Cancellous bone : Extract structure information from 3D binary images

Jérôme Bouzillard

March 25, 2014

Client	Pascal Desbarats

Abstract

The goal of this project is to provide information about the structure of cancellous bone using segmented 3D binary images given by Computed Tomography. The main part of the process is to compute the skeleton of tubular object (like cancellous bone), a frequently used shape feature that represent the general form of an object, and that we need for the structure analysis. A simple graph (tree) can be then calculated to identify nodes and trabeculae from the skeleton. Thus, we can add all the features needed for the structure analysis. The source code is done in C++, without any dependance, given as a quite simple library, and is originally intended for the *LaBRI*, a research unit associated with the CNRS (UMR 5800), the University of Bordeaux and the IPB.

Contents

1 Introduction

There are two major ways today to explore the interior of bones in medicine: Biopsy, which is quite destructive, and the more recent 3D micro Computed Tomography (microCT) which is non-destructive (not more than radiography). In many cases, we can identify health problems by analyzing the structure of the cancellous bone, which doesn't need biopsies. The main example is for osteoporosis.

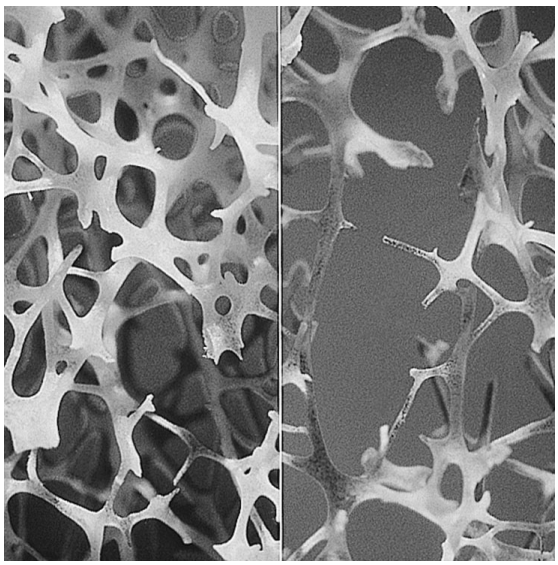

Figure 1: Microarchitecture of bone in health (left) and in osteoporosis (right). Source: Wikipedia[1]

To analyze the cancellous bone structure, the microCT images are segmented with a threshold, to binarize the 3D image, that separates bone from the marrow. In bone research, several key measures are usually done, such as bone volume ratio (BV/TV), Trabecular Thickness (Tb.Th), Trabecular Spacing (Tb.Sp), and Trabecular shape (rods/plates). We can also measure connectivity, number of trabeculae, average trabecular length, anysotropy, etc. Given the 3D binary image, providing those measures requires first a skeleton of the tubular structure, and the skeletonization process is the main step of this project.

First of all, I am going to state the specifications of the project given by the *LaBRI*, after which I will explore the state of the art in 3D skeletonization and explain the choices made to fit the needs, then I will present the architecture of my solution, afterwards I will explain in details the algorithms chosen and developped in the project, and at last, I will talk about tests, limitations, and future work.

[1] http://en.wikipedia.org/wiki/Trabecular_bone_score

2 Functional Requirements

First of all, there is not any non-functional needs required in the specification, anyway, there was no way to develop and implement something that is much slower or heavier in memory than what currently exists on the state of the art. However, there is no particular constraint about real time, and the images provided are small enough for the memory use in any case.

Let's talk about the specification itself:

- Images reader : The 3D segmented images given are in a format called Analyze (a format widely used in medical imaging and detailed in the Implementation section). We have to read the binary data from these files.

- Skeleton : The ImageJ plugin used at the *LaBRI* provides a skeletonization technique that extract mostly medial surfaces on the cancellous bone samples, the core of this project is to extract only one voxel width skeleton from our 3D images.

- Graph : Once the skeleton is done, we need to make it as a structure, to identify the trabeculae and their junction, this will be done by building a clever graph of the skeleton and is another important part of the project.

- Measures on cancellous bone sample: This part is what this project will intend to provide thanks to the skeleton/graph. These measures includes simple information:
  - Bone volume ratio (BV/TV)
  - Number of trabeculae
  - Average trabecular length
  - Connectivity histogram (number of trabeculae on junctions)

  And more elaborated information:
  - Average Trabecular Thickness (Tb.Th)
  - Average Trabecular Spacing (Tb.Sp)
  - Average Trabecular shape (rods/plates)
  - Anisotropy of the object

The main risk of this project is to not be able to compute some of the required measures with the extracted skeleton, anyway, it is always possible to provide in the future different kind of skeletonization to the functions.
In bone research, most of above measures are standard for medical applications. Let's explore now the state of the art about the core need of the project, namely 3D skeletonization.

3 State of the art

The aim of the skeletonization process is to extract a region-based shape feature representing the general form of an object. It generally provides medial lines, or medial surface, from the original object.

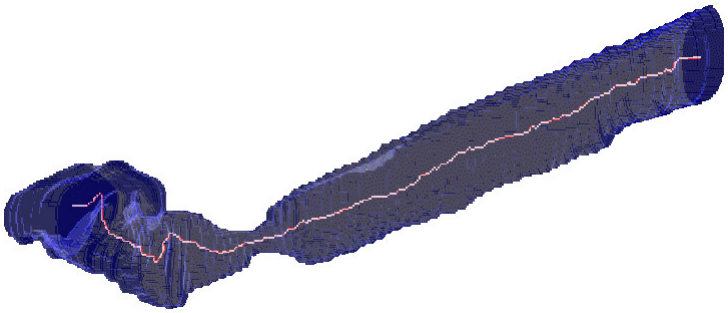

Figure 2: Skeletonization that extract the central path of a 3D "tubular" object. Source: K. Palágyi's publication[1]

There are three major skeletonization techniques:

- detecting ridges in distance map of the boundary points
- computing the Voronoi diagram generated by the boundary points
- the layer by layer erosion, called thinning.

Since we work in digital spaces, only an approximation to the true skeleton can be extracted. There are two requirements to be complied with:

- topological (keep the topology of the original object)
- geometrical (invariance under translation, rotation, and scaling)

Table 1: Comparison of skeletonization techniques

Method	Geometrical	Topological
distance transform	Yes	No
Voronoi	Yes	Yes
thinning	No	Yes

As the goal of the project is to provide structural information about our tubular objects (cancellous bone), we need to get a skeleton that preserves the topology of the original object, therefore the distance transform method. Moreover, the Voronoi skeletonization is an expensive process. Thus, the thinning method seems best to fill the needs in bone research.

Many thinning algorithms has been proposed in the last 20 years. Most of them extract medial lines and medial surfaces depending on the shape of the object, it is the case of the skeletonization plugin from MATLAB and also the software ImageJ[2], a Java-based image processing program developed at the *National Institutes of Health*, and used by the *LaBRI*. Since one of the needs of the project is to do the calculations using a one voxel width skeleton, we keep only the curve thinning algorithms, which provides only medial lines of the original object.
Most of the curve thinning algorithms are designed for parallel implementation, however, the deadline of the project being short, I decided to adopt and implement an interesting sequential curve thinning algorithm, proposed by *K. Palágyi*[1].

4 Architecture

Despite the fact this project deals mainly with utilities and procedures, I decided to make a simple library, which provides a Tubular Object class, with various member functions.
This include the load from a file, computation of the skeleton of this object, writing the skeleton data in a new file, building a graph, doing some of the measures we need for the structure analysis, and printing those on a text file.
Let's give a class diagram of this architecture:

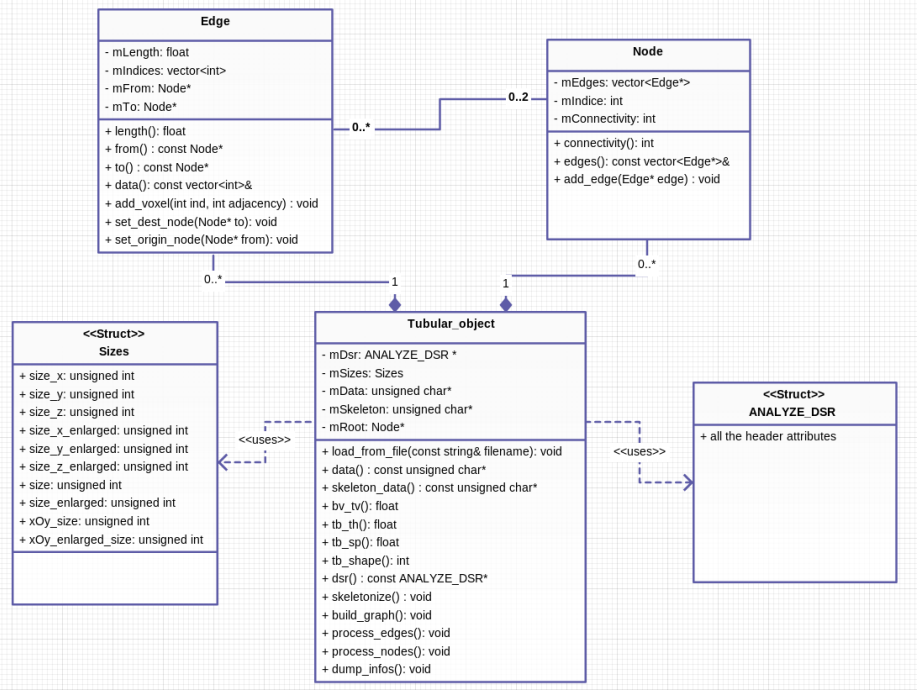

Figure 3: Class diagram

Given this structure, a typical User program could be presented as simple as this :

```
Tubular_object : new()
Tubular_object : load_from_file(myAnalyzeImagePath)
Tubular_object : skeletonize()
Tubular_object : build_graph()
Tubular_object : bv_tv()
Tubular_object : tb_th()
Tubular_object : dump_infos()
```

The current architecture could have been stronger, but the core of the project was about the development of the algorithms, from which I am going to give detailed explanations now.

5 Implementation

5.1 Load from a file

The segmented images given by the *LaBRI* is in Analyze 7.5 format. This format was developped by *Biomedical Imaging Resource (BIR)* for their software package and is commonly used in medical applications, such as microCT.
One Analyze item consists of two files: One file with the actual data in a binary format with the filename extension .img and another file (header with filename extension .hdr) with information about the data such as voxel size and number of voxel in each dimension.
A detailed description of this format is given on Sourceforge[2].
To gain time, I decided to use a simple and short C source code from LPGL licensed library *libtpcimgio* [3] which read Analyze files. I added to this code a function to write image data in a file (the function only write data from unsigned char values).
When loading is done, the data and the header descriptor are stored in our Tubular Object and we may proceed to skeletonization, or direct measures that don't need it.

5.2 Skeletonization

This process is the main and longest part of the project. It is an implementation of a sequential 3D curve-thinning algorithm proposed by *K. Palágyi*[1]. I am going to give you only the main steps of my implementation since the detailed algorithm can be found on the K. Palágyi publication[1]

The algorithm works just as peeling an apple: We peel the apple until no skin remains. The algorithm repeats 6 subiterations consisting of thinning the skin from the 6-adjacency directions (See FIGURE 4). To do so, we simply check all the voxels that are border points to the current direction and delete them if they satisfy topological conditions. (See FIGURE 5)

[2] http://eeg.sourceforge.net/ANALYZE75.pdf
[3] http://www.turkupetcentre.net/software/libdoc/libtpcimgio/index.html

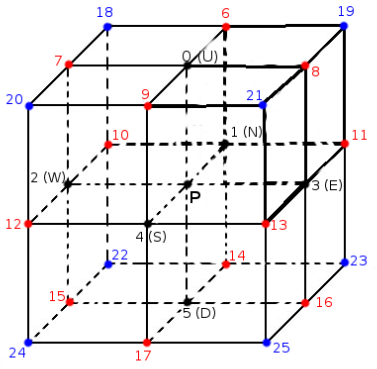

Figure 4: The 26-adjacent neighbours of a 3D point P. The 6-adjacency neighbours contains black points marked Up, North, West, East, South, Down. The 18-adjacency neighbours contains these points and the 12 red points. The 26-adjacency neighbours contains all of those points and the 8 blue points. Each of the neighbours have a unique indice i from 0 to 25.

Unit Test : Every step of the algorithm have been tested with a computed Cylinder and displayed with a personal openGL visualizer to validate those steps, including end points and border points conditions (See FIGURE 5), and each iterations.

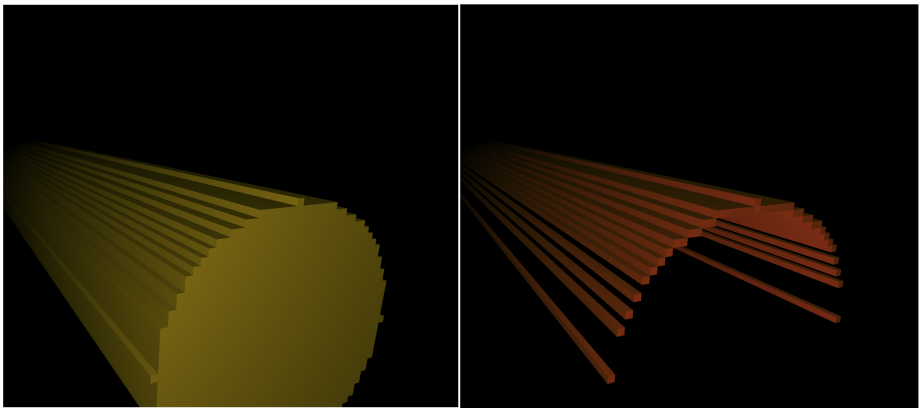

Figure 5: A fullfilled Cylinder (left) and its U-border points (right).

Since we want a curve skeleton, we need topological conditions from which medial lines will be naturally extracted by the thinning process:

- First, since a line has 2 end points (points with only one neighbour), we don't want to delete the endline points, or the thinning would remove everything, therefore each marked point for deletion have to not be an endpoint to be deleted.

- Second, and the key condition of this algorithm, the border points in a subiteration marked for deletion must remains a simple point to be deleted.

A point is called simple point if its deletion does not alter the topology of the object. In fact, it means 2 things:

- All the object points in 26-adjacency must remain connected after the marked point is deleted. For instance, in FIGURE 4, let's say the point P has 2 neighbours with indice 19 and 24, when we delete the point P, 19 and 24 are disconnected, changing the topology.

- All the background points in 6-adjacency must be already connected in 18-adjacency before deleting the marked point. For instance, imagine the Up and Down points of the FIGURE 4 are background points, and are not connected, it means that our point P potentially separates exterior from interior, and deleting it would fusion both, changing the topology.

Again, for more details about those definitions, see the K. Palágyi publication[1].

Checking if points are simple are done with recursive functions that visit all the neighbours. These functions use an optimised static array storing the neighbours of each neighbours of point P.

Unit Test : Most of the tests were done on the simple point condition, given several neighborhood configuration, the 2 conditions of the simple point have been validated.

When the skeletonization is complete, we have a one voxel width skeleton from which data is stored, and can be saved and displayed as shown in FIGURE 6

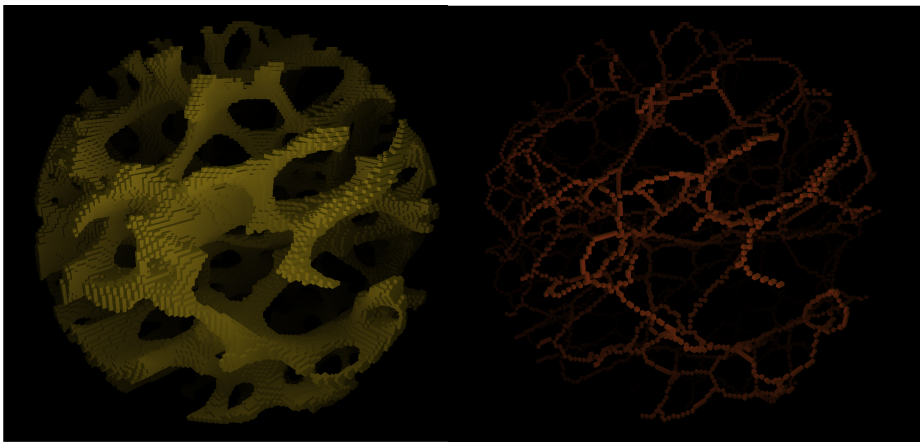

Figure 6: Cancellous Bone sample from segmented binarized image data (left) and its Skeleton (right) from my implementation of this algorithm[1].

5.3 Graph

When we have the skeleton data, we need to identify the nodes and the edges, which define the structure of cancellous bone, each node being a junction of trabeculae, the edges. The graph was also a long but shorter part of this project. It is a simple and non-cycle graph (a tree) computed in 3 different passes.

Before I explore them, let's define some properties to build the graph.
We can separate 3 kinds of voxel in our skeleton :

- Endpoint : already defined, endpoints are the end part of branches (edges connected to only one node).
- Edge point : a point which have 2 and only 2 neighbours.
- Node point : a point which have at least 3 neighbours.
- First pass: Nodes and Edge first identification.

  This pass of the algorithm compute a depth-first search that creates edges and nodes, each voxel being identified to an edge or a node. While edges are built in a sequential loop that just find the next non visited voxel of the edge, the nodes are a set of voxels that are built with a breadth-first search algorithm. (A voxel without neighbours is an isolated point, and is therefore never taken into the graph)

  Unit Test : check if all the skeleton voxels are identified with an edge or a node (again, isolated points are ignored). It is currently assumed that there is only one skeleton in the image, with no disjoint sets of voxels, this test then fails everytime there are disjoint sets of voxels in the skeleton, since only one graph is computed at the moment.

- Second pass: Refine nodes to the minimum number of voxels.

  The first pass could be sufficient to build a graph, but is not satisfactory enough. Indeed, most of the nodes have more voxels than necessary and have to be refined. As shown on Figure 7, 5 voxels are identified as nodes because they have each at least 3 neighbours. However, only one voxel is needed to connect the 4 edges. This pass removes the unnecessary node voxels and extend the corresponding edges to these voxels (or simply delete them if they are obsolete).

  Unit Test : Each node should keep the same number of edge junctions as before, also every edge extended should still be one voxel width.

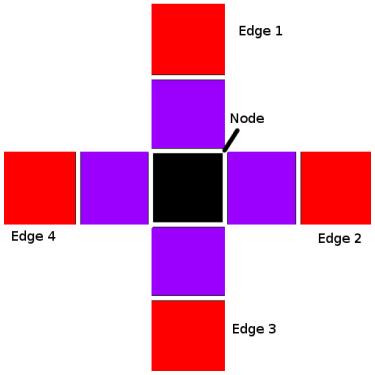

Figure 7: Second pass : the purple voxels were part of the node because they each have at least 3 neighbours. They all become extension of the 4 edges, the node being refined to the center voxel only.

- Third pass: Remove unwanted branches and node fusion.

  We need to perform another pass due to the skeletonization process which bring noisy branches (ending edges with length shorter than a threshold : $2\sqrt{3}$ by default) to the computed skeleton, creating unnecessary nodes. More, edges between 2 nodes with a length shorter than another given threshold($2\sqrt{3}$ by default) should be actually considered part of one single node containing the edge and node voxels.

  This last pass try to remove the noisy unwanted branches from the skeleton and fusion nodes that are separated by too small edges:
  - If the edge has an endpoint, the edge is simply removed, and if the node has only 2 remaining edges connected, all the voxels from these edges including the node forms only one edge (See FIGURE 8)
  - If the edge is between 2 nodes, then all the voxels including the 2 nodes forms only one node. (See Figure 9)

  Unit Test : There is not any node with less than 3 neighbours, 2 nodes cannot be connected, and 2 edges cannot be connected without a node, this is what have to be checked after this last pass.

Once these 3 passes are done, a depth-first search algorithm is performed to build the tree, starting from a root node and visiting only once each edges and nodes. A graph traversal function is implemented and provided also to the user. The connectivity of each nodes and the length of each edges are computed during this graph construction phase.

Unit Test : We simply check that every voxels, edge and nodes, are visited during the graph traversal, we also check the node connectivity attribute is the value expected.

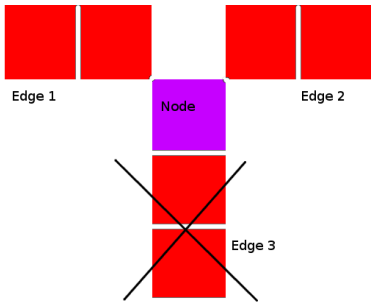

Figure 8: Third pass : Edge 3 is shorter than a threshold, it is removed, the node having only 2 remaining neighbours, Edge 1, Edge 2 and Node become all voxels of Edge 1.

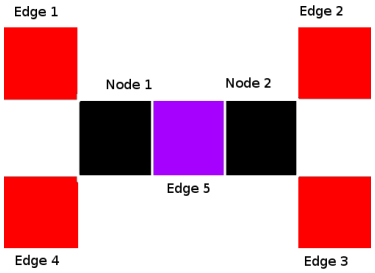

Figure 9: Third pass : Edge 5 is too short to be considered an Edge. Node 1, Edge 5 and Node 2 become 3 voxels of the unique Node 1, which is connected now to the 4 first edges.

5.4 BV/TV

BV/TV is one of the most common measure done on cancellous bone. It doesn't need a skeleton, and states the bone volume pourcentage on the total image volume. Since the 3D microCT images provided represent a sphere and not the cube dimensions of the image, the total volume is given by the sphere volume formula $\frac{4}{3}\pi r^3$, where the radius "r" is half of the image width dimension. The bone volume is meanwhile a simple sum of all the bone voxels in the image.

Unit Test : We fill the data with an entire sphere of bone voxels to check the ratio being equal to 1.

5.5 Average Trabecular length

Length of trabeculae are computed during the graph building phase. The average, minimum, maximum and deviation length of the trabeculae are calculated by the function "average_trabecular_length()" which call the graph traversal function to access all of the edges. The length of each edges are precomputed during the graph building phase:

Each time a voxel is added to an edge, we check its adjacency to the previous voxel. Considering that a voxel width is unit, the distance in 6-adjacency, 18-adjacency and 26-adjacency are different (See FIGURE 10). The length is then simply the edge unit length multiplied by the physical voxel width given in the Tubular Object descriptor (stored from the Analyze header file).

This function assumes that a voxel is isotropic, meaning that physical width, height and depth from the Analyze Header are the same.

Unit Test : We create data of a single edge and build it, then the length member function should return the expected value.

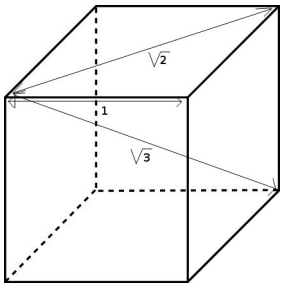

Figure 10: The length of an edge depends on the adjacency of its voxels

5.6 Number of Trabeculae and Connectivity

This value is computed by a simple counter of the edges when performing the graph traversal function. As regards the connectivity, a simple table contains for each value the number of nodes that have a certain number of neighbours (most of them will have 3 neighbours, rarely 5 or 6, and as far as I tested on the given images, never more than 6). This is also computed with the graph traversal function.

Unit Test : No tests were done on this method, only comparison with internet database or research to validate it.

6 Tests, limitations, and future work

6.1 Memory

A standard valgrind memory check have been done during the typical user program execution:

```
" valgrind -- tool = memcheck -- leak - check = full "
HEAP SUMMARY:
==31713==     in use at exit: 7,989,272 bytes in 12,579
    blocks
==31713==   total heap usage: 528,749 allocs, 516,170 frees,
     34,776,703 bytes allocated

LEAK SUMMARY:
==31657==    definitely lost: 32 bytes in 1 blocks
==31657==    indirectly lost: 395,840 bytes in 12,575 blocks
==31657==      possibly lost: 7,592,832 bytes in 2 blocks
==31657==    still reachable: 568 bytes in 1 blocks
==31657==         suppressed: 0 bytes in 0 blocks
```

Most of the lost memory is due to the lost of Nodes and Edge pointers that are not destroyed in the graph when we destroy our Tubular Object, this function is on doing, or will have to be done.

6.2 Performance

Since the images given from the *LaBRI* are all defined with 154x154x154 dimensions, the execution took in all the cases less than 5 seconds to complete, with quad core i7 960, on Unix 64 bits.

6.3 Quality of Results

As mentionned by the client, the measures done for structure analysis can be used in a comparative way and thus, do not have to reflect the true values. However, I tried along this project to get decent and consistent results (comparing them with internet database to validate its quality). For instance, the graph construction has been set such that all the simple measures should be closer to the reality. Also, as far as I am concerned, I think after many attempts and tests, the skeletonization implementation works as intended.

6.4 Limitations

Since the code is not that big and architecture is extremely simple, with no high dependencies, everything can be reworked with no particular difficulties if needed. Nevertheless, there are several limitations on the current implementation, from which I can enumerate a few:

- At this moment, it is not possible to have multiple graph on one 3D Image yet, and this can make complete irrelevant results if the image contains 2 big disjoint objects.
- It is also not possible with the current implementation to compute the real edge length if the voxel physical dimensions are not isotropic.
- The bone volume ratio function assumes the 3D images taken from microCT to be contained in a sphere, if this could be not the case, the current implementation don't handle it.

6.5 Future Work

The 4 elaborated measures presented in the specification couldn't be done and must be features to add and implement in the future. (Average Trabecular Thickness (Tb.Th), Average Trabecular Spacing (Tb.Sp), Average Trabecular shape (rods/plates), Anisotropy of the object). All the remaining work that have to be done to end the project have been also added in the code with TODOS comments. The initial goal of the project was to build it as a simple library which could still be done in a near future. Finally, if this curve skeletonization procedure works fine to establish classification of rod-like and plate-like trabeculae, the project could open its application to something else than cancellous bone or generally any tubular structures.

References

[1] K. Palágyi, E. Sorantin, E. Balogh, A. Kuba, Cs. Halmai, B. Erdôhelyi, K. Hausegger *: A sequential 3D thinning algorithm and its medical applications.* Department of Applied Informatics, University of Szeged, Hungary, 2001.

[2] Ta-Chih Lee and Rangasami L. Kashyap *: Building skeleton models via 3-D medial surface/axis thinning algorithms.* Computer Vision, Graphics, and Image Processing, 56(6):462–478, 1994.

[3] Saha PK, Xu Y, Duan H, Heiner A, Liang G. *: Volumetric topological analysis: a novel approach for trabecular bone classification on the continuum between plates and rods.* Proc. SPIE 7259, Medical Imaging: Image Processing, 725950 2009.