Title:
Efficient Voxelization-Based Construction of Finite Element Meshes Originated from Micro-Computed Tomography Data

Authors:
Mohammadreza Faieghi, mfaieghi@uwo.ca, Western University, London, ON, Canada
Nikolas Knowles, nknowle@uwo.ca, Western University, London, ON, Canada
O. Remus Tutunea-Fatan, rtutunea@eng.uwo.ca, Western University, London, ON, Canada
Louis Ferreira, louis.ferreira@sjhc.london.on.ca, Western University, London, ON, Canana

Keywords:
Micro Finite Element Model (μFEM), Micro Computed Tomography (μCT), Hexahedral Mesh, Cartesian Mesh, Voxelization

DOI: 10.14733/cadconfP.2018.21-25

Introduction:
Micro-finite element models play a critical role in microscale structural analysis. These models are generally obtained through processing μ-CT data [7] that encompasses images with resolutions as high as few microns per pixel. The 3D geometry of the scanned sample can be reconstructed through appropriate segmentation operations that are to be performed on the μ-CT images yielded through scanning. This 3D representation of the sample becomes then the basis of the finite-element volumetric meshes to be further integrated in the subsequent μ-FEMs generated for structural analysis purposes [5]. According to the fundamentals of the finite element methods, hexahedral elements are more computationally efficient than their tetrahedral counterparts [3], such that their preferential use is desirable. However, the tools that are presently available to generate hexahedral μ-FE meshes either require considerable computing time and power [6] or are limited to rather simple geometries [1].

In addition, the development of an algorithm capable to efficiently discretize/mesh μ-CT data poses a number of challenges that are primarily derived from the size of the μ-CT data often consisting of tens of millions of elements/voxels. As it can be inferred, the processing of this amount of data is a time-consuming operation even if performed on a powerful hardware. Moreover, the available tools rely on unoptimized codes that are subjected to random crashes caused by poor/obsolete graphic memory management routines. Robust number representation techniques are mandatory in this context since the size of the numbers to be handled exacerbates all known flaws associated with the commonly employed floating point format.

Along these lines, the present study represents an attempt to propose a computationally efficient algorithm capable to generate hexahedral meshes from μ-CT data. The core idea of the algorithm is that since μ-CT data consist of uniform voxels, it could be stored in a uniform 3D grid. If this grid is determined/known, then fast and robust integer operations can be used to manage the computations to be performed on grid coordinates. Following this, since the high-resolution μ-CT voxels is considered accurate enough for μ-FEMs, the algorithm can be restricted to output only cubes/voxels as a simple hexahedral element [3], and this should result in significant reductions of the computational time. Another prominent feature of the algorithm is related to the use of hashing techniques for indexing of the nodes and materials in the FEM [8]. The implementation of the hashing techniques allows the search operations to be performed in a constant time and this in turn reduces the time complexity of the algorithm to *O(n)* thus making it extremely efficient when dealing with large data sets. As a result, the

algorithm will end up being remarkably fast even when running on a hardware characterized by average specifications.

Implementation:
Fig.1 outlines the major steps involved in meshing the μ-CT data. The input to the algorithm is a text file enclosing the center coordinates $\mathbf{c} \in \mathbb{R}^3$ along with intensity gray-values values $r \in \mathbb{R}$ for each of the μ-CT voxels obtained after the necessary segmentation and filtering operations to be followed by a 6-connectivity check. The algorithm starts with the import of the $(\mathbf{c}, r)$ pairs from the text file.

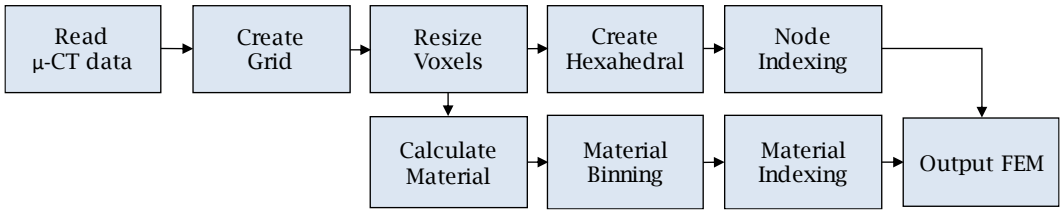


Fig. 1: Overview of the voxelization-based generation of μ-FE meshes.

Since the voxel size $\mathbf{s} \in \mathbb{R}^3$ is *a priori* known from the μ-CT resolution, the determination of the minimum and maximum of the center coordinates during file stream enables the inference of a uniform voxel grid $\mathcal{G}$ whose dimension $\mathbf{d} \in \mathbb{N}^3$ is given by the element-wise calculation of $(\mathbf{c}_{\max} - \mathbf{c}_{\min})/\mathbf{s}$. The maximum number of voxels to be held by the grid is therefore $n_{\max} = d_x d_y d_z$. Subsequently, the algorithm proceeds with the dynamic allocation of a linear array of integers, hereafter called density array characterized by an $n_{\max}$ length. The density array stores the intensity values for each voxel in a certain location. This array is populated in such a way that the memory location can be used to identify the coordinates of the voxel corresponding to that element. Specifically, let $(x, y, z) \in \mathbb{Z}^3$ ranging from $\mathbf{o}_3$ to $(d_x - 1, d_y - 1, d_z - 1)$, be the coordinates of a voxel $\mathcal{V} \in \mathcal{G}$. Then, the i-th element of the density array corresponds to a voxel $\mathcal{V}(x, y, z)$ whose grid coordinates are expressed by:

$$z = i / d_x d_y \,,\; y = \left(i - z d_x d_y \right) / d_x \text{ and } x = i - z d_x d_y - y d_x \,. \tag{8.1}$$

Conversely, a voxel $\mathcal{V}(x, y, z)$ corresponds to the i-th element number in the density array, where

$$i = x + y d_x + z d_x d_y \,. \tag{8.2}$$

Note that with the above formulation, the center coordinates of a voxel $\mathcal{V}(x, y, z)$ is determined by:

$$\mathbf{c} = \mathbf{c}_{\min} + < \mathbf{s}, (x, y, z) > \,, \tag{8.3}$$

where $< , >$ denotes the dot product. One of the major advantages of representing the μ-CT data in a grid is that the coordinate transformation can be handled by Eq. (8.1) and Eq. (8.2). These equations involve only integer-based calculations whereas the direct use of the μ-CT coordinates would set the need for floating-point operations that – given the number of decimal places in μ-CT data – is expected to be error-prone as well as slow.

If changes in voxel resolution are needed, the grid constructed in the previous step is to be passed to the so-called ***Resize Voxels*** routine that creates a new grid by placing its minimum corner at $\mathbf{c}_{\min} - \mathbf{s}/2$. If the new voxel size is $\mathbf{s}' = k^{-1}.\mathbf{s}, k \in \mathbb{N}$, then the dimension of the new grid will be $\mathbf{d}' = k\,.\,\mathbf{d}$. Then, by allocating a new density array of $d'_x d'_y d'_z$ length, the algorithm can move forward by iterating through the elements whose intensity values are correspondingly assigned. This is the case of

the dividing voxels for which the intensity of the new voxels is inherited from the parent μ-CT voxel. Note that $k \in \mathbb{N}$ ensures that $\mathbf{s}'$ is a divisor of $\mathbf{s}$ and this represents a prerequisite for volume conservation. In case of voxel merging, $\mathbf{s}'$ can be a multiple of $\mathbf{s}$ and therefore the intensity of the new voxels can be obtained by averaging the intensities of μ-CT voxels that are present in the same location.

Next, the ***Create Hexahedral*** phase receives grid information, iterates over the density array, and creates eight nodes with respect to the center coordinate of the voxels that are occupied in the grid. These nodes must follow the particular order illustrated in Fig. 2. If the number of occupied voxels is represented by n, the length of the output array will be $8n$, in which the elements from $8i$ to $8i+7$ correspond to a single voxel. At this phase, the obtained array constitutes a complete representation of the FE mesh. However, the array also includes duplicates and thereby must be separated into a unique nodes coordinates array and an indices array enclosing the topology of the geometry. This is the task of the ***Node Indexing*** phase.

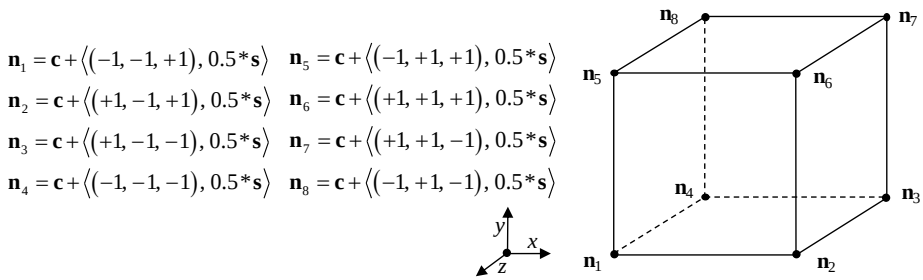


Fig. 2: Topology of the hexahedral element.

One simple way to remove duplicates is to continuously loop over the array while looking for identical instances of the element being scrutinized. However, since the average time complexity for array searching operations is *O(n)*, the worst-case time complexity associated with this procedure is $O(n^2)$. This number is clearly inefficient in case of μ-CT data encompassing millions of voxels. To circumvent this, the developed algorithm relies on a more efficient hash mapping technique. In essence, hash map is a data container that stores every node coordinates paired with a key value. These key values are computed by a hash function assigned to the hash map. Despite the linear array that stores the nodes sequentially, the hash map places the nodes according to the keys. This reduces the time complexity associated with the search for an element in the hash map to *O(1)*. By employing this idea, ***Node Indexing*** iterates over the array of the nodes and looks up duplicate nodes by means of the hash table. This reduces the time complexity of the process to *O(n)*, leading to a much higher efficiency compared the nested loop approach (Tab. 1). As such, the geometry of the hexahedral elements is fully characterized by the two arrays that have been generated at the end of the current processing phase.

Number of μ-CT voxels	***Total number of nodes***	***Number of duplicate nodes***	***Runtime (ms)***	
			Nested loops	***Hash mapping***
500	4,000	3,147	2.696	0.212
1,000	8,000	7,167	10.476	0.435
5,000	40,000	33,735	162.167	1.795
15,000	120,000	101,123	15,92.704	6.175
40,000	320,000	270,113	10,986.821	15.416

Tab. 1: Comparative assessment of hash mapping efficiency.

Once duplicates are removed, ***Calculate Material*** starts to assign material values to the hexahedral elements. This subroutine consists of a user-defined function $f(\,)$ that correlates the intensity value of each μ-CT voxel r to a material value $m = f(r)$. However, to reduce the complexity of the FEM, it is sometimes desirable to reduce the number of materials generated through this technique. To this end, materials of different Hounsfield intensities/densities are categorized in bins with a user-selected width within the ***Material Binning*** step. Once this is completed, ***Material Indexing*** pairs voxel's index and its material values and then pools these pairs by means of a hash map. By iterating over the pool, it is possible to group all the voxels enclosing the same material value to be then categorized in distinct element sets. The element sets and their corresponding material values are then combined with the output of the ***Node Indexing*** phase in order to build the final μ-FE mesh.

Results:

The performance the developed C++11 algorithm was assessed on sample μ-CT data consisting of a trabecular bone specimen and a synthetic bone model (cellular foam), both scanned at an isotropic 32μm voxel size, as well as a cadaveric glenoid specimen scanned at an isotropic 64μm voxel size. To evaluate the performance of the algorithm for clinical CT data, a human scapula model scanned at a voxel size of (0.472656mm, 0.472656mm, 1mm) was also considered. Voxel resolution was kept constant in all test cases. Material properties of the large samples (cellular foam and cadaveric glenoid) were binned with a bucket size of 10. The computing time of each step was measured with microsecond resolution by means of the chrono timer available in C++11 standard library [4]. The hardware used in all tests included a common Core-i7 6700K CPU equipped with 16 GB RAM.

Fig. 3 depicts the resulting μ-FE mesh for the analyzed samples, while the corresponding computing time is presented in Tab. 2. To eliminate confounding errors, I/O times were not considered. The results indicate that a model as complex as the glenoid (36.2M voxels) required only 55.16 s for mesh generation, whereas a scapula scanned at clinical CT resolution needed less than a second. To determine the functional relationship between model size and algorithm runtime, various decimations of the glenoid size were tested and the data plotted in Fig. 4 suggests that a linearithmic dependence between the model size and mesh generation time exists for datasets larger than $10^{4.5}$ voxels. The algorithm seems to exhibit an *O(1)* behavior for small datasets and this might be a consequence of the significant initialization overhead that is present.

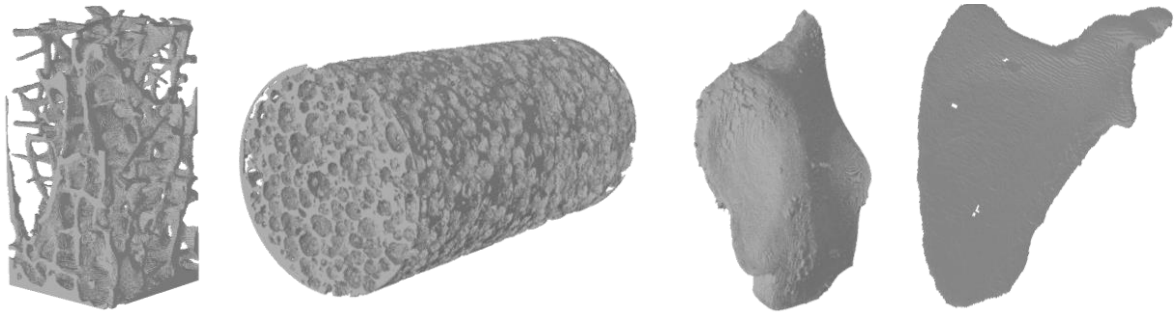
Phase	***Runtime (ms)***			
	Trabecular specimen (1.7M voxels)	***Cellular foam (12.4M voxels)***	***Glenoid (36.2M voxels)***	***Scapula (558K voxels)***
Create Voxel Grid	25.057	189.135	641.875	17.358
Create Hexahedral	196.085	1,383.37	4631.19	69.359
Nodes Indexing	1,354.804	12,888.198	37,013.259	343.325
Calculate Material	15.743	147.407	500.394	9.335
Material Binning	N/A	35.414	97.415	N/A
Material Indexing	387.306	3,301.615	12,279.429	109.241
Sum	1,978.995	17,945.139	55,163.562	548.798

Tab. 2: Runtime for different phases of the voxelization-based μ-FE generation process.

Conclusion:

The present study proposed a fast algorithm capable to generate hexahedral μ-FE meshes using data obtained from μ-CT scans. The results presented imply that the projection of CT voxels into a uniform 3D grid coupled with the use of the hash mapping techniques for searching and indexing can significantly decrease the mesh generation time as long as eight-node brick elements are used. The tests performed revealed that the time complexity of the algorithm is *O(n)* and that it can mesh a domain with 36.2M voxels in 55.16 seconds. Future developments will be focused on adaptively-sized voxels that are capable to further enhance the computational performance of the proposed technique. However,

these extensions of the current method are far from being trivial, primarily due to the hanging nodes to be present between adjacent but differently-sized hexahedrons.



(a) (b) (c) (d)

Fig. 3 Generated μ-FE mesh for: (a) trabecular, (b) cellular foam, (c) glenoid, and (d) scapula samples.

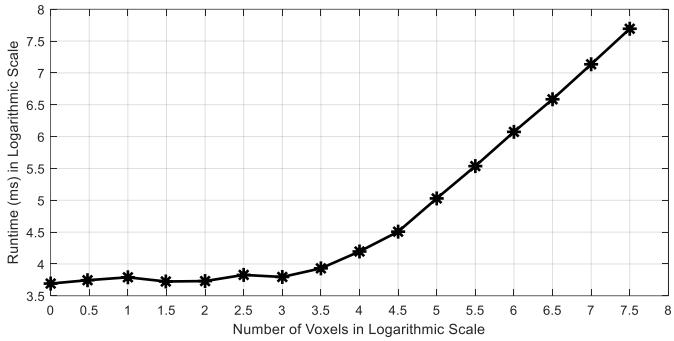


Fig. 4: Relationship between model size and μ-FE generation time.

Acknowledgement:
The work presented in this study was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Canadian Institutes of Health Research (CIHR) under the framework of the Collaborative Health Research Program (CHRP).

References:

[1] Abaqus FEA, http://www.simulia.com/
[2] Bayraktar, H. H.: Nonlinear micro finite element analysis of human trabecular bone, Circle 141-Abaqus Inc., 2004, 22-25.
[3] Ericson, C.: Real-time collision detection. CRC Press; 2004.
[4] Josuttis, N. M: The C++ standard library: a tutorial and reference. Addison-Wesley: 2012.
[5] Lacroix, D.; Chateau, A.; Ginebra, M. P.; Planell, J. A: Micro-finite element models of bone tissue-engineering scaffolds, Biomaterials, 27(30), 2006, 5326-5334. https://doi.org/10.1016/j.biomaterials.2006.06.009
[6] Simpleware FE Module, https://www.simpleware.com/software/fe-module/
[7] Verdonschot, N.; Fennis, W. M.; Kuijs, R. H.; Stolk, J.; Kreulen, C. M.; Creugers, N. H: Generation of 3-D finite element models of restored human teeth using micro-CT techniques, International Journal of Prosthodontics, 14(4), 2001, 310-315.
[8] Wang, E.; Nelson, T.; Rauch, R: Back to elements-tetrahedra vs. hexahedra. In: Proceedings of the 2004 International ANSYS Conference 2004, Pennsylvania, USA.