Software Engineering : Laplace Spectral Representation

Due on , January 14, 2015

Yohan Fougerolle

Pramita Winata
Dragutin

January 14, 2015

Contents

Introduction

This application is done as part of Software Engineering course project. The aim of this project is to show basic Laplace spectral representation and to implement C++ programming skill. It is developed under Qt IDE 5.3.0 using OpenGL and Eigen library. I will discuss the project implementation in two main sections; basic laplace representation and application development. Paper [1] is mainly used in the Laplace implementation. The development of the application is the extension of the existing project where improvement in the Graphical User Interface (GUI) is done and Laplacian representation is cooperated.

Section 1 - Basic Laplace representation

This part of the implementation is to satisfied the second and third objectives of the project where we need to implement section 2.1 of paper [**?**] by using the *Neighbourmesh* class given in *Meshloader.pro*. Inside *Meshloader.pro*, I can already find many extremely useful function that can be used to implement [**?**], especially functions in *Neighbourmesh.cpp*. Eigen library [3] also very useful on implementing this project, especially *SparseMatrix* class that I use to store the cotangent weight and the Laplace matrix.

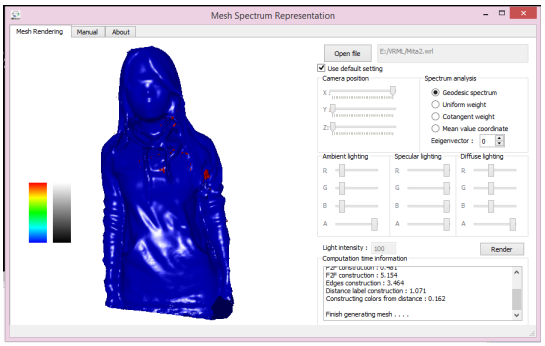

Figure 1: 3D Representation

The Figure1 below is a 3D representation of me. The *wrl* file is generated with kinect in 3D lab in IUT.

To start the implementation, I need to understand what is mesh representation. Mesh representation in 3D is a way to represent 3D objects in a computer. Construction of a data structure that can be recognize by a computer for it to manipulate, analyze 3d objects. There are many 3D representation has been developed until now, such as boundary representations, solid representations, image-based representation, and etc. Each of the representation has also subsections of representing 3D objects. For this project, we will deal with 3D boundary triangle mesh representation.

Listing 1: DrawTriangularMesh function in *LaplacianMesh.cpp*

```
void LaplacianMesh::DrawTriangularMesh()
{
    Vector3d vi, vj;
    for ( int i = 0; i <100; i ++ )
    {
        vi = vertices[i];
        for ( set <int> :: iterator it =  mesh.P2P_Neigh[i].begin();
                it !=  mesh.P2P_Neigh[i].end(); ++it)
        {
```

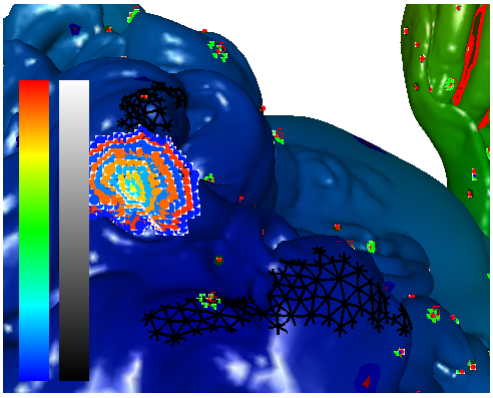

Figure 2: Triangular mesh Representation

```
            vj= vertices[*it]; //point to point

            //Draw mesh
            glLineWidth(3);
            glColor3f(0,0,0);
            glBegin(GL_LINES);
            glVertex3f(vi[0],vi[1],vi[2]);
            glVertex3f(vj[0],vj[1],vj[2]);
            glEnd();
        }
    }
}
```

In this function, Glut function from OpenCV is used and also notice that I only iterates 100 times since the purpose is just to illustrate triangular mesh .
Triangular meshes representation of a 3D point consists of a set of triangles built from the given point and its neighbor. We will have an 'umbrella' like shape for each of the 3D points. In the *Meshloader.pro*, data structure to read *wrl* file has already being built. The necessary and useful calculation such as getting the neighborhood vertices's and faces' has already being computed and nicely set.

The next step is to understand basic Laplace mesh representation. In Laplacian framework, each vertex is represented with consideration to its neighbors unlike Cartesian coordinates that only represent the spatial location. Laplacian meshes store the location of a vertex relative to its neighboring vertices. This is useful when the information of object's shape are important for example when deforming the object itself. With Laplacian mesh representation, we can use manipulate one vertex, as known as 'anchor' based on [**?**], to have the overall objects deformed in a correct way. In regular meshes, for example, we need to manipulate many vertices to get this done.

In Laplacian framework, we need to obtain a variable for each vertex that represent its location based on its immediate neighbours' vertices. This variable is called $differential$ or $\delta - coordinates$ of each vertex. In paper [1] the $differential$ or $\delta - coordinates$ of v_i is the difference between the absolute coordinates of v_i and the center of mass of its immediate neighbors in the mesh.

$$\delta_i = (\delta_i^{(x)}, \delta_i^{(y)}, \delta_i^{(z)}) = v_i - \frac{1}{d-i} \sum_{j \in N(i)} v_j,$$

where $N(i) = j|(i,j) \in E$ and $d_i = |N(i)|$ is the number of immediate neighbors of i (the degree or valence of i).
It is important to understand that each vertex will have weight correspond to each of its neighbor. In this project, I will implement three method to get the weight of the vertex. These method are uniform weight, cotangent weight, and mean-value-coordinate.

Uniform weight

In uniform weight method, the weight of each vertex depends on the size of its immediate neighbor. According to [1], the symmetric Laplace Matrix can be denoted by:

$$L_{s_{ij}} = \begin{cases} d_i & \text{if } i = j \\ -1 & \text{if } (i,j) \in \varepsilon \\ 0 & otherwise \end{cases}$$

In implementing this, I created new class *LaplaceMesh.cpp*, this class contains the calculation for this. Below are the header file of *LaplaceMesh.cpp*.

Listing 2: Header file for *LaplaceMesh* class

```
#ifndef LAPLACIANMESH_H
#define LAPLACIANMESH_H

#include <Eigen/Core>
#include <Eigen/Geometry>
#include<vector>
#include "NeighborMesh.h"

using namespace Eigen;

class LaplacianMesh
{
public:
    LaplacianMesh( NeighborMesh m );
    virtual~LaplacianMesh();

    //mesh detail
    NeighborMesh mesh;
    vector<Vector3d> vertices;
    vector<Vector3i> faces;
    int vertexNumber;

    //Laplace attributes
    MatrixXd laplaceMatrix;
    VectorXd myEigen;

    // Laplace functions
    void GetLaplacianUniformWeight();
    void GetLaplacianCotangentWeight();

    void CalculateEigenDecomposition();
```

```
    double CalculateCotangent(  Vector3d v1, Vector3d v2,
                                        Vector3d v3, double &A  );
    double CalculateCotangentWeight ( int vi_index, int vj_index );

    //Draw and render
    void DrawTriangularMesh();
    void BuildColorMap();
};

#endif // LAPLACIANMESH_H
```

Listing 3: Function to calculate Laplacian Matrix with uniform weight

```
void LaplacianMesh::GetLaplacianUniformWeight()
{
    laplaceMatrix.resize(vertexNumber, vertexNumber);

    //set everything to zero first
    laplaceMatrix.setZero(vertexNumber, vertexNumber);

    for ( int i = 0; i <vertexNumber; i ++ )
    {
       laplaceMatrix(i,i) = mesh.P2P_Neigh[i].size();//neighbour size here
       for ( set <int> :: iterator it =  mesh.P2P_Neigh[i].begin();
               it !=  mesh.P2P_Neigh[i].end(); ++it)
       {
           laplaceMatrix(i,*it) = -1;
           laplaceMatrix(*it, i) = -1;
        }
     }

    QString path1 = "laplacian.txt";
    SaveMatrixXd( laplaceMatrix, path1 );
    cout<<"Laplacian matrix can be found in "<<path1.toStdString()<<endl;
}
```

From here, I can construct the Laplace matrix from the Laplace definition above. The differential coordinates for each of vertex can be obtained by multiplying the Laplacian by the original coordinates. It represents the difference between the average position of its neighbors and the vertex's actual position, which corresponds to local detail.

In order to check whether the matrix and vector generated are fine, I created two function $SaveMatrix$ and $SaveVector$ in $Useful$ class. These function will store the corresponding matrix or vector to a text file in a given path.

Listing 4: Function to save matrix and vector to a text file

```
void SaveMatrixXd(MatrixXd &src, QString filepath)
{
    std::ofstream file(filepath.toStdString().c_str());
    if(file.is_open()){
        file<<"Matrix :\n"<<src<<"\n";
        file.close();return;
```

```
    }
    std::cerr<<"file cannot be opened";
}

void SaveVectorXd(VectorXd &src , QString filepath)
{
    std::ofstream file(filepath.toStdString().c_str());
    if(file.is_open()){
        file<<"Vector :\n"<<src<<"\n";
        file.close();return;
    }
    std::cerr<<"file cannot be opened";
}
```

I tested the the function with *Icosaedron.wrl* file. The matrix generation takes 2.452 seconds. The snipplet of Laplace matrix generated is shown in Figure 3.

```
 5 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
-1  6 -1 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
-1 -1  6  0 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0 -1  0  6 -1  0  0  0 -1  0 -1  0  0  0  0  0  0  0  0  0  0
 0 -1 -1 -1  6 -1  0  0  0  0 -1  0  0  0 -1  0  0  0  0  0  0
 0  0 -1  0 -1  6  0  0  0  0  0  0 -1  0 -1  0  0  0  0  0  0
 0  0  0  0  0  0  6 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0 -1  6 -1 -1 -1  0  0  0  0  0  0  0  0  0  0
 0  0  0 -1  0  0 -1 -1  6  0 -1  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0 -1  0  6 -1  0  0 -1 -1  0  0  0  0  0  0
 0  0  0 -1 -1  0  0 -1 -1 -1  6  0  0  0 -1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  6 -1 -1  0  0  0  0  0  0  0
 0  0  0  0  0 -1  0  0  0  0  0 -1  6 -1 -1  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0 -1  0 -1 -1  6 -1  0  0  0  0  0  0
 0  0  0  0 -1 -1  0  0  0 -1 -1  0 -1 -1  6  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  5 -1 -1  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  6 -1 -1 -1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1  6  0 -1 -1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  6 -1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1 -1 -1  6 -1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0 -1  6
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 3: Laplace matrix for *Icosaedron.wrl*

It is realized that

$$L_s * vertex_i = D\delta_i$$

where D is diagonal matrix with $D_i i = d_i$. Eigen decomposition then done in Laplace matrix. When eigen decomposition is done on the Laplace matrix, one eigenvector can be chosen to be our new set of weighted-vertices.

Listing 5: Function to calculate eigen decomposition of Laplacian Matrix

```
void LaplacianMesh::CalculateEigenDecomposition()
{
    if( laplaceMatrix.data() == NULL ){ return; }
    else
    {
        cout<<"Computing Eigen decomposition "<<endl;
        EigenSolver<MatrixXd> eigen;
        eigen.compute(laplaceMatrix);

        MatrixXd eigenvector = eigen.eigenvectors().real();
        QString path2 = "eigenvectors_matrix.txt";
        SaveMatrixXd( eigenvector, path2 );

        myEigen.resize( laplaceMatrix.rows() );
        myEigen = eigenvector.col(2);
        QString path3 = "eigenvector.txt";
        SaveVectorXd( myEigen, path3 );
    }
}
}
```

0.0394669	0.0700537	0.000151946	0.0389834	-0.112401	-0.0967413	-0.00267289	0.0010582	-0.126553	-0.000401903	0.00222432
0.0394669	0.0691081	0.00711076	0.0351237	-0.104116	-0.0928035	-0.00101131	0.00151221	-0.110617	-0.0100159	-0.00933163
0.0394669	0.0691081	0.00463018	0.0325333	-0.104116	-0.092841	-0.00396702	0.00140856	-0.110617	-0.00361671	-0.00666408
0.0394669	0.0667155	0.0157609	0.0296473	-0.0843647	-0.0831742	0.00572869	0.00360401	-0.0746479	-0.0198968	-0.0238954
0.0394669	0.0674994	0.0135673	0.0267028	-0.090662	-0.086312	-0.00229299	0.00580186	-0.0858107	-0.0133415	-0.0228301
0.0394669	0.0667155	0.0101959	0.0238359	-0.0843644	-0.0832472	-0.00988154	0.0030566	-0.0746479	-0.00762262	-0.0173172
0.0394669	0.0577666	0.0338032	0.0160179	-0.0240467	-0.0502663	0.0360739	0.0135383	0.0149722	-0.0304715	-0.0520914
0.0394669	0.0599088	0.0320832	0.0131109	-0.0368369	-0.0577947	0.0194058	0.0184388	-0.00111666	-0.0245298	-0.0550485
0.0394669	0.0628896	0.0248653	0.0231543	-0.0560572	-0.0685249	0.0184106	0.00769979	-0.0283433	-0.0271829	-0.0387552
0.0394669	0.0606242	0.0294799	0.00998934	-0.0414533	-0.0604094	-0.00151863	0.0196292	-0.0074164	-0.0198059	-0.0531181
0.0394669	0.0643967	0.0228701	0.0201772	-0.0667987	-0.0742281	0.00542142	0.0118021	-0.0451528	-0.0207338	-0.0395389
0.0394669	0.0577666	0.0218017	0.00348498	-0.0240456	-0.0503637	-0.0378831	0.0109448	0.0149722	-0.0152419	-0.0357995
0.0394669	0.0628896	0.0160529	0.0139517	-0.0560566	-0.0686197	-0.0214352	0.0063025	-0.0283433	-0.0116637	-0.0274886
0.0394669	0.0599088	0.0260108	0.00676956	-0.0368363	-0.0578452	-0.0221196	0.0169827	-0.00111666	-0.0166411	-0.0464705
0.0394669	0.0643967	0.0199202	0.0170967	-0.0667984	-0.0742603	-0.00920309	0.0112893	-0.0451528	-0.015486	-0.0356528
0.0394669	0.031329	0.059174	-0.0108714	0.0502658	0.0194104	0.0958858	0.0339239	0.0253107	-0.00664835	-0.0697337
0.0394669	0.0329456	0.0597469	-0.0140228	0.0476858	0.0156746	0.0880084	0.032079	0.029319	-0.0021493	-0.0779161
0.0394669	0.0375062	0.0552765	-0.00556104	0.0451698	0.00699033	0.0894397	0.0316743	0.0429868	-0.0153826	-0.0711557
0.0394669	0.0344116	0.0594315	-0.0177525	0.0436704	0.0117576	0.0676938	0.0289251	0.0313929	0.00188418	-0.084229
0.0394669	0.0403213	0.0549906	-0.00828677	0.0391756	2.17153e-005	0.0763747	0.0320346	0.0472128	-0.0115851	-0.0798419
0.0394669	0.0446429	0.0494026	0.00123082	0.0303826	-0.010422	0.0752532	0.0267879	0.0524813	-0.0238093	-0.0689667
0.0394669	0.0357017	0.0554318	-0.0249217	0.0389074	0.0079688	0.000259641	0.023664	0.0317637	0.00313696	-0.0837946
0.0394669	0.0430463	0.0517116	-0.0149394	0.0293447	-0.00795774	0.0179789	0.0303541	0.0459359	-0.00664107	-0.084366
0.0394669	0.0353707	0.0579956	-0.0214766	0.0402372	0.00896621	0.0369085	0.025758	0.0318102	0.00387046	-0.0864144
0.0394669	0.0492742	0.046779	-0.00462287	0.013217	-0.0240514	0.0316057	0.030675	0.0460531	-0.0159461	-0.0788471
0.0394669	0.0421383	0.0538453	-0.0115592	0.0330775	-0.00516788	0.051311	0.0314685	0.0470577	-0.00825761	-0.0844878
0.0394669	0.0515725	0.0421143	0.00858446	0.00631945	-0.0302197	0.0562041	0.0203025	0.0438062	-0.0292429	-0.062486
0.0394669	0.0542106	0.040768	0.00577508	-0.0060028	-0.0386481	0.038088	0.0246687	0.0335885	-0.0240794	-0.0677258
0.0394669	0.0475617	0.0485292	-0.00149657	0.0206287	-0.0187068	0.058393	0.0294662	0.0503129	-0.0194712	-0.0762766
0.0394669	0.031329	0.0381109	-0.0328672	0.0502687	0.0194566	-0.0952149	0.0272225	0.0253107	-0.0195421	-0.0397218
0.0394669	0.0375062	0.0356071	-0.0261014	0.0451725	0.00699262	-0.0888963	0.0254205	0.0429868	-0.0200613	-0.0427028

Figure 4: Computed eigenvectors

I utilize the *EigenSolver* class from Eigen library in C++. The implementation is very straightforward. I can just instantiate the solver with specifying the datatype of Matrix to be solved, in this case $MatrixXd$, and call the *compute* function. Eigenvectors and eigenvalues of the matrix will be stored in a $MatrixXd$. Figure 4 shows the snipplet of the eigenvectors' matrix. This is stored in text file for later references.
From here, I get one real eigenvector. This eigenvector will represent new weighted value of the vertices. Figure 5 shows the snipplet of the eigenvectors' matrix. This is stored in text file for later references.

After obtaining this new set of values, I build a colormap to illustrate the eigen decomposition of the mesh.

```
-0.0967413
-0.0928035
-0.092841
-0.0831742
-0.086312
-0.0832472
-0.0502663
-0.0577947
-0.0685249
-0.0604094
-0.0742281
-0.0503637
-0.0686197
-0.0578452
-0.0742603
0.0194104
0.0156746
0.00699033
0.0117576
2.17153e-005
-0.010422
0.0079688
-0.00795774
```

Figure 5: Sniplet of eigen vector chosen

Listing 6: Function to get the new colormap based on the eigen vector of Laplace matrix

```
void LaplacianMesh::BuildColorMap(  NeighborMesh *m )
{
    double eMin =  myEigen.coeff( 0 ) ;
    double eMax =  myEigen.coeff( myEigen.size()-1);

    m->colors.clear();

   //translate values into [0,1], then convert into color, and store
   for(int i=0; i< myEigen.size(); i++)
    {
      Vector3d d2C = DoubleToColor ( (myEigen.coeff(i) - eMin )/(eMax-eMin) );
       m->colors.push_back( d2C );

    }

   QString path2 = "color2.txt";
   SaveVector3d( (m->colors) , path2);
}
```

Generating the color map takes s
Total time to finish the whole process is shown in Figure .
It is also worth notes that the size of the *.wrl* file used is playing a huge role in this implementation, especially the size of the vertices inside it. I have encountered many problem in regards of this. First, the problem when instantiating the Laplace matrix using $MatrixXd$. Laplace matrix needs to be instantiated as nxn matrix, with n is the number of vertices in the mesh. If this variable gets too big, Qt will not be able to handle it. I am running the Qt with MinGW 32 bit, the maximum memory size can be allocated is 4 GB. Mesh with total vertices 100.000, will need to instantiate 100.000 x 100.000 size of Matrix, which is far greater than the limit of the application.
There are three solution for this :

1. Memory increment
   Increasing the memory by change the Qt compiler to get be work on MinGw 64 bit. These will also require changing the Qt Desktop application to be 64 bit version and setting everything up again.

2. Code modification

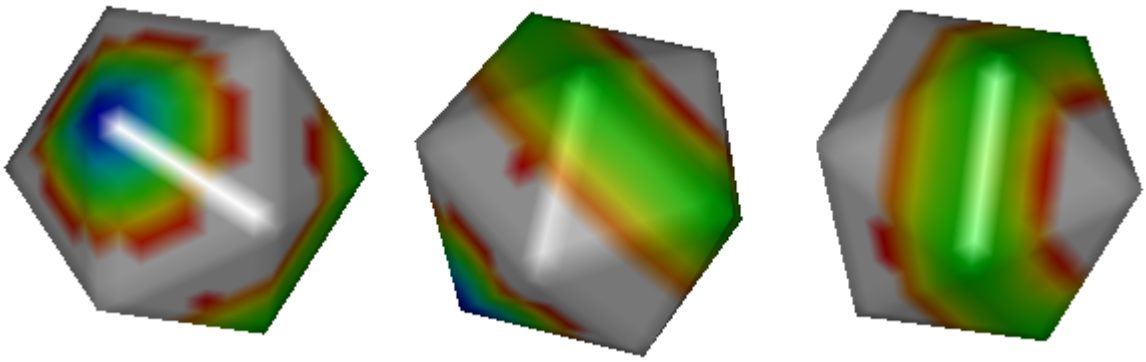

Figure 6: New mesh colormap based on the chosen 4_th column eigenvector

```
READ FILE
Extension=WRL
Reading IV/VRML
Loading time :0.033 s
P2P construction :0.007 s
P2F construction :0.004 s
F2F construction :0.042 s
Egdes construction :0.023 s
Laplacian matrix can be found in C:/Users/User/Documents/Master_VIBOT/Semester1/laplacian.txt
Calculating laplacian matrix: 2.498 s
Computing Eigen decomposition
Calculating eigen decomposition: 165.906 s
Building eigen color map: 0.011 s
```

Figure 7: Computation time

Changing the way Laplace matrix instantiated could be a solution. In Eigen library, *SparseMatrix* class can be used to store Laplacian since Laplace matrix is extremely sparse (most of the value is zero). It can be instantiated with a huge number without taking much memory. However this solution has one big drawback, *Sparse* matrix class in Eigen is not supported in *EigenSolver*. External library like ARPACK needs to be cooperated in Qt.

3. Data modification
   The easiest and more practical solution is to change the data having a reasonable amount of vertices. This what I have chosen to be the solution for this project. Meshes like dolphin, Icoheadral, or mannequin can be computed in 32 bit compiler. This is not the best solution but it is an acceptable solution in my opinion.

Cotangent weight

The weight use previously, is improved by [5] We can also generate the Laplacian matrix by using the cotangent weight instead of its uniform weight .

$$\delta_i^c = \frac{1}{|\Omega_i|} \sum_{j\in} \frac{1}{2}(cot\alpha_{ij} + cot\beta_i j)(v_i - v_j),$$

where $|\Omega_i|$ is the size of the Voronoi cell of i and α_{ij}, β_{ij} denote the two angles opposite of edge (i, j)
Thus I get, the cotangent weight:

$$\omega_i j = \frac{1}{2}(cot\alpha_{ij} + cot\beta_i j)$$

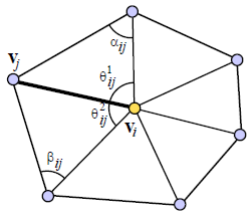

Figure 8: Cotangent weight

After computing$\omega_i j$, we can get the Laplacian matrix, L ,that comprises :

- $L_(ij) = \omega_i j$, if i and j are neighbors
- $L_(ij) = 1$, if $i = j$ are neighbors
- 0, everywhere else

$$L_{s_{ij}} = \begin{cases} 1 & \text{if } i = j \\ \omega_i j & \text{if } (i,j) \in \varepsilon \\ 0 & otherwise \end{cases}$$

To compute the cotangent, I use the equation illustrated in [5].
If we have a triangle with vertices $vi; vj; vk$, the triangle area A_{ijk} is defined by [6]:

$$A_{ijk} = \sqrt[2]{r(r - l_{ij})(r - l_{jk})(r - l_{ki})}$$

where r is the semiperimeter

$$r = \frac{1}{2}(l_{ij} + l_{jk} + l_{ki}).$$

The law of cosines:

$$cos\alpha_{ij} = \frac{-l_{ij}^2 + l_{jk}^2 + l_{ki}^2}{2 l_{jk} l_{ki}}.$$

For sine:

$$sin\alpha_{ij} = \frac{2A_{ijk}}{l_{jk} l_{ki}}$$

I can get the cotangent by:

$$cot\alpha_{ij} == \frac{cos\alpha_{ij}}{in\alpha_{ij}}$$

Listing 7: Function to calculate the cotangent of a triangle

```
double LaplacianMesh::CalculateCotangent(  Vector3d vi, Vector3d vj,
                                           Vector3d vk , double &A )
{
    //Based on "Cotangent Laplacian for Images as Surfaces paper "

    //get length of each edge
    double lij,ljk,lki;
    double r, cos_alphaij, sin_alphaij, cot_alphaij;

    lij = (vi - vj).norm();
```

```
    ljk = (vj - vk).norm();
    lki = (vk - vi).norm();

    //r is semiparameter
    r = 0.5 * ( lij + ljk + lki );

    //triangle area
    A = sqrt( r*(r-lij)*(r-ljk)*(r-lki) );

    //derive the cosine and sine
    // lij^2 =  ljk^2 + lki^2 - 2*ljk*lki*cos_alphaij
    cos_alphaij = ( ljk*ljk + lki*lki - lij*lki ) / ( 2*ljk*lki );

    sin_alphaij = 2*A / ( ljk*lki );

    cot_alphaij = cos_alphaij / sin_alphaij;

    return cot_alphaij;
}
```

Before calculating the cotangent weight, I need to find the correct triangle to be passed in to *CalculateCotangent*, in another case find the other vertex that formed the face. I iterates the faces of the vertex, and find the faces that contains the edge we are having. Code below shows the function *CalculateCotangentWeight* that compute this.

Listing 8: Function to calculate the cotangent weight of the vertices

```
double LaplacianMesh::CalculateCotangentWeight ( int vi_index, int vj_index )
{
    Vector3i f;
    double cotangent_weight = 0, area;
    int v1_index, v2_index ,v3_index, vk_index ;
    Vector3d vi, vj, vk;

    double cot_alphaij = 0;
    //Get neighbourhood faces
    for ( set <int> :: iterator it2 =  mesh.P2F_Neigh[vi_index].begin();
          it2 !=  mesh.P2F_Neigh[vi_index].end(); ++it2)
    {

        //cout<<"Face index "<<*it2<<endl;
        f = faces[*it2];

        // the face that contains the point vi, vj is correct face
        if( mesh.FaceHasVertex( *it2 , vj_index ) )
        {
            //get vk
            v1_index = mesh.FaceIndex(*it2,0);
            v2_index = mesh.FaceIndex(*it2,1);
            v3_index = mesh.FaceIndex(*it2,2);

            if ( ( v1_index == vi_index  && v2_index == vj_index ) ||
                 ( v1_index == vj_index  && v2_index == vi_index ) )
                vk_index = v3_index;
```

```
            else if( ( v2_index == vi_index  && v3_index == vj_index )||
                ( v2_index == vj_index  && v3_index == vi_index ) )
                vk_index = v1_index;
            else
                vk_index = v2_index;

            vi = vertices[vi_index];
            vj = vertices[vj_index];
            vk = vertices[vk_index];

           cot_alphaij = CalculateCotangent( vi, vj, vk, area);
             //initiate the weight
           if( cotangent_weight == 0.0 ){ cotangent_weight = 0.5; }

           cotangent_weight = cotangent_weight * cot_alphaij;
        }
    }
    return cotangent_weight;
}
```

After having the cotangent calculation, we can build the Laplace Matrix using the rule given previously.

Listing 9: Function to construct Laplace matrix with cotangent weight

```
void LaplacianMesh::GetLaplacianCotangentWeight()
{
    double cotangent_weight = 0;
    Vector3d vi, vj;

    laplaceMatrix.resize(vertexNumber, vertexNumber);

    //set everything to zero first
    laplaceMatrix.setZero(vertexNumber, vertexNumber);

    for ( int i = 0; i <vertexNumber; i ++ )
    {
        laplaceMatrix(i,i) = 1;
        vi = vertices[i];
       //Get neighbourhood points
       for ( set <int> :: iterator it =  mesh.P2P_Neigh[i].begin();
               it !=  mesh.P2P_Neigh[i].end(); ++it)
       {
            vj = vertices[*it]; //point to point
            cotangent_weight = CalculateCotangentWeight( i, *it );
            laplaceMatrix(i,*it) = cotangent_weight;
            laplaceMatrix(*it, i) = cotangent_weight;
       }
    }

    cout<<"Write Laplacian matrix to text file."<<endl;
    QString path1 = "laplacian_cot.txt";
    SaveMatrixXd( laplaceMatrix, path1 );
}
```

Here is a sniplet of the Laplace matrix.

1	0.166665	0.166666	0	0	0	0	0	0
0.166665	1	0.166666	0.166668	0.166667	0	0	0	0
0.166666	0.166666	1	0	0.166667	0.166665	0	0	0
0	0.166668	0	1	0.166668	0	0	0	0.166665
0	0.166667	0.166667	0.166668	1	0.166665	0	0	0
0	0	0.166665	0	0.166665	1	0	0	0
0	0	0	0	0	0	1	0.166664	0.166669
0	0	0	0	0	0	0.166664	1	0.166667
0	0	0	0.166665	0	0	0.166669	0.166667	1
0	0	0	0	0	0	0	0.166668	0
0	0	0	0.166666	0.166668	0	0	0.166665	0.166664
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.166668	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0.166665	0.166668	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Figure 9: Sniplet of Laplace matrix using Cotangent weight

The routine goes the same with previous method, we need to perform eigen decomposition and build the color map using the chosen eigenvector. I am using the same function built previously. The result of the color map is shown in Figure **??**

-0.0309129	-0.0512952	-0.0120381	0.0129857	-0.0716195	-0.0102757	-0.00249393
-0.0370046	-0.0610387	-0.00797246	0.0198671	-0.0815361	-0.0210688	-0.0120339
-0.0370047	-0.0631856	-0.00833883	0.0110476	-0.0826558	-0.0181196	0.00794205
-0.0391062	-0.0628693	-0.00055802	0.0257822	-0.0766132	-0.0306744	-0.0227387
-0.038729	-0.0654842	-0.000336618	0.0159534	-0.0798576	-0.0272072	-0.00127858
-0.0391061	-0.0676669	-0.00137661	0.00607177	-0.0789899	-0.0241669	0.0206186
-0.0402844	-0.0575045	0.0165624	0.0354479	-0.0470887	-0.0404353	-0.0396996
-0.0402704	-0.0622851	0.0172294	0.0256597	-0.055609	-0.0357255	-0.0213561
-0.0400172	-0.0613484	0.00792579	0.0310455	-0.0641659	-0.0374817	-0.0324356
-0.0402564	-0.0656009	0.0171945	0.0152537	-0.05925	-0.0318077	-0.000381479
-0.0397924	-0.0651001	0.00834271	0.0210091	-0.0703983	-0.0333494	-0.0117023
-0.0402844	-0.0676627	0.0148291	-0.00628491	-0.0512798	-0.027766	0.0413477
-0.0400169	-0.0688922	0.00663843	5.15278e-005	-0.0676258	-0.0275684	0.0323744
-0.0402703	-0.0674309	0.0163513	0.00451956	-0.0578084	-0.0290682	0.021209
-0.0397924	-0.0676282	0.00791137	0.010623	-0.0715831	-0.0299468	0.0105213
-0.0309133	-0.0250341	0.0340225	0.0340667	0.0158061	-0.0141787	-0.0292603
-0.0370053	-0.0339768	0.0428922	0.0344952	0.0137347	-0.0100266	-0.0309482
-0.0370053	-0.0360113	0.0359251	0.0399584	0.00697844	-0.0250205	-0.039134
-0.0391067	-0.0400408	0.0470783	0.0282488	0.00930866	-0.00327559	-0.0252039
-0.0387295	-0.0429164	0.0392466	0.03407	-0.000741757	-0.0198021	-0.033776
-0.0391066	-0.0445874	0.0315078	0.0404582	-0.00893456	-0.0336958	-0.0434595
-0.0402845	-0.0478781	0.0495297	0.00959686	0.00400584	0.00634649	-0.00325528
-0.0402705	-0.0525502	0.042451	0.0164648	-0.010591	-0.00948676	-0.00965701

Figure 10: Sniplet of eigenvectors matrix for the cotangent weight method

Total time to finish the whole process is shown in Figure .

Mean value coordinate

Mean value coordinate is introduce by [8] to tackle the problem when the cotangent weights may be negative and in the case where the angle of the vertices is very big. [8] introduce this weight computation to solve this problem.

$$\delta_i^j = \frac{\tan((\theta_{ij}^1)/2) + \tan((\theta_{ij}^2)/2))}{||v_i - v_j||},$$

where θ is illustrated in Figure **??**.

```
-0.0716195
-0.0815361
-0.0826558
-0.0766132
-0.0798576
-0.0789899
-0.0470887
-0.055609
-0.0641659
-0.05925
-0.0703983
-0.0512798
-0.0676258
-0.0578084
-0.0715831
0.0158061
0.0137347
0.00697844
0.00930866
-0.000741757
-0.00893456
0.00400584
-0.010591
```

Figure 11: Sniplet of the chosen eigenvector for the cotangent weight method

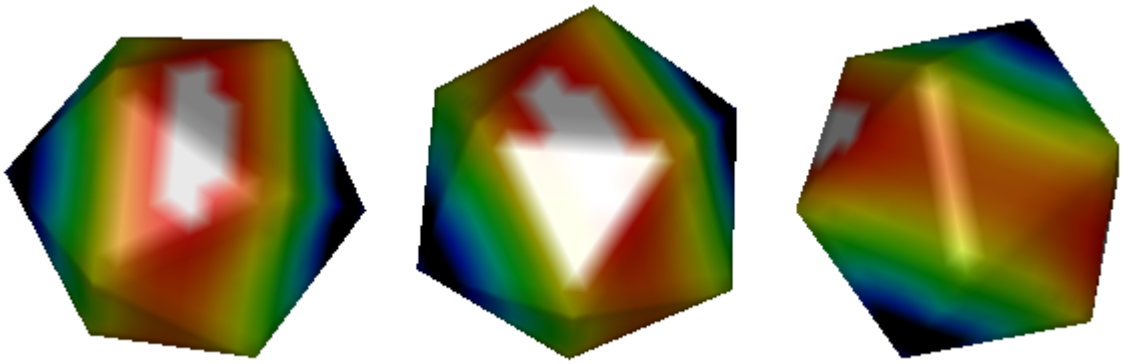

Figure 12: New mesh colormap based on the chosen 4_th column eigenvector for cotangent weight method

```
READ FILE
Extension=WRL
Reading IV/VRML
Loading time :0.019 s
P2P construction :0.004 s
P2F construction :0.002 s
F2F construction :0.017 s
Egdes construction :0.011 s
Write Laplacian matrix to text file.
Calculating laplacian matrix: 2.409 s
Computing Eigen decomposition
Calculating eigen decomposition: 167.785 s
Building eigen color map: 0.014 s
```

Figure 13: Computation time

The Laplacian matrix, L, comprises:

- $L_(ij) = \omega_i j$, if i and j are neighbors
- $L_(ij) = 1$, if $i = j$ are neighbors
- 0, everywhere else

$$L_{s_{ij}} = \begin{cases} 1, i = j \\ w_{ij}, (i,j) \in \varepsilon \\ 0, otherwise) \end{cases}$$

To compute the cotangent, I use the same method as in calculation cotangent computation illustrated in [5].

Listing 10: Function to calculate the tangent for mean value coordinates

```
double LaplacianMesh::CalculateTangent(  Vector3d vi, Vector3d vj, Vector3d vk )
{
    //get length of each edge
    double lij,ljk,lki,A;
    double r, cos_alphaij, sin_alphaij, tan_alphaij;

    lij = (vi - vj).norm();
    ljk = (vj - vk).norm();
    lki = (vk - vi).norm();

    //r is semiparameter
    r = 0.5 * ( lij + ljk + lki );

    //triangle area
    A = sqrt( r*(r-lij)*(r-ljk)*(r-lki) );

    //derive the cosine and sine
    // lij^2 =  ljk^2 + lki^2 - 2*ljk*lki*cos_alphaij
    cos_alphaij = ( ljk*ljk + lki*lki - lij*lki ) / ( 2*ljk*lki );

    sin_alphaij = 2*A / ( ljk*lki );

    tan_alphaij = cos_alphaij / sin_alphaij;

    return tan_alphaij;


}
```

Same steps I did in computing the cotangent weight, I need to find the correct triangle to be passed in to *CalculateTangent*, in another case find the other vertex that formed the face. I iterates the faces of the vertex, and find the faces that contains the edge we are having. Code below shows the function *CalculateMeanValueWeight* that compute this.

Listing 11: Function to calculate the mean value coordinate of the vertices

```
double LaplacianMesh::CalculateMeanValueWeight( int vi_index, int vj_index )
{
    Vector3i f;
    int v1_index, v2_index ,v3_index, vk_index ;
    Vector3d vi, vj, vk;

    double tan_thetaij = 0, sumAngle = 0;

    //Get neighbourhood faces
```

```
    for ( set <int> :: iterator it2 =  mesh.P2F_Neigh[vi_index].begin();
              it2 !=  mesh.P2F_Neigh[vi_index].end(); ++it2)
    {

        //cout<<"Face index "<<*it2<<endl;
        f = faces[*it2];

        // the face that contains the point vi, vj is correct face
        if( mesh.FaceHasVertex( *it2 , vj_index ) )
        {
            //get vk
            v1_index = mesh.FaceIndex(*it2,0);
            v2_index = mesh.FaceIndex(*it2,1);
            v3_index = mesh.FaceIndex(*it2,2);

            if ( ( v1_index == vi_index  && v2_index == vj_index ) ||
                 ( v1_index == vj_index  && v2_index == vi_index ) )
                vk_index = v3_index;
            else if( ( v2_index == vi_index  && v3_index == vj_index )||
                ( v2_index == vj_index  && v3_index == vi_index ) )
                vk_index = v1_index;
            else
                vk_index = v2_index;

            vi = vertices[vi_index];
            vj = vertices[vj_index];
            vk = vertices[vk_index];

           tan_thetaij = CalculateTangent( vj, vk, vi);

           sumAngle = sumAngle + tan_thetaij/2;
        }
    }

    return ( 1 / ( (vi-vj).norm() ) ) * sumAngle;
}
```

After having the cotangent calculation, we can build the Laplace Matrix using the rule given previously.

Listing 12: Function to construct Laplace matrix with mean value coordinates

```
void LaplacianMesh::GetLaplacianMeanValue()
{
    double cotangent_weight = 0;
    Vector3d vi, vj;

    laplaceMatrix.resize(vertexNumber, vertexNumber);

    //set everything to zero first
    laplaceMatrix.setZero(vertexNumber, vertexNumber);

    for ( int i = 0; i <vertexNumber; i ++ )
    {
        laplaceMatrix(i,i) = 1;
```

```
        vi = vertices[i];
        //Get neighbourhood points
        for ( set <int> :: iterator it =  mesh.P2P_Neigh[i].begin(); it !=  mesh.P2P_Neigh[i].end
        {
            vj = vertices[*it]; //point to point
            cotangent_weight = CalculateMeanValueWeight( i, *it );
            laplaceMatrix(i,*it) = cotangent_weight;
            laplaceMatrix(*it, i) = cotangent_weight;
        }
    }

    cout<<"Write Laplacian matrix to text file."<<endl;
    QString path1 = "C:/Users/User/Documents/Master_VIBOT/Semester1/laplacianMeanValue.txt";
    SaveMatrixXd( laplaceMatrix, path1 );
}
```

Here is a sniplet of the Laplace matrix.

1	3.73675	3.73672	0	0	0	0	0	0	0	0
3.73675	1	3.73671	3.7367	3.73671	0	0	0	0	0	0
3.73672	3.73671	1	0	3.73668	3.7367	0	0	0	0	0
0	3.7367	0	1	3.73668	0	0	0	3.73668	0	3.73669
0	3.73671	3.73668	3.73668	1	3.73668	0	0	0	0	3.7367
0	0	3.7367	0	3.73668	1	0	0	0	0	0
0	0	0	0	0	0	1	3.73668	3.73668	0	0
0	0	0	0	0	0	3.73668	1	3.73667	3.73669	3.7367
0	0	0	3.73668	0	0	3.73668	3.73667	1	0	3.73666
0	0	0	0	0	0	0	3.73669	0	1	3.73669
0	0	0	3.73669	3.7367	0	0	3.7367	3.73666	3.73669	1
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	3.73672	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.73669	0
0	0	0	0	3.73667	3.7367	0	0	0	3.73666	3.73672
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure 14: Sniplet of Laplace matrix using mean value coordinates

The routine goes the same with previous method, we need to perform eigen decomposition and build the color map using the chosen eigenvector. I am using the same function built previously. The result of the color map is shown in Figure **??**
Total time to finish the whole process is shown in Figure .

-0.0309149	0.0506827	0.0193387	0.00152981	0.00510778	0.0685501
-0.0370065	0.0595778	0.0233294	0.00951493	0.000660701	0.0749531
-0.0370065	0.0624545	0.0161689	0.00472125	0.00959262	0.0751374
-0.0391073	0.0604823	0.0244113	0.0190711	-0.00424983	0.0664335
-0.0387303	0.063873	0.0161024	0.0142798	0.00617533	0.0677169
-0.0391075	0.0669115	0.00840865	0.00835799	0.0135137	0.0668216
-0.0402848	0.0533552	0.0231817	0.0381245	-0.00780321	0.031268
-0.040271	0.0588712	0.0147498	0.0339314	0.00404478	0.0358036
-0.0400177	0.0580435	0.0242441	0.0288699	-0.00736911	0.050772
-0.040257	0.0630211	0.00611771	0.0286828	0.0127088	0.0374252
-0.0397933	0.0625709	0.0156983	0.024239	0.00405485	0.0540183
-0.040285	0.0669674	-0.0107008	0.0154415	0.0176964	0.0319374
-0.0400179	0.0681529	-0.000919498	0.0120239	0.0163478	0.0513315
-0.0402709	0.0657664	-0.00241379	0.022441	0.0173816	0.0361547
-0.0397933	0.0659586	0.00726579	0.0185937	0.012137	0.0542097
-0.0309131	0.0201415	0.0114329	0.0490754	0.009918	-0.0279211
-0.0370051	0.028486	0.00737611	0.0576039	0.0215519	-0.0329605
-0.0370053	0.0304715	0.0157951	0.054828	0.00553094	-0.0227193
-0.0391068	0.0347845	9.25924e-005	0.0583755	0.0317057	-0.0340875
-0.0387295	0.037628	0.0093088	0.0551395	0.0164009	-0.0211145

Figure 15: Sniplet of eigenvectors matrix for the mean value weight method

```
0.00510778
0.000660701
0.00959262
-0.00424983
0.00617533
0.0135137
-0.00780321
0.00404478
-0.00736911
0.0127088
0.00405485
0.0176964
0.0163478
0.0173816
0.012137
0.009918
0.0215519
0.00553094
0.0317057
0.0164009
-0.000525932
0.0403795
0.0316253
```

Figure 16: Sniplet of the chosen eigenvector for the mean value coordinate method

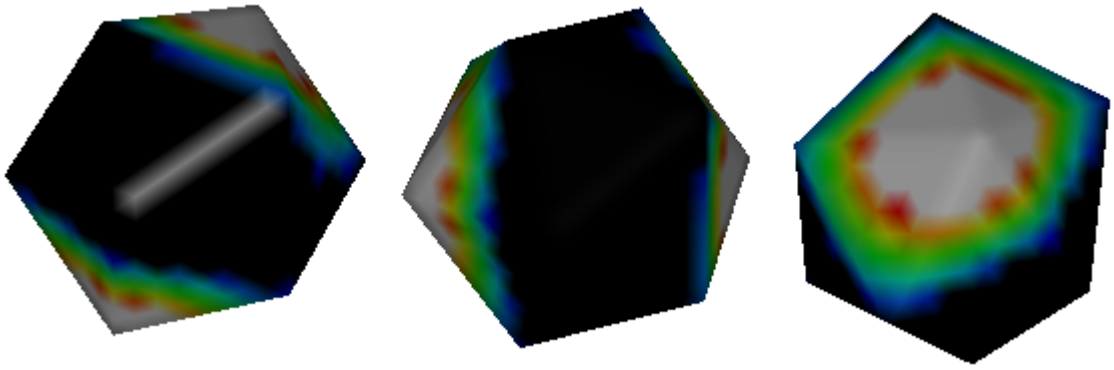

Figure 17: New mesh colormap based on the chosen 4_th column eigenvector for mean value coordinate method

Section 2 - Graphical User Interface

This chapter will discuss on the development of the Graphical User Interface (GUI) to display what I have calculated previously. The implementation started with the *MeshLoader* bundle given beforehand. *MeshLoader.pro* is developed in *QtSDK*. This bundle has already have many implementations that is

```
READ FILE
Extension=WRL
Reading IV/VRML
Loading time :0.044 s
P2P construction :0.008 s
P2F construction :0.006 s
F2F construction :0.056 s
Egdes construction :0.035 s
Write Laplacian matrix to text file.
Calculating laplacian matrix: 2.663 s
Computing Eigen decomposition
Calculating eigen decomposition: 160.61 s
Building eigen color map0.009 s
Geodesic construction :0 s
```

Figure 18: Computation time

extremely useful in developing this GUI. However, the $Widget$ is still missing in this bundle. I will extend the implementation of this bundle to include GUI, developed in $QtWidget$ and cooperated Laplacian operators. First, I need to know what exactly classes in $MeshLoader$ do and try to use it in the new Widget application. There are 4 classes implemented. Below are a short description on what each of the classes do.

1. $Mesh$
   This class provides the base structure of the Mesh, which includes the vertices, faces, colors, textures, face normals, and vertex normals. The mesh stores each attribute in a *vector* of $Vector3d$. This implementation use one of $C++$ Standard Template Library (STL), *vector* class. It is z resizeable array that stores the datatype specified. The datatype used is from vector family of *Eigen* library. Eigen library gives a convinient for matrix an vector manupulation which what we are dealing with. It also provides eigen solver that is very useful.

2. $NeighborMesh$
   $NeighborMesh$ is a class inherited from $Mesh$ class. It has all the traits from $Mesh$, which means it will have all the vertices and faces stored, and also have additional traits defined in this class. Here, the surrounding environment of a vertex is computed. The neighborhood points , faces, edges, and distance are computed. I am mainly utilizing this class when computing the Laplacian.

3. $Scene$
   This class provides functions to draw in $OpenGL$ (camera position and lighting) and handling user interaction (like keyboard or ball tracking). I will modified this class to support $QWidget$ for $OpenGL$ which is$QGLWidget$.

4. $Useful$ This class provides some additional function that is used when calculating, rendering, or trying to corporate with C++.

The Widget start with making the main window for the Application. The main file is very simple as, I only wants to instantiate $MainWindow$ class and show it out. The design of the window will be done in *ui* file.

Listing 13: $Main.cpp$

```
/*************************************************************************
                                main.cpp
                             -------------------
    update               : 2014 - 01 - 12
    copyright            : (C) 2013 by Pramita Winata
    email                : pramita.winata@gmail.com
 *                                                                       *
```

```
 *   Description:
 *   This project is done as part of Software Engineering course.
 *   The aim of this project is to show basic Laplace specral representation
 *                                                                          *
 ****************************************************************************/

#include <QApplication>
#include <iostream>
#include "mainwindow.h"

using namespace std;
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow mainwindow;
    mainwindow.show();
    mainwindow.setWindowTitle("Mesh Spectrum Representation");
    return a.exec();
}
```

MainWindow class acts as a base in my GUI. The button pushed, slider changed, check boxed checked will be registered in this class as slots. I can specified the post-processing of the action in these handler functions. The design in Qt is an xml-based UI, but Qt prohibits to directly changed the xml file, but instead use its designer interface. I utilized check boxes, radio buttons, buttons, horizontal sliders, and spin boxes.

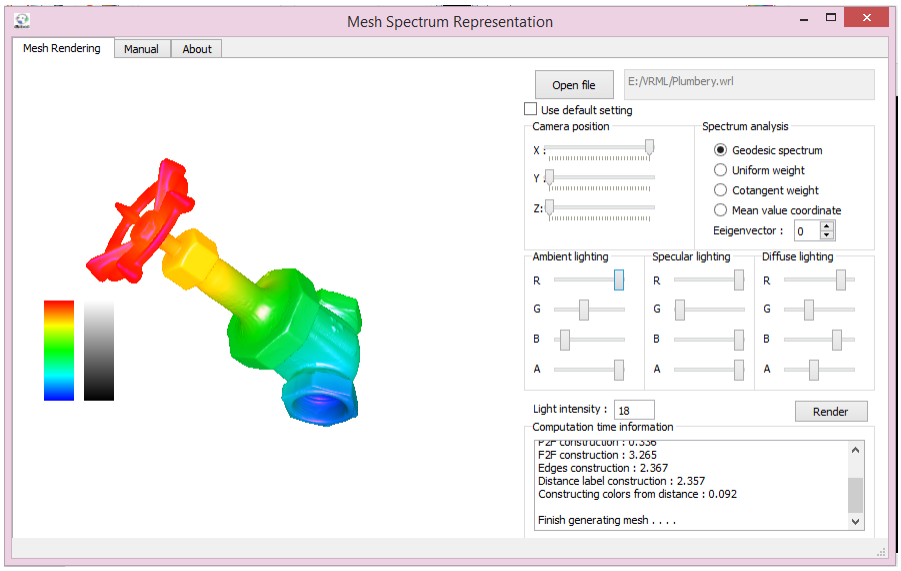

Figure 19: Main window

As it shown in the main window, I have added the camera and lighting to be configurable from the interfaces. Checking the checkbox "Use default setting" will set the parameter to be the default setting of the application. Text box to show the computation time to generate the mesh also is added to give more information of the mesh rendering.

Listing 14: Header file for *MainWindow* class

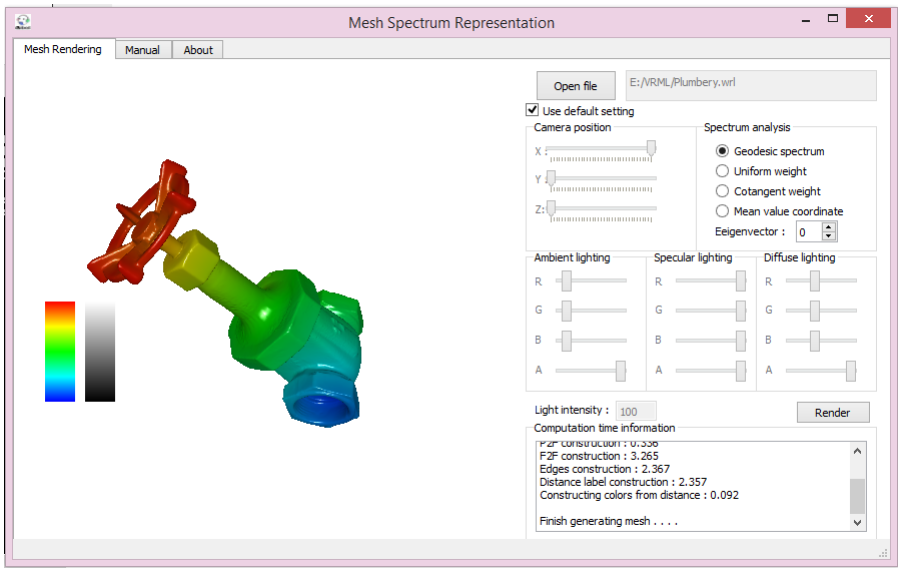

Figure 20: Default setting example

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include<QFutureWatcher>
#include <QMainWindow>
#include "myGLWidget.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Vector3d GetCameraPosition();
    void UpdateCamera();
    void UpdateLighting();
    void UpdateLightingValue();

    GLfloat* GetAmbient();
    GLfloat* GetDiffuse();
    GLfloat* GetSpecular();
    GLfloat* GetIntensity();

private slots:

    void on_searchButton_clicked();
```

```
    void on_pushButton_clicked();
    void on_checkBox_toggled(bool checked);
    void on_xHorizontalSlider_valueChanged(int value);
    void on_yHorizontalSlider_2_valueChanged(int value);
    void on_zHorizontalSlider_valueChanged(int value);

    void on_AmRHorizontalSlider_valueChanged(int value);

    void on_AmAHorizontalSlider_valueChanged(int value);

    void on_AmGHorizontalSlider_valueChanged(int value);

    void on_AmBHorizontalSlider_valueChanged(int value);

    void on_IntensitylineEdit_textChanged(const QString &arg1);

    void on_SpecRHorizontalSlider_valueChanged(int value);

    void on_SpecGHorizontalSlider_valueChanged(int value);

    void on_DiffRHorizontalSlider_valueChanged(int value);

    void on_DiffGHorizontalSlider_valueChanged(int value);

    void on_DiffBHorizontalSlider_valueChanged(int value);

    void on_DiffAHorizontalSlider_valueChanged(int value);

    void on_SpecAHorizontalSlider_valueChanged(int value);

    void on_SpecBHorizontalSlider_valueChanged(int value);

private:
    Ui::MainWindow *ui;
 };

#endif // MAINWINDOW_H
```

One of the tricky part of the implementation is to cooperate OpenGL in QWidget. Having a very good reference from the class, I made *myGLWidget* as the class on the widget to render the mesh. As shown in Figure 19, area where the mesh is drawn is genereted from *myGLWidget* class.
Now getting on the main problem which is rendering the 3D mesh to this area. As implemented in the bundle given, GLUT is used. I need to understand the Glut in C++ for this. GLUT is one of the OpenGL toolkit to implement GUI for OpenGL program. So, GLUT initiation and implementation will be done in *myGLWidget* class. Figure

Listing 15: Header file for *myGLWidget* class

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QtOpenGL>
#include <QGLWidget>
```

```
#include <GL/glu.h>
#include <string>
#include "NeighborMesh.h"
#include "Stopwatch.h"
#include "Constante.h"
#include "scenegl.h"
#include <iostream>

class myGLWidget : public QGLWidget
{
     Q_OBJECT

public:

    //explicit constructor
    myGLWidget(QWidget *parent = 0);

    // message string to show the computation time in the widget
    string message;

    //to store the camera position and lighting
    SceneGL sceneGL;

    //initialization
    void initializeGL();

    // functions for Mesh rendering
    void InitGLForRender();
    void renderMesh( string filename, int option,
                     int eigenCol,
                         double isDefaultPos, Vector3d camPos  );
    void settingUpMeshBasic( double isDefaultPos, Vector3d camPos  );
    void renderToLaplaceRepresentation( int weightOption, int eigenCol );
    void renderToGeodesicRepresentation();

    //function called when the widget is resized
    void resizeGL (int width, int height );

    //function called when the widget should be drawn or updated
    void paintGL();

    //update camera position
    void UpdateCamera( bool isDefaultPos, Vector3d camPos );
    //update lighting
    void UpdateLighting();

    //function handler for user interaction
    void keyPressEvent( QKeyEvent *keyEvent );
    void mousePressEvent(QMouseEvent *event);
    void mouseMoveEvent(QMouseEvent * mouse);
    void mouseReleaseEvent(QMouseEvent* mouse);
    void Mouse (QMouseEvent* event);
    void Motion( int x ,int y);
```

```
    void MouseRelease(QMouseEvent* event);

public slots:
    void timeOutSlot();

private:
    //timer for frame rate
    QTimer *t_Timer;

    //frames per second
    float framesPerSecond;

    //frame counter
    int n;

    // 3D variables initiation
    int window_width = 640;
    int window_height = 200;
    int window_number = 0;
    double view_angle = 45;
    Vector3d translations = Vector3d(0,0,0);
    Vector3d rotations = Vector3d(0,0,0);
    Vector2i previous_mouse_position = Vector2i(0,0);
    Vector3d previous_trackball_position = Vector3d(0,0,0);
    Vector3d rotation_axis = Vector3d(0,0,0);
    double rotation_angle = 0;
    float trackball_transform[4][4] = {{1,0,0,0}, {0,1,0,0}, {0,0,1,0}, {0,0,0,1}};

    //timer to calculate computation time
    Stopwatch timer;

    //mesh to be drawn
    NeighborMesh globalmesh;
    int id_globalmesh;

    //Mouse handling variables
      //Trackball mapping
    static Vector3d TrackballMapping( int x, int y );
    //Variable to control old values of x and y of the mouse
    int oldx,oldy;
    float Current_Matrix[4][4];

    //Controls the light
    int lights_on;
    //function to create lighting and update the camera position
    void createLighting();

    //Object movement variables
    //Object move speed;
    Vector3d objectMove;
    // for rotation and translation
    int rotate,translate;
    int Axis;
```

```
    int Axis_Id_List;

    // for the color bar
    vector<Vector3d> Color_Steps;
    //Function to draw color bar on the side
    void Draw_Color_Bar(int size_x, int size_y,int x_init,int y_init);

    //function to cast double to string
    //usefull when computation time need to be shown n Qstring
    string DoubleToString ( double d );

};


#endif // MYGLWIDGET_H
```

Listing 16: *myGLWidget* class

```
#include "myGLWidget.h"
#include "laplacianmesh.h"


myGLWidget::myGLWidget(QWidget *parent)
    : QGLWidget(parent)
{

    if(framesPerSecond == 0)
        t_Timer = NULL;
    else
    {
        int seconde = 1000; // 1 seconde = 1000 ms
        int timerInterval = seconde / framesPerSecond;
        t_Timer = new QTimer(this);
        connect(t_Timer, SIGNAL(timeout()), this, SLOT(timeOutSlot()));
        t_Timer->start( timerInterval );
    }
}

void myGLWidget::keyPressEvent(QKeyEvent *keyEvent)
{
    glMatrixMode(GL_PROJECTION);

    switch(keyEvent->key())
    {
        case Qt::Key_F :
            glPolygonMode(GL_FRONT_AND_BACK,GL_FILL);
            break;
        case Qt::Key_L :
            glLineWidth(1.0);
            glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);
            break;
        case Qt::Key_Plus :
            translations[2] += objectMove[2];
            break;
```

```
        case Qt::Key_Minus :
            translations[2] -= objectMove[2];
            break;
        case Qt::Key_Up :
            translations[1] += objectMove[0];
            break;
        case Qt::Key_Left:
            translations[0] -= objectMove[0];
            break;
        case Qt::Key_Right:
            translations[0] += objectMove[1];
            break;
        case Qt::Key_Down:
            translations[1] -= objectMove[1];
            break;

    }

    glMatrixMode(GL_MODELVIEW);

    updateGL();
}


#include<iostream>
using namespace std;

void myGLWidget::timeOutSlot()
{
    updateGL();
}
void myGLWidget::initializeGL()
{
    rotate=0;
    Axis=0;

    //inititate matrix for trackball
    for(int i=0;i<4;i++)
        for(int j=0;j<4;j++)
            Current_Matrix[i][j]=0;

    Current_Matrix[0][0]=Current_Matrix[1][1]=Current_Matrix[2][2]=Current_Matrix[3][3]=1;

    //initiation for Color bar
    Vector3d COL_BLUE(0,0,1);
    Vector3d COL_CYAN(0,1,1);
    Vector3d COL_GREEN(0,1,0);
    Vector3d COL_YELLOW(1,1,0);
    Vector3d COL_RED(1,0,0);

    Color_Steps.push_back(COL_BLUE);
    Color_Steps.push_back(COL_CYAN);
    Color_Steps.push_back(COL_GREEN);
```

```
90      Color_Steps.push_back(COL_YELLOW);
        Color_Steps.push_back(COL_RED);

        glClearColor(1,1,1,1);
    }
95
    //Mesh Rendering
    void myGLWidget::InitGLForRender()
    {
        // set the drawing mode to full rendering
100       glPolygonMode(GL_FRONT,GL_FILL);

          //activate Z buffer (hide elements in the back)
          glEnable(GL_DEPTH_TEST);

105       //useful if you want to superpose the rendering in full mode and in wireless mode
          glEnable(GL_POLYGON_OFFSET_FILL);

          //convenient settings for polygon offset, do not change this
          glPolygonOffset(1.0,1.0);
110
          // unit normals, in case you would forget to compute them
          glEnable(GL_NORMALIZE);

          // now you can associate a color to a point...
115       glEnable(GL_COLOR_MATERIAL);

          //background color is white (better for screenshot when writing a paper)
          glClearColor(1.0f,1.0f,1.0f,0.0f);

120       //you can activate blending for better rendering...
          glEnable( GL_BLEND );

          // careful with those parameters, results depend on your graphic card
          glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
125       glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

          //activate / disable those and compare ;)
          glEnable( GL_POINT_SMOOTH );
          glHint( GL_POINT_SMOOTH, GL_NICEST );
130
          //clear up the message output
          message = "";


135   }

    void myGLWidget::renderMesh( string filename, int option, int eigenCol,
                                      double isDefaultPos, Vector3d camPos )
    {
140       InitGLForRender();

          timer.Reset();
```

```
      timer.Start();
      if (!globalmesh.ReadFile(filename)) exit(0);
      timer.Stop();
      cout<<"Loading time :"<< timer.GetTotal()/1000.0<<" s"<<endl;
      message = message + "Loading time : "
                    + DoubleToString(timer.GetTotal()/1000.0) + "\n";

      settingUpMeshBasic( isDefaultPos, camPos );

     if( option == GEODESIC )
       renderToGeodesicRepresentation();
      else
       renderToLaplaceRepresentation( option, eigenCol );

      //ok now here render
      id_globalmesh=glGenLists(1);
      glNewList(id_globalmesh,GL_COMPILE_AND_EXECUTE);
      globalmesh.Draw(VERTEX_NORMAL_RGB);

      glEndList();

      resizeGL(width(),height());
      updateGL();

}

void myGLWidget::settingUpMeshBasic( double isDefaultPos, Vector3d camPos )
{
    //construct P2P : point to point connectivity
    timer.Reset();
    timer.Start();
    globalmesh.Build_P2P_Neigh();
    timer.Stop();
    cout<<"P2P construction :"<< timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "P2P construction : "
                    + DoubleToString(timer.GetTotal()/1000.0) + "\n";

    //construct P2F : point to face connectivity
    timer.Reset();
    timer.Start();
    globalmesh.Build_P2F_Neigh();
    timer.Stop();
    cout<<"P2F construction :"<< timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "P2F construction : "
                    + DoubleToString(timer.GetTotal()/1000.0)+ "\n";

    //construct F2F : face to face connectivity
    timer.Reset();
    timer.Start();
    globalmesh.Build_F2F_Neigh();
    timer.Stop();
    cout<<"F2F construction :"<< timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "F2F construction : "
```

```
                    + DoubleToString(timer.GetTotal()/1000.0)+ "\n";

    //construct Edges
    timer.Reset();
    timer.Start();
    globalmesh.Build_Edges();
    timer.Stop();
    cout<<"Edges construction :"<< timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "Edges construction : "
                    + DoubleToString(timer.GetTotal()/1000.0)+ "\n";


    sceneGL.UpdateCamera(globalmesh, isDefaultPos, camPos);
    //Init camera parameters based on the globalmesh
    double distance = sceneGL.GetDistance();

    //adjust displacements consequently
    objectMove[0] = distance/20;
    objectMove[1] = distance/20;
    objectMove[2] = distance/20;
    //creates lights accordingly to the position and size of the object
     createLighting();

    //compute normals
    globalmesh.ComputeFaceNormals(); //normal to faces
    globalmesh.ComputeVertexNormals(); //normals to vertex

}

void myGLWidget::renderToGeodesicRepresentation()
{
    int startpointindex = 0;
    timer.Reset();
    timer.Start();
    //will compute approximate geodesic distance from this point to all other points
    //consider the mesh as a graph and run shortest path algo
    //then stores distances in the label array
    globalmesh.BuildDistanceLabels(startpointindex);
    timer.Stop();
    cout<<"Distance label construction :"
          << timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "Distance label construction : "
                    + DoubleToString(timer.GetTotal()/1000.0) + "\n";

   //construct colors from labels
   timer.Reset();
   timer.Start();
   globalmesh.SetColorsFromLabels();
   timer.Stop();
   cout<<"Constructing colors from distance :"
          << timer.GetTotal()/1000.0<<" s"<<endl;
   message = message + "Constructing colors from distance : "
               + DoubleToString(timer.GetTotal()/1000.0) + "\n";
```

```
}
void myGLWidget::renderToLaplaceRepresentation( int weightOption, int eigenCol )
{

    LaplacianMesh lapMesh(globalmesh);
    switch( weightOption )
    {
        case UNIFORMWEIGHT:
        {
            timer.Reset();
            timer.Start();
            lapMesh.GetLaplacianUniformWeight();
            timer.Stop();
            cout<<"Calculating laplacian matrix: "
                    <<timer.GetTotal()/1000.0<<" s"<<endl;
            message = message + "Calculating laplacian matrix:  "
                              + DoubleToString(timer.GetTotal()/1000.0) + "\n";

        }break;
        case COTANGENTWEIGHT:
        {
            timer.Reset();
            timer.Start();
            lapMesh.GetLaplacianCotangentWeight();
            timer.Stop();
            cout<<"Calculating laplacian matrix: "
                    << timer.GetTotal()/1000.0<<" s"<<endl;
            message = message + "Calculating laplacian matrix:  "
                              + DoubleToString(timer.GetTotal()/1000.0) + "\n";
        }break;
        case MEANVALUE:
        {
            timer.Reset();
            timer.Start();
            lapMesh.GetLaplacianMeanValue();
            timer.Stop();
            cout<<"Calculating laplacian matrix: "
                    << timer.GetTotal()/1000.0<<" s"<<endl;
            message = message + "Calculating laplacian matrix:  "
                              + DoubleToString(timer.GetTotal()/1000.0) + "\n";

        }break;

    }

    //test laplace
    timer.Reset();
    timer.Start();
    lapMesh.CalculateEigenDecomposition( eigenCol );
    timer.Stop();
    cout<<"Calculating eigen decomposition: "
              << timer.GetTotal()/1000.0<<" s"<<endl;
```

```
    message = message + "Calculating eigen decomposition: "
                    + DoubleToString(timer.GetTotal()/1000.0) + "\n";

    //test laplace
    timer.Reset();
    timer.Start();
    lapMesh.BuildColorMap( &globalmesh );
    timer.Stop();
    cout<<"Building eigen color map"<< timer.GetTotal()/1000.0<<" s"<<endl;
    message = message + "Building eigen color map : "
                    + DoubleToString(timer.GetTotal()/1000.0) + "\n";
}

void myGLWidget::UpdateCamera( bool isDefaultPos, Vector3d camPos )
{
    sceneGL.UpdateCamera( globalmesh, isDefaultPos, camPos );
    createLighting();
    resizeGL(width(),height());
    updateGL();
}
//update the camera position and lighting
void myGLWidget::UpdateLighting()
{
    createLighting();
    resizeGL(width(),height());
    updateGL();
}

//OpenGL Scene3D rendering
void myGLWidget::resizeGL(int width, int height)
{

    glPushMatrix();
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(  45,
                    (float)width/height,
                    sceneGL.GetZnear(),
                    sceneGL.GetZfar()
                    );

    // wonderful gluLookAt function
    gluLookAt(
    sceneGL.GetPosition()[0], sceneGL.GetPosition()[1], sceneGL.GetPosition()[2],
    sceneGL.GetTarget()[0], sceneGL.GetTarget()[1], sceneGL.GetTarget()[2],
    0,1,0);//Up Vector, do not change this

    glMatrixMode(GL_MODELVIEW);
```

```
}
void myGLWidget :: paintGL()
{
     //this function will be called when calling updateGL

    GLboolean lights_on[1];
    glGetBooleanv(GL_LIGHTING,lights_on);

    GLint polygon_draw_mode[2];
    glGetIntegerv(GL_POLYGON_MODE,polygon_draw_mode);

    GLint viewport[4];
    glGetIntegerv(GL_VIEWPORT,viewport);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //////
    //
    //    Windows informations display (color bar, referential,...)
    //
    /////

    glDisable(GL_LIGHTING);

    //render 3D ref at the bottom left corner of the Opengl window

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //Trackball Transformation
    glRotatef( rotation_angle, rotation_axis[0],
               rotation_axis[1], rotation_axis[2] );
    glMultMatrixf((GLfloat *)trackball_transform);
    glGetFloatv(GL_MODELVIEW_MATRIX, (GLfloat *)trackball_transform);

    //Color Bar Draw
    Draw_Color_Bar(30,100,30,170);

    //translation in windows coordinates
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();

    //local 3d projection to draw 3D referential(size=30 pixels)
    glOrtho(0,viewport[2],0,viewport[3],0,max(viewport[2],viewport[3]));
    //to avoid clip planes
    glTranslatef(10,90,-50);

    glPushMatrix();

    glMatrixMode(GL_MODELVIEW);
    //disable lights before drawing 3d axis
    glDisable(GL_LIGHTING);
```

```
    glMatrixMode(GL_PROJECTION);

    glPopMatrix();

    glPopMatrix();

   //render mesh
    glPolygonMode(GL_FRONT,polygon_draw_mode[0]|polygon_draw_mode[1] );

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    Vector3d center(0,0,0);
    glTranslatef(translations[0],translations[1],translations[2]);

    glMultMatrixf((GLfloat *)trackball_transform);
    if(lights_on) glEnable(GL_LIGHTING);
    else glDisable(GL_LIGHTING);
    glCallList(id_globalmesh);

    //illustrate results for a point
    rotation_axis = Vector3d(0,0,0);
    rotation_angle = 0;

}

void myGLWidget::Draw_Color_Bar(int size_x, int size_y,int x_init,int y_init)
{
    // store draw mode (light and polygon mode)
    GLint polygon_draw_mode[2];
    glGetIntegerv(GL_POLYGON_MODE,polygon_draw_mode);

    vector<Vector3d>::iterator it(Color_Steps.begin());
    Vector3d current,prev;

    GLint viewport[4];
    glGetIntegerv(GL_VIEWPORT,viewport);

    glDisable ( GL_LIGHTING );
    glPolygonMode(GL_FRONT,GL_FILL);

    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D(0, viewport[2], 0, viewport[3]);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glTranslatef(x_init, y_init,0);
    double coef_x(size_x),coef_y(size_y/double(Color_Steps.size()-1) );

    for(int i=0;i<Color_Steps.size()-1;i++){
        prev=(*it);
```

```
        it++;
        current=(*it);

        glBegin(GL_QUADS);

        glNormal3d(0,0,1.0);
        glColor3d(prev[0], prev[1], prev[2]);
        glVertex3d(size_x,(i)*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(current[0],current[1],current[2]);
        glVertex3d(size_x,(i+1)*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(current[0],current[1],current[2]);
        glVertex3d(0,(i+1)*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(prev[0], prev[1], prev[2]);
        glVertex3d(0,(i)*coef_y,0);


        glEnd();


        double t0=(double)(i)/((double)Color_Steps.size()-1);
        double t1=(double)(i+1)/((double)Color_Steps.size()-1);

        glBegin(GL_QUADS);

        glNormal3d(0,0,1.0);
        glColor3d(t0,t0,t0);
        glVertex3d(2*size_x+10,i*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(t1,t1,t1);
        glVertex3d(2*size_x+10,(i+1)*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(t1,t1,t1);
        glVertex3d(size_x+10,(i+1)*coef_y,0);

        glNormal3d(0,0,1.0);
        glColor3d(t0,t0,t0);
        glVertex3d(size_x+10,i*coef_y,0);


        glEnd();
    }

    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
```

```
    glMatrixMode(GL_MODELVIEW);

    //old polygon drawing mode restoration
    glPolygonMode(GL_FRONT,polygon_draw_mode[0]|polygon_draw_mode[1]);
}

void myGLWidget::createLighting()
{
    //function to play around with open GL lights
       glEnable(GL_COLOR_MATERIAL);

       //roughly speaking : color of reflected light
       GLfloat* s = sceneGL.GetSpecular();
       GLfloat specular[]={ *(s+0),  *(s+1),  *(s+2),  *(s+3)};

        glMaterialfv(GL_FRONT,GL_SPECULAR, specular);

       //intensity of the shining...

       GLfloat* i = sceneGL.GetIntensity();
       GLfloat intensity[]={ *(i+0)};

       glMaterialfv(GL_FRONT,GL_SHININESS,intensity);

       //ambient and diffuse light
       GLfloat* a = sceneGL.GetAmbient();
       GLfloat ambient[]={ *(a+0),  *(a+1),  *(a+2),  *(a+3)};

       GLfloat* diff = sceneGL.GetDiffuse();
       GLfloat diffuse[]={ *(diff+0),  *(diff+1),  *(diff+2),  *(diff+3)};
       glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
       glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);

       //set spot light cone, direction, angle, etc
       Vector3d Dir = -( sceneGL.GetPosition() -sceneGL.GetTarget()).normalized();
       GLfloat spot_direction[] = {Dir[0],Dir[1],Dir[2]};
       glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 360.0);
       glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
       glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);

       //activate the first light
       glEnable(GL_LIGHT0);

 }

//Movement
Vector3d myGLWidget::TrackballMapping( int x, int y )
{
    static const double PI_2 = 3.14159265358979323846264338327/2.0;

    Vector3d v(0,0,0);
    static GLint params[4];
```

```
    glGetIntegerv(GL_VIEWPORT,params);

    v[0] = ( 2.0 * x - params[2] ) / params[2];
    v[1] = ( params[3] - 2.0 * y ) / params[3];

    double  d = v.norm();
    if( d > 1.0 ) d = 1.0;

    v[2] = cos((PI_2) * d);

    return v.normalized();
}
void myGLWidget::Motion( int x ,int y)
{
    static Vector3d current_position;

    if(rotate)
     {
          // Map the mouse position
        current_position = TrackballMapping( x, y );
        // Rotate about the axis that is perpendicular
          //to the great circle connecting the mouse movements.
        rotation_axis = previous_trackball_position .cross ( current_position );
        rotation_angle =
          90.0 *( current_position - previous_trackball_position ).norm() * 1.5;
        previous_trackball_position = current_position;
    }

    if(translate){

        if( (y - oldy) > 0)
            translations[2] += objectMove[2];
        else
            translations[2] -= objectMove[2];

        oldx=x;
        oldy=y;
    }

}

void myGLWidget::Mouse (QMouseEvent* event)
{
    rotate=0;
    if (event->buttons() && Qt::LeftButton)
    {
        rotate=1;
        previous_trackball_position = TrackballMapping( event->x(), event->y() );
    }
    else if (event->buttons() && Qt::RightButton)
    {
        translate=1;
        oldx = event->x();
```

```
        oldy = event->y();
    }
}

void myGLWidget::mouseMoveEvent(QMouseEvent * mouse)
{
    Motion(mouse->x(),mouse->y());
    updateGL();
}

void myGLWidget::MouseRelease(QMouseEvent* event)
{
    if (event->buttons() && Qt::LeftButton)
    {
        rotate=0;
    }
    else if ( event->buttons() && Qt::RightButton)
    {
        translate=0;
        oldx = event->x();
        oldy = event->y();

    }
}

void myGLWidget::mouseReleaseEvent(QMouseEvent* mouse)
{
    MouseRelease(mouse);
    updateGL();
}

void myGLWidget::mousePressEvent(QMouseEvent* event)
{
    Mouse(event);
    updateGL();
}

string myGLWidget::DoubleToString ( double d )
{
    ostringstream strs;
    strs << d;
    string str = strs.str();

    return str;
}
```

The implementation of this class is a modification from *main.cpp* and *Scene* class from previous bundle. In here, I make all the initiation needed for GLUT rendering and as well handling the user interaction, such as *keyPressEvent* and *mouseEvent*. It is worth noted that when making *keyPressEvent*, I need to set the focus for *myGLWidget* to be *StrongFocus*. Otherwise, it will not execute the handler correctly.
In order to make the camera position and lighting configurable, I make a new class *SceneGL*. This class handles the camera position and the lighting of the mesh rendering.

Listing 17: Header file for *SceneGL* class

```
#ifndef SCENEGL_H
#define SCENEGL_H

#include "NeighborMesh.h"
#include <GL/glu.h>

using namespace Eigen;

class SceneGL
{
public:
    SceneGL();

    Vector3d GetTarget();
    Vector3d GetPosition();
    double GetDistance();

    //camera
    double SetCameraDefaultPosition(const NeighborMesh& mesh);
    double GetZnear ();
    double GetZfar ();

    //lighting variables
    void SetLightingDefaultPosition();
    GLfloat* GetAmbient();
    GLfloat* GetDiffuse();
    GLfloat* GetSpecular();
    GLfloat* GetIntensity();

    //update camera ang lighting
    void UpdateCamera    (    const NeighborMesh& mesh, bool isDefault,
                              Vector3d userPosition );
    void UpdateLighting  (    bool isDefault, GLfloat a[4],
                              GLfloat s[4], GLfloat d[4], GLfloat i);


private:
    //camera variables
    Vector3d position;
    Vector3d target;
    double znear;
    double zfar;
    double distance;

    //lighting variables
    GLfloat specular[4]={1.0, 1.0, 1.0, 1.0};
    //intensity of the shining...
    GLfloat intensity[1]={100};
    //ambient and diffuse light
    GLfloat ambient[4]={0.1, 0.1, 0.1, 1.0};
    GLfloat diffuse[4]={0.4, 0.4, 0.4, 1.0};
```

```
};


#endif // SCENEGL_H
```

Listing 18: *SceneGL* class

```
#include "SceneGL.h"
#include <iostream>
SceneGL::SceneGL()
{
}


double SceneGL::SetCameraDefaultPosition(const NeighborMesh& mesh)
{
    //roughly adjust view frustrum to object and camera position
    Vector3d Pmin(mesh.vertices[0]), Pmax(mesh.vertices[0]);
    Vector3d Center(0,0,0);

    for( int i=0; i< mesh.vertices.size(); i++)
    {
        Vector3d P(mesh.vertices[i]);
        for( int j=0; j<2; j++)
        {
            Pmin[j] = min(Pmin[j],P[j]);
            Pmax[j] = max(Pmax[j],P[j]);
        }
        Center += P;
    }

    Center/= mesh.vertices.size();
    target = Center;

    //length of the diagonal of the bouding box
    double distance = (Pmax-Pmin).norm();

    //set arbitraty position to camera and adjusts max and min view planes
    position = target + Vector3d( 0,0, distance*3);
    znear = distance*0.1;
    zfar = distance*5;

    return distance;
}

Vector3d SceneGL::GetTarget()
{
    return target;
}

Vector3d SceneGL::GetPosition()
{
    return position;
```

```
}
double SceneGL::GetDistance()
{ return distance; }

double SceneGL::GetZnear ()
{
    return znear;
}
double SceneGL::GetZfar ()
{
    return zfar;
}


GLfloat* SceneGL:: GetAmbient()
{
    return ambient;
}
GLfloat* SceneGL:: GetDiffuse()
{
    return diffuse;
}
GLfloat* SceneGL:: GetSpecular()
{
    return specular;
}
GLfloat* SceneGL:: GetIntensity()
{
    return intensity;
}

void SceneGL::UpdateCamera( const NeighborMesh& mesh,
                                    bool isDefault, Vector3d userPosition )
{
    Vector3d Center(0,0,0);

    if( isDefault )
        distance =  SetCameraDefaultPosition( mesh );
    else
    {

        for( int i=0; i< mesh.vertices.size(); i++)
        {
            Vector3d P(mesh.vertices[i]);
            Center += P;
        }

        Center /= mesh.vertices.size();
        position = userPosition;
        distance = (Center - position).norm();

        znear = distance*0.1;
        zfar = distance*3;
```

```
    }

}

void SceneGL::UpdateLighting( bool isDefault, GLfloat a[4],
                                  GLfloat s[4], GLfloat d[4], GLfloat i)

{
    if( isDefault )
        SetLightingDefaultPosition();
    else
    {
        for(int i=0; i<4;i++)
        {
            diffuse[i]= d[i];
            specular[i]= s[i];
            ambient[i]= a[i];
        }
        intensity[0] = i;
    }
  }

void SceneGL::SetLightingDefaultPosition()
{
    color of reflected light
    GLfloat s[4]={1.0, 1.0, 1.0, 1.0};
    //intensity of the shining
    GLfloat i[1]={100};
    //ambient and diffuse light
    GLfloat a[4]={0.1, 0.1, 0.1, 1.0};
    GLfloat d[4]={0.4, 0.4, 0.4, 1.0};

  diffuse[0]= *(d+0);diffuse[1]= *(d+1);diffuse[2]= *(d+2);diffuse[3]= *(d+3);
  specular[0]= *(s+0);specular[1]= *(s+1);specular[2]= *(s+2);specular[3]= *(s+3);
  ambient[0]= *(a+0);ambient[1]= *(a+1);ambient[2]= *(a+2);ambient[3]= *(a+3);
  intensity[0] = *(i+0);

}
```

I use this class to store all of the camera and lighting related variables, update it when there is changes from the front page and render it back in the QGLWidget. Here I made all of the variables to be *private* and make one public function for each of the variable. This class will be instantiated in *myGLWidget* class. Furthermore, I have added manual page directly in the widget to help the user to use the application. I notice that most of the standalone application built does not have a manual to be referred to and the shortcut is not mentioned eventhough there are great shortcuts have been implemented. I found this very inconvenient. I have added a short user manual in the application tab for this.

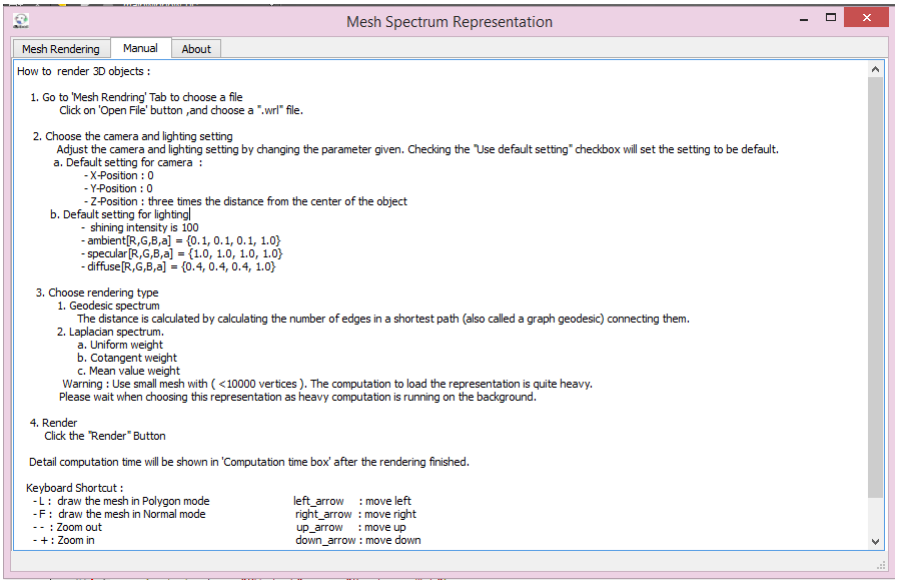

Figure 21: Manual tab

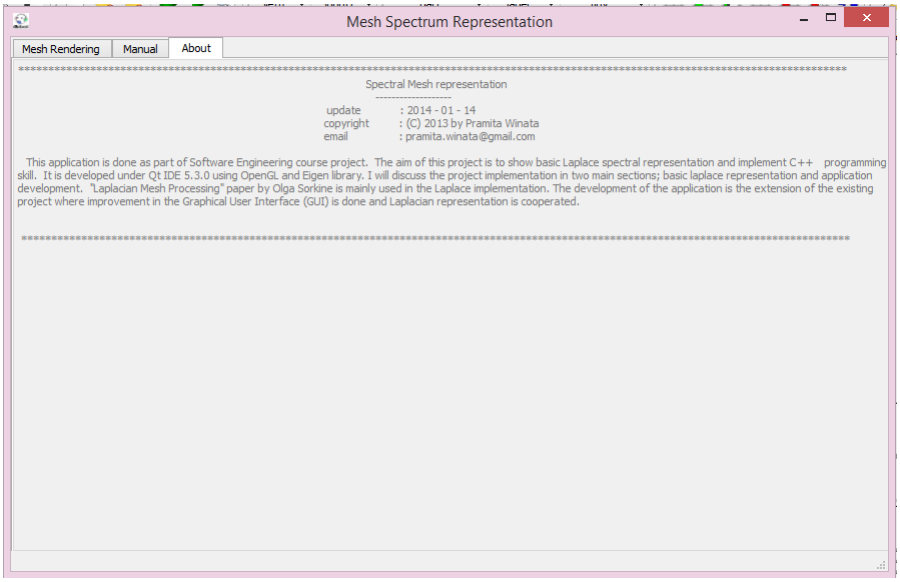

Figure 22: About tab

Conclusion

This project has been a very informative, useful, interesting, and tedious project. Not only that my programming skill is improved but also in researching scientific paper. Understanding Laplace operator is not as easy as it sounds. Reading many papers and researching need to be done to understand it. I have known C++ much better after implementing this project especially in integrating OpenGL and utilizing Eigen library. Not forget to mention, dealing with C++ standard library and overall behaviors are definitely very informative and useful.

Further Improvement

- There are many improvements can be made to this application. In term of Laplacian representation, many additional features can be added, for example adding the anchor. Anchor implementation will enable us to get a deform mesh structure but still keeping the overall structure in place. This is definitely a very interesting and useful improvement.
- In the GUI point of view, many other additional features can be added, especially the implementation of Thread. I definitely noticed that rendering Laplacian representation takes a lot of time. It will be useful to implement a progress bar in order to show that the application actually is doing something in behind. A real time information also can be an option.

References

[1] Olga Sorkine,*Laplacian Mesh Processing*. STAR report, Eurographics, 2005.

[2] Mesh Processing, G. Peyr,
`https://www.ceremade.dauphine.fr/ peyre/teaching/manifold-sci/`

[3] Eigen library,
`http://eigen.tuxfamily.org/index.php?title=Main_Page`

[4] The C++ Standard library, see
`http://www.yolinux`.

[5] MEYER M., DESBRUN M., SCHRDER P., BARR A. H.: Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Hege H.-C., Polthier K., (Eds.). Springer-Verlag, Heidelberg, 2003, pp. 3557.

[6] HERON. 60.
`Metrica`. Alexandria, Roman Egypt.

[7] FIEDLER M.: Algebraic connectivity of graphs.
`Czech. Math. Journal 23` (1973), 298305.

[8] LOATER M. S.: Mean value coordinates.
`CAGD 20`, 1 (2003), 1927.