Software Engineering Project - Robust principal axes determination for point-based shapes

Quim Sanchez Nicuesa
Gerard Bahi Vila

January 13, 2013

Contents

1 Introduction **2**

2 Methodology **2**

3 Time line **3**

4 Understanding the problem **3**

5 Research and decisions **4**
- 5.1 Eigen 5
- 5.2 OpenMesh 5
- 5.3 Qt4 5
- 5.4 OpenGL 6
- 5.5 Octree data structure 6

6 System analysis and design **6**
- 6.1 Graphical interface design 7
- 6.2 Paper implementation 7
  - 6.2.1 RANSAC and LMS 8
  - 6.2.2 Forward Search 9
  - 6.2.3 Implementation and Algorithm minor details 9
  - 6.2.4 Alignment 10
- 6.3 Class design 11
  - 6.3.1 Class diagram 11
  - 6.3.2 QGLViewerWidget 12
  - 6.3.3 MeshViewerWidgetT 12
  - 6.3.4 MeshViewerWidget 12
  - 6.3.5 ColorMap 12
  - 6.3.6 Axis3D 13
  - 6.3.7 QDebugStream 13
  - 6.3.8 LinalgTools 13
  - 6.3.9 PrincipalAxisGenerator 13
  - 6.3.10 QPrincipalAxisGenerator 13
  - 6.3.11 MeshTranslator 13
  - 6.3.12 MainWindow 14

7 Results **14**

8 Conclusions **16**

9 Future work **17**

1 Introduction

The main goal of this project is to implement the algorithm described in [1].

This algorithm tries to find the axis of a cloud of points. This problem is usually addressed with PCA. PAC will provide a main axis using the minimum mean distance from all the points to its projection on the axis lines. This is a good approach when you have clean data without noise. But a some noise on the cloud of points or a small change on the represented 3d shape and PCA will give a completely different result.

In 3D models clouds of points are used to represent shapes. This shapes can be obtained from laser scanners which are not very accurate and will provide noisy data. If we want to get the principal axis of the scanned shape then PCA is not the best way to go.

Getting the principal axis from a 3D shape is something very common. For example it is the first step if we want to match two shapes. Here they provide a method to obtain the axis of the cloud of points in a robust way, that it is a way resistant to noise and small changes on the shape that will give better results that PCA.

2 Methodology

We did not find necessary to choose an specific software developing methodology, as we do not have a real client and we would need to apply some modifications to the chosen methodology. We decided to choose a methodology group and follow its main features

For developing this project we based our working methodology on a software development method group called Agile. This methods are based on the iterative and incremental development.

Agile focuses on generating working software rather than documents to clients. It also has an adaptive planning. The development follows a time boxed iterative approach which makes it easy to adapt to changes in the requirements and new needs.

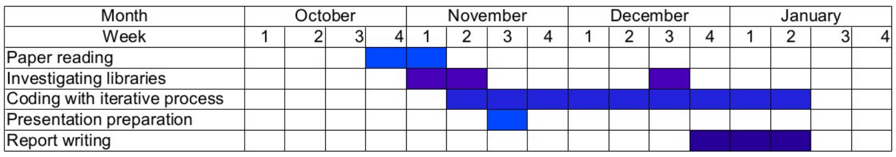
This group of methodology is intended for small groups of developers which have different programming skills. It is focused on short projects rather than long therm projects, as its iterative approach makes it difficult to plan the work flow for a long period of time. A more detailed explanation of the Agile method can be found in the Agile Manifesto [2].

As in this project we do not have a client that would change the requirements. We approach it as us being our own clients. We sated down a sketch of all the features we wanted our final system to have. This features would be modified due to complexity or developing time consuming, hence the requirements of the final change would be modified as well.

This way we needed to check periodically the state of our beta application. Which is a good way for making us realize if it would be possible to finish all the desired tasks by the death-line and re-schedule the work time if necessary.

3 Time line

After understanding the main task given by the paper. We set up a meeting to discuss the best way to approach the problem, chose a working method and diving the workload for finding the libraries that would fit our needs better. After this we sated a second meeting to generate a time line for our project. In the following figure an approximation of our time line is shown.



Month	October				November				December				January			
Week	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Paper reading				🟦	🟦											
Investigating libraries					🟪	🟪					🟪					
Coding with iterative process						🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
Presentation preparation							🟦									
Report writing												🟦	🟦	🟦		

Figure 1: Timeline table

4 Understanding the problem

We want to implement a method that correctly obtains the principal axis of a 3D shape represented by a cloud of points. This points might be more or less accurate and they might even have some noise. We also want to be able to recover the principal axis from a truncated shape (i.e. a shape from which some part of it has been removed).

The algorithm treys to solve this problem by dividing the cloud of points between a major region and a minor region. In the major region we want to have the significant points of the shape that will be used to compute the axis and in the minor region we want to have all the outliers, including noise and port of the shape that are far from the main part of the object and that would give a big change on the axis (i.e. in a shape of a dog the legs would be included in the minor region because they are not part of the main body).

To split the cloud of points we could follow two different iterative approaches. The first one we would be a backward method in which we start with all the points and we keep removing them on every iteration using some rule. The second one is called forward search. In this method we start with a small subset of points which we know to be representative of the whole object (i.e. all the points of the subset have to be part of what will later become the major region) and from that we iteratively keep adding points following some rules until no more points meets the rule.

In this algorithm forward search is used, so we need a method to get an initial clean subset of points. We need now a method to get this initial clean subset. We are going to use the RANSAC algorithm (RANdom SAmple Consensus) where you run a method T times with random seeding and you keep the best gotten solution.

This method gets a random subset and then it calculates the main principal axis (one points and one vector) of those points using PCA. We can measure how good this subset is by the LMS (Least Median of Squares) measure. LMS will calculate the distance from all points not belonging to the choose subset to its projection onto the the line defined by the calculated principal axis and gets the median of the distances. It uses the median because it is resistant to outliers, more resistant that for example the mean.

After we get a clean subset of the cloud of points it uses the forward search technique. We will call now this subset the major region of the clouds and all the other points the minor region, the goal of this step is to keep adding new points to the major region. On every iteration the algorithm will calculate the principal axis of the major region and the distances from all the points from the minor region to the line defined by the principal axis. Then it moves the closest points from the minor region to the major region. The algorithm will converge when all the points from the minor region are too far away to be added to the major region.

Finally from the major region we calculate the principal axis using PCA.

5 Research and decisions

After reading the paper we saw that we needed libraries that would help us to do the linear algebra calculus. We also clearly needed libraries that would be able to load a mesh, view it and provide us a list of points. In addition we also needed to find an implementation of the octree data structure that would fit our necessities.

We decided to choose Eigen libraries as it was the advice of our professor. For choosing the mesh handling libraries we took in account that they would use STL data structure as it makes easy to joint different libraries for doing all the tasks needed. In addition we also were looking for libraries that would have support for working with the libraries we are used to work such as Qt, OpengGL or STL between others. Finally we decided to use OpenMesh as it has Qt and OpengGL support and uses STL data structures to store the mesh.

5.1 Eigen

Eigen libaries are known for being a good support libraries for doing linear algebra calculus. They are known for being efficient and reliable.

The Egien project was started by Benoit Jacob (founder) and Gael Guennebaud (guru). It is a free software community so anyone can contribute to the project. The libraries have a Free software license. However as they also use some third party code some features are under LGPL license. The support material for Eigen libraries can be found in [3]

5.2 OpenMesh

These libraries offer support for loading several types of standard mesh file formats. They can represent meshes with any polygonal shape although we are going to use triangles.

They have Qt and OpengGL for displaying the mesh. Which creates a layer above OpenGL and merges it with the Qt objects making it easy to attach it to an existing Qt application. They also offer sample codes which include a mesh viewer for a single mesh.

These libraries are multi-platform and have LGPL license and all the documentation needed including the sample codes can be found at [4].

5.3 Qt4

We are going to use Qt for creating the main application as we have already worked with them before.

The main working principle of these libraries is the design patter called Model/View (MV). Which is a modification of the Model-View-Controller (MVC) pattern. MV merges the objects of type Controller and the objects of type View.

They base the interaction between objects in a signal slot system. Each Qt object has a set of signals that are emitted when an event happens. For example: a button is clicked or an attribute is modified. These signals can be connected to other objects through slots. Slots are a set of methods that can be seen as receivers, which are going to be executed automatically every time one of the signals is connected to is emitted.

The Qt libraries are multi-platform. They are property of Digia since 2012 and they have a LGPL license. Their documentation and manuals can be found at the Qt project website [5].

5.4 OpenGL

These libraries are used for rendering 2D and 3D. We are using these libraries for rendering the mesh points. However we are not working directly in OpenGL as OpenMesh create an intermediate layer which simplifies its use.
THe OpenGL libraries are managed by a non-profit consortium called Khronos Group, which is composed by Apple, AMD/ATI, Google, Nvidia among other companies. It is a multi-platform library. Their license information can be found in [6]. The support material can be found in [7].

5.5 Octree data structure

In this project we had the option to use an octree to accelerate the computation. In order to do that part we looked for some implementations on the Internet, but most of them would not match our needs. Basically we need to fill the octree with data and the be able to access this data per cells. Most of the libraries would only allow us to access per point position.

We finally found on [8] one that has an iterator per cell. It also has an transversal method which allows us to apply one *callback* to all the cells in one call. *callback* is a class with method that gets executed on every cell. One good thing of using a class instead of a function is that it has its own memory and it can act differently depending on what he did on the previous cells.

6 System analysis and design

The application is going to have two main groups of classes: the ones for the linear algebra calculations that would compute the axes given a set of points, and the ones in charge of mesh loading and displaying. The main application classes are going to connect the two groups of classes making them totally independent. Programming the system these way the calculus classes can be easily exported to another interface. What's more the mesh handling classes can be used for different purposes rather than the one we are using it for now.

For the mesh handling classes we had an starting point, which was a sample code from the OpenMesh library. That sample code could load and display a single mesh using Qt and OpenGL. So we adapted it to be able to handle N meshes.

Although the main goal of the project is to implement the paper algorithm we found important to find a way to present the data in a intuitive way. In addition we did not want only to focus on displaying the paper method properly. We wanted to create an interface that would let different methods to be compared visually. In order to see their drawbacks and advantages in a graphical way.

From that point we divided our system in two main tasks:

- Create an interface capable of fulfill our requirements without being tough to use for the user.
- Implement the paper method...

6.1 Graphical interface design

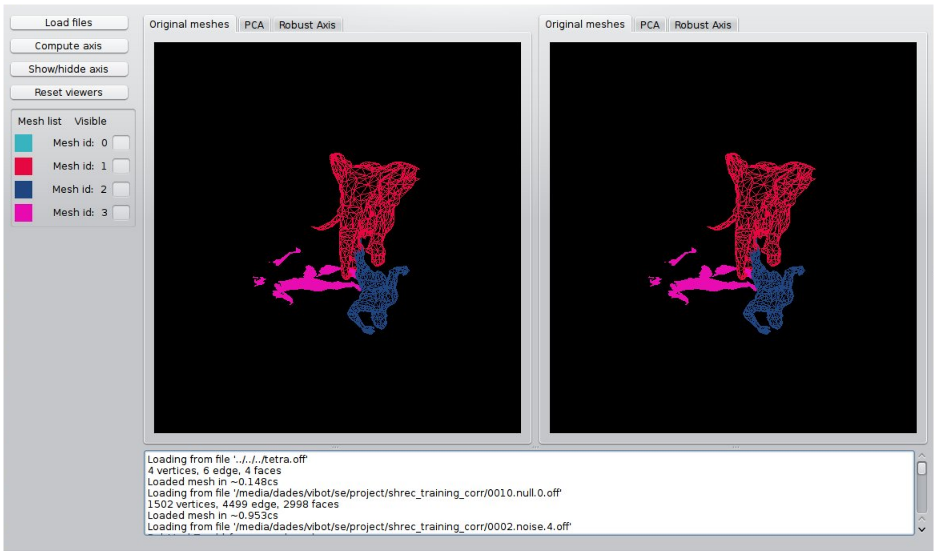
In figure 2 the final design of our main application is shown. We decided to put two viewers next to each other so comparison between results can be made easily. Those viewers have tabs in order to choose in which method are you displaying the data. We decided to set three ways of displaying the point clouds:

Points Only the points of the mesh are displayed

Wireframe Faces are displayed but they are transparent so you can see through them

Hidden-Line Faces are discuss and they are solid.

We also connected the mouse events between the viewers so when a movement is applied to one viewer the other viewer moves its elements too. This way we achieve a better comparison between methods.



```
Loading from file '../../../tetra.off'
4 vertices, 6 edge, 4 faces
Loaded mesh in ~0.148cs
Loading from file '/media/dades/vibot/se/project/shrec_training_corr/0010.null.0.off'
1502 vertices, 4499 edge, 2998 faces
Loaded mesh in ~0.953cs
Loading from file '/media/dades/vibot/se/project/shrec_training_corr/0002.noise.4.off'
```

Figure 2: Interface design of the application

6.2 Paper implementation

To implement the this algorithm we used the Eigen library. This library provide a few classes to work with matrices and vectors. This classes allows us to work in a similar way to Matlab. As we are going to use some of the functions provided by Eigen, for example to calculate eigenvalues and eigenvector or to calculate the inverse of a matrix, we tried to not mix it with any other libraries (to avoid translations), so all the used variables are going to be part of the Eigen library.

The first thing we do is to translate the mesh structure to a 3xM matrix where each column is one point from the cloud, we call it the vertexes matrix.

This algorithm can mainly be divided in two important methods, the first one calculates the initial subset and the second part applies the forward search to this subset.

6.2.1 RANSAC and LMS

This algorithm starts with the vertexes matrix. From this matrix it is intended to obtain a subset of n points (where n is a small number like 5 or 7) which is supposed to be clean of outliers.

We start by selecting n elements using a pseudo-random number generator. This points are going to be copied (not moved) from the vertexes matrix to a the subset matrix. We keep track of this movement with a mask vector where in each position well have a 0 if the the point on that same position has not been selected to be part of the subset or a 1 if the point on that position is now part of the subset. This vector will have more zeros than ones so it should be implemented as a space vector.

On [1] it is proposed to use an octree to get a better random subset. When using just random points many iterations will get points from the same regions, this iterations will be useless because they will not have enough information from the shape. Using an octree we can force the random selection to be spread over all the shape with provability proportional to the density each octree cell.

Next step is to calculate the principal axis using PCA using the subset matrix and apply LMS to the points that are not part of the selected subset. We give all the vertexes to the function that calculates the distances but it knows which ones should be used looking for 0s in the mask vector. To calculate the distance from a point to a line defined by:

$$\|(a-p)-((a-p)\cdot n)n\|$$

where p is the point, a is one point that already belongs to the line and n is the vector that defines the direction of the line. Here an octree can be used too. Instead of calculating the distance from all the points to the principal axis we can get only the distance of a representative point from each octree cell.

To find the median of the distances we need first to sort them. We use the STL *sort* method to sort the distances and get the median one, this is the only part of the algorithm where we get out of Eigen classes.

If the new media is the smallest median so far we keep this subset as the best one, and move to the next iteration with a new random subset.

6.2.2 Forward Search

Ones we have an initial clean of outliers subset we start with the Forward Search algorithm. This subset is now the major region and we are going to fill it with more points.

On each iteration we calculate the principal axis of the major region and add to it the closest points of the minor region. To calculate and sort the distances we use the method explained in the section above.

As we know the major region can be at most as large as the set of points we initialize its matrix with the same size as the vertexes matrix. Every time we move one point from the minor region the the major region we will update the mask vector that codes the belonging or not belonging to the major region. It is used to calculate the distances from the points that belong to the minor region.

Ones the distance from all the major point to the principal axis is larger than a concrete threshold we end the search and leave the major and minor region as they are.

In [1] they propose to have a threshold proportional to the maximum distance from the initial subset to its calculated principal axis. We have this proportion as a parameter of the method.

6.2.3 Implementation and Algorithm minor details

Once we have the shape divided into the major and minor region we can calculate the first principal axis using PCA. To calculate the second principal axis we do as follows. We firs need to project all the points to the plane composed by the central gravity points and the computed principal axis as its normal vector. With this set of projected points we apply the same method described above (RANSAC, LMS and Forward Search). The third principal axis is the corss product of the first and the second.

To do the projection of the points onto a plane we use the formula:

$$p - ((p - o) \cdot n)n$$

where p is the point to project, o is a point of the plane and n is the normal of the plane.

Many times we need to compute PCA of a set of points. We used the code from [9] but it was wrong. When computing the mean of the points it computes the mean of the columns instead of the mean of the rows, so we updated the code to correct this code. We also forced the code to return up pointing eigenvectors (i.e. vectors that move positively on the Z coordinate) otherwise even if the direction was correct the objects would be upside down matched.

6.2.4 Alignment

Finally when we have to shapes with its principal axis we want to align those axis. All different axis can be mapped to each other using a rotation and a translation. The translation is easy to compute, to translate from point p to point q we need to add to p the increment $q - p$.

A rotation is defined by a $3x3$ matrix:

$$R = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$V_{rot} = RV$$

We need this matrix to rotate all points from one mesh to match the axis of the other mesh.

As we have 3 axis with 3 coordinates each that we want to match with with the 3 axis from the other. For every axis we know that we want R to hold:

$$X_{mesh1} = RX_{mesh2}$$

If X_{mesh1} is a vector with three components

$$X_{mesh1} = \begin{pmatrix} x_{1x} \\ x_{1y} \\ x_{1z} \end{pmatrix} X_{mesh2} = \begin{pmatrix} x_{2x} \\ x_{2y} \\ x_{2z} \end{pmatrix}$$

then we can get the equations from the rotation of the first axis:

$$x_{1x}a_{11}x_{1y}a_{12}x_{1z}a_{13} = x_{2x}$$

$$x_{1x}a_{21}x_{1y}a_{22}x_{1z}a_{23} = x_{2y}$$

$$x_{1x}a_{31}x_{1y}a_{32}x_{1z}a_{33} = x_{2z}$$

As we have two more axis we end up with 9 equations and 9 unknowns from R. We can use linear algebra to find R solving $Ax = b$ where:

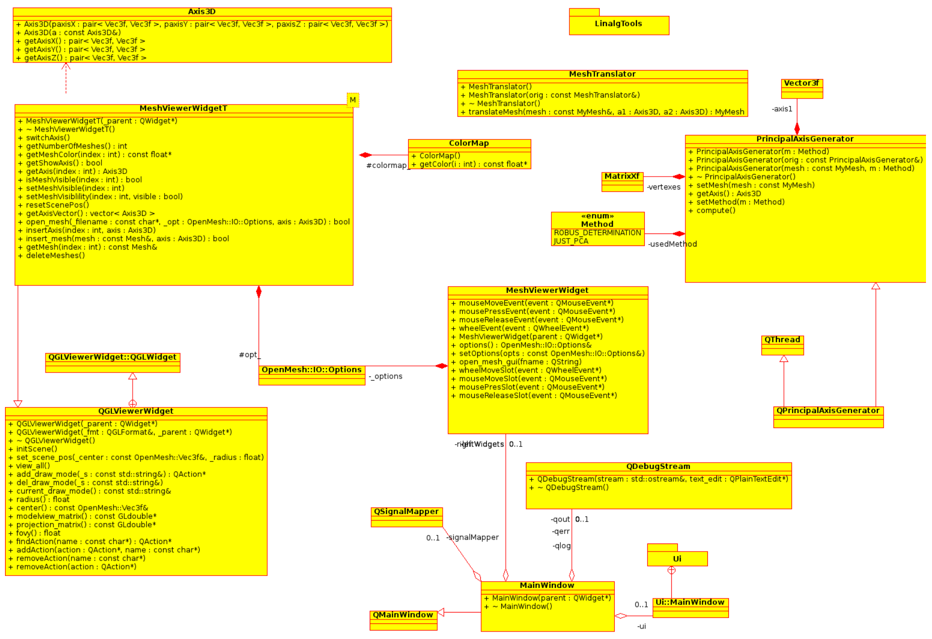
$$A = \begin{pmatrix} x_{1x} & x_{1y} & x_{1z} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{1x} & x_{1y} & x_{1z} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x_{1x} & x_{1y} & x_{1z} \\ y_{1x} & y_{1y} & y_{1z} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & y_{1x} & y_{1y} & y_{1z} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & y_{1x} & y_{1y} & y_{1z} \\ z_{1x} & z_{1y} & z_{1z} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & z_{1x} & z_{1y} & z_{1z} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & z_{1x} & z_{1y} & z_{1z} \end{pmatrix}$$

$$x = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \\ a_{33} \end{pmatrix} \quad b = \begin{pmatrix} x_{2x} \\ x_{2y} \\ x_{2z} \\ y_{2x} \\ y_{2y} \\ y_{2z} \\ z_{1x} \\ z_{1y} \\ z_{1z} \end{pmatrix}$$

6.3 Class design

In this section we are going to make a brief explanation of the relationship between all the classes we used in this projects, the main role each class have and what does it represent.

6.3.1 Class diagram



Axis3D
+ Axis3D(paxisX : pair< Vec3f, Vec3f >, paxisY : pair< Vec3f, Vec3f >, paxisZ : pair< Vec3f, Vec3f >)
+ Axis3D(a : const Axis3D&)
+ getAxisX() : pair< Vec3f, Vec3f >
+ getAxisY() : pair< Vec3f, Vec3f >
+ getAxisZ() : pair< Vec3f, Vec3f >
LinalgTools
MeshTranslator
+ MeshTranslator()
+ MeshTranslator(orig : const MeshTranslator&)
+ ~ MeshTranslator()
+ translateMesh(mesh : const MyMesh&, a1 : Axis3D, a2 : Axis3D) : MyMesh
Vector3f
-axis1
M
MeshViewerWidgetT
+ MeshViewerWidgetT(_parent : QWidget*)
+ ~ MeshViewerWidgetT()
+ switchAxis()
+ getNumberOfMeshes() : int
+ getMeshColor(index : int) : const float*
+ getShowAxis() : bool
+ getAxis(index : int) : Axis3D
+ isMeshVisible(index : int) : bool
+ setMeshVisible(index : int)
+ setMeshVisiblility(index : int, visible : bool)
+ resetScenePos()
+ getAxisVector() : vector< Axis3D >
+ open_mesh(_filename : const char*, _opt : OpenMesh::IO::Options, axis : Axis3D) : bool
+ insertAxis(index : int, axis : Axis3D)
+ insert_mesh(mesh : const Mesh&, axis : Axis3D) : bool
+ getMesh(index : int) : const Mesh&
+ deleteMeshes()
#colormap_
ColorMap
+ ColorMap()
+ getColor(i : int) : const float*
PrincipalAxisGenerator
+ PrincipalAxisGenerator(m : Method)
+ PrincipalAxisGenerator(orig : const PrincipalAxisGenerator&)
+ PrincipalAxisGenerator(mesh : const MyMesh, m : Method)
+ ~ PrincipalAxisGenerator()
+ setMesh(mesh : const MyMesh)
+ getAxis() : Axis3D
+ setMethod(m : Method)
+ compute()
MatrixXf
-vertexes
«enum»
Method
ROBUS_DETERMINATION
JUST_PCA
-usedMethod
MeshViewerWidget
+ mouseMoveEvent(event : QMouseEvent*)
+ mousePressEvent(event : QMouseEvent*)
+ mouseReleaseEvent(event : QMouseEvent*)
+ wheelEvent(event : QWheelEvent*)
+ MeshViewerWidget(parent : QWidget*)
+ options() : OpenMesh::IO::Options&
+ setOptions(opts : const OpenMesh::IO::Options&)
+ open_mesh_gui(fname : QString)
+ wheelMoveSlot(event : QWheelEvent*)
+ mouseMoveSlot(event : QMouseEvent*)
+ mousePresSlot(event : QMouseEvent*)
+ mouseReleaseSlot(event : QMouseEvent*)
QGLViewerWidget::QGLWidget
#opt_
OpenMesh::IO::Options
-_options
QThread
QPrincipalAxisGenerator
QGLViewerWidget
+ QGLViewerWidget(_parent : QWidget*)
+ QGLViewerWidget(_fmt : QGLFormat&, _parent : QWidget*)
+ ~ QGLViewerWidget()
+ initScene()
+ set_scene_pos(_center : const OpenMesh::Vec3f&, _radius : float)
+ view_all()
+ add_draw_mode(_s : const std::string&) : QAction*
+ del_draw_mode(_s : const std::string&)
+ current_draw_mode() : const std::string&
+ radius() : float
+ center() : const OpenMesh::Vec3f&
+ modelview_matrix() : const GLdouble*
+ projection_matrix() : const GLdouble*
+ fovy() : float
+ findAction(name : const char*) : QAction*
+ addAction(action : QAction*, name : const char*)
+ removeAction(name : const char*)
+ removeAction(action : QAction*)
-rightWidgets
-leftWidgets
0..1
QDebugStream
+ QDebugStream(stream : std::ostream&, text_edit : QPlainTextEdit*)
+ ~ QDebugStream()
-qout
0..1
-qerr
-qlog
QSignalMapper
0..1
-signalMapper
Ui
MainWindow
+ MainWindow(parent : QWidget*)
+ ~ MainWindow()
0..1
Ui::MainWindow
-ui
QMainWindow

Figure 3: Class diagram

6.3.2 QGLViewerWidget

This class belongs to the OpenMesh sample code. Inherits from QGLWidget wich is a Qt class that uses OpenGL.The main goal of this class is to deal with projection and model view matrices, mouse and keyboard events and handle points viewing types.

This class makes it easier to render objects creating a layer over QGLWidget. We only needed to slightly modify this class adding one method in order to be able to reset the model view matrix and scene position. Adding this method the different viewers can have the same position in both viewers as we need them to move synchronized.

6.3.3 MeshViewerWidgetT

This class belongs to the OpenMesh sample code. Inherits from QGLViewer-Widget. It is a template class in order to be able to handle several types of polygons. The main goal of the class is handle the points, faces and axis rendering of the meshes and compute its bounding box.

We had to apply a lot of modifications in this class in order to make it support more than one mesh. We also added the possibility to display the axis of every mesh. Initially this class had support for textures but we took it off as we do not need it for our purposes and would make the code more complicated unnecessarily.

As all the meshes have no default color assigned we also added a feature in order to have a color map in the case the mesh has no color assigned.

6.3.4 MeshViewerWidget

This class belongs to the OpenMesh sample code. Inherits from MeshViewerWidgetT but it sets the template to a triangle mesh representation. It also inherits from QObject to be able to handle signals and slots. This class is necessary because template classes cannot handle signals and slots that are necessary for the Qt objects interaction.

We applied several modifications in this class adding the necessary signals and slots to the mouse events in order to connect the viewers events.

6.3.5 ColorMap

The goal of this class is to provide with a color map. Given an index this class would return a color. The color map is hard-coded and has been generated with Matlab. The difference between colors with a similar index are very different. This way the consecutive meshes can have different colors making it easier to compare.

6.3.6 Axis3D

The goal of this class is provide with a data structure to store the origin and the direction of the 3D axis.

6.3.7 QDebugStream

This class is provided by [10] . This class inherits from $std :: basic\_streambuf < char >$ its purpose is to redirect the output of STD to a Qt object. Using this class we are able to redirect the console output to a text box in the main application.

6.3.8 LinalgTools

In this namespace used to gathered all the Linear Algebra and Math functions. There we can find functions for:

- Prjection onto a plane
- Projection onto a line
- Mean and Median
- Sorting
- Pint rotation and translation

6.3.9 PrincipalAxisGenerator

This class is in charge of computing the principal axis of a cloud of points. We have two different methods implemented:

1. Normal PCA method: this method only applies PCA to the cloud of points to get the first axis. Then it projects all the points to the plane made up of the mean point and the principal axis as its normal. It is used to compare results.
2. Robust axis: this is the method described in [1].

6.3.10 QPrincipalAxisGenerator

This class inherits from PrincipalAxisGenerator and QThread in order to be able to execute the axis comput function in a secondary thread. This is necessary because if an operation that is time consuming then it won't let the main application tho update the window, which would lead to think that the application is frozen. Executing the time consuming operation in a secondary thread this problem is solved.

6.3.11 MeshTranslator

This class is used to translate and rotate all the points from one mesh to be aligned with the main axis of an other mesh.

6.3.12 MainWindow

This class is the Viewer class of the Qt design pattern. This class works as the main applications and ties all the classes together, connects all the signals and slots of the different widgets. This class is attached to an .ui Qt form to create the graphical interface.

7 Results

As we can see in the figure 4 the Robust axis method is more invariant to modifications on the object. Looking at the axis we can see that PCA axis are influenced by the movement of the arms. However The axis of the Robust axis method does not get altered by this modification.

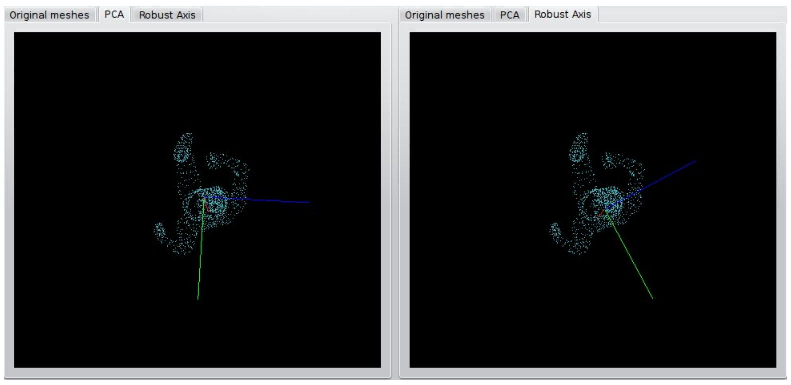


Figure 4: PCA, Robust axis method: Comparing computed axis for a single mesh

Analysing the images from the database we found out that in the meshes in which the noise is compensated PCA works better than robust axis as can be seen in figure 5. After applying rotation and translation to the objects we can see that the PCA meshes are overlapped almost perfectly. Whereas the Robust axis method look shifted.

We run some tests with meshes in which we cut some points out of the mesh. In the figure 6 we took a mesh of a camel and deleted two legs of it. We can see that PCA method does not get a good alignment but Robust axis makes almost a perfect match. However when the number of points deleted is higher even Robust axis method does not achieve a good match as the centroid varies too much, see figure 7. Although we think that the result is still good in therms of direction as shifting the centroid you would get a perfect match as can be seen in the figure.

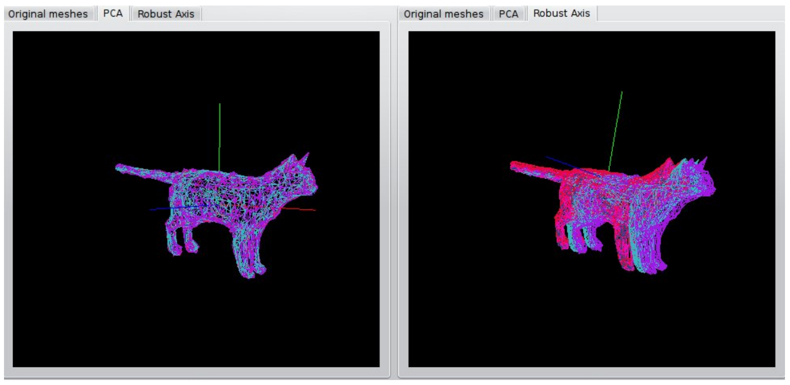


Figure 5: PCA, Robust axis method: Comparing compensated noise alignment

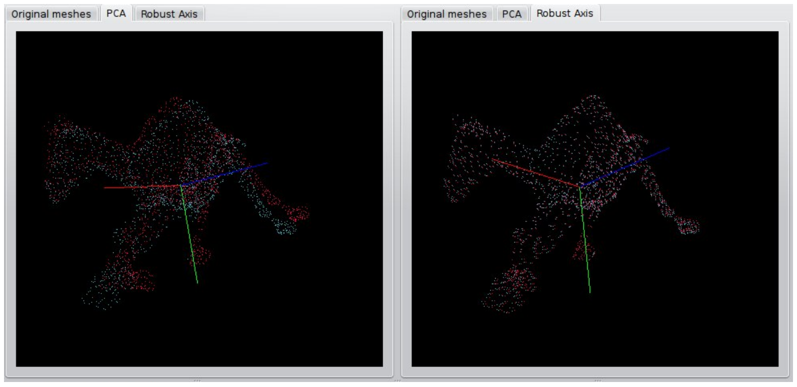


Figure 6: PCA, Robust axis method: Comparing partial mesh suppression

When trying to align the objects we found out a problem regarding to axis orientation. As you cans see in figure 8 the directions of the axes obtained are good. However as the sign of some of them is different which leads to a rotation when aligning them so they cannot be compared. We found a way to solve the sign problem for the main axis but solving it for the second axis was more difficult than expected.

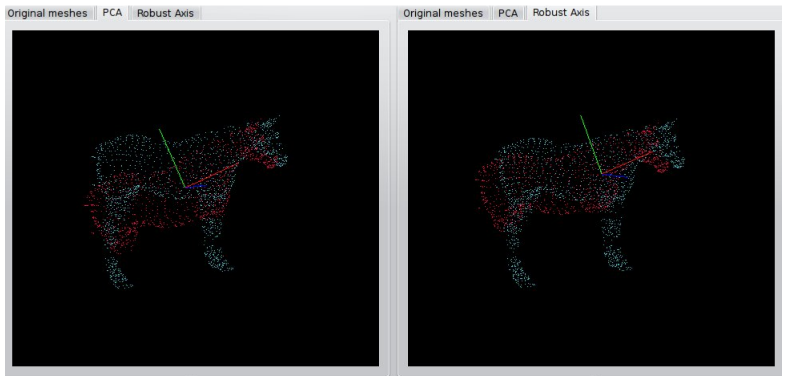


Figure 7: PCA, Robust axis method: Comparing partial mesh suppression

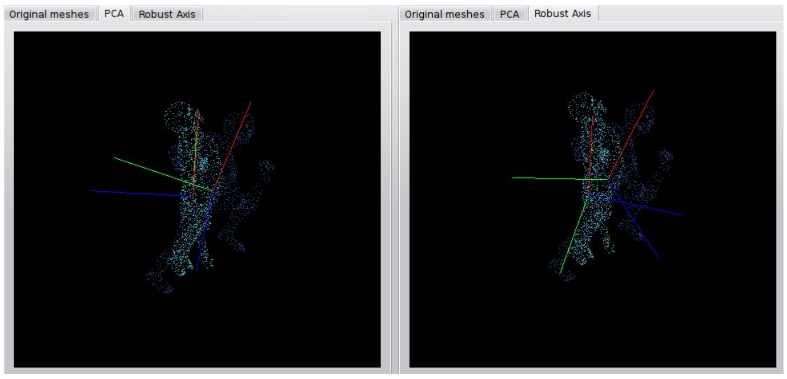


Figure 8: PCA, Robust axis method: Wrong axis sign

8 Conclusions

During the coding process we had to redefine some requirements as they would carry much more workload than expected. We found usefully to make the decision of following a methodology because this way the next step for the working process was always clear. What is more the need to have periodically meetings forced us to dedicate a regular amount of time in the project.

As we explained in the results section we found out that Robust axis method gives better results but with the compensated noise. Firstly that made us think that would be due to an error in our implementation. However looking deeply in the implementation and the different results given by trying different kinds of noise, we could not find any error.

9 Future work

Although we managed to get some results we think that some work can be done in order to improve our final application.

The problem of the second axis sign needs to be solved in order to get a proper alignment with all the meshes.

To emprove the execution time we should implement the algorithm using the octree structure. This way less iterations in the RANSAC algorithm should be needed and less distances calculated.

Referring with the graphical interface some controls could be added in order to be able to just delete one mesh instead of having to delete all of them. It could also be a good improvement to make it able to undo the mesh alignment as now if the user wants to un-align the meshes, he has to compute the axis again which can be very time consuming if the mesh is big.

It also would be a good idea to test its robustness between different kinds of meshes more than triangle meshes as the viewer is build over a template class that can handle any polygonal mesh. However we have not run any test in those kinds of meshes.

Inheritance should be applied to MeshTranslator adding it Qt Thread support as we have seen that when aligning a large number of meshes the main window would seem frozen. We applied this solution to the mesh axis computer class as it is more time consuming. Although with the mesh translation only happens with large number of meshes is a bug that should be fixed in order to improve the application fluency.

References

[1] Y.S. Liu and K. Ramani. Robust principal axes determination for point-based shapes using least median of squares. *Computer-Aided Design*, 41(4):293–305, 2009.

[2] Agile Alliance. Manifesto for Agile Software Development. June 2010.

[3] http://eigen.tuxfamily.org/.

[4] http://www.openmesh.org/.

[5] http://qt-project.org/.

[6] http://www.sgi.com/products/software/opengl/license.html.

[7] http://www.opengl.org/.

[8] Eduardo Poyart. https://github.com/Eduardop/Octree. 2012.

[9] http://codingplayground.blogspot.fr/2010/01/pca-dimensional-reduction-in-eigen.html.

[10] Username: Joonhwan. http://stackoverflow.com/questions/12978973/unknown-ouput-with-qdebugstream-and-qtextedit.