

Software-Project 2017

Detailed Design

Real-Time Mesh Utilities

Petros Simidyan
Julius Lerm

Blerta Hamzallari
Lars Debor

Felix Griesau
Simon Heinke

Marco Klamke
Sugandha Sachdeva

last change: June 5, 2017

figures/mne-cpp.png

Contents

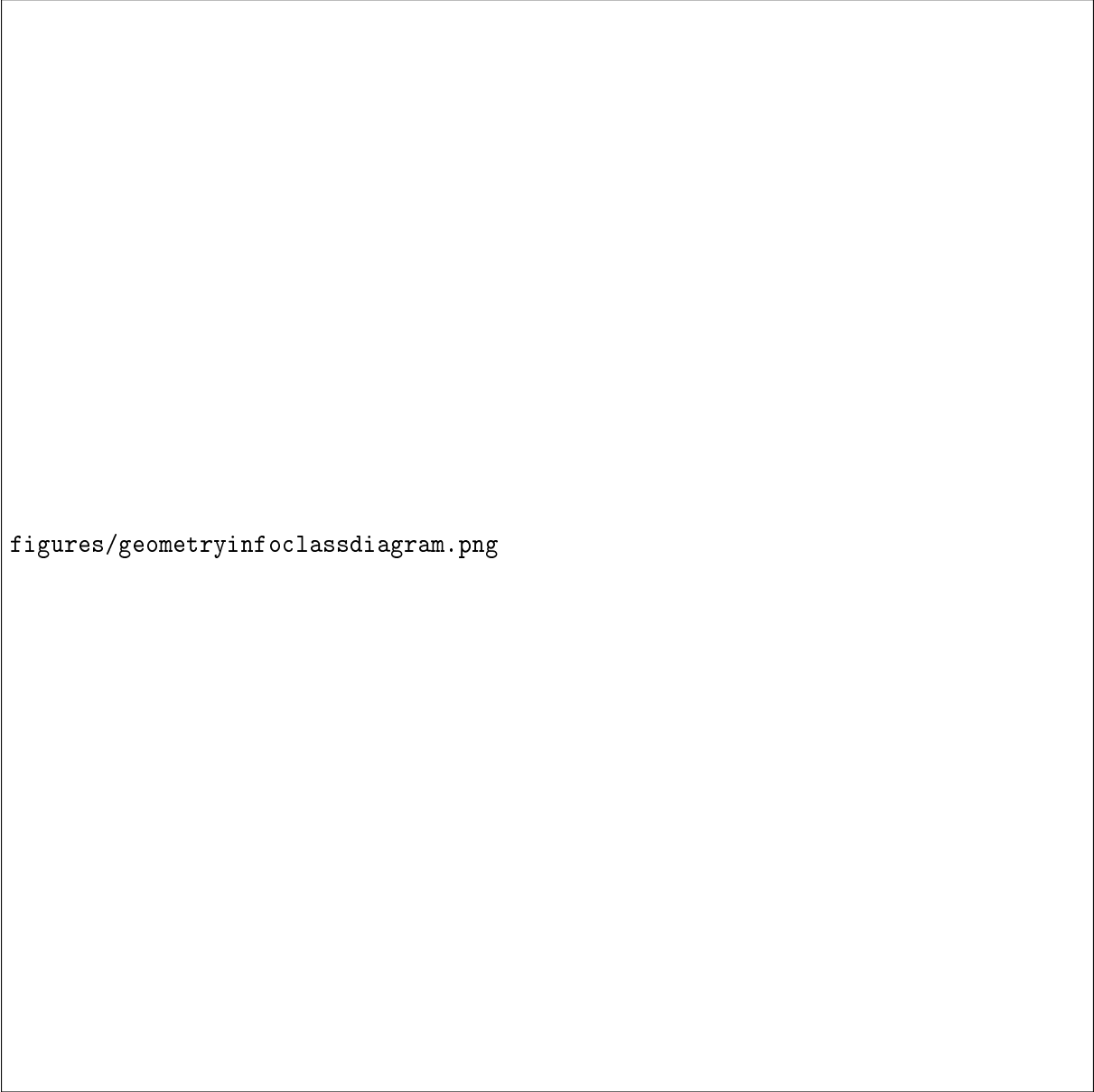
1	General Architecture	2
2	GeometryInfo	3
2.1	Member Functions	4
2.1.1	scdc	4
2.1.2	iterativeDijkstra	4
2.1.3	projectSensors	4
2.1.4	nearestNeighbor	5
2.1.5	matrixDump	5
2.1.6	squared	5
3	Interpolation	6
3.1	Member Objects	7
3.1.1	Weight Matrix	7
3.2	Member Functions	7
3.2.1	createInterpolationMatrix	7
3.2.2	interpolateSignal	7
3.2.3	clearInterpolationMatrix	7
3.2.4	getResult	7
3.2.5	linear, gaussian, square	7
4	Glossary	8

1 General Architecture

The features of the mesh utilities are divided into two classes...

2 GeometryInfo

The class *GeometryInfo* holds all needed functions for the Surface Constrained Distance Calculation (from now on abbreviated with SCDC) and the Sensor Projecting (see Functional Specification for further details). Since all functions are static, the class itself does not have to be instantiated, thus the class also is declared static and the default constructor is forbidden. The class *GeometryInfo* does not have any class members.



figures/geometryinfoclassdiagram.png

Figure 1: Interface of GeometryInfo

2.1 Member Functions

2.1.1 `scdc`

The method `scdc` originally was thought to receive only one input argument, i.e. the `MNEBemSurface` (object that holds the needed adjacency information). Based on this, it was supposed to calculate the full distance table. Since the latter would have needed around 160 gigabyte of memory for some meshes, it was decided that the method also receives a subset of vertices. It then only calculates all distances for each vertex of said subset. The subset argument is defaulted with an empty vector, in case a subset could not be provided or is not wanted.

After first tests revealed that an unrestricted distance calculation would take far too long, a third argument was introduced: A double value that acts as a distance threshold and thus restricts the distance calculation to vertices which lie within the said radius. This seems a bit imprecise or careless at first glance, but it actually is reasonable for this specific application: Bilateral stimulation of cells inside the brain only occurs up to a certain distance. The third argument is defaulted with the maximum double-value, in case a distance threshold could not be provided or is not wanted.

After these restrictions and adaptations were agreed upon, the question which algorithm to choose arose. While P.a.R.'s algorithm had the best asymptotic run-time, further research revealed that an implementation would go far beyond the scope of this project, since the algorithm uses several other algorithms as subroutines (such as Jarnik and Prim). Another possible candidate for solving the problem was Thorups algorithm. Testing of a publicly available implementation showed that it actually did worse than another algorithm, when the above mentioned restrictions to the problem were applied. The one algorithm that resulted in a lower computation time was an iterative version of Dijkstras algorithm (see below for more details).

Since two instances of iterative Dijkstra would not interfere with each other (because they work on different columns of the distance table), it was decided for the `scdc` method to start several threads. Before any threads are started, the distance table is allocated. According to the number of available threads (or cores, depending on hyperthreading etc.), the passed subset of vertices gets split into equally sized subvectors, whose limiting indices are then passed to an instance of iterative Dijkstra.

2.1.2 `iterativeDijkstra`

The method `iterativeDijkstra` receives six input arguments: Firstly a pointer to the distance table which the results are to be written to. Secondly, a reference to an `MNEBemSurface` which holds the necessary adjacency information. As described above, the `iterativeDijkstra` method receives a vector of indices which point to vertices inside the `MNEBemSurface`. Added to that, it gets passed two indices which limit the section of said vector that the respective instance of `iterativeDijkstra` is responsible for. Since the external requirements (i.e. low computation time and overall high efficiency) induce some restrictions (see description of `scdc` for more details), a distance threshold gets passed as the sixth argument. Because Dijkstras algorithm is so well known, its general implementation is not described in this document. Apart from a standard version of Dijkstras algorithm, `iterativeDijkstra` ignores vertices that have a higher distance than said threshold, that means it does not check their neighbors for possibly better paths.

This method is private, since begründung

2.1.3 `projectSensors`

asdf

2.1.4 nearestNeighbor

asödlkfj

2.1.5 matrixDump

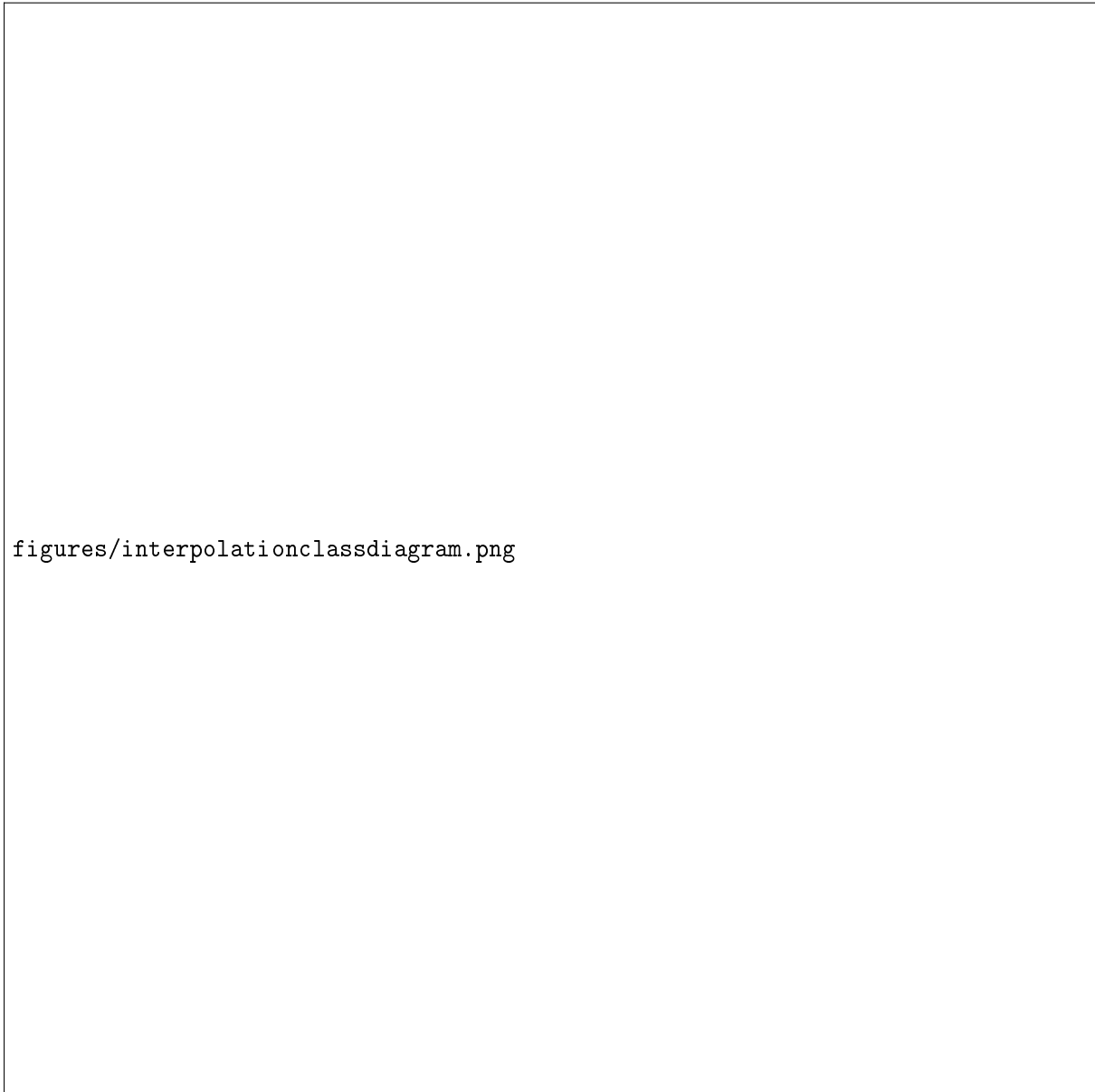
This method is implemented for testing and verifying purposes only. It receives a MatrixXd and a string. It then writes the content of the matrix into a file that is named with said string.

2.1.6 squared

This method is implemented for better readability only, thus it is declared private. It receives a double-value and returns it squared. In order to avoid efficiency losses through stack frames, it is declared inline.

3 Interpolation

The class *Interpolation* holds all ...



figures/interpolationclassdiagram.png

Figure 2: Interface of Interpolation

3.1 Member Objects

3.1.1 Weight Matrix

3.2 Member Functions

3.2.1 `createInterpolationMatrix`

asdf

3.2.2 `interpolateSignal`

The method *interpolateSignal* calculates the full

3.2.3 `clearInterpolationMatrix`

The method *clearInterpolationMatrix* resets the weight matrix, i.e. frees all memory allocated for the respective member object.

3.2.4 `getResult`

This method is implemented in case the calculated weight matrix is needed as an object itself. It returns a smart pointer to the weight matrix (see member objects for more details).

3.2.5 `linear, gaussian, square`

These methods are implemented to provide a minimal preset of functions to use during the calculation of the interpolation weight matrix (see method *createInterpolationMatrix* for more details).

4 Glossary

mesh MatrixXd dijkstra verweis, jarnik und prim