjectIDs, m_vertexBufferID, and m_vertexOffsets.

Referenced by initialize(), and myGLWindow::sendDataToOpenGL().

```
166 {
167     const uint NUM_FLOATS_PER_VERTICE = 9;
168     const uint VERTEX_BYTE_SIZE = NUM_FLOATS_PER_VERTICE * sizeof(float);
169
170     GLsizeiptr totalVertexBufferSize = 0;
171     GLsizeiptr totalElementBufferSize = 0;
172     for (size_t i = 0; i < m_shapes.size(); i++)
173     {
174         totalVertexBufferSize += m_shapes[i]->vertexBufferSize();
175         totalElementBufferSize += m_shapes[i]->indexBufferSize();
176     }
177
178     glGenBuffers(1, &m_vertexBufferID);
179     glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferID);
180     glBufferData(GL_ARRAY_BUFFER, totalVertexBufferSize, 0, GL_DYNAMIC_DRAW);
181     GLsizeiptr currentVertexOffset = 0;
182     for (size_t i = 0; i < m_shapes.size(); i++)
183     {
184         m_vertexOffsets[i] = currentVertexOffset;
185         glBufferSubData(GL_ARRAY_BUFFER, currentVertexOffset, m_shapes[i]->vertexBufferSize(),
    m_shapes[i]->vertices);
186         currentVertexOffset += m_shapes[i]->vertexBufferSize();
187     }
188
189     glGenBuffers(1, &m_elementBufferID);
190     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_elementBufferID);
191     glBufferData(GL_ELEMENT_ARRAY_BUFFER, totalElementBufferSize, 0, GL_STATIC_DRAW);
192     GLsizeiptr currentElementOffset = 0;
193     for (size_t i = 0; i < m_shapes.size(); i++)
194     {
195         m_elementOffsets[i] = currentElementOffset;
196         glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, currentElementOffset, m_shapes[i]->indexBufferSize
```

```
      (), m_shapes[i]->indices);
197         currentElementOffset += m_shapes[i]->indexBufferSize();
198     }
199
200     for (size_t i = 0; i < m_shapes.size(); i++)
201     {
202         //glDeleteVertexArrays(1, &m_vertexArrayObjectIDs[i]);
203         glGenVertexArrays(1, &m_vertexArrayObjectIDs[i]);
204
205         glBindVertexArray(m_vertexArrayObjectIDs[i]);
206         glEnableVertexAttribArray(0);
207         glEnableVertexAttribArray(1);
208         glEnableVertexAttribArray(2);
209         glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferID);
210         glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
      m_vertexOffsets[i]));
211         glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
      m_vertexOffsets[i] + sizeof(float)* 3));
212         glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, VERTEX_BYTE_SIZE, (void*)(
      m_vertexOffsets[i] + sizeof(float)* 6));
213         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_elementBufferID);
214     }
215     //glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_elementBufferID);
216
217 }
```

Here is the caller graph for this function:



6.6.3.34 setBoundingSphereRadius()

```
void myRenderer::setBoundingSphereRadius (
            float _radius )  [inline]
```

Definition at line 209 of file myRenderer.h.

References boundingSphereRadius.

```
209 {boundingSphereRadius = _radius;}
```

6.6.3.35 setDefaultValues()

```
void myRenderer::setDefaultValues ( )
```

setDefaultValues - set custom renderer parameters

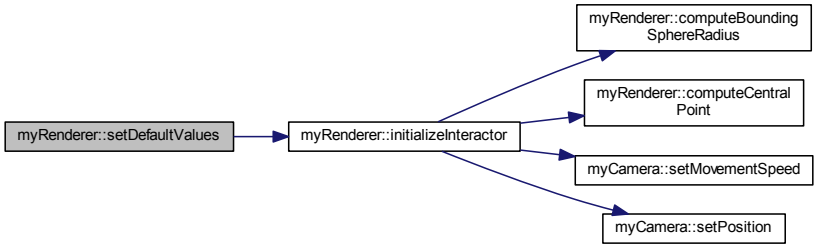
Definition at line 238 of file myRenderer.cpp.

References initializeInteractor(), m_far, m_fov, m_height, m_near, m_width, modelToWorldMatrix, and sceneCentralPoint.

Referenced by myGLWindow::setShape().

```
239 {
240     //currentDrawFlag = e_draw_faces;
241     sceneCentralPoint = glm::vec3(0.0f, 0.0f, 0.0f);
242     m_width = 1000;
243     m_height = 800;
244     m_fov = 45.0f;
245     m_near = 0.01f;
246     m_far = 1000.0f;
247     modelToWorldMatrix = glm::translate(glm::vec3(0.0f, 0.0f, 0.0f));
248     initializeInteractor();
249 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.36 setFar()

```
void myRenderer::setFar (
            GLfloat _far )  [inline]
```

Definition at line 206 of file myRenderer.h.

References m_far.

```
206 {m_far = _far;}
```

6.6.3.37 setFOV()

```
void myRenderer::setFOV (
            GLfloat _fov )  [inline]
```

Definition at line 204 of file myRenderer.h.

References m_fov.

```
204 {m_fov = _fov;}
```

6.6.3.38 setHeight()

```
void myRenderer::setHeight (
            GLuint _height )  [inline]
```

Definition at line 203 of file myRenderer.h.

References m_height.

Referenced by myGLWindow::paintGL().

```
203 {m_height = _height;}
```

Here is the caller graph for this function:

6.6.3.39 setModelToWorldMatrix()

```
void myRenderer::setModelToWorldMatrix (
            glm::mat4 & _modelToWorldMatrix )  [inline]
```

Definition at line 207 of file myRenderer.h.

References modelToWorldMatrix.

```
207 {modelToWorldMatrix = _modelToWorldMatrix;}
```

6.6.3.40 setNear()

```
void myRenderer::setNear (
            GLfloat _near )  [inline]
```

Definition at line 205 of file myRenderer.h.

References m_near.

```
205 {m_near = _near;}
```

6.6.3.41 setSceneCentralPoint()

```
void myRenderer::setSceneCentralPoint (
            glm::vec3 & _point )  [inline]
```

Definition at line 208 of file myRenderer.h.

References sceneCentralPoint.

```
208 {sceneCentralPoint = _point;}
```

6.6.3.42 setShapeDrawMode()

```
void myRenderer::setShapeDrawMode (
            size_t _shape_index,
            myDrawFlags _mode )
```

setShapeDrawMode - set rendering mode for a specific shape

Parameters

_shape_index	index of the target shape
_mode	rendering mode for the target shape

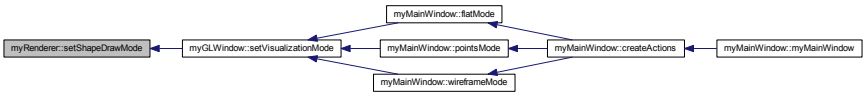
Definition at line 98 of file myRenderer.cpp.

References m_draw_modes.

Referenced by myGLWindow::setVisualizationMode().

```
99 {
100     m_draw_modes[_shape_index] = _mode;
101 }
```

Here is the caller graph for this function:



6.6.3.43 setWidth()

```
void myRenderer::setWidth (
            GLuint _width )  [inline]
```

setters

Definition at line 202 of file myRenderer.h.

References m_width.

Referenced by myGLWindow::paintGL().

```
202 {m_width = _width;}
```

Here is the caller graph for this function:

6.6.3.44 translateCamera()

```
void myRenderer::translateCamera (
            const glm::vec2 & newMousePosition )
```

translateCamera - translate all shapes (model to world matrix manipulation) regarding a mouse movement interaction

Parameters

newMousePosition	current mouse position

Definition at line 433 of file myRenderer.cpp.

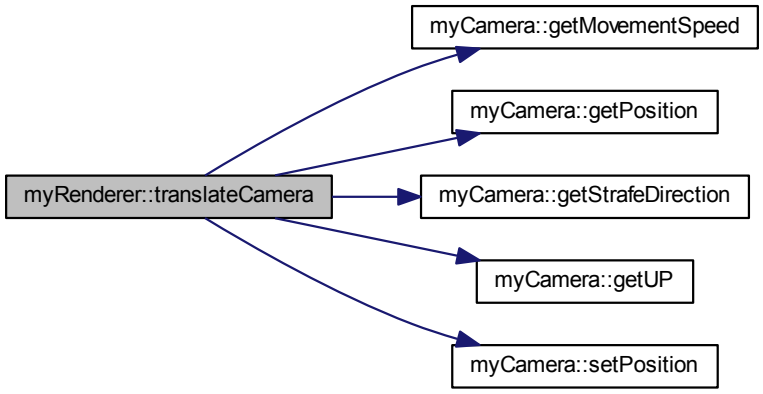
References camera, myCamera::getMovementSpeed(), myCamera::getPosition(), myCamera::getStrafeDirection(), myCamera::getUP(), oldMousePosition, and myCamera::setPosition().

Referenced by myGLWindow::event().

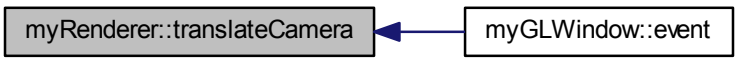
```
434 {
435     glm::vec2 mouseDelta = newMousePosition - oldMousePosition;
436     if (glm::length(mouseDelta) > 20.0f)
437     {
```

```
438         oldMousePosition = newMousePosition;
439         return;
440     }
441     float TRANSLATION_SPEED = camera.getMovementSpeed();
442
443     float valx = mouseDelta.x * TRANSLATION_SPEED;
444     float valy = mouseDelta.y * TRANSLATION_SPEED;
445
446     camera.setPosition(camera.getPosition()-
      camera.getUP()*valy+camera.getStrafeDirection()*valx);
447
448     oldMousePosition = newMousePosition;
449     return;
450 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.45 updateVertexBuffer()

```
void myRenderer::updateVertexBuffer (
            size_t index )
```

updateVertexBuffer - update and send vertex data for a single shape

Parameters

index	target shape index

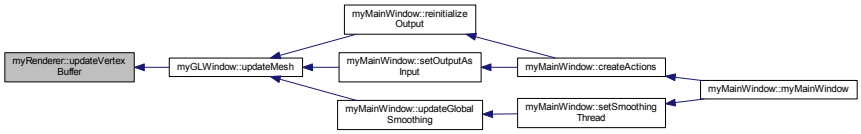
Definition at line 370 of file myRenderer.cpp.

References m_shapes, m_vertexBufferID, and m_vertexOffsets.

Referenced by myGLWindow::updateMesh().

```
371 {
372     glBindBuffer(GL_ARRAY_BUFFER, m_vertexBufferID);
373     glBufferSubData(GL_ARRAY_BUFFER, m_vertexOffsets[index],
      m_shapes[index]->vertexBufferSize(), m_shapes[index]->vertices);
374 }
```

Here is the caller graph for this function:



6.6.3.46 zoom()

```
void myRenderer::zoom (
            float delta )
```

zoom - translate camera position (forward/backward) regarding a step size

Parameters

delta	- step size

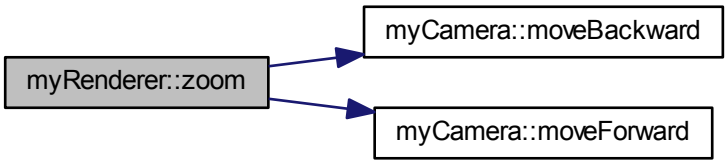
Definition at line 452 of file myRenderer.cpp.

References camera, myCamera::moveBackward(), and myCamera::moveForward().

Referenced by myGLWindow::event().

```
453 {
454     if (delta >= 0.0f)
455         camera.moveForward();
456     else
457         camera.moveBackward();
458 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.4 Field Documentation

6.6.4.1 boundingSphereRadius

```
float myRenderer::boundingSphereRadius  [private]
```

boundingSphereRadius - bounding sphere radius (regarding the central point)

Definition at line 268 of file myRenderer.h.

Referenced by computeBoundingSphereRadius(), getBoundingSphereRadius(), initializeInteractor(), and setBoundingSphereRadius().

6.6.4.2 camera

```
myCamera myRenderer::camera  [private]
```

camera - main camera for renderer

Definition at line 239 of file myRenderer.h.

Referenced by draw(), getCamera(), getRayDirection(), initializeInteractor(), translateCamera(), and zoom().

6.6.4.3 light

```
glm::vec4 myRenderer::light  [private]
```

light - main light "color"

Definition at line 243 of file myRenderer.h.

Referenced by draw(), and myRenderer().

6.6.4.4 lightPosition

```
glm::vec3 myRenderer::lightPosition  [private]
```

lightPosition - main light position

Definition at line 247 of file myRenderer.h.

Referenced by draw(), and myRenderer().

6.6.4.5 m_draw_modes

```
vector<myDrawFlags> myRenderer::m_draw_modes  [private]
```

m_draw_modes - vector containing all shape rendering modes

Definition at line 308 of file myRenderer.h.

Referenced by addShape(), clearAndDeleteShapes(), clearShapes(), draw(), removeShape(), and setShapeDraw↩Mode().

6.6.4.6 m_elementBufferID

```
GLuint myRenderer::m_elementBufferID  [private]
```

m_elementBufferID - main element buffer ID

Definition at line 284 of file myRenderer.h.

Referenced by resendDataSingleBuffer(), sendDataSingleBuffer(), and ∼myRenderer().

6.6.4.7 m_elementOffsets

```
vector<GLsizeiptr> myRenderer::m_elementOffsets  [private]
```

m_elementOffsets - vector containing all shape indices offsets

Definition at line 300 of file myRenderer.h.

Referenced by addShape(), clearAndDeleteShapes(), clearShapes(), draw(), getIndexOffsetAt(), removeShape(), resendDataSingleBuffer(), and sendDataSingleBuffer().

6.6.4.8 m_far

```
GLfloat myRenderer::m_far  [private]
```

m_far - far plane distance

Definition at line 235 of file myRenderer.h.

Referenced by draw(), getFar(), getRayDirection(), myRenderer(), setDefaultValues(), and setFar().

6.6.4.9 m_fov

```
GLfloat myRenderer::m_fov  [private]
```

m_fov - field of view (FOV)

Definition at line 227 of file myRenderer.h.

Referenced by draw(), getFOV(), getRayDirection(), myRenderer(), setDefaultValues(), and setFOV().

6.6.4.10 m_height

```
GLuint myRenderer::m_height  [private]
```

m_height - window height

Definition at line 223 of file myRenderer.h.

Referenced by draw(), getHeight(), getRayDirection(), myRenderer(), setDefaultValues(), and setHeight().

6.6.4.11 m_near

```
GLfloat myRenderer::m_near  [private]
```

m_near - near plane distance

Definition at line 231 of file myRenderer.h.

Referenced by draw(), getNear(), getRayDirection(), myRenderer(), setDefaultValues(), and setNear().

6.6.4.12 m_programID

```
GLuint myRenderer::m_programID  [private]
```

Data.

m_programID - main program ID

Definition at line 276 of file myRenderer.h.

Referenced by clearAndDeleteShaders(), createProgram(), draw(), getProgramID(), installShaders(), and ~myRenderer().

6.6.4.13 m_shaders

```
vector<myShader*> myRenderer::m_shaders  [private]
```

m_shaders - vector containing all shaders

Definition at line 288 of file myRenderer.h.

Referenced by addShader(), clearAndDeleteShaders(), clearShaders(), and installShaders().

6.6.4.14 m_shapes

```
vector<ShapeData*> myRenderer::m_shapes  [private]
```

m_shapes - vector containing all shapes

Definition at line 292 of file myRenderer.h.

Referenced by addShape(), clearAndDeleteShapes(), clearShapes(), computeBoundingSphereRadius(), computeCentralPoint(), draw(), getNumberOfShapes(), removeShape(), resendDataSingleBuffer(), sendDataSingleBuffer(), and updateVertexBuffer().

6.6.4.15 m_vertexArrayObjectIDs

```
vector<GLuint> myRenderer::m_vertexArrayObjectIDs  [private]
```

m_vertexArrayObjectIDs - vector containing all vertex array objects IDs

Definition at line 304 of file myRenderer.h.

Referenced by addShape(), clearAndDeleteShapes(), clearShapes(), draw(), getVertexArrayObjectIDAt(), removeShape(), resendDataSingleBuffer(), and sendDataSingleBuffer().

6.6.4.16 m_vertexBufferID

```
GLuint myRenderer::m_vertexBufferID  [private]
```

m_vertexBufferID - main vertex buffer ID

Definition at line 280 of file myRenderer.h.

Referenced by resendDataSingleBuffer(), sendDataSingleBuffer(), updateVertexBuffer(), and ~myRenderer().

6.6.4.17 m_vertexOffsets

```
vector<GLsizeiptr> myRenderer::m_vertexOffsets  [private]
```

m_vertexOffsets - vector containing all shape vertices offsets

Definition at line 296 of file myRenderer.h.

Referenced by addShape(), clearAndDeleteShapes(), clearShapes(), removeShape(), resendDataSingleBuffer(), sendDataSingleBuffer(), and updateVertexBuffer().

6.6.4.18 m_width

```
GLuint myRenderer::m_width  [private]
```

Rendering.

m_width - window width

Definition at line 219 of file myRenderer.h.

Referenced by draw(), getRayDirection(), getWidth(), myRenderer(), setDefaultValues(), and setWidth().

6.6.4.19 modelToWorldMatrix

```
glm::mat4 myRenderer::modelToWorldMatrix  [private]
```

Interaction.

modelToWorldMatrix - 4x4 model to world matrix (for all shapes)

Definition at line 255 of file myRenderer.h.

Referenced by draw(), getModelToWorldMatrix(), myRenderer(), rotateObjects(), setDefaultValues(), and setModelToWorldMatrix().

6.6.4.20 oldMousePosition

```
glm::vec2 myRenderer::oldMousePosition  [private]
```

oldMousePosition - old mouse position for mouse move interactions

Definition at line 259 of file myRenderer.h.

Referenced by rotateObjects(), and translateCamera().

6.6.4.21 sceneCentralPoint

```
glm::vec3 myRenderer::sceneCentralPoint  [private]
```

sceneCentralPoint - scene central point

Definition at line 264 of file myRenderer.h.

Referenced by computeBoundingSphereRadius(), computeCentralPoint(), getSceneCentralPoint(), initializeInteractor(), myRenderer(), rotateObjects(), setDefaultValues(), and setSceneCentralPoint().

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

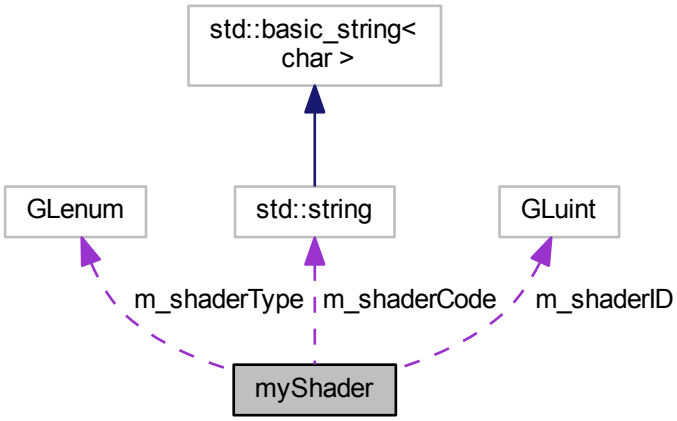
- visualization/myRenderer.h
- visualization/myRenderer.cpp

6.7 myShader Class Reference

The myShader class - shader data for OpenGL.

`#include <myShader.h>`

Collaboration diagram for myShader:



Public Member Functions

- myShader ()
  myShader - default constructor
- myShader (GLenum _shaderType)
  myShader - create empty shader data of a given shader type
- myShader (GLenum _shaderType, const string &_code)
  myShader - create shader data given a shader type and a shader code
- virtual ~myShader ()
  ~myShader - destructor
- bool readShaderCode (const char *_fileName)
  *readShaderCode - read shader code from an input file (file name as const char *)*
- bool readShaderCode (const string &_fileName)
  readShaderCode - read shader code from an input file (file name as string)
- void createShader ()
  createShader - create shader in the current OpenGL context
- void compileShader ()
  compileShader - compile shader in the current OpenGL context
- string getShaderCode ()
  getters
- GLuint getShaderID ()
- GLenum getShaderType ()
- void setShaderCode (string _m_shaderCode)
  setters
- void setShaderID (GLuint _m_shaderID)
- void setShaderType (GLenum _m_shaderType)

Private Attributes

- string m_shaderCode

  m_shaderCode shader code
- GLuint m_shaderID

  m_shaderID shader ID in the current OpenGL context
- GLenum m_shaderType

  m_shaderType OpenGL shader type

6.7.1 Detailed Description

The myShader class - shader data for OpenGL.

Definition at line 20 of file myShader.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 myShader() [1/3]

```
myShader::myShader ( )
```

myShader - default constructor

Definition at line 3 of file myShader.cpp.

References m_shaderCode, m_shaderID, and m_shaderType.

```
4 {
5     m_shaderType = GL_VERTEX_SHADER;
6     m_shaderCode = "";
7     m_shaderID = 0;
8 }
```

6.7.2.2 myShader() [2/3]

```
myShader::myShader (
            GLenum _shaderType )
```

myShader - create empty shader data of a given shader type

Parameters

_shaderType	shader type

Definition at line 10 of file myShader.cpp.

References m_shaderCode, m_shaderID, and m_shaderType.

```
11 {
12     m_shaderType = _shaderType;
13     m_shaderCode = "";
14     m_shaderID = 0;
15 }
```

6.7.2.3 myShader() [3/3]

```
myShader::myShader (
            GLenum _shaderType,
            const string & _code )
```

myShader - create shader data given a shader type and a shader code

Parameters

_shaderType	
_code	

Definition at line 17 of file myShader.cpp.

References m_shaderCode, m_shaderID, and m_shaderType.

```
18 {
19     m_shaderType = _shaderType;
20     m_shaderCode = _code;
21     m_shaderID = 0;
22 }
```

6.7.2.4 ~myShader()

```
myShader::~myShader ( )  [virtual]
```

~myShader - destructor

Definition at line 24 of file myShader.cpp.

References m_shaderID.

```
25 {
26     glDeleteShader(m_shaderID);
27 }
```

6.7.3 Member Function Documentation

6.7.3.1 compileShader()

```
void myShader::compileShader ( )
```

compileShader - compile shader in the current OpenGL context

Definition at line 51 of file myShader.cpp.

References m_shaderCode, and m_shaderID.

```
52 {
53     const GLchar* adapter[1];
54     adapter[0] = m_shaderCode.c_str();
55     glShaderSource(m_shaderID, 1, adapter, 0);
56     glCompileShader(m_shaderID);
57 }
```

6.7.3.2 createShader()

```
void myShader::createShader ( )
```

createShader - create shader in the current OpenGL context

Definition at line 46 of file myShader.cpp.

References m_shaderID, and m_shaderType.

```
47 {
48     m_shaderID = glCreateShader(m_shaderType);
49 }
```

6.7.3.3 getShaderCode()

```
string myShader::getShaderCode ( )  [inline]
```

getters

Definition at line 66 of file myShader.h.

```
66 { return m_shaderCode; }
```

6.7.3.4 getShaderID()

```
GLuint myShader::getShaderID ( )  [inline]
```

Definition at line 67 of file myShader.h.

```
67 { return m_shaderID; }
```

6.7.3.5 getShaderType()

```
GLenum myShader::getShaderType ( )  [inline]
```

Definition at line 68 of file myShader.h.

```
68 { return m_shaderType; }
```

6.7.3.6 readShaderCode() [1/2]

```
bool myShader::readShaderCode (
            const char * _fileName )
```

readShaderCode - read shader code from an input file (file name as const char *)

Parameters

_fileName	file name

Returns

true if file can be read

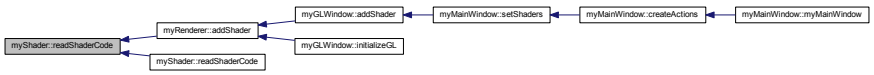
Definition at line 29 of file myShader.cpp.

References m_shaderCode.

Referenced by myRenderer::addShader(), and readShaderCode().

```
30 {
31     ifstream ifile(fileName);
32     if (!ifile.good())
33     {
34         cout << "Cant read file " << fileName << endl;
35         return false;
36     }
37     m_shaderCode = string(istreambuf_iterator<char>(ifile), istreambuf_iterator<char>());
38     return true;
39 }
```

Here is the caller graph for this function:



6.7.3.7 readShaderCode() [2/2]

```
bool myShader::readShaderCode (
            const string & _fileName )
```

readShaderCode - read shader code from an input file (file name as string)

Parameters

_fileName	file name

Returns

true if file can be read

Definition at line 41 of file myShader.cpp.

References readShaderCode().

```
42 {
43     return readShaderCode(_fileName.c_str());
44 }
```

Here is the call graph for this function:

6.7.3.8 setShaderCode()

```
void myShader::setShaderCode (
            string _m_shaderCode )  [inline]
```

setters

Definition at line 73 of file myShader.h.

```
73 { m_shaderCode = _m_shaderCode; }
```

6.7.3.9 setShaderID()

```
void myShader::setShaderID (
            GLuint _m_shaderID )  [inline]
```

Definition at line 74 of file myShader.h.

```
74 { m_shaderID = _m_shaderID; }
```

6.7.3.10 setShaderType()

```
void myShader::setShaderType (
            GLenum _m_shaderType )  [inline]
```

Definition at line 75 of file myShader.h.

```
75 { m_shaderType = _m_shaderType; }
```

6.7.4 Field Documentation

6.7.4.1 m_shaderCode

```
string myShader::m_shaderCode  [private]
```

m_shaderCode shader code

Definition at line 81 of file myShader.h.

Referenced by compileShader(), myShader(), and readShaderCode().

6.7.4.2 m_shaderID

```
GLuint myShader::m_shaderID  [private]
```

m_shaderID shader ID in the current OpenGL context

Definition at line 85 of file myShader.h.

Referenced by compileShader(), createShader(), myShader(), and ~myShader().

6.7.4.3 m_shaderType

```
GLenum myShader::m_shaderType  [private]
```

m_shaderType OpenGL shader type

Definition at line 89 of file myShader.h.

Referenced by createShader(), and myShader().

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

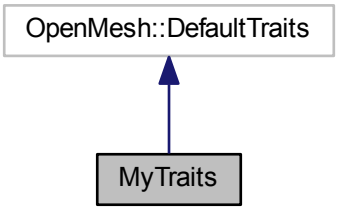
- visualization/myShader.h
- visualization/myShader.cpp

6.8 MyTraits Struct Reference

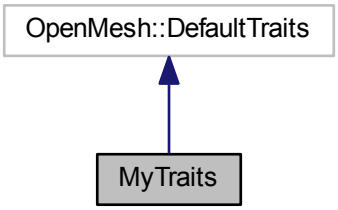
The MyTraits struct - OpenMesh custom traits.

```
#include <mesh.h>
```

Inheritance diagram for MyTraits:



Collaboration diagram for MyTraits:



Public Types

- typedef OpenMesh::Vec3f Point
- typedef OpenMesh::Vec3f Normal
- typedef double TexCoord1D
- typedef OpenMesh::Vec2f TexCoord2D
- typedef OpenMesh::Vec3f TexCoord3D

Public Member Functions

- VertexAttributes (OpenMesh::Attributes::Status|OpenMesh::Attributes::Normal|OpenMesh::Attributes::Color)
- HalfedgeAttributes (OpenMesh::Attributes::Status|OpenMesh::Attributes::PrevHalfedge)
- FaceAttributes (OpenMesh::Attributes::Status|OpenMesh::Attributes::Normal|OpenMesh::Attributes::Color)
- EdgeAttributes (OpenMesh::Attributes::Status|OpenMesh::Attributes::Color)

6.8.1 Detailed Description

The MyTraits struct - OpenMesh custom traits.

Definition at line 21 of file mesh.h.

6.8.2 Member Typedef Documentation

6.8.2.1 Normal

```
typedef OpenMesh::Vec3f MyTraits::Normal
```

Definition at line 25 of file mesh.h.

6.8.2.2 Point

```
typedef OpenMesh::Vec3f MyTraits::Point
```

Definition at line 24 of file mesh.h.

6.8.2.3 TexCoord1D

```
typedef double MyTraits::TexCoord1D
```

Definition at line 28 of file mesh.h.

6.8.2.4 TexCoord2D

```
typedef OpenMesh::Vec2f MyTraits::TexCoord2D
```

Definition at line 30 of file mesh.h.

6.8.2.5 TexCoord3D

```
typedef OpenMesh::Vec3f MyTraits::TexCoord3D
```

Definition at line 32 of file mesh.h.

6.8.3 Member Function Documentation

6.8.3.1 EdgeAttributes()

```
MyTraits::EdgeAttributes (
            OpenMesh::Attributes::Status|OpenMesh::Attributes::Color  )
```

6.8.3.2 FaceAttributes()

```
MyTraits::FaceAttributes (
            OpenMesh::Attributes::Status|OpenMesh::Attributes::Normal|OpenMesh::Attributes::↩
Color  )
```

6.8.3.3 HalfedgeAttributes()

```
MyTraits::HalfedgeAttributes (
            OpenMesh::Attributes::Status|OpenMesh::Attributes::PrevHalfedge  )
```

6.8.3.4 VertexAttributes()

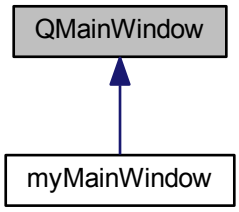
```
MyTraits::VertexAttributes (
            OpenMesh::Attributes::Status|OpenMesh::Attributes::Normal|OpenMesh::Attributes::↩
Color  )
```

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

- mesh/mesh.h

6.9 QMainWindow Class Reference

Inheritance diagram for QMainWindow:

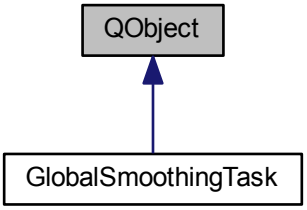


The documentation for this class was generated from the following file:

- myMainWindow.h

6.10 QObject Class Reference

Inheritance diagram for QObject:

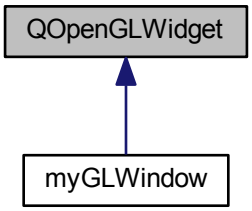


The documentation for this class was generated from the following file:

- smoothing.h

6.11 QOpenGLWidget Class Reference

Inheritance diagram for QOpenGLWidget:



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

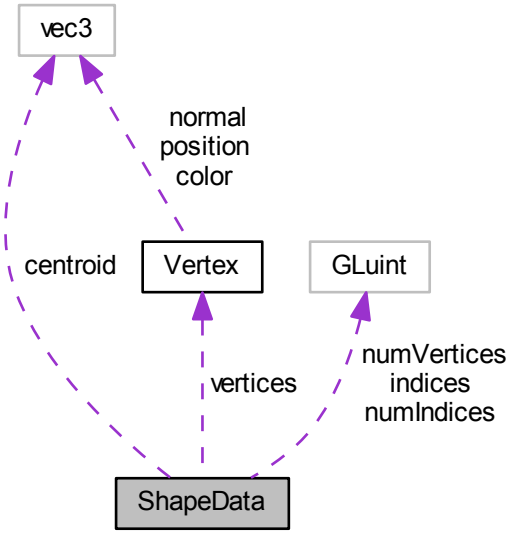
- myGLWindow.h

6.12 ShapeData Struct Reference

The ShapeData struct - triangular mesh for OpenGL buffers manipulation.

`#include <myShape.h>`

Collaboration diagram for ShapeData:



Public Member Functions

- ShapeData ()

  ShapeData - default constructor.
- ShapeData (string fileName)

  ShapeData - generates shape data using a triangular mesh file supported in OpenMesh library.
- void loadFromFile (string fileName)

  loadFromFile - load shape data using a triangular mesh file supported in OpenMesh library
- void loadMesh (TriMesh &_mesh)

  loadMesh - load shape data from a data structure defined in OpenMesh library
- void loadMeshVertexSelection (TriMesh &_mesh, vector< size_t > &selected_vertices)

  loadMeshVertexSelection - load as shape data a subset of vertices (only vertices) from a data structure defined in OpenMesh library
- GLsizeiptr vertexBufferSize () const

  vertexBufferSize - get size of the array of vertices
- GLsizeiptr indexBufferSize () const

  indexBufferSize - get size of the array of indices
- void clear ()

  clear - erase arrays and reinitialize values

Data Fields

- GLuint numVertices

  numVertices - number of vertices of the mesh
- GLuint numIndices

  numIndices - number of indices of all triangles
- Vertex * vertices

  vertices - array containing all vertices
- GLuint * indices

  indices - array containing all indices
- glm::vec3 centroid

  centroid - shape centroid

6.12.1 Detailed Description

The ShapeData struct - triangular mesh for OpenGL buffers manipulation.

Definition at line 36 of file myShape.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 ShapeData() [1/2]

```
ShapeData::ShapeData ( )  [inline]
```

ShapeData - default constructor.

Definition at line 64 of file myShape.h.

```
64 :vertices(0), numVertices(0),indices(0), numIndices(0){}
```

6.12.2.2 ShapeData() [2/2]

```
ShapeData::ShapeData (
            string fileName )  [inline]
```

ShapeData - generates shape data using a triangular mesh file supported in OpenMesh library.

Parameters

fileName	- mesh file name

Definition at line 69 of file myShape.h.

```
70     {
71         loadFromFile(fileName);
72     }
```

6.12.3 Member Function Documentation

6.12.3.1 clear()

```
void ShapeData::clear ( )  [inline]
```

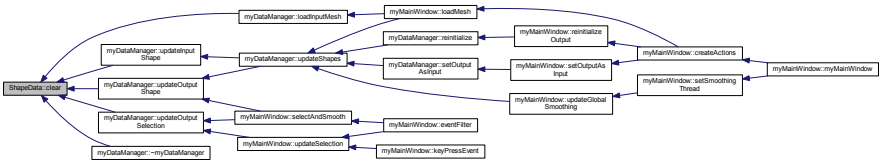
clear - erase arrays and reinitialize values

Definition at line 174 of file myShape.h.

Referenced by myDataManager::loadInputMesh(), myDataManager::updateInputShape(), myDataManager::updateOutputSelection(), myDataManager::updateOutputShape(), and myDataManager::~myDataManager().

```
175     {
176         delete[] vertices;
177         delete[] indices;
178         vertices = 0;
179         indices = 0;
180         numVertices = numIndices = 0;
181         centroid = glm::vec3(0.0f,0.0f,0.0f);
182     }
```

Here is the caller graph for this function:



6.12.3.2 indexBufferSize()

```
GLsizeiptr ShapeData::indexBufferSize ( ) const  [inline]
```

indexBufferSize - get size of the array of indices

Returns

- size of array of indices

Definition at line 167 of file myShape.h.

```
168     {
169         return numIndices * sizeof(GLuint);
170     }
```

6.12.3.3 loadFromFile()

```
void ShapeData::loadFromFile (
            string fileName )  [inline]
```

loadFromFile - load shape data using a triangular mesh file supported in OpenMesh library

Parameters

fileName	- mesh file name

Definition at line 77 of file myShape.h.

References importMesh().

```
78     {
79         TriMesh _mesh;
80         importMesh(_mesh, fileName);
81         _mesh.request_face_normals();
82         _mesh.request_vertex_normals();
83         _mesh.update_normals();
84         loadMesh(_mesh);
85     }
```

Here is the call graph for this function:

6.12.3.4 loadMesh()

```
void ShapeData::loadMesh (
            TriMesh & _mesh )  [inline]
```

loadMesh - load shape data from a data structure defined in OpenMesh library

Parameters

_mesh	- triangular mesh

Definition at line 90 of file myShape.h.

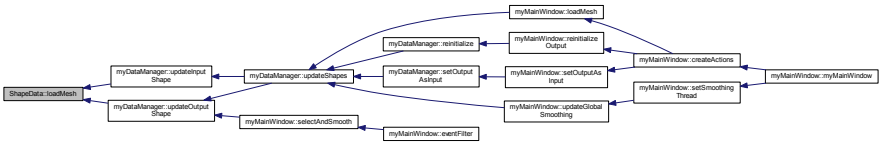
References Vertex::color, Vertex::normal, and Vertex::position.

Referenced by myDataManager::updateInputShape(), and myDataManager::updateOutputShape().

```
91     {
92         numVertices = static_cast<GLuint>(_mesh.n_vertices());
93         vertices = new Vertex[numVertices];
94         numIndices = static_cast<GLuint>(_mesh.n_faces() * 3);
95         indices = new GLuint[numIndices];
96 
97         GLuint currentIndex = 0;
98         TriMesh::Point c(0, 0, 0);
99         float n = 0.0f;
100         for (TriMesh::VertexIter v_it = _mesh.vertices_begin(); v_it != _mesh.vertices_end(); v_it++)
101         {
102             TriMesh::Color current_color = _mesh.color(*v_it);
103             TriMesh::Point current_point = _mesh.point(*v_it);
104             TriMesh::Normal current_normal = _mesh.normal(*v_it);
105             c += current_point;
106             n++;
107             vertices[currentIndex].position = glm::vec3(current_point[0], current_point[1],
      current_point[2]);
108             vertices[currentIndex].color = glm::vec3(0.5f, 0.5f, 0.5f);
109             vertices[currentIndex].normal = glm::vec3(current_normal[0], current_normal[1],
     current_normal[2]);
110             currentIndex++;
111         }
112         c = c / n;
113         centroid[0] = c[0];
114         centroid[1] = c[1];
115         centroid[2] = c[2];
116         currentIndex = 0;
117         for (TriMesh::FaceIter f_it = _mesh.faces_begin(); f_it != _mesh.faces_end(); f_it++)
118         {
119             for (TriMesh::FaceVertexIter fv_it = _mesh.fv_iter(*f_it); fv_it.is_valid(); fv_it++)
120             {
121                 indices[currentIndex] = static_cast<GLuint>(fv_it->idx());
```

```
122                 currentIndex++;
123             }
124         }
125     }
```

Here is the caller graph for this function:



6.12.3.5 loadMeshVertexSelection()

```
void ShapeData::loadMeshVertexSelection (
            TriMesh & _mesh,
            vector< size_t > & selected_vertices )  [inline]
```

loadMeshVertexSelection - load as shape data a subset of vertices (only vertices) from a data structure defined in OpenMesh library

Parameters

_mesh	- triangular mesh
selected_vertices	- subset of vertices ids of _mesh

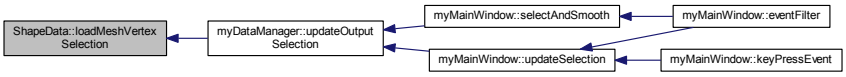
Definition at line 131 of file myShape.h.

References Vertex::color, Vertex::normal, and Vertex::position.

Referenced by myDataManager::updateOutputSelection().

```
132     {
133         numVertices = static_cast<GLuint>(_mesh.n_vertices());
134         vertices = new Vertex[numVertices];
135         numIndices = 0;
136         indices = new GLuint[numIndices];
137         TriMesh::Point c(0, 0, 0);
138         float n = 0.0f;
139         for (size_t i = 0; i < selected_vertices.size(); i++)
140         {
141             TriMesh::VertexHandle vh((int)selected_vertices[i]);
142             TriMesh::Point current_point = _mesh.point(vh);
143             TriMesh::Normal current_normal = _mesh.normal(vh);
144             c += current_point;
145             n++;
146             vertices[i].position = glm::vec3(current_point[0], current_point[1],
    current_point[2]);
147             vertices[i].color = glm::vec3(1.0f, 0.0f, 0.0f);
148             vertices[i].normal = glm::vec3(current_normal[0], current_normal[1],
    current_normal[2]);
149         }
150         c = c / n;
151         centroid[0] = c[0];
152         centroid[1] = c[1];
153         centroid[2] = c[2];
154     }
```

Here is the caller graph for this function:



6.12.3.6 vertexBufferSize()

```
GLsizeiptr ShapeData::vertexBufferSize ( ) const  [inline]
```

vertexBufferSize - get size of the array of vertices

Returns

- size of array of vertices

Definition at line 159 of file myShape.h.

```
160     {
161         return numVertices * sizeof(Vertex);
162     }
```

6.12.4 Field Documentation

6.12.4.1 centroid

```
glm::vec3 ShapeData::centroid
```

centroid - shape centroid

Definition at line 59 of file myShape.h.

6.12.4.2 indices

```
GLuint* ShapeData::indices
```

indices - array containing all indices

Definition at line 54 of file myShape.h.

6.12.4.3 numIndices

```
GLuint ShapeData::numIndices
```

numIndices - number of indices of all triangles

Definition at line 45 of file myShape.h.

6.12.4.4 numVertices

```
GLuint ShapeData::numVertices
```

numVertices - number of vertices of the mesh

Definition at line 41 of file myShape.h.

6.12.4.5 vertices

```
Vertex* ShapeData::vertices
```

vertices - array containing all vertices

Definition at line 50 of file myShape.h.

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

- visualization/myShape.h

6.13 Vertex Struct Reference

The Vertex struct - single vertex with its corresponding attributes.

```
#include <myShape.h>
```

Collaboration diagram for Vertex:

Data Fields

- glm::vec3 position
  position - vertex position (x,y,z)
- glm::vec3 color
  color - vertex color (r,g,b)
- glm::vec3 normal
  normal - vertex normal (nx,ny,nz)

6.13.1 Detailed Description

The Vertex struct - single vertex with its corresponding attributes.

Definition at line 17 of file myShape.h.

6.13.2 Field Documentation

6.13.2.1 color

`glm::vec3 Vertex::color`

color - vertex color (r,g,b)

Definition at line 26 of file myShape.h.

Referenced by ShapeData::loadMesh(), and ShapeData::loadMeshVertexSelection().

6.13.2.2 normal

`glm::vec3 Vertex::normal`

normal - vertex normal (nx,ny,nz)

Definition at line 30 of file myShape.h.

Referenced by ShapeData::loadMesh(), and ShapeData::loadMeshVertexSelection().

6.13.2.3 position

`glm::vec3 Vertex::position`

position - vertex position (x,y,z)

Definition at line 22 of file myShape.h.

Referenced by ShapeData::loadMesh(), and ShapeData::loadMeshVertexSelection().

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

- visualization/myShape.h

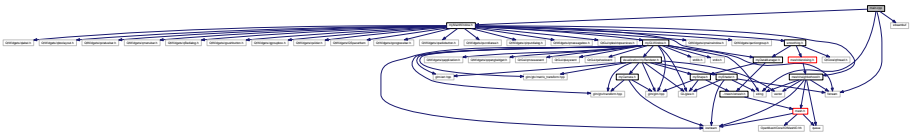
Chapter 7

File Documentation

7.1 main.cpp File Reference

```
#include "myMainWindow.h"
#include <string>
#include <fstream>
#include <streambuf>
```

Include dependency graph for main.cpp:



Functions

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

7.1.1 Function Documentation

7.1.1.1 main()

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

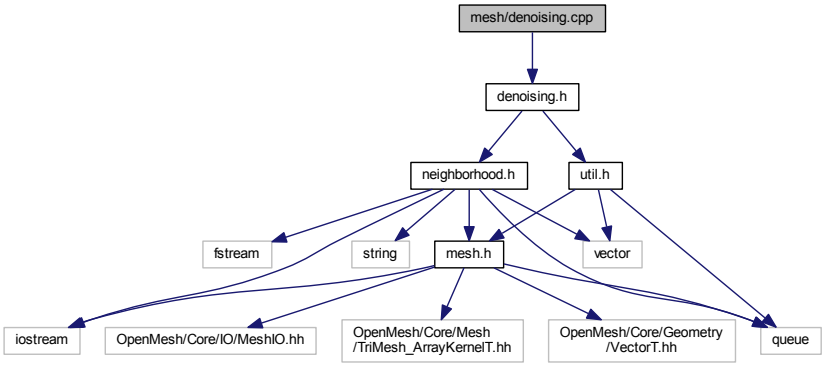
Definition at line 17 of file main.cpp.

```
18 {
19     QApplication app(argc, argv);
20     myMainWindow main_window;
21     main_window.show();
22     /*myGLWindow myWindow;
23     myWindow.show();*/
24     printf("OpenGL version supported by this platform (%s): \n",glGetString(GL_VERSION));
25     return app.exec();
26 }
```

7.2 mesh/denoising.cpp File Reference

```
#include "denoising.h"
```

Include dependency graph for denoising.cpp:



Functions

- bool hasIND (TriMesh::Point &p)

  Auxiliar functions.
- void updateVertexPositions (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, int iteration_number, bool fixed_boundary)

  updateVertexPositions - computes new vertex positions adapted to an input normal field.
- num_t getSigmaC (TriMesh &mesh, vector< TriMesh::Point > &face_centroids, num_t sigma_c_scalar)

  getSigmaC - sigma_c computation based on the average of face centroid distances (only between adjacent faces), and multiplied by a scalar
- num_t getRadius (TriMesh &mesh, num_t scalar)

  getRadius - computation of radius for radius-based face neighborhood, based on the average of face centroid distances (only between adjacent faces) and multiplied by a scalar
- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale)

  Uniform Laplacian smoothing.
- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale, vector< size_t > &vertex_ids)

  uniformLaplacian - denoise a subset of vertices of an input mesh using Uniform Laplacian Smoothing Algorithm
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta)

  HC Laplacian smoothing (Vollmer et al.)
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta, vector< size_t > &vertex_ids)

  HCLaplacian - denoise a subset of vertices of an input mesh using HC Laplacian Smoothing Algorithm.
- void updateFilteredNormals (TriMesh &mesh, int normal_iteration_number, num_t sigma_c_scalar, num_t sigma_s, vector< TriMesh::Normal > &filtered_normals)

  Bilateral normal filtering for mesh denoising (Zheng et al.)
- TriMesh bilateralNormal (TriMesh &_mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s)

  bilateralNormal - Denoise the input mesh using Bilateral Normal Filtering Algorithm
- void getAllFaceNeighborsGMNF (TriMesh &mesh, FaceNeighborType face_neighbor_type, num_t radius, bool include_central_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

Guided mesh normal filtering (Zhang et al.)

- void getAllGuidedNeighborsGMNF (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors)

  getAllGuidedNeighborsGMNF - neighborhood computation for guidance signal
- void getFaceNeighborsInnerEdges (TriMesh &mesh, vector< TriMesh::FaceHandle > &face_neighbors, vector< TriMesh::EdgeHandle > &inner_edges)

  getFaceNeighborsInnerEdges - get all inner edges of a face neighborhood
- void getConsistenciesAndMeanNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > &consistencies_and_mean_normals)

  getConsistenciesAndMeanNormals - consistency and mean normal computation for all patches
- void getGuidedNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > consistencies_and_mean_normals, vector< TriMesh::Normal > &guided_normals)

  getGuidedNormals - guided normal computation for each face of the mesh
- void updateFilteredNormalsGuided (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, num_t radius_scalar, num_t sigma_c_scalar, int normal_iteration_number, num_t sigma_s, int vertex_iteration_number)

  updateFilteredNormalsGuided - guided bilateral filtering of a normal field (face based) of a mesh
- TriMesh guided (TriMesh _mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s, num_t radius_scalar)

  guided - Denoise the input mesh using Guided Mesh Normal Filtering Algorithm

7.2.1 Function Documentation

7.2.1.1 hasIND()

```
bool hasIND (
            TriMesh::Point & p )
```

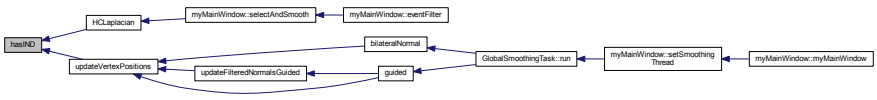
Auxiliar functions.

Definition at line 7 of file denoising.cpp.

Referenced by HCLaplacian(), and updateVertexPositions().

```
8 {
9     if (p[0] != p[0] || p[1] != p[1] || p[2] != p[2])
10         return true;
11     else return false;
12 }
```

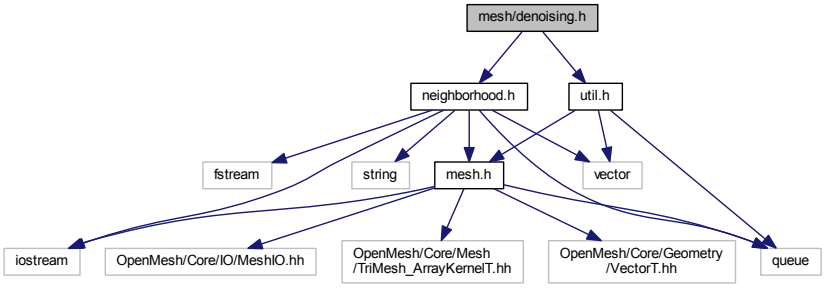
Here is the caller graph for this function:



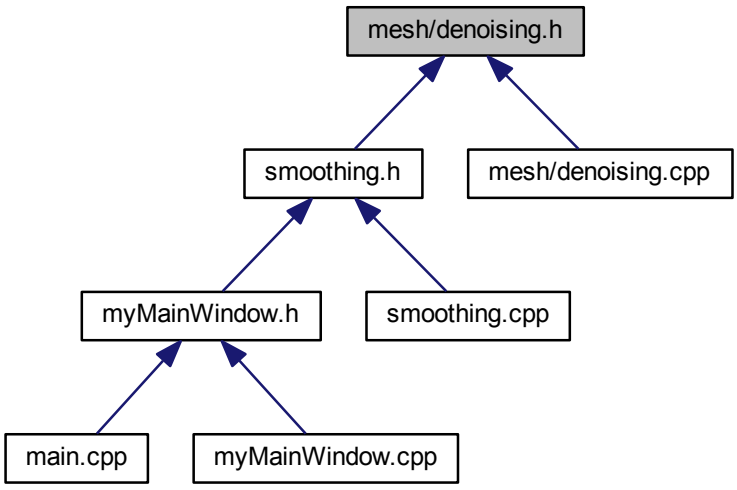
7.3 mesh/denoising.h File Reference

```
#include "neighborhood.h"
#include "util.h"
```

Include dependency graph for denoising.h:



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



Enumerations

- enum FaceNeighborType { kVertexBased, kEdgeBased, kRadiusBased }

  The FaceNeighborType enum - determines the face neighborhood features.

Functions

- void updateVertexPositions (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, int iteration_number, bool fixed_boundary)

  updateVertexPositions - computes new vertex positions adapted to an input normal field.
- num_t getSigmaC (TriMesh &mesh, vector< TriMesh::Point > &face_centroids, num_t sigma_c_scalar)

  getSigmaC - sigma_c computation based on the average of face centroid distances (only between adjacent faces), and multiplied by a scalar
- num_t getRadius (TriMesh &mesh, num_t scalar)

  getRadius - computation of radius for radius-based face neighborhood, based on the average of face centroid distances (only between adjacent faces) and multiplied by a scalar
- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale)

  Uniform Laplacian smoothing.
- TriMesh uniformLaplacian (TriMesh &_mesh, int iteration_number, num_t scale, vector< size_t > &vertex_ids)

  uniformLaplacian - denoise a subset of vertices of an input mesh using Uniform Laplacian Smoothing Algorithm
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta)

  HC Laplacian smoothing (Vollmer et al.)
- TriMesh HCLaplacian (TriMesh &_mesh, int iteration_number, num_t alpha, num_t beta, vector< size_t > &vertex_ids)

  HCLaplacian - denoise a subset of vertices of an input mesh using HC Laplacian Smoothing Algorithm.
- void updateFilteredNormals (TriMesh &mesh, int normal_iteration_number, num_t sigma_c_scalar, num_t sigma_s, vector< TriMesh::Normal > &filtered_normals)

  Bilateral normal filtering for mesh denoising (Zheng et al.)
- TriMesh bilateralNormal (TriMesh &_mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s)

  bilateralNormal - Denoise the input mesh using Bilateral Normal Filtering Algorithm
- void getAllFaceNeighborsGMNF (TriMesh &mesh, FaceNeighborType face_neighbor_type, num_t radius, bool include_central_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  Guided mesh normal filtering (Zhang et al.)
- void getAllGuidedNeighborsGMNF (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors)

  getAllGuidedNeighborsGMNF - neighborhood computation for guidance signal
- void getFaceNeighborsInnerEdges (TriMesh &mesh, vector< TriMesh::FaceHandle > &face_neighbors, vector< TriMesh::EdgeHandle > &inner_edges)

  getFaceNeighborsInnerEdges - get all inner edges of a face neighborhood
- void getConsistenciesAndMeanNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > &consistencies_and_mean_normals)

  getConsistenciesAndMeanNormals - consistency and mean normal computation for all patches
- void getGuidedNormals (TriMesh &mesh, vector< vector< TriMesh::FaceHandle > > &all_guided_neighbors, vector< num_t > &face_areas, vector< TriMesh::Normal > &normals, vector< pair< num_t, TriMesh::Normal > > consistencies_and_mean_normals, vector< TriMesh::Normal > &guided_normals)

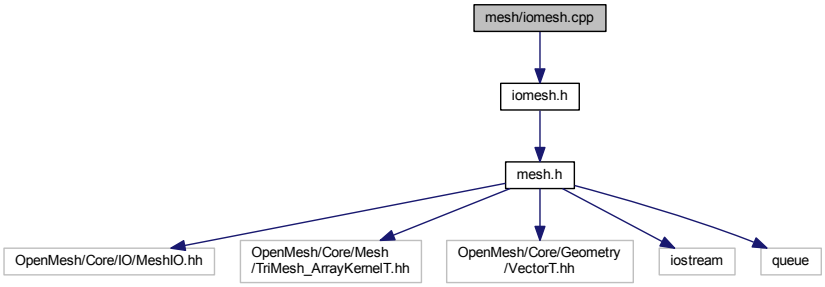
  getGuidedNormals - guided normal computation for each face of the mesh
- void updateFilteredNormalsGuided (TriMesh &mesh, vector< TriMesh::Normal > &filtered_normals, num_t radius_scalar, num_t sigma_c_scalar, int normal_iteration_number, num_t sigma_s, int vertex_iteration_number)

  updateFilteredNormalsGuided - guided bilateral filtering of a normal field (face based) of a mesh
- TriMesh guided (TriMesh _mesh, int normal_iteration_number, int vertex_iteration_number, num_t sigma_c_scalar, num_t sigma_s, num_t radius_scalar)

  guided - Denoise the input mesh using Guided Mesh Normal Filtering Algorithm

7.4 mesh/iomesh.cpp File Reference

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



Functions

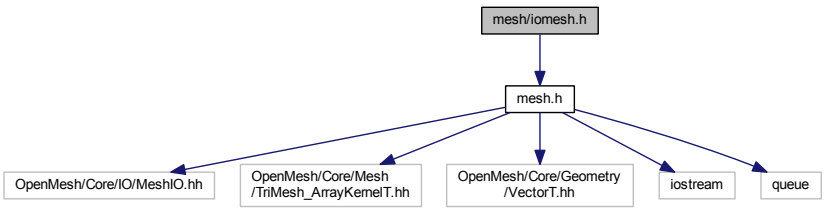
- bool importMesh (TriMesh &mesh, string filename)

  importMesh - read mesh file (.obj, .off, .ply, .stl)
- bool exportMesh (TriMesh &mesh, string filename)

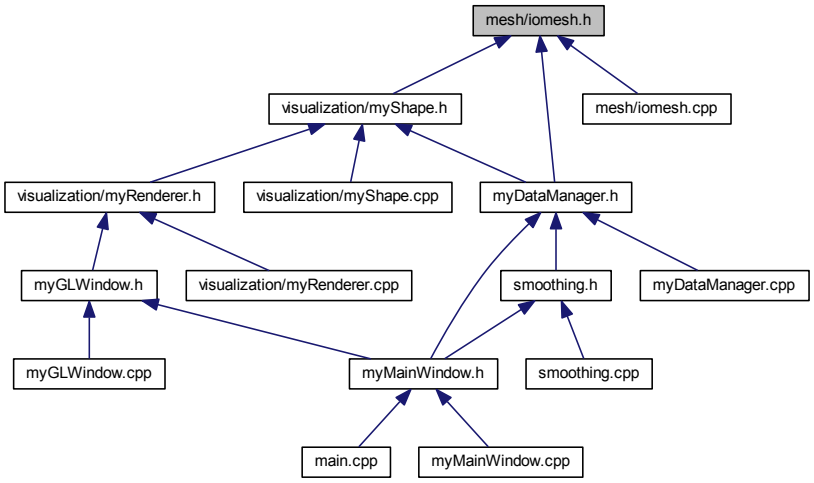
  exportMesh - write mesh file (.obj, .off, .ply, .stl)

7.5 mesh/iomesh.h File Reference

`#include "mesh.h"`
Include dependency graph for iomesh.h:



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



Functions

- bool importMesh (TriMesh &mesh, string filename)

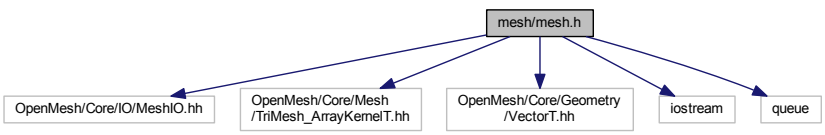
  importMesh - read mesh file (.obj, .off, .ply, .stl)
- bool exportMesh (TriMesh &mesh, string filename)

  exportMesh - write mesh file (.obj, .off, .ply, .stl)

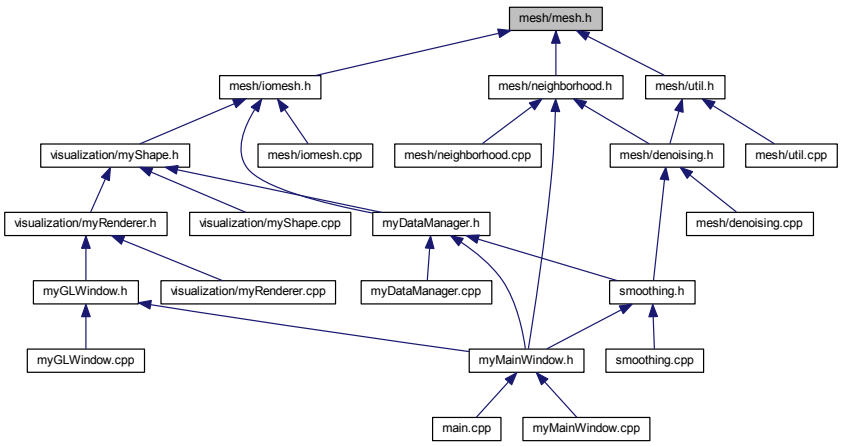
7.6 mesh/mesh.h File Reference

```
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/TriMesh_ArrayKernelT.hh>
#include <OpenMesh/Core/Geometry/VectorT.hh>
#include <iostream>
#include <queue>
```

Include dependency graph for mesh.h:



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



Data Structures

- struct MyTraits

  The MyTraits struct - OpenMesh custom traits.

Typedefs

- typedef OpenMesh::TriMesh_ArrayKernelT< MyTraits > TriMesh

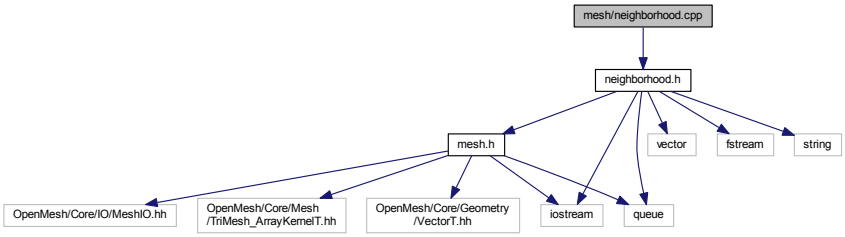
  TriMesh - triangular mesh definition.
- typedef float num_t

  num_t - type definition for numbers used in mesh processing module

7.7 mesh/neighborhood.cpp File Reference

```
#include "neighborhood.h"
```

Include dependency graph for neighborhood.cpp:



Functions

- void getVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, int k, vector< TriMesh::VertexHandle > &vertex_neighbors)

  Vertex neighborhood.
- void getAdaptiveVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, num_t radius, vector< TriMesh::VertexHandle > &vertex_neighbors)

  getAdaptiveVertexNeighbors: get all neighboring vertices of a given vertex regarding a radius
- void getAllVertexNeighbors (TriMesh &mesh, int k, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllVertexNeighbors: get all neighboring vertices of all vertices regarding a depth k (k-ring)
- void getAllAdaptiveVertexNeighbors (TriMesh &mesh, num_t radius, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllAdaptiveVertexNeighbors: get all neighboring vertices of all vertices regarding a radius
- void getFaceNeighbors_EdgeBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  Face neighborhood.
- void getFaceNeighbors_VertexBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_VertexBased: get neighboring faces based on face vertices
- void getFaceNeighbors_RadiusBased (TriMesh &mesh, TriMesh::FaceHandle fh, num_t radius, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_RadiusBased: get neighboring faces regarding a radius
- void getAllFaceNeighbors_EdgeBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

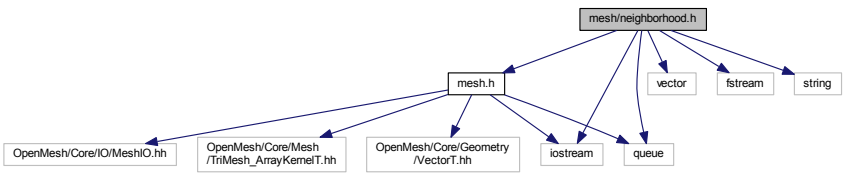
  getAllFaceNeighbors_EdgeBased: get all neighboring faces of all faces (edge based)
- void getAllFaceNeighbors_VertexBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  getAllFaceNeighbors_VertexBased: get all neighboring faces of all faces (vertex based)

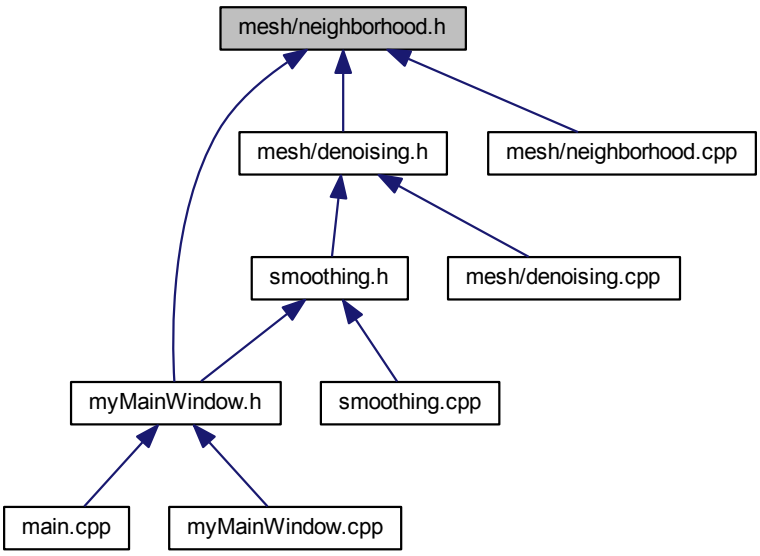
7.8 mesh/neighborhood.h File Reference

```
#include "mesh.h"
#include <vector>
#include <queue>
#include <fstream>
#include <iostream>
#include <string>
```

Include dependency graph for neighborhood.h:



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



Functions

- void getVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, int k, vector< TriMesh::VertexHandle > &vertex_neighbors)

  Vertex neighborhood.
- void getAdaptiveVertexNeighbors (TriMesh &mesh, TriMesh::VertexHandle vh, num_t radius, vector< TriMesh::VertexHandle > &vertex_neighbors)

  getAdaptiveVertexNeighbors: get all neighboring vertices of a given vertex regarding a radius
- void getAllVertexNeighbors (TriMesh &mesh, int k, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllVertexNeighbors: get all neighboring vertices of all vertices regarding a depth k (k-ring)
- void getAllAdaptiveVertexNeighbors (TriMesh &mesh, num_t radius, vector< vector< TriMesh::VertexHandle > > &all_vertex_neighbors)

  getAllAdaptiveVertexNeighbors: get all neighboring vertices of all vertices regarding a radius
- void getFaceNeighbors_EdgeBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  Face neighborhood.
- void getFaceNeighbors_VertexBased (TriMesh &mesh, TriMesh::FaceHandle fh, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_VertexBased: get neighboring faces based on face vertices
- void getFaceNeighbors_RadiusBased (TriMesh &mesh, TriMesh::FaceHandle fh, num_t radius, vector< TriMesh::FaceHandle > &face_neighbors)

  getFaceNeighbors_RadiusBased: get neighboring faces regarding a radius
- void getAllFaceNeighbors_EdgeBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

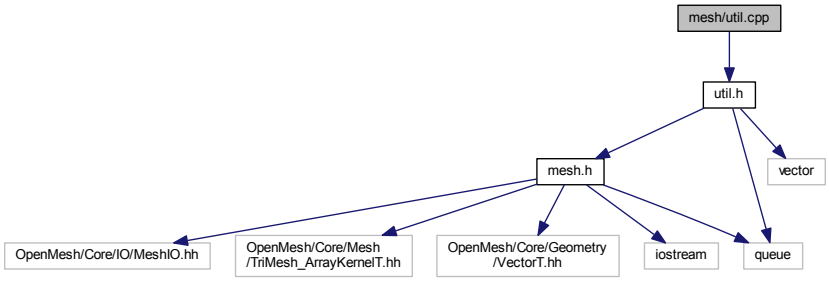
  getAllFaceNeighbors_EdgeBased: get all neighboring faces of all faces (edge based)
- void getAllFaceNeighbors_VertexBased (TriMesh &mesh, bool include_target_face, vector< vector< TriMesh::FaceHandle > > &all_face_neighbors)

  getAllFaceNeighbors_VertexBased: get all neighboring faces of all faces (vertex based)

7.9 mesh/util.cpp File Reference

```
#include "util.h"
```

Include dependency graph for util.cpp:



Functions

- num_t getArea (TriMesh &mesh)

  getArea - area computation of a mesh (sum of triangle areas)
- num_t getVolume (TriMesh &mesh)

  getVolume - volume computation of a mesh
- num_t getAverageEdgeLength (TriMesh &mesh)

  getAverageEdgeLength - average edge length computation of a mesh
- void getAllFaceAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllFaceAreas - area computation for all faces
- void getAllFaceCentroids (TriMesh &mesh, vector< TriMesh::Point > &centroids)

  getAllFaceCentroids - centroid computation for all faces
- void getAllFaceNormals (TriMesh &mesh, vector< TriMesh::Normal > &normals)

  getAllFaceNormals - normal computation for all faces
- void getFaceVertexAngle (TriMesh &mesh, TriMesh::FaceHandle fh, TriMesh::VertexHandle vh, num_t &angle)

  getFaceVertexAngle - angle computation for a given vertex included in a face
- num_t getVertexArea (TriMesh &mesh, TriMesh::VertexHandle vh, vector< num_t > &areas)

  getVertexArea - vertex area computation (barycentric area)
- void getAllVertexAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllVertexAreas - vertex area computation for all vertices (barycentric areas)
- void getAllPoints (TriMesh &mesh, vector< TriMesh::Point > &points)

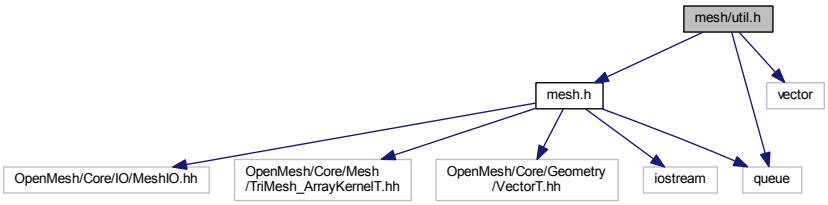
  getAllPoints - get all vertex coordinates

7.10 mesh/util.h File Reference

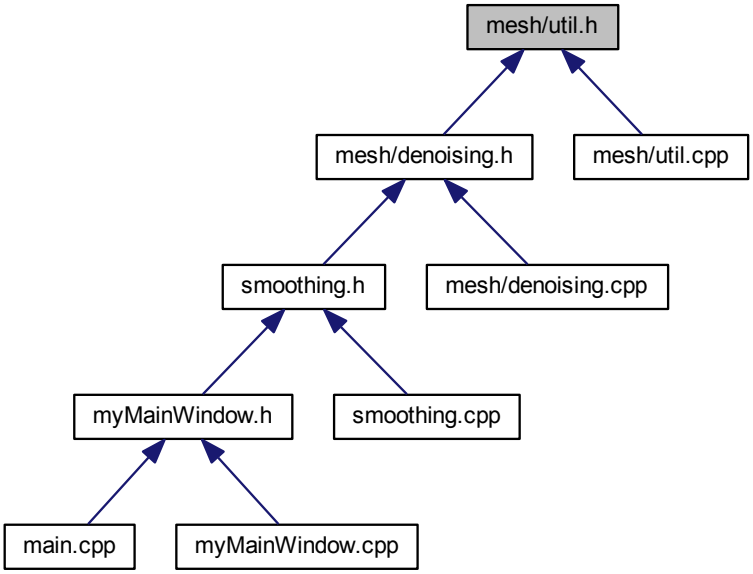
```
#include "mesh.h"
#include <vector>
```

```
#include <queue>
```

Include dependency graph for util.h:



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



Macros

- #define PI 3.14159265359

Functions

- num_t getArea (TriMesh &mesh)

  getArea - area computation of a mesh (sum of triangle areas)
- num_t getVolume (TriMesh &mesh)

  getVolume - volume computation of a mesh
- num_t getAverageEdgeLength (TriMesh &mesh)

  getAverageEdgeLength - average edge length computation of a mesh

- void getAllFaceAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllFaceAreas - area computation for all faces
- void getAllFaceCentroids (TriMesh &mesh, vector< TriMesh::Point > &centroids)

  getAllFaceCentroids - centroid computation for all faces
- void getAllFaceNormals (TriMesh &mesh, vector< TriMesh::Normal > &normals)

  getAllFaceNormals - normal computation for all faces
- void getFaceVertexAngle (TriMesh &mesh, TriMesh::FaceHandle fh, TriMesh::VertexHandle vh, num_t &angle)

  getFaceVertexAngle - angle computation for a given vertex included in a face
- num_t getVertexArea (TriMesh &mesh, TriMesh::VertexHandle vh, vector< num_t > &areas)

  getVertexArea - vertex area computation (barycentric area)
- void getAllVertexAreas (TriMesh &mesh, vector< num_t > &areas)

  getAllVertexAreas - vertex area computation for all vertices (barycentric areas)
- void getAllPoints (TriMesh &mesh, vector< TriMesh::Point > &points)

  getAllPoints - get all vertex coordinates
- num_t GaussianWeight (num_t distance, num_t sigma)

  GaussianWeight - gaussian function computation used for bilateral filtering.
- num_t NormalDistance (const TriMesh::Normal &n1, const TriMesh::Normal &n2)

  NormalDistance - computation of normal distance: $|n1 - n2|$.

7.10.1 Macro Definition Documentation

7.10.1.1 PI

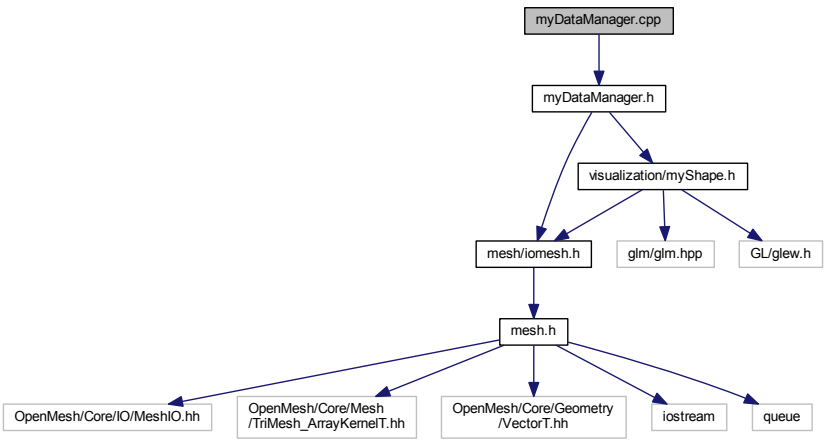
```
#define PI 3.14159265359
```

Definition at line 10 of file util.h.

7.11 myDataManager.cpp File Reference

```
#include "myDataManager.h"
```

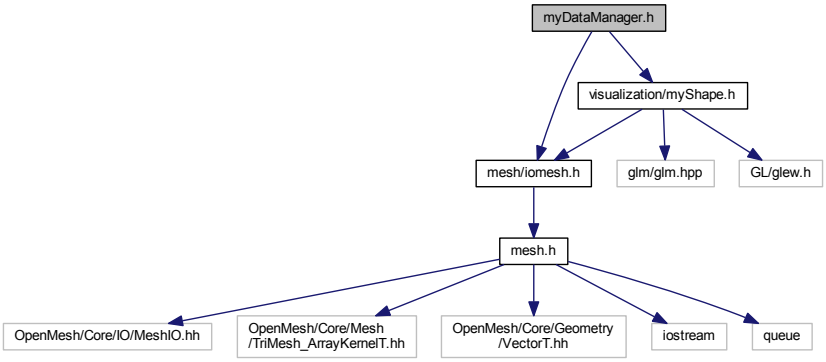
Include dependency graph for myDataManager.cpp:



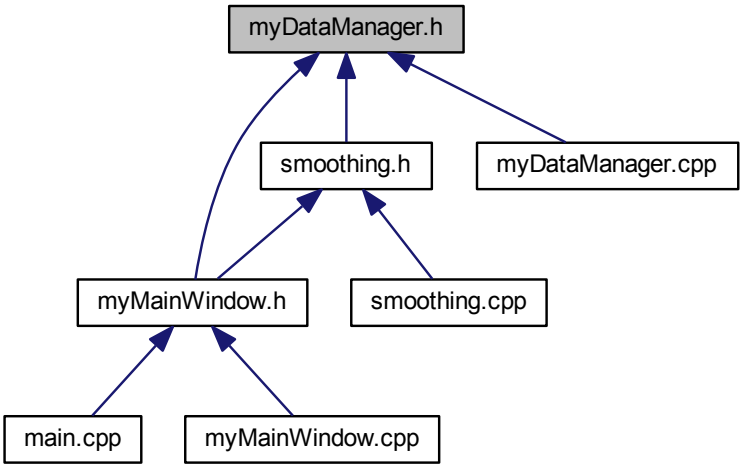
7.12 myDataManager.h File Reference

```
#include "mesh/iomesh.h"
#include "visualization/myShape.h"
```

Include dependency graph for myDataManager.h:



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



Data Structures

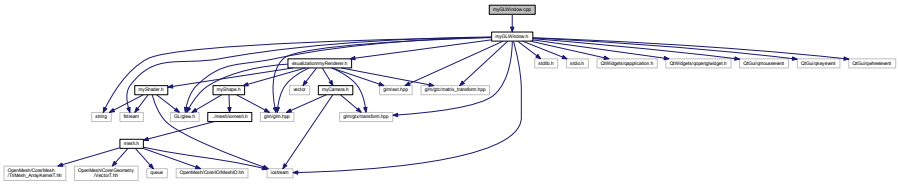
- class myDataManager

  The myDataManager class - data manager for mesh smoothing application.

7.13 myGLWindow.cpp File Reference

```
#include "myGLWindow.h"
```

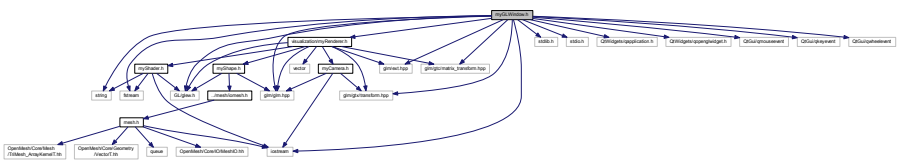
Include dependency graph for myGLWindow.cpp:



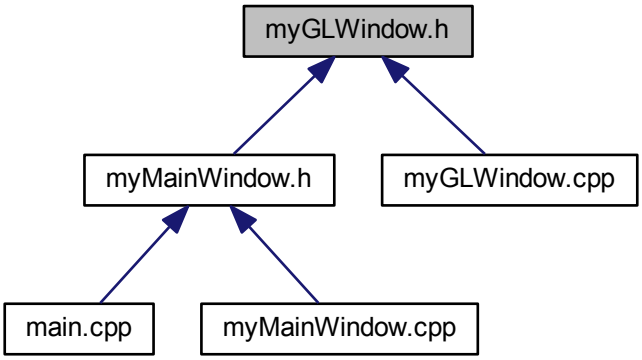
7.14 myGLWindow.h File Reference

```
#include <string>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <GL/glew.h>
#include <glm/glm.hpp>
#include <glm/ext.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform.hpp>
#include <QtWidgets/qapplication.h>
#include <QtWidgets/qopenglwidget.h>
#include <QtGui/qmouseevent>
#include <QtGui/qkeyevent>
#include <QtGui/qwheelevent>
#include "visualization/myRenderer.h"
```

Include dependency graph for myGLWindow.h:



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



Data Structures

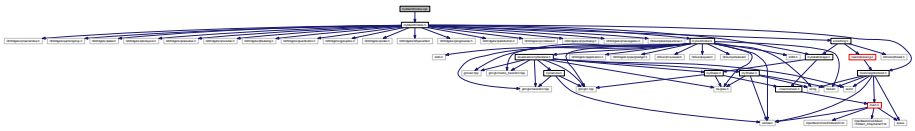
- class myGLWindow

  The myGLWindow class - Qt Widget for OpenGL Designed to support only one shape and one selection at time. So in the containers of the renderer there are two shapes at position 0 and 1. The first one for the current shape and the second one for the current selection. This object adapts a renderer to a QT OpenGL Widget which manages a single OpenGL context. Also, it simplifies the usage of the renderer for the mesh smoothing application.

7.15 myMainWindow.cpp File Reference

```
#include "myMainWindow.h"
```

Include dependency graph for myMainWindow.cpp:



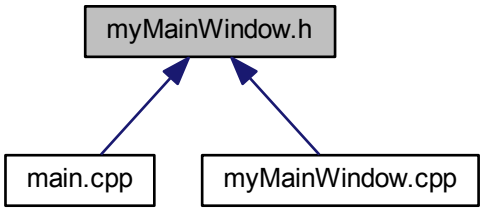
7.16 myMainWindow.h File Reference

```
#include <QtWidgets/qmainwindow.h>
#include <QtWidgets/qactiongroup.h>
#include <QtWidgets/qlabel.h>
#include <QtWidgets/qboxlayout.h>
#include <QtWidgets/qstatusbar.h>
#include <QtWidgets/qmenubar.h>
#include <QtWidgets/qfiledialog.h>
#include <QtWidgets/qpushbutton.h>
```

```
#include <QtWidgets/qgroupbox.h>
#include <QtWidgets/qslider.h>
#include <QtWidgets/QSpacerItem>
#include <QtWidgets/qprogressbar.h>
#include <QtWidgets/qradiobutton.h>
#include <QtWidgets/qscrollarea.h>
#include <QtWidgets/qinputdialog.h>
#include <QtWidgets/qmessagebox.h>
#include <QtGui/qdesktopservices.h>
#include "myGLWindow.h"
#include "myDataManager.h"
#include "mesh/neighborhood.h"
#include "smoothing.h"
```

Include dependency graph for myMainWindow.h:

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



Data Structures

- class myMainWindow

  The myMainWindow class - Main Window for mesh smoothing application.

Enumerations

- enum globalSmoothingStatus { gs_status_init, gs_status_started, gs_status_stopping, gs_status_stopped, gs_status_continuing }

  The globalSmoothingStatus enum - global smoothing status.
- enum globalSmoothingAlgorithm { gs_algorithm_bilateral_normal, gs_algorithm_guided }

  The globalSmoothingAlgorithm enum - global smoothing algorithms.
- enum focalizedSmoothingAlgorithm { fs_algorithm_uniform_laplacian, fs_algorithm_hc_laplacian }

  The focalizedSmoothingAlgorithm enum - focalized smoothing algorithms.

Variables

- const vector< string > globalSmoothingAlgorithmLabels

  Global smoothing algorithm names.
- const vector< string > focalizedSmoothingAlgorithmLabels

  Focalized smoothing algorithm names.

7.16.1 Enumeration Type Documentation

7.16.1.1 focalizedSmoothingAlgorithm

```
enum focalizedSmoothingAlgorithm
```

The focalizedSmoothingAlgorithm enum - focalized smoothing algorithms.

Enumerator

fs_algorithm_uniform_laplacian	uniform laplacian smoothing
fs_algorithm_hc_laplacian	hc laplacian smoothing

Definition at line 55 of file myMainWindow.h.

```
55                                {
56     fs_algorithm_uniform_laplacian,
57     fs_algorithm_hc_laplacian
58 };
```

7.16.1.2 globalSmoothingAlgorithm

```
enum globalSmoothingAlgorithm
```

The globalSmoothingAlgorithm enum - global smoothing algorithms.

Enumerator

gs_algorithm_bilateral_normal	bilateral normal filtering
gs_algorithm_guided	guided mesh denoising

Definition at line 41 of file myMainWindow.h.

```
41                               {
42     gs_algorithm_bilateral_normal,
43     gs_algorithm_guided
44 };
```

7.16.1.3 globalSmoothingStatus

```
enum globalSmoothingStatus
```

The globalSmoothingStatus enum - global smoothing status.

Enumerator

gs_status_init	status init (global smoothing: not running and not started)
gs_status_started	status started (global smoothing: running)
gs_status_stopping	status stopping (global smoothing: running and will be stopped)
gs_status_stopped	status stopped (global smoothing: not running)
gs_status_continuing	status continuing (global smoothing: running)

Definition at line 30 of file myMainWindow.h.

```
30                          {
31     gs_status_init,
32     gs_status_started,
33     gs_status_stopping,
34     gs_status_stopped,
35     gs_status_continuing
36 };
```

7.16.2 Variable Documentation

7.16.2.1 focalizedSmoothingAlgorithmLabels

```
const vector<string> focalizedSmoothingAlgorithmLabels
```

Initial value:

```
= {
    "Uniform Laplacian",
    "HC Laplacian"
}
```

Focalized smoothing algorithm names.

Definition at line 61 of file myMainWindow.h.

Referenced by myMainWindow::setFocalizedSmoothingAlgorithm().

7.16.2.2 globalSmoothingAlgorithmLabels

```
const vector<string> globalSmoothingAlgorithmLabels
```

Initial value:

```
= {
    "Bilateral Normal Filtering",
    "Guided Mesh Denoising"
}
```

Global smoothing algorithm names.

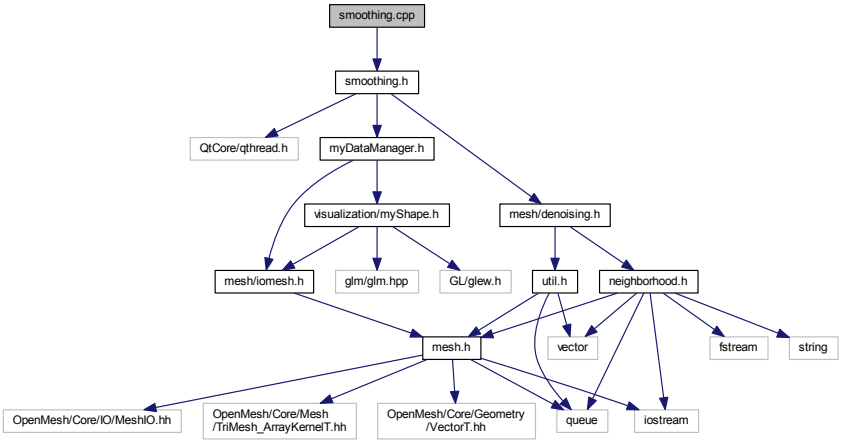
Definition at line 47 of file myMainWindow.h.

Referenced by myMainWindow::setGlobalSmoothingAlgorithm().

7.17 smoothing.cpp File Reference

```
#include "smoothing.h"
```

Include dependency graph for smoothing.cpp:

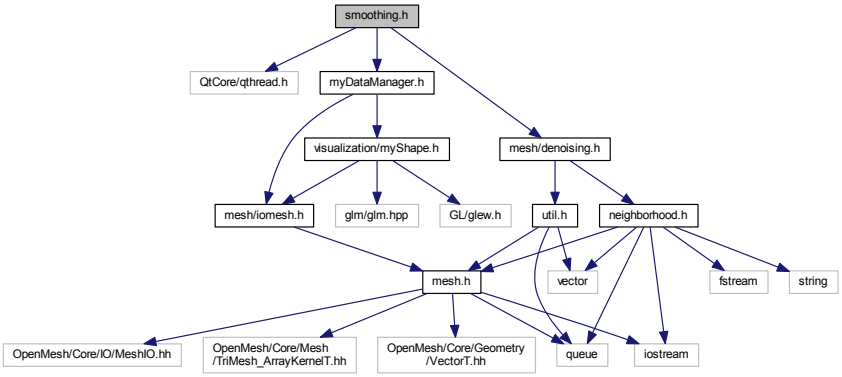


7.18 smoothing.h File Reference

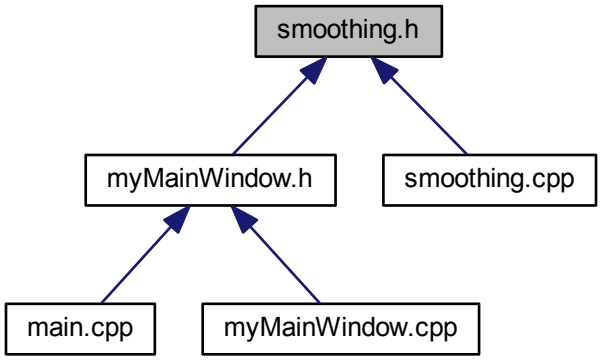
```
#include <QtCore/qthread.h>
#include "myDataManager.h"
```

```
#include "mesh/denoising.h"
```

Include dependency graph for smoothing.h:



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



Data Structures

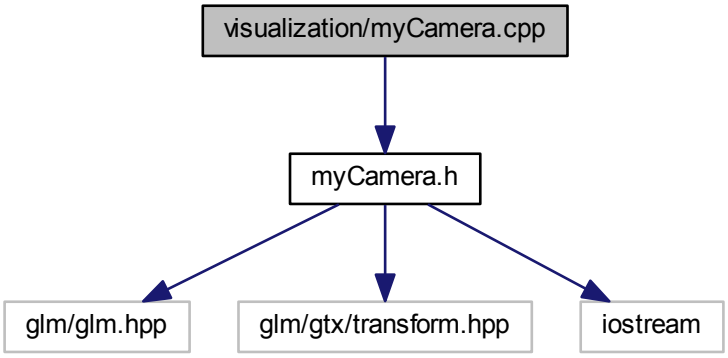
- class GlobalSmoothingTask

  The GlobalSmoothingTask class - global smoothing task.

7.19 visualization/myCamera.cpp File Reference

```
#include "myCamera.h"
```

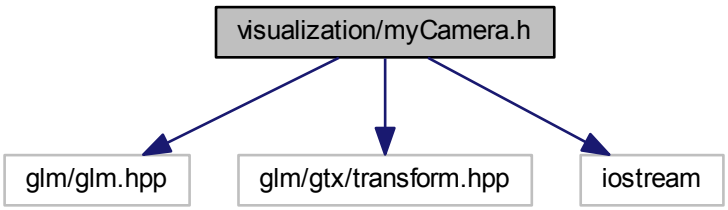
Include dependency graph for myCamera.cpp:



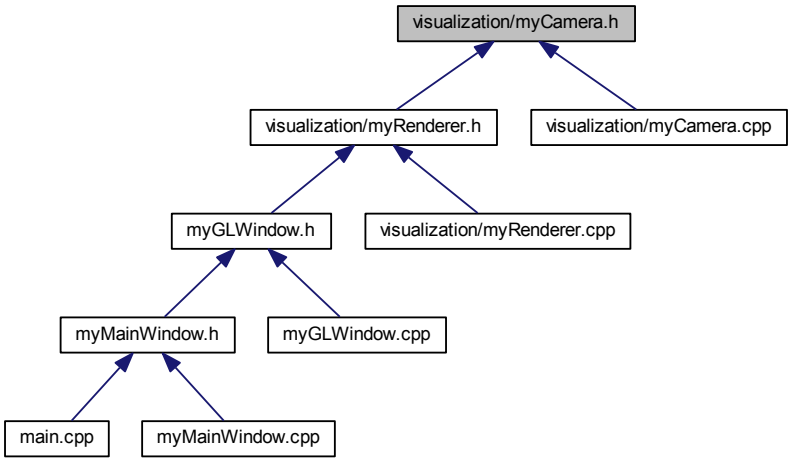
7.20 visualization/myCamera.h File Reference

```
#include <glm/glm.hpp>
#include <glm/gtx/transform.hpp>
#include <iostream>
```

Include dependency graph for myCamera.h:



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



Data Structures

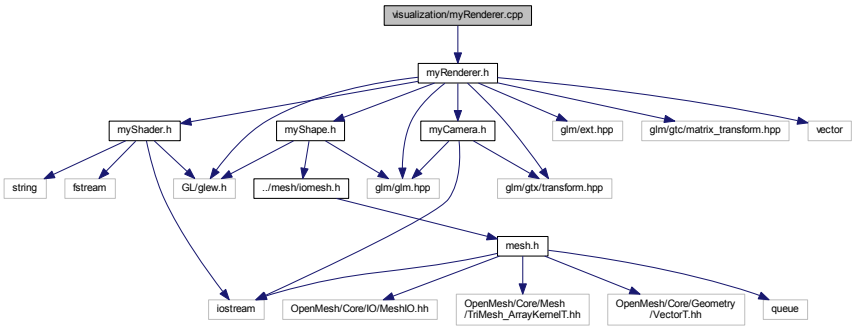
- class myCamera

  The myCamera class - camera representation in a typical rendering pipeline (i.e. OpenGL)

7.21 visualization/myRenderer.cpp File Reference

```
#include "myRenderer.h"
```

Include dependency graph for myRenderer.cpp:



Functions

- bool checkStatus (GLuint objectID, PFNGLGETSHADERIVPROC objectPropertyGetterFunc, PFNGLGET↩ SHADERINFOLOGPROC getInfoLogFunc, GLenum statusType)
- bool checkShaderStatus (GLuint shaderID)
- bool checkProgramStatus (GLuint programID)

7.21.1 Function Documentation

7.21.1.1 checkProgramStatus()

```
bool checkProgramStatus (
            GLuint programID )
```

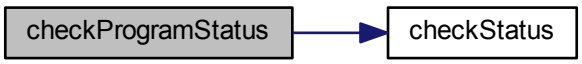
Definition at line 28 of file myRenderer.cpp.

References checkStatus().

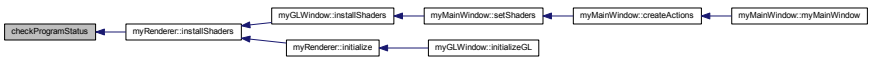
Referenced by myRenderer::installShaders().

```
29 {
30     return checkStatus(programID, glGetProgramiv, glGetProgramInfoLog, GL_LINK_STATUS);
31 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.1.2 checkShaderStatus()

```
bool checkShaderStatus (
            GLuint shaderID )
```

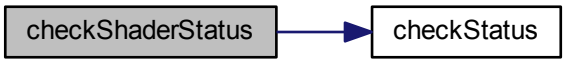
Definition at line 23 of file myRenderer.cpp.

References checkStatus().

Referenced by myRenderer::installShaders().

```
24 {
25     return checkStatus(shaderID, glGetShaderiv, glGetShaderInfoLog, GL_COMPILE_STATUS);
26 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

7.21.1.3 checkStatus()

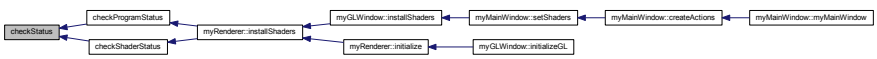
```
bool checkStatus (
            GLuint objectID,
            PFNGLGETSHADERIVPROC objectPropertyGetterFunc,
            PFNGLGETSHADERINFOLOGPROC getInfoLogFunc,
            GLenum statusType )
```

Definition at line 5 of file myRenderer.cpp.

Referenced by checkProgramStatus(), and checkShaderStatus().

```
6 {
7     GLint status;
8     objectPropertyGetterFunc(objectID, statusType, &status);
9     if (status != GL_TRUE)
10     {
11         GLint infoLogLength;
12         objectPropertyGetterFunc(objectID, GL_INFO_LOG_LENGTH, &infoLogLength);
13         GLchar * buffer = new GLchar[infoLogLength];
14         GLsizei bufferSize;
15         getInfoLogFunc(objectID, infoLogLength, &bufferSize, buffer);
16         cout << buffer << endl;
17         delete[] buffer;
18         return false;
19     }
20     return true;
21 }
```

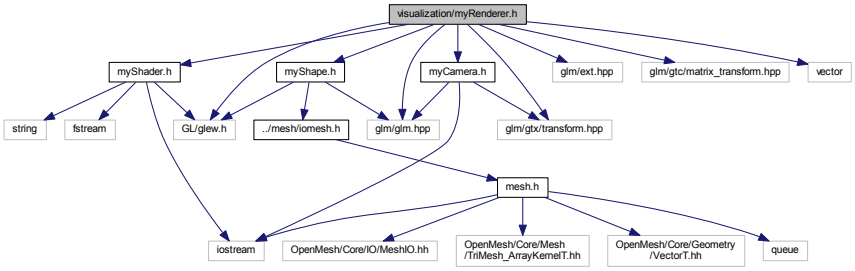
Here is the caller graph for this function:



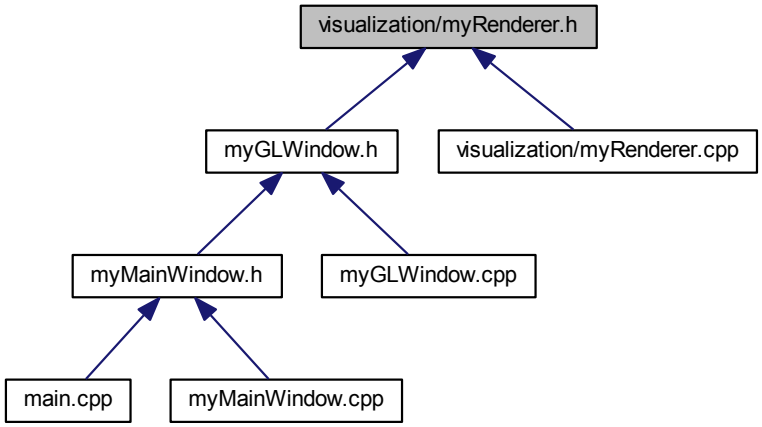
7.22 visualization/myRenderer.h File Reference

```
#include <GL/glew.h>
#include <glm/glm.hpp>
#include <glm/ext.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtx/transform.hpp>
#include <vector>
#include "myShader.h"
#include "myShape.h"
#include "myCamera.h"
```

Include dependency graph for myRenderer.h:



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



Data Structures

- class myRenderer

  The myRenderer class - Renderer for multiple shapes (triangular meshes) visualization.

Enumerations

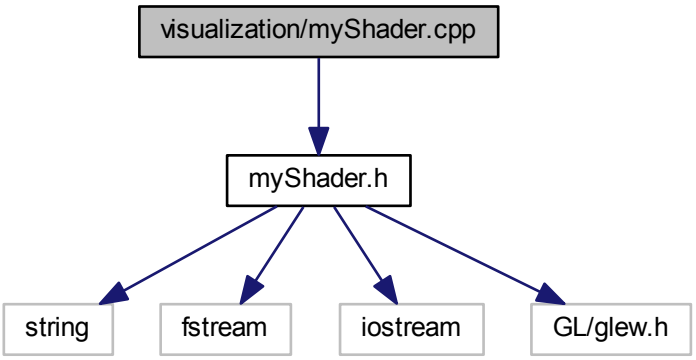
- enum myDrawFlags { e_draw_faces, e_draw_wireframe, e_draw_points, e_draw_selection }

  The myDrawFlags enum - Rendering mode.

7.23 visualization/myShader.cpp File Reference

```
#include "myShader.h"
```

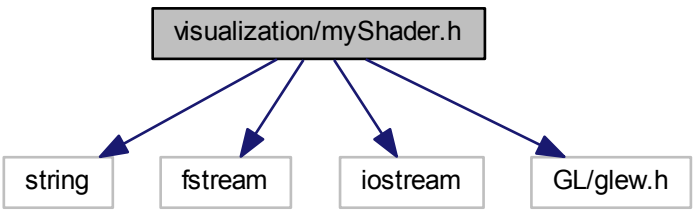
Include dependency graph for myShader.cpp:



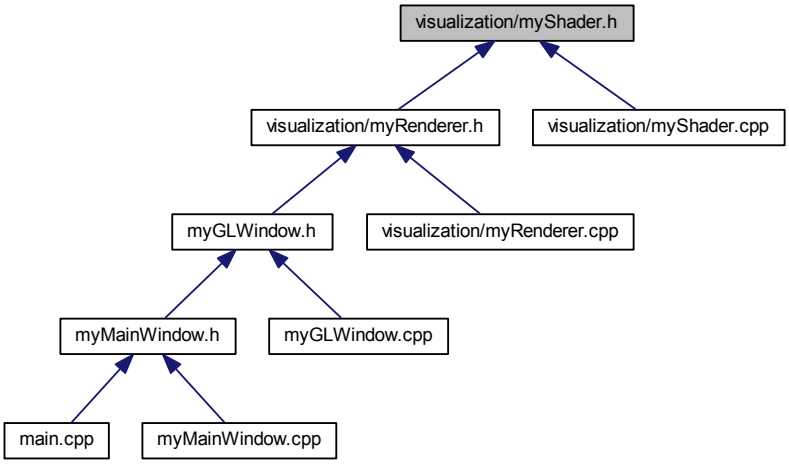
7.24 visualization/myShader.h File Reference

```
#include <string>
#include <fstream>
#include <iostream>
#include <GL/glew.h>
```

Include dependency graph for myShader.h:



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



Data Structures

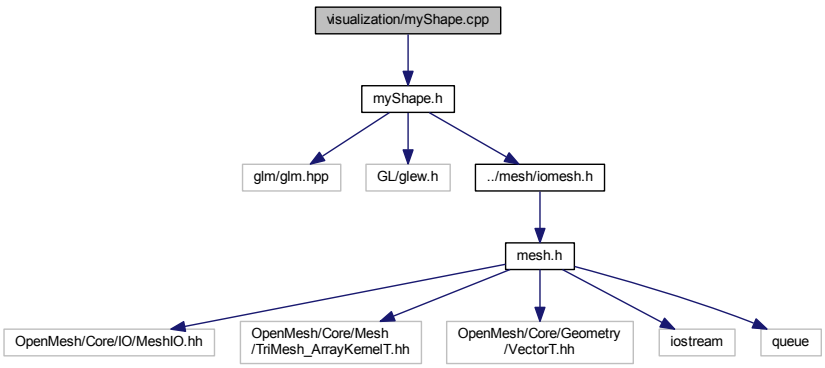
- class myShader

  The myShader class - shader data for OpenGL.

7.25 visualization/myShape.cpp File Reference

```
#include "myShape.h"
```

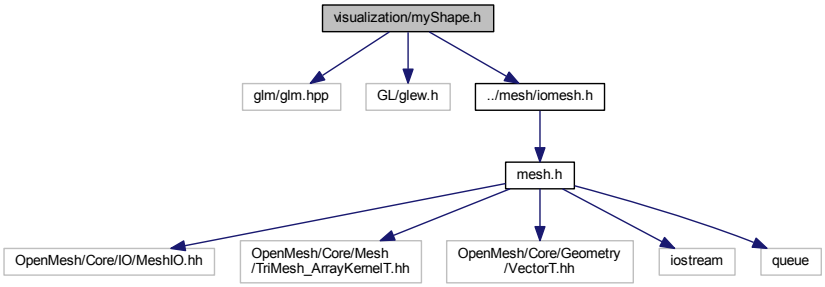
Include dependency graph for myShape.cpp:



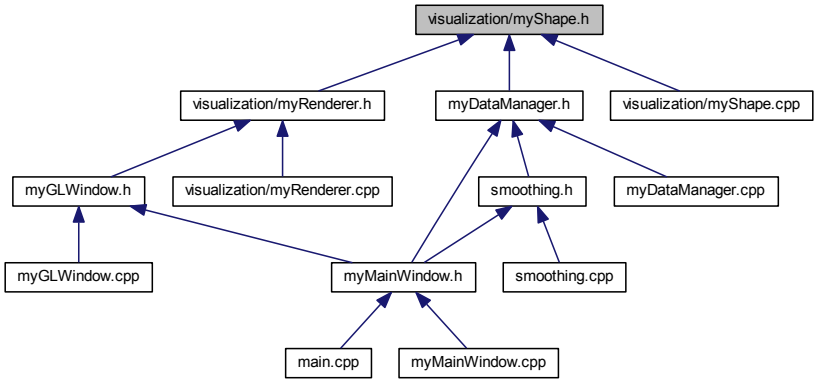
7.26 visualization/myShape.h File Reference

```
#include <glm/glm.hpp>
#include <GL/glew.h>
#include "../mesh/iomesh.h"
```

Include dependency graph for myShape.h:



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



Data Structures

- struct Vertex

  The Vertex struct - single vertex with its corresponding attributes.
- struct ShapeData

  The ShapeData struct - triangular mesh for OpenGL buffers manipulation.

Index