Taylor & Francis
Taylor & Francis Group

Computer Methods in Biomechanics and Biomedical Engineering

ISSN: 1025-5842 (Print) 1476-8259 (Online) Journal homepage: http://www.tandfonline.com/loi/gcmb20

Morphing the feature-based multi-blocks of normative/healthy vertebral geometries to scoliosis vertebral geometries: development of personalized finite element models

Prasannaah Hadagali, James R. Peters & Sriram Balasubramanian

To cite this article: Prasannaah Hadagali, James R. Peters & Sriram Balasubramanian (2018): Morphing the feature-based multi-blocks of normative/healthy vertebral geometries to scoliosis vertebral geometries: development of personalized finite element models, Computer Methods in Biomechanics and Biomedical Engineering, DOI: 10.1080/10255842.2018.1448391

To link to this article: https://doi.org/10.1080/10255842.2018.1448391

Published online: 12 Mar 2018.

COMPUTER METHODS IN BIOMECHANICS AND BIOMEDICAL ENGINEERING, 2018
https://doi.org/10.1080/10255842.2018.1448391

Taylor & Francis
Taylor & Francis Group

Morphing the feature-based multi-blocks of normative/healthy vertebral geometries to scoliosis vertebral geometries: development of personalized finite element models

Prasannaah Hadagali[‡], James R. Peters and Sriram Balasubramanian

Orthopedic Biomechanics Laboratory, School of Biomedical Engineering Science and Health Systems, Drexel University, Philadelphia, PA, USA

ABSTRACT

Personalized Finite Element (FE) models and hexahedral elements are preferred for biomechanical investigations. Feature-based multi-block methods are used to develop anatomically accurate personalized FE models with hexahedral mesh. It is tedious to manually construct multi-blocks for large number of geometries on an individual basis to develop personalized FE models. Mesh-morphing method mitigates the aforementioned tediousness in meshing personalized geometries every time, but leads to element warping and loss of geometrical data. Such issues increase in magnitude when normative spine FE model is morphed to scoliosis-affected spinal geometry. The only way to bypass the issue of hex-mesh distortion or loss of geometry as a result of morphing is to rely on manually constructing the multi-blocks for scoliosis-affected spine geometry of each individual, which is time intensive. A method to semi-automate the construction of multi-blocks on the geometry of scoliosis vertebrae from the existing multi-blocks of normative vertebrae is demonstrated in this paper. High-quality hexahedral elements were generated on the scoliosis vertebrae from the morphed multi-blocks of normative vertebrae. Time taken was 3 months to construct the multi-blocks for normative spine and less than a day for scoliosis. Efforts taken to construct multi-blocks on personalized scoliosis spinal geometries are significantly reduced by morphing existing multi-blocks.

ARTICLE HISTORY
Received 4 May 2017
Accepted 1 March 2018

KEYWORDS
ANSYS ICEM CFD; multi-block hex-meshing; dual-kriging; scoliosis spine; personalized Fe models

1. Introduction

Spine deformity patterns in cases like scoliosis varies within individuals. This has necessitated the need for personalized finite element (FE) models of the thoracolumbar spine to design personalized intervention for deformity correction. Existing personalized FE models of the scoliosis-affected thoracolumbar spine vary in their levels of complexity from simple beam element models to more complex voxel-based, tetrahedral or hexahedral element models and limited by anatomical inaccuracies (Hadagali 2014; Wang et al. 2014). While beam element and voxel-based models are incapable of accurately representing the anatomical surface contours, tetrahedral element-based models are proven to exhibit inaccuracies *in silico*. Consequently, hexahedral element-based models have been preferred over other types for their superior element quality, as well as their ability to more accurately simulate biomechanical phenomena (Tadepalli et al. 2011; Mao et al. 2013).

Developing personalized FE models using hexahedral elements for anatomical structures without compromising the anatomical details is a time intensive process (Mao et al. 2013). This process usually involves segmentation of the subject's skeletal geometry from radiographic data followed by manual mesh generation. The complex anatomical geometries of the spine and associated structures present additional challenges to the personalized FE modeling procedures using hexahedral elements. Different studies have used feature-based multi-block hexahedral meshing method for personalized FE modeling purposes (Grosland et al. 2009; Kallemeyn et al. 2009; Shivanna et al. 2010; Jiang et al. 2012; Dong et al. 2013; Mao et al. 2013). Such a method would enable the development of high-quality hexahedral meshes and provides the advantage of adjusting the mesh density and quality of the hexahedral elements after they have been developed (Grosland et al. 2009; Shivanna et al. 2010; Mao et al. 2013). In spite of the attractive features that feature-based multi-block (also called hex-box, hex-block) hex-meshing method offers, usage of the technique to generate hex-mesh for geometry of every normative or scoliotic subject on an individual basis is tedious.

CONTACT Prasannaah Hadagali phadagali@mcw.edu
[‡]Department of Neurosurgery, Medical College of Wisconsin, Milwaukee, WI, USA.

Alternatively, in order to reduce the time taken to develop FE models for each subject, various interpolation-based techniques have been used to morph baseline FE mesh to subject-specific geometries (Stytz and Parrott 1993; Trochu 1993; Carr et al. 1997; Bennink et al. 2007). Morphing methods to generate personalized FE models have been previously reported for the brain (Li et al. 2011), spine (O'Reilly and Whyne 2008; Sigal and Whyne 2010; Lalonde et al. 2013), pelvis (Salo et al. 2013), femur (Bah et al. 2009; Grassi et al. 2011) and knee (Baldwin et al. 2010). Morphing methods can be surface-matching-based (laplace smoothing, deformable registration algorithm, mesh-matching algorithm, elastic-volumetric algorithm) or landmark-based (Kriging, radial basis functions, landmark based parametric meshing, dual-kriging) (Jingwen et al. 2012). Morphing approaches have limitations associated with element distortion leading to degradation of element quality (more prevalent with hex-elements) and simulation results (Couteau et al. 2000; Tada et al. 2005; Ji et al. 2011; Mao et al. 2013), as well as computational errors stemming from alterations in source geometry. In a recent study, dual kriging, which is a landmark-based method, was used to develop FE models of a pediatric and age-old normative thoraco-lumbar-sacrum (TLS) complex from an average-sized adult's baseline FE model of TLS complex. Although this method was shown to effectively retain the geometrical details of complex anatomical structures, the study dealt with usage of tetrahedral elements, the disadvantage of which was already discussed above (Lalonde et al. 2013). In the aforementioned studies pertaining to FE morphing, baseline and target geometries had relatively minimal variations in the anatomical features (Example: geometry of a 6-month old brain to a 3-month old, adult TLS to pediatric or age-old TLS) compared to the discrepancies between features of a healthy (baseline) and scoliosis-affected (target) spine. While relatively minimal variations in geometry could cause quality of elements to degrade after morphing as reported in literature, it can be predicted that these issues would increase in magnitude when an FE model of healthy spine is morphed to geometry of scoliosis spine (Figure 1).

The problem of mesh-distortion on hexahedral and tetrahedral elements as a result of morphing cannot be circumvented. However, replacing the entire distorted morphed mesh with new tetrahedral elements using robust re-meshing algorithms in commercially available tools seem to be an alternate option to rapidly generate personalized FE models for scoliosis spine. Several methods have been developed and incorporated in commercial tools to automatically generate tetrahedral elements on arbitrary geometries like vertebra. Although replacing distorted hexahedral elements with automatically generated tetrahedral elements is a viable option, the choice of elements for wide-range of biomechanical applications is hexahedral. This is due to the established fact that application of tetrahedral elements for nearly incompressible materials tend to lock and become overly stiff, producing inaccurate stress results (Fougeron et al. 2017). Creating new hexahedral mesh for every scoliosis spine is the only solution for the aforementioned problem at present, although the process demands significant time in manually constructing the multi-blocks. The goal of this study is to arrive at a possible solution to eliminate the need for manual construction of multi-blocks for spinal geometry of scoliosis-affected patients individually, thereby minimizing time and labor while maintaining the quality of hexahedral elements. The possible solution is to semi-automate the process of constructing multi-blocks on personalized geometries of scoliosis spine from the existing multi-blocks of normative spine (Appendix 1).

2. Materials and methods

Institutional Review Board (IRB) approvals from Drexel University and the Children's Hospital of Philadelphia (CHOP) were obtained to scan the chest regions using computed-tomography (CT). The retrospectively obtained (CT) scans of a skeletally normal 10 year old (YO) male subject (source geometry) and a 12 YO scoliosis affected subject (target geometry) were manually segmented and digitally reconstructed using Mimics (Materialise Inc., Belgium) to extract the 3D surface geometry of the entire bony thoracic spine (T1-T12). The spine of the 12 YO subject was deformed with a cobb angle of 90 degrees in the coronal plane.

Using the feature-based multi-block hexahedral meshing algorithm in ANSYS ICEM CFD 14.5 (ANSYS, Canonsburg, PA), blocks consisting 8 vertices, 12 edges and 6 faces were created for each vertebra of the 10 YO normative thoracic spine (Figure 2). Vertices, similar to nodes of an element, have coordinates in three dimensions (x, y and z) and flexible to be adjusted. Shape of a block structure relies on the positions of 8 vertices. Surface geometry (in .stl format) of each healthy thoracic vertebra was cross-sectioned in the mid-sagittal plane in order to simplify the process of multi-block development. The development process was initiated from the spinous process, and ended up in anterior region (vertebral body). Unlike a top-down approach where a single block is created around the geometry and later broken to accommodate the geometry (Mao et al. 2013), bottom-up approach was followed in the current attempt. A single block was created covering the tip of the spinous-process (vertebral tail). Another block was created from the existing block via face-extrusion method and their vertices were manually adjusted to fit the surface contour. Similar process

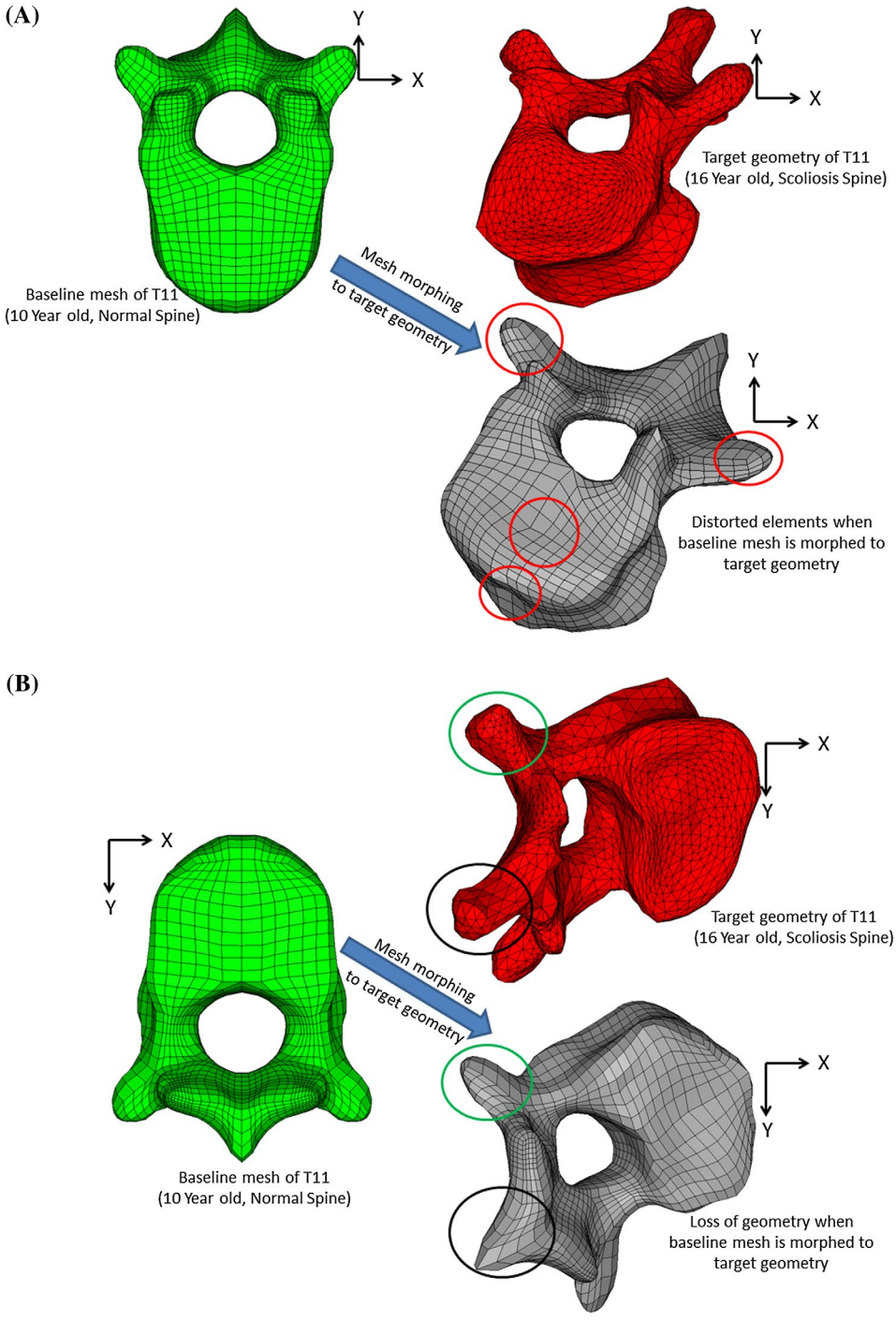


Figure 1. Existing problem while morphing mesh of healthy vertebra to scoliosis affected vertebra: (A)-Mesh distortion and (B)-Geometry loss. Green: Baseline mesh, Red: Target geometry, Grey: Baseline mesh morphed to target geometry.

was followed for all the sectioned thoracic vertebrae of the healthy subject. The multi-blocks and surface of each segment were mirrored and hexahedral elements with size ranging from 2 to 4 mm were generated. The multi-blocks of each thoracic vertebra of the 10 YO normative thoracic spine will henceforth be termed as 'multi-block template'.

The baseline FE model met the established standards for acceptable mesh quality (Li et al. 2011; Dong et al. 2013). Considering all the elements from T1 to T12, 99.86% had Jacobian ≥0.5, 99.72% had warpage ≤40, 99.26% had skewness ≤60, 99.82% had aspect ratio ≤5, 98.86% had minimum angle ≥30° and 97.76% had a maximum angle ≤150°. Material property descriptions are not provided, since these details are beyond the scope of our objective.

The next step was to create an FE model for the target geometry of 12 YO deformed thoracic spine using the existing baseline mesh developed using multi-block

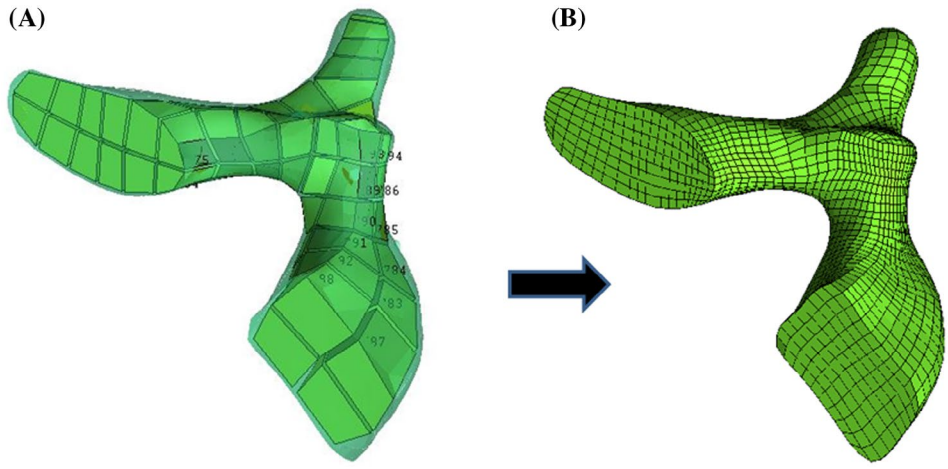


Figure 2. (A) Multi-blocks on the medially sectioned geometry of the T3 vertebra of the 10 YO thoracic spine. (B) Hexahedral mesh of the medially sectioned T3 of the 10 YO thoracic spine.

hex-meshing method. Before exploring ways to achieve the intended objective, it is necessary to understand the geometrical features of scoliosis-affected spines. The deformity is not only in the coronal plane, but also in sagittal (hyper- or hypo-kyphosis depending upon the individual) and transverse (vertebral rotation around the vertical axis) planes. Further observations into intra-vertebral morphology reveal wedged-vertebral bodies, uneven facet angles, increased angulation of pedicles in the concave side of apical region, decreasing width of pedicles in the concave side, increasing width of pedicles in convex, intra-vertebral rotation along transverse plane resulting in angulation of pedicles on concave side, shortening of the intervertebral disc, left or right bending of vertebral tail, etc. (Liljenqvist et al. 2002; Hu et al. 2014). Substantial differences between the normative and the scoliotic vertebral geometry leads to unevenness in the mesh density in the scoliosis FE model, in the process when baseline mesh is trying to accommodate to the changing surface contours (Figure 3). The mesh-morphing process could eventually warp the elements especially in the narrower and curved regions.

Previous methods have suggested constructing the multi-block (which would be tedious process considering the complexity of our problem) and obtain new hexahedral mesh for each patient. It is also possible to use the existing multi-block of the source mesh and manually move its vertices to the surface of the target. Presence of numerous vertices in the multi-blocks makes the process more complicated. One of the features in ANSYS ICEM CFD 14.5 enables the user to manually script a function for creating a node, creating a block, etc. using the Tool Command Language (TCL). It also enables saving the multi-blocks as a separate file containing details of vertices, edges and faces (.blk file). Command for transforming vertices was identified (Figure 4). Goal was to automatically 'snap' (transform) the vertices of existing multi-block template to the target surface with the help of vertex-transforming command, thereby accommodating the already created blocks to another contour surface and generating a new set of hexahedral elements. Automatic vertex-transforming operation to the target surface was executed with the help of dual-kriging, a global statistical interpolation method (Figure 4). Dual-kriging method finds detailed explanation in Lalonde et al. 2013 who used it to morph a baseline FE model of mid-sized adult's TLS complex to the geometry of 10 YO and 82 YO TLS successfully (Lalonde et al. 2013).

The technique used in this study relies on mapping the vertices of multi-block associated with the source geometry to that of the target. This requires both, the target geometry and point correspondences (landmarks) between the source and target. These sets of corresponding points are used to derive a mapping from the source to the target. In the current study, we used 30 surface landmark points identified on the vertebral bodies, pedicles, facets, and major processes of each vertebra of both the source (10 YO) and target (12 year old) geometries. These landmark points are consistent with those reported in the literature for geometry quantification and stereographic reconstruction of the vertebrae (Figure 5(a)) (Peters et al. 2015).

Initially, surface (.stl format), coordinates of the vertices pertaining to the multi-block (.blk file) and the corresponding landmark points of 10 YO T1 along with the target geometry (12 YO scoliosis T1 vertebrae) were imported to Matlab (Mathworks, Natick, MA). The 10 YO T1 surface, its vertices, and 12 YO T1 surface geometry were superimposed using the centroids of their associated landmark points (Figure 4(b)). The densities of the source

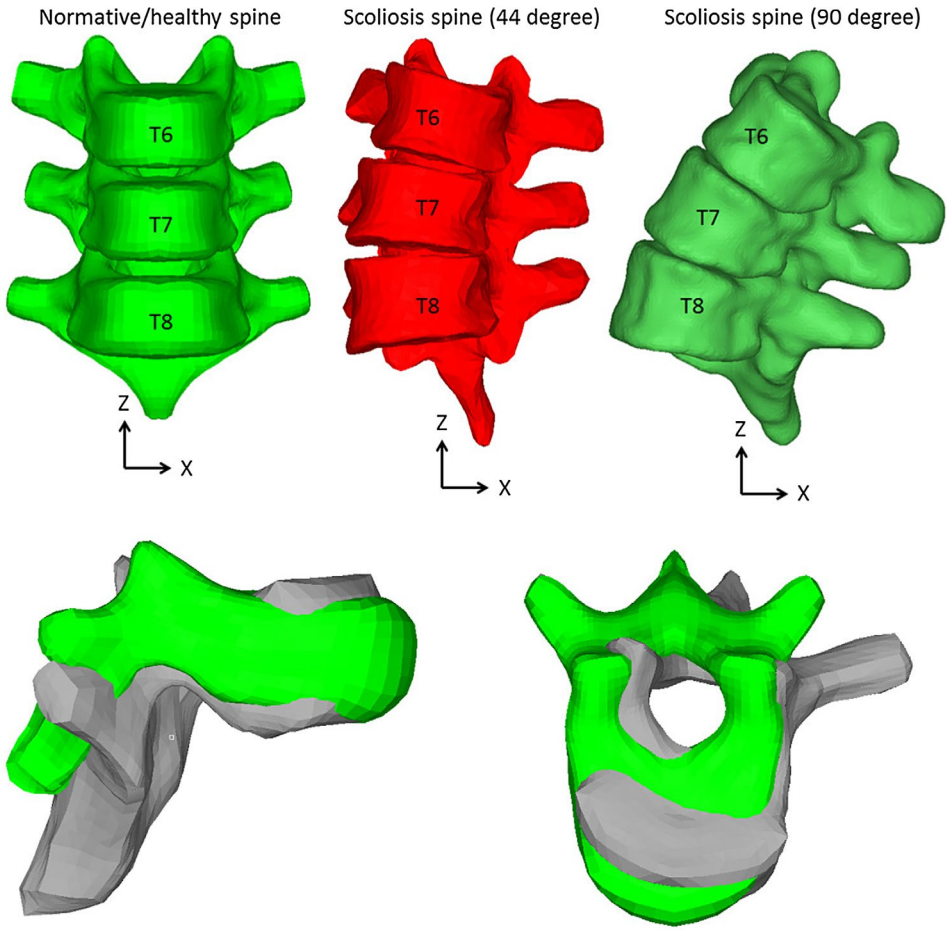


Figure 3. Vertebral geometries of T6-T8 for normal spine and different curves of scoliosis (Top). Geometry of T10 of a normative thoracic spine (green) and scoliosis deformed T10 (grey) in lateral view (bottom-left) and top view (bottom-right).
Notes: Figure 3 highlights the significant geometrical variation between normative and scoliotic vertebrae and the complexity of mesh morphing for such applications.

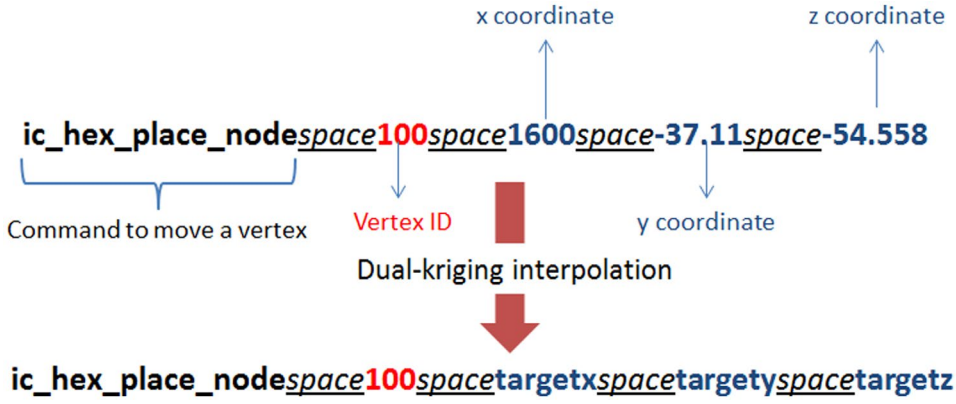


Figure 4. Operation of automatic transform of Multi-block vertices on the geometry of a source vertebra to the surface of target geometry using Dual-kriging.

and target surface mesh (.stl file) were increased after it was imported to Matlab. It is to be noted that in this case, the centroid of 12 YO target T1 geometry along with its surface and landmark points were translated for superimposing it on the unmoved centroids and surface of 10 YO source T1 geometry. A dual-kriging interpolation system was then used to map the displacements between source and target landmark points on to the multi-block template vertices and source geometries effectively distorting their original shapes to match the target (Figure 5) (Equations 1 and 2) (Trochu 1993; Lalonde et al. 2013). This was the first-pass in the morphing procedure.

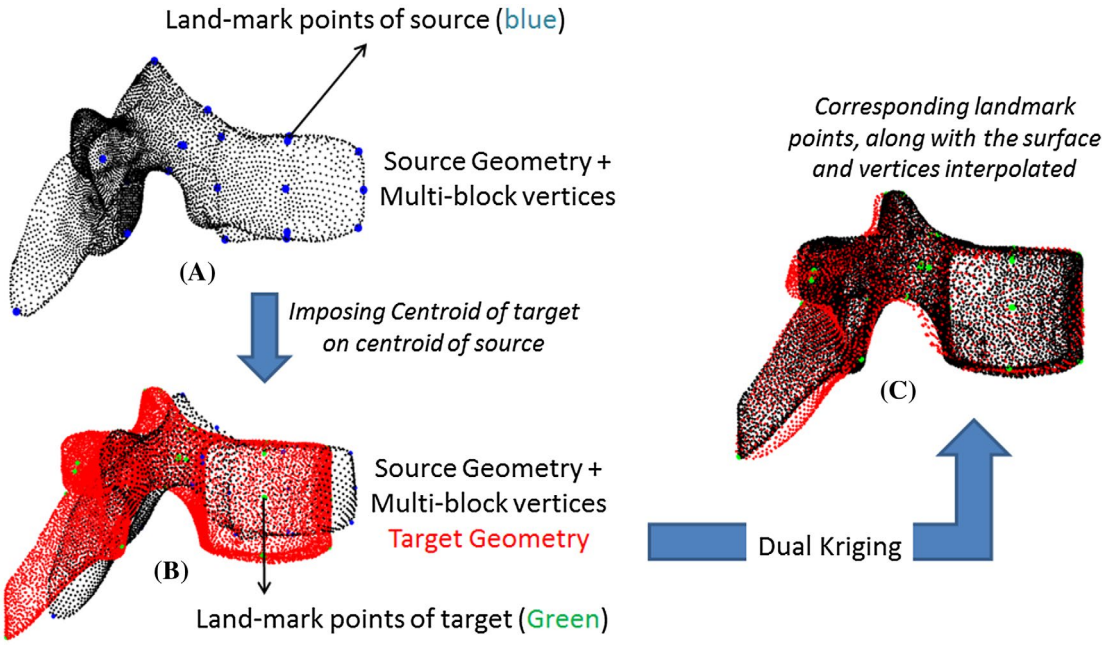


Figure 5. (a) Source geometry, vertices of multi-blocks and the landmark points. (b)Imposing centroids of source and target. (c) Interpolating the land-marks using dual-kriging.

$$\begin{bmatrix} K_{1,1} & \cdots & K_{1,n} & p_{1,1} & \cdots & p_{1,m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ K_{n,1} & \cdots & K_{n,n} & p_{n,1} & \cdots & p_{n,m} \\ p_{1,1} & \cdots & p_{1,n} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & \cdots & p_{m,n} & 0 & \cdots & 0 \end{bmatrix} \begin{vmatrix} b_1 \\ \vdots \\ b_n \\ \hline a_1 \\ \vdots \\ a_m \end{vmatrix} = \begin{vmatrix} u_1 \\ \vdots \\ u_n \\ \hline 0 \\ \vdots \\ 0 \end{vmatrix} \ldots \tag{1}$$

(Dual-Kriging System)

$$K = x - x_i \ldots \tag{2}$$

In the equation, $K_{n,n}$ is an entry in pseudo covariance matrix, $p_{n,m}$ is the nth data point in the mth dimension, b_n is the coefficient of the dual-kriging system corresponding to $K_{n,n}$, a_m is the coefficient of the linear mth dimension equation and u_n is the target data. To further improve the fit between the matched surfaces (second-pass) a second dual-kriging system was created using 20 coincident, 1 mm thick, radial cross sections. Each cross section was subdivided using a 4 by 5, evenly spaced, rectangular grid and semi-landmark points were created using the centroid of the surface vertices in each grid. This new set of 400 control points was then utilized to perform a second interpolation of the source geometry to improve the surface-to-surface fit (Figure 6). Vertices-transforming commands were finally exported from Matlab as a text file that could be fed into ICEM CFD.

Target T1 geometry of deformed spine and the multi-block template of the healthy T1 vertebra were imported in a newly created project file in ICEM CFD. Vertex-transforming commands for all the vertices in T1 Multi-block were produced detailing the new x, y and z coordinates in the form of text file. Customized codes for each of the aforementioned processes have been furnished in the appendix section with appropriate comments (Appendix 1).

To the same project, text file containing the vertices-transforming commands was uploaded to the scripting window in user-interface of ANSYS ICEM CFD 14.5 to 'snap' the vertices of the source T1 surface to the target T1. Vertices the blocks were visually inspected and manually adjusted if needed and volumetric hexahedral meshes of the 12 YO target T1 thoracic vertebrae were created. Similar procedure was followed for T2 to T12 vertebrae (Figure 7).

3. Results

More than 98% of the hexahedral elements jacobian >0.5, 6.23% of elements had warpage >20 and only 4.6% of elements had aspect ratio >4. The internal angles in more than 95% of elements had maximum angle <140°, minimum angle >30° and skew <50°. These values satisfied the criteria required for a high-quality FE model with very little manual effort required to refine the quality of the mesh (Figure 8). The first-pass morphing resulted in an average root-mean-square error (RMSE) between the morphed source and target surface geometries of 0.9 ± 0.6 mm with a maximum error of 3.75 mm across vertebral levels. The

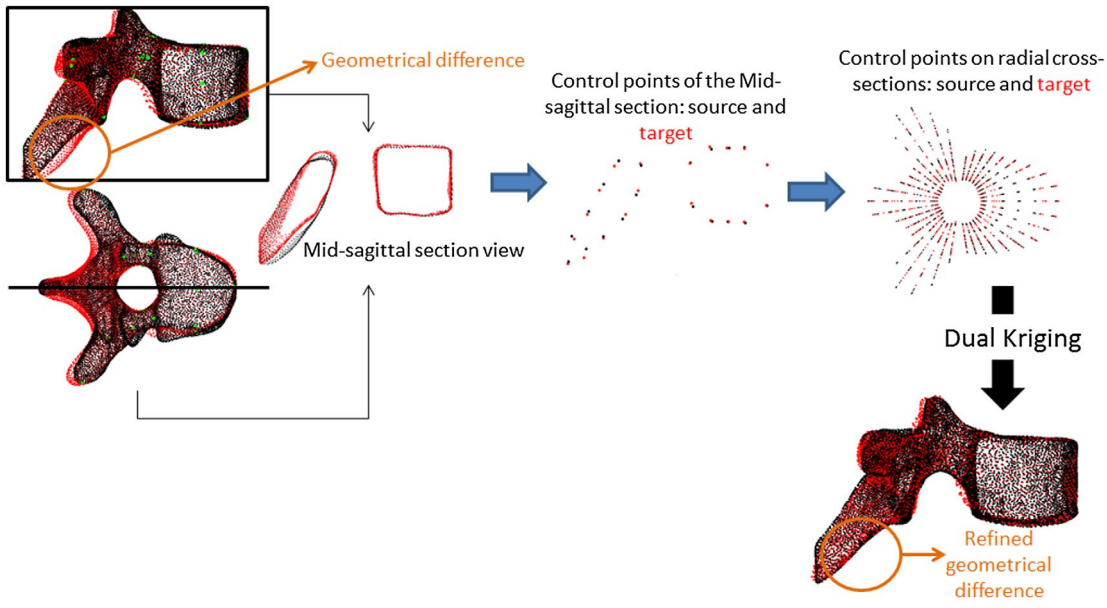


Figure 6. Interpolated source geometry and vertices of the multi-blocks to target geometry (red) and further refinement to accurately match the target geometry.

second-pass morphing using the resulting 400 semi-landmarks reduced the average RMSE to 0.6 ± 0.3 mm with a maximum error of 2.21 mm between the morphed source and target surface geometries.

4. Discussion

This paper echoes the views put forth by (Mao et al. 2013) regarding the advantages of multi-block technique over mesh-morphing for development of personalized FE meshes. Although advantageous, there is tediousness involved in multi-block meshing procedure; specifically in the development of FE models for numerous scoliosis affect spinal geometries. The alternative method reported in the current study eliminates the need to create multi-block for all the geometries from the beginning, thereby semi-automating the multi-block construction process on each personalized geometries. The process of rapidly 'snapping' the vertices of baseline multi-block to the target geometry is equivalent to creating new multi-block. Blocks do not carry the same set of elements; rather, generate new meshes that are uniform and comply with the new geometry. This implies that even when the vertices are 'translated' from source to the target, meshes created in the source are not carried to the target geometry.

The multi-block template of the baseline geometry (10 YO thoracic spine) was developed within a span of three months. However, it only required less than a day to transform the multi-blocks to fit to the geometry of scoliosis affected vertebra, using dual-kriging interpolation algorithm on the vertices of existing multi-block. Although applying dual-kriging morphing to the multi-blocks enabled development of personalized model in a relatively shorter time period, more time-efficient methods have been reported in previous studies (O'Reilly and Whyne 2008; Bah et al. 2009; Sigal and Whyne 2010; Grassi et al. 2011; Li et al. 2011; Lalonde et al. 2013; Salo et al. 2013). But a majority of those studies were limited by their use of tetrahedral elements, reduced geometrical congruence and negative volume issues.

The choice of element (tetrahedral or hexahedral) in previously reported studies can be attributed to anatomical complexity of the structures being modeled. While tetrahedral elements, typically used to model complex geometries like the pelvis (Salo et al. 2013), spine (Sigal et al. 2008; Sigal and Whyne 2010; Lalonde et al. 2013), and femur (Bah et al. 2009; Grassi et al. 2011) are computationally less expensive, they could have been preferred due to the sophisticated geometry of the deformed spine. Due to issues such as element locking and uneven stress distribution in models with tetrahedral elements and also due to the reason that many researchers prefer hexahedral elements over other types of elements, hexahedral elements were used in the present study (Tadepalli et al. 2011; Fougeron et al. 2017). This method improved the overall efforts needed to develop high-quality hexahedral element-based models of scoliosis vertebrae from a baseline model of normative/healthy vertebrae.

Surface errors and loss of geometrical accuracy have been observed in a majority of existing volumetric FE morphing approaches, albeit minimal variation of features between source and target geometries (Barratt et al. 2008; Sigal and Whyne 2010; Grassi et al. 2011; Lalonde et al. 2013; Salo et al. 2013). However, more recent studies have

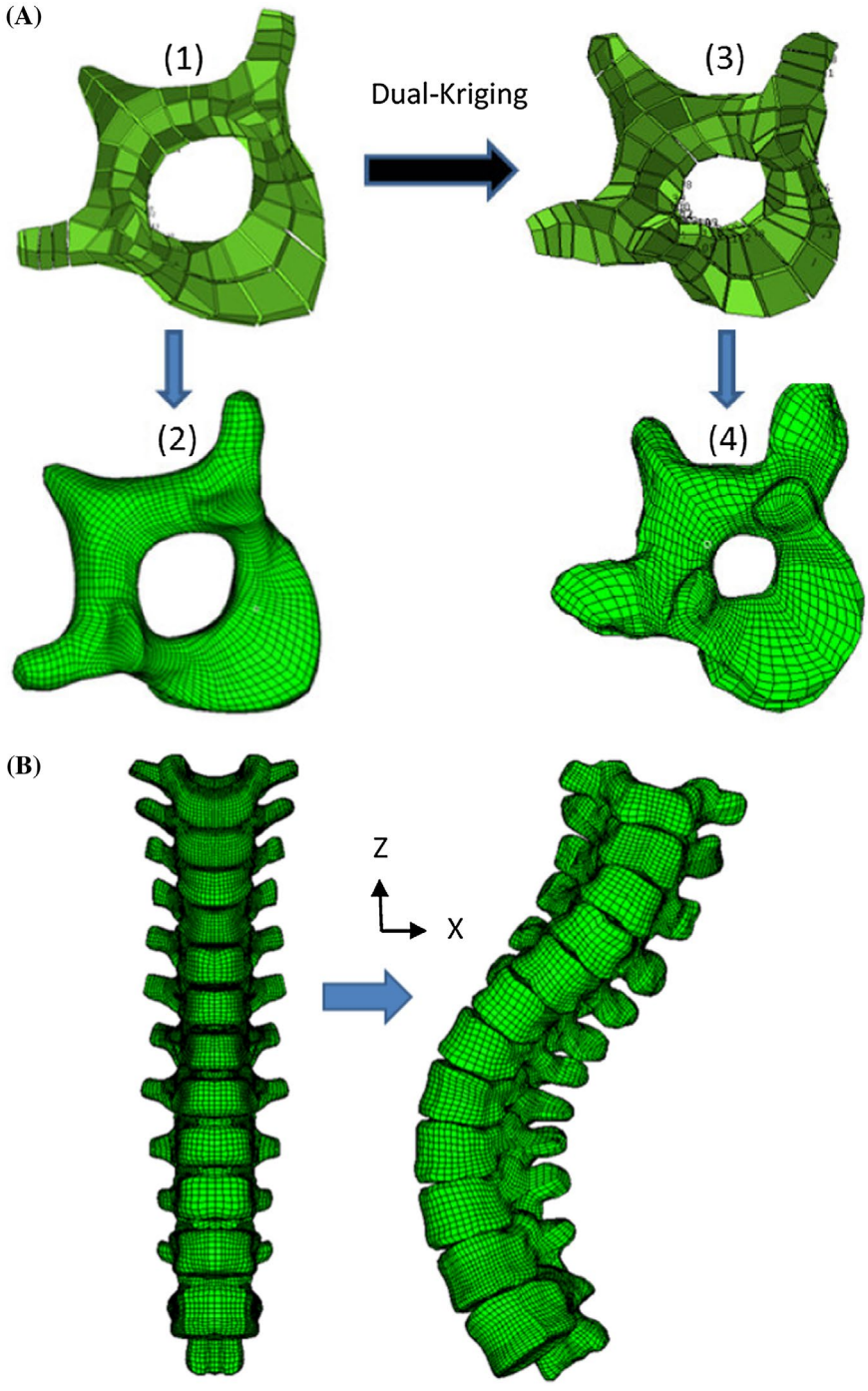


Figure 7. A1- T3 vertebral Template blocks, A2- Resulting mesh from the template blocks, A3- Template blocks of T3 morphed to T3 vertebral geometry of scoliotic 12 YO, A4- Resulting mesh from morphed template blocks. B- Resulting meshes of 12 YO scoliosis thoracic vertebrae from morphed multi-blocks of 10 YO T1-T12.

shown significant improvements in surface variations. Salo et al. (2013) created a personalized FE model of the pelvis and reported an average node-to-surface deviation of 0.9 ± 0.8 mm, and a maximum error of 6 mm (Salo et al. 2013). Another recent study observed average deviation of 2 mm on more than 90% of the surface geometry of the adult and pediatric models of the TL spine (Lalonde et al. 2013). Such errors in the prior mentioned models are justifiable considering the geometrical complexity of the pelvis and spine respectively. In the current study, errors were least significant as the final surface-fit error was approximately 0.6 ± 0.3 mm despite significant variations between the geometries of normative spine (source) and scoliosis-affected spine (target). Vertices that were not 'snapped' close to the target surface as a result of surface-fit error were manually adjusted to ensure the mapping of

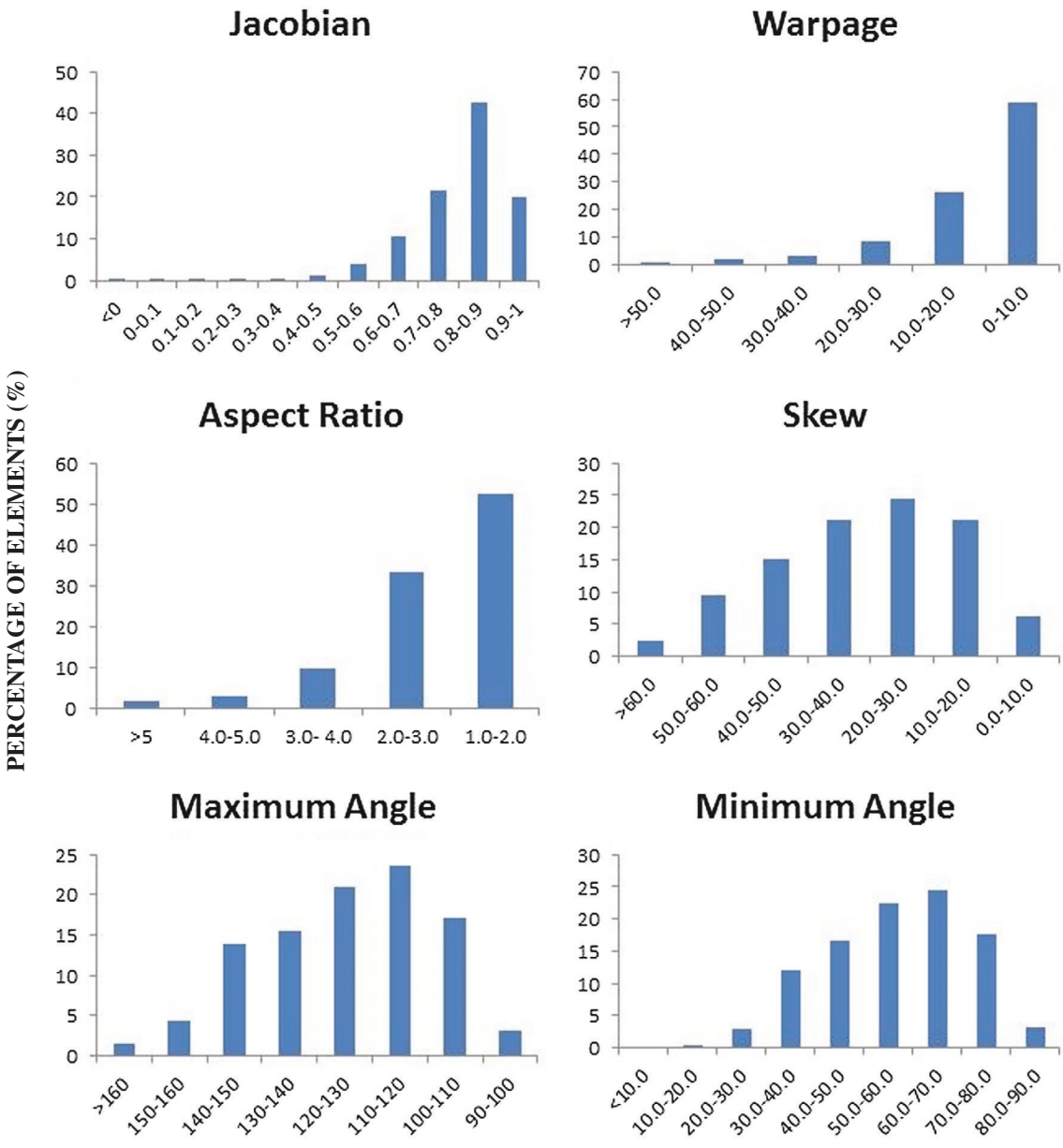


Figure 8. Quality of hexahedral elements in the morphed 12 YO thoracic vertebrae.

the vertices to the target surface. Convenient adjustments of the vertices in ICEM-CFD consequently improved the mesh quality with ease.

Prior studies using conventional surface- or landmark-based morphing to create volumetric subject-specific models have reported adverse warping and negative volume element issues (Couteau et al. 2000; Tada et al. 2005; O'Reilly and Whyne 2008; Grosland et al. 2009; Salo et al. 2013). Algorithmically complicated and time consuming mesh repair codes (Bucki et al. 2010) are prominently used to refine the distorted elements. However, mesh smoothing algorithms had no effect on morphed FE models of pelvis (Salo et al. 2013) and vertebral body (O'Reilly and Whyne 2008). The reasons for such inconsistencies have not been detailed and could be due to the complex geometry of pelvis and use of hexahedral elements in baseline model of vertebral body, respectively. On the contrary, personalized FE models of the brain that were obtained using RBF interpolation methods did not report degradation of element quality although hexahedral elements were used (Li et al. 2011). Any negative volume elements in the FE model of deformed vertebrae created using automatic vertices-transforming method could be conveniently overcome using the automatic mesh smoothing algorithm in ICEM CFD.

The current study is limited to the development of personalized FE models for the vertebrae of the thoracic spine. This method can be further extended to complete

spinal and ribcage structures of scoliosis spine geometries. This procedure can also be incorporated into publicly and commercially available multi-block meshing tools like IA-FEMESH (University of Iowa) and Truegrid (XYZ Scientific Applications Inc., Livermore, CA) respectively, as an additional feature.

5. Conclusion

The goal of this study was not to critique various morphing methods, but to highlight the limitations of applying such techniques to morph a normative spine FE model to scoliosis spine geometry. However, morphing technique can still be used on the blocks that are used to create the baseline mesh, instead of directly morphing the mesh. With the reported method, extensive warping of the volumetric elements can be avoided as old elements will be replaced. Efforts taken to create multi-blocks on personalized spinal geometries is significantly reduced due to dual-kriging morphing.

Acknowledgment

The authors would like to thank Mr. Samir Kadam (ANSYS Inc, Canonsburg, PA) for his technical support. Param Shah, Drexel University also deserves acknowledgement for helping the authors in organizing this manuscript. This work was not funded by any agencies.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Bah MT, Nair PB, Browne M. 2009. Mesh morphing for finite element analysis of implant positioning in cementless total hip replacements. Med Eng Phys. 31(10):1235–1243.

Baldwin MA, Langenderfer JE, Rullkoetter PJ, Laz PJ. 2010. Development of subject-specific and statistical shape models of the knee using an efficient segmentation and mesh-morphing approach. Comput Methods Programs Biomed. 97(3):232–240.

Barratt DC, Chan CS, Edwards PJ, Penney GP, Slomczykowski M, Carter TJ, Hawkes DJ. 2008. Instantiation and registration of statistical shape models of the femur and pelvis using 3D ultrasound imaging. Med Image Analy. 12(3):358–374.

Bennink HE, Korbeeck JM, Janssen BJ, Haar Romeny B. 2007. Warping a neuro-anatomy Atlas on 3D MRI Data with radial basis functions 3rd Kuala Lumpur International Conference on Biomedical Engineering 2006. In: Ibrahim F, Osman N, Usman J, Kadri N, editors. Kuala Lumpur: Springer Berlin Heidelberg. p. 28–32.

Bucki M, Lobos C, Payan Y. 2010. A fast and robust patient specific Finite Element mesh registration technique: Application to 60 clinical cases. Med Image Anal. 14(3):303–317.

Carr JC, Fright WR, Beatson RK. 1997. Surface interpolation with radial basis functions for medical imaging. IEEE Trans Med Imaging. 16(1):96–107.

Couteau B, Payan Y, Lavallée S. 2000. The mesh-matching algorithm: an automatic 3D mesh generator for finite element structures. J Biomech. 33(8):1005–1009.

Dong L, Li G, Mao H, Marek S, Yang KH. 2013. Development and validation of a 10-year-old child ligamentous cervical spine finite element model. Ann Biomed Eng. 41(12):2538–2552.

Fougeron N, Macron A, Pillet H, Skalli W, Rohan PY. 2017. Subject specific hexahedral Finite Element mesh generation of the pelvis from bi-Planar X-ray images. Comput Methods Biomech Biomed Eng. 20(sup1):75–76.

Grassi L, Hraiech N, Schileo E, Ansaloni M, Rochette M, Viceconti M. 2011. Evaluation of the generality and accuracy of a new mesh morphing procedure for the human femur. Med Eng Phys. 33(1):112–120.

Grosland NM, Shivanna KH, Magnotta VA, Kallemeyn NA, DeVries NA, Tadepalli SC, Lisle C. 2009. IA-FEMesh: an open-source, interactive, multiblock approach to anatomic finite element model development. Comput Methods Programs Biomed. 94(1):96–107.

Hadagali, P. 2014. Subject specific finite element modeling of the adolescent thoracic spine for scoliosis research master's. Philadelphia (PA): Drexel University.

Hu X, Siemionow KB, Lieberman IH. 2014. Thoracic and lumbar vertebrae morphology in Lenke type 1 female adolescent idiopathic scoliosis patients. Int J Spine Surgery. 8:30.

Ji S, Ford JC, Greenwald RM, Beckwith JG, Paulsen KD, Flashman LA, McAllister TW. 2011. Automated subject-specific, hexahedral mesh generation via image registration. Finite Elem Anal Des. 47(10):1178–1185.

Jiang B, Cao L, Mao H, Wagner C, Marek S, Yang KH. 2012. Development of a 10-year-old paediatric thorax finite element model validated against cardiopulmonary resuscitation data. Comput Methods Biomech Biomed Eng. 17(11):1185–1197.

Jingwen H, Rupp JD, Reed MP. 2012. Focusing on vulnerable populations in crashes: recent advances in finite element human models for injury biomechanics research. J Automot Saf Eng. 3(4):295–307.

Kallemeyn NA, Tadepalli SC, Shivanna KH, Grosland NM. 2009. An interactive multiblock approach to meshing the spine. Comput Methods Programs Biomed. 95(3):227–235.

Lalonde NM, Petit Y, Aubin CE, Wagnac E, Arnoux PJ. 2013. Method to geometrically personalize a detailed finite-element model of the spine. IEEE Trans Biomed Eng. 60(7):2014–2021.

Li Z, Hu J, Reed MP, Rupp JD, Hoff CN, Zhang J, Cheng B. 2011. Development, validation, and application of a parametric pediatric head finite element model for impact simulations. Ann Biomed Eng. 39(12):2984–2997.

Liljenqvist, UR, Allkemper T, Hackenberg L, Link TM, Steinbeck J, Halm HF. 2002. Analysis of vertebral morphology in idiopathic scoliosis with use of magnetic resonance imaging and multiplanar reconstruction. J Bone Joint Surg Am 84-a(3):359–368.

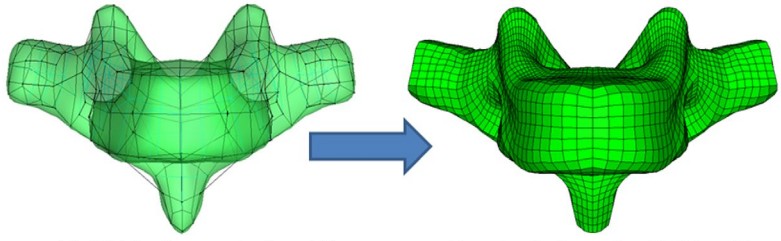
Mao H, Gao H, Cao L, Genthikatti VV, Yang KH. 2013. Development of high-quality hexahedral human brain

meshes using feature-based multi-block approach. Comput Methods Biomech Biomed Engin. 16(3):271–279.
O'Reilly MA, Whyne CM. 2008. Comparison of computed tomography based parametric and patient-specific finite element models of the healthy and metastatic spine using a mesh-morphing algorithm." Spine (Phila Pa 1976) 33(17):1876–1881.
Peters JR, Chandrasekaran C, Robinson LF, Servaes SE, Campbell RM Jr, Balasubramanian S. 2015. Age- and gender-related changes in pediatric thoracic vertebral morphology. Spine J. 15(5):1000–1020.
Salo Z, Beek M, Whyne CM. 2013. Evaluation of mesh morphing and mapping techniques in patient specific modeling of the human pelvis. Int j numer method biomed eng. 29(1):104–113.
Shivanna KH, Tadepalli SC, Grosland NM. 2010. Feature-based multiblock finite element mesh generation. Comput Aided Des. 42(12):1108–1116.
Sigal IA, Whyne CM. 2010. Mesh morphing and response surface analysis: quantifying sensitivity of vertebral mechanical behavior. Ann Biomed Eng. 38(1):41–56.
Sigal IA, Hardisty MR, Whyne CM. 2008. Mesh-morphing algorithms for specimen-specific finite element modeling. J Biomech. 41(7):1381–1389.
Stytz MR, Parrott RW. 1993. Using Kriging for 3D medical imaging. Comput Med Imaging Graph. 17(6):421–442.
Tada M, Yoshida H, Mochimaru M. 2005. Geometric modeling of living tissue for subject-specific finite element analysis. Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society IEEE Engineering in Medicine and Biology Society Conference.
Tadepalli SC, Erdemir A, Cavanagh PR. 2011. Comparison of hexahedral and tetrahedral elements in finite element analysis of the foot and footwear. J Biomech. 44(12):2337–2343.
Trochu F. 1993. A contouring program based on dual kriging interpolation. Eng with Comput. 9(3):160–177.
Wang W, Baran GR, Betz RR, Samdani AF, Pahys JM, Cahill PJ. 2014. The use of finite element models to assist understanding and treatment for scoliosis: a review paper. Spine Deformity. 2(1):10–27.

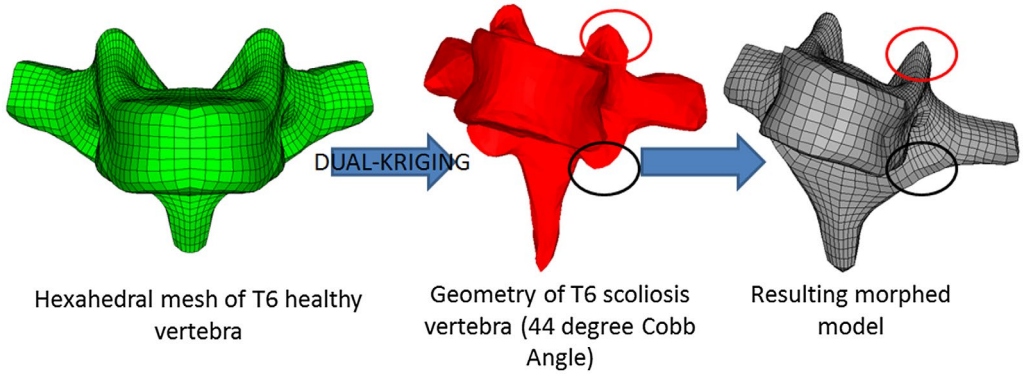
Appendix 1

PICTORIAL REPRESENTATION OF PROBLEM AND ARRIVAL OF SOLUTION

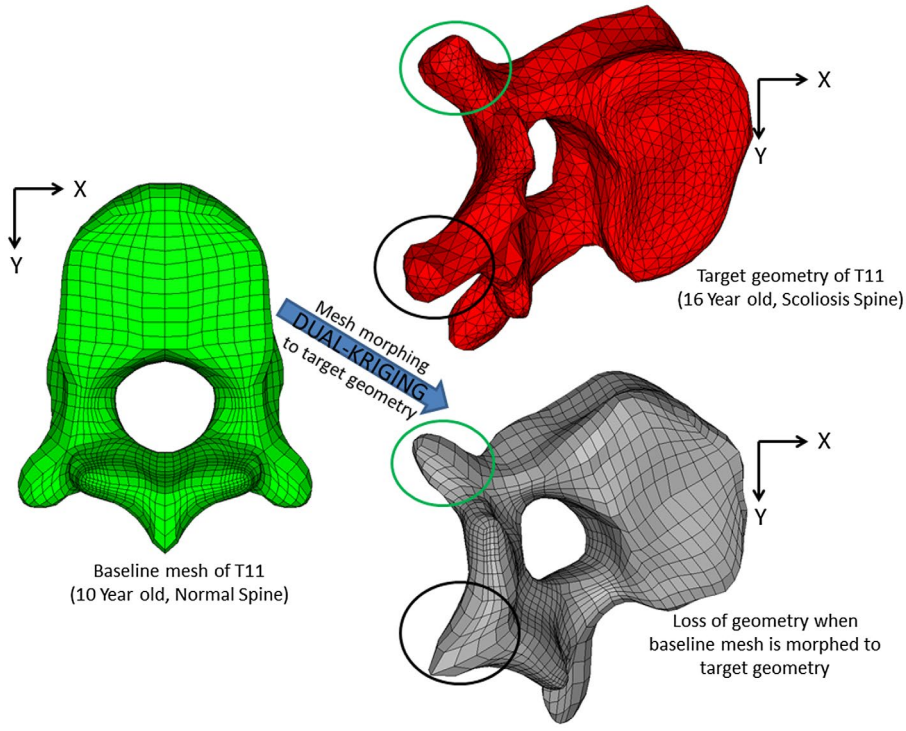
Step 1 (Hex-meshing using multi-block method):



Step 2 (Issues with morphing-Refer to Figure 1 in Introduction):



Available solutions to the existing problems:

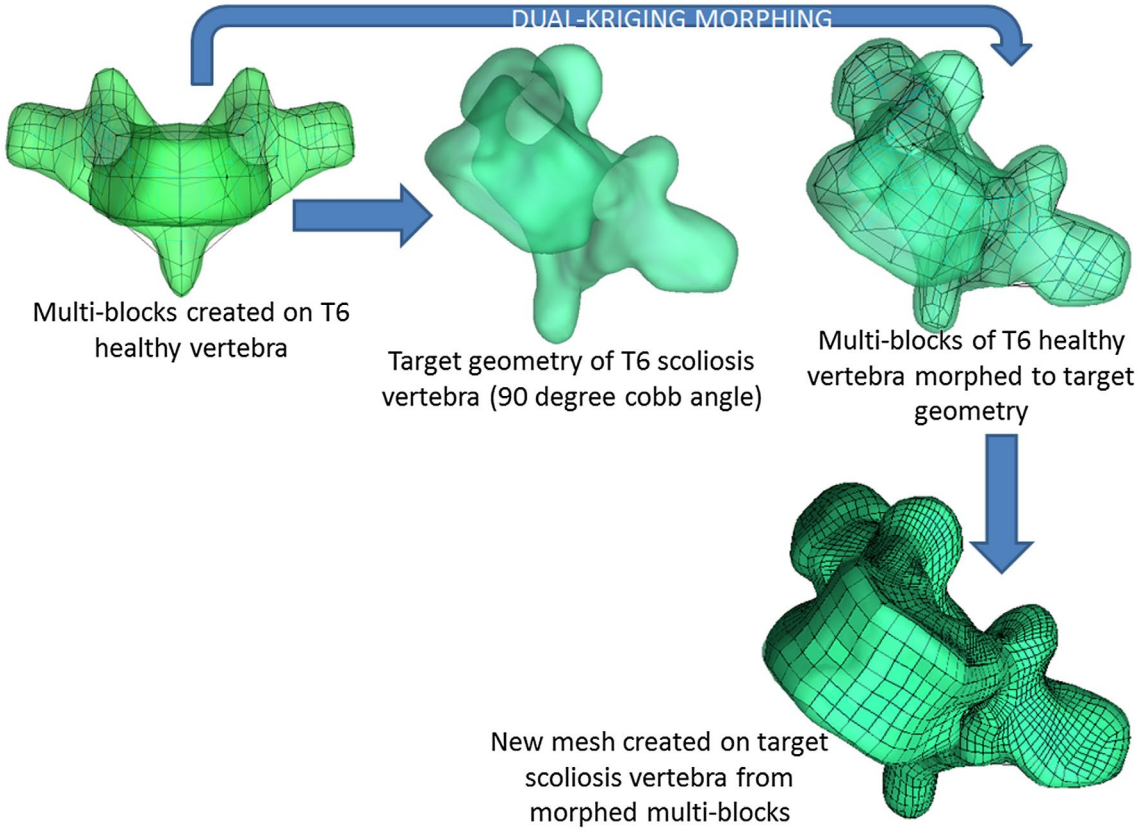


- Manually adjust distorted elements (impractical and time intensive)
- Redo meshing using tetra-elements (not preferred by many)
- Avoid morphing and create multi-blocks from scratch (time intensive procedure)

Proposed solution:

- Apply morphing algorithm on existing multi-blocks, generate new hexa-elements
- Morphing procedure will save time from creating new blocks on vertebrae
- **Hexa-elements can be generated**
- **Density can be adjusted at will due to availability of blocks**
- Unavoidable distortion in vertices of blocks due to morphing can be easily countered

Step 3 (Implementing proposed solution for existing problem):



TRANSFORMING VERTICES OF T5 MULTI-BLOCK (443 VERTICES)

```
% James R. Peters 8/12/2014 ==============================================
%                                Morphing                                vertebrae
==========================================================
clear all
close all
clc

VertebralLevel = 5;

% Extracting block vertices from .blk file ================================
BlkFileIDs = dir('*.blk'); % look for block file names
BlkFileIDs = {BlkFileIDs.name}; % convert the structure to a cell array

FileID = fopen(BlkFileIDs{1}, 'r'); % open the first block file

% Finding the start of text describing node locations and numbers =========
Count = 0;
Stop = false;

TextMatch = 3print('NODES new_numbering\r\n{'); % \r is a hard return and \n is a new line
TextMatch = TextMatch';

while ~Stop
   Text = fread(FileID, 22, '*char'); % '*char' reads the text in as a string column vector
   if strcmp(Text, TextMatch)
      StartPosition = Count + 24;
      Stop = true;
   else
      Count = Count + 1;
      fseek(FileID, Count, -1);
   end
end

fseek(FileID, StartPosition, -1); % set starting read position to beginning of vertex data

Data = textscan(FileID,'%f %f %f %d %d %f %*[^\n]'); % %*[^\n] skips everything up to a
new line
% Data = fscanf(FileID, '%f %f %f %d %d %d %*[^\n]'); % textscan works much faster here
```

```
BlkVertices = [Data{1, 1}, Data{1, 2}, Data{1, 3}];
BlkNodeNumber = Data{1, 6};

Fig1 = figure;
Ax1 = axes('Parent', Fig1, 'NextPlot', 'add', 'DataAspectRatioMode', 'manual',…
    'PlotBoxAspectRatioMode', 'manual', 'DataAspectRatio', [1, 1, 1], 'PlotBoxAspectRatio',…
    [4, 4, 4]);

% Import the matching surface mesh ========================================
[TemplateVertices, TemplateFaces, ~] = import_stl_fast('T5.stl', 1);

% Offsetting and rotating the mesh ========================================
Offset3D = mean(TemplateVertices);

TemplateVertices = bsxfun(@minus, TemplateVertices, Offset3D);
BlkVertices = bsxfun(@minus, BlkVertices, Offset3D);

% Initial rotation to orient mesh in same direction as target surface =====
RotationMatrix = makehgtform('zrotate', pi/2);
RotationMatrix = RotationMatrix(1 : 3, 1 : 3)';
% TemplateVertices = TemplateVertices * RotationMatrix;
% BlkVertices = BlkVertices * RotationMatrix;

TemplatePlot = trimesh(TemplateFaces, TemplateVertices(:, 1), TemplateVertices(:, 2),…
    TemplateVertices(:, 3), 'Parent', Ax1, 'EdgeColor', [0, 0, 0], 'FaceColor', [0, 1, 0]);

BlkPlot = line('Parent', Ax1, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2),…
    'Zdata', BlkVertices(:, 3), 'LineStyle', 'none', 'Marker', '.', 'MarkerSize', 15,…
    'Color', [0, 0, 0]);

% Find the average and max lengths of edges of each surface triangle ======
NumFaces = length(TemplateFaces(:, 1));

EdgeLengths = zeros(NumFaces, 1);

for i = 1 : NumFaces
    Face = TemplateFaces(i, ☺;
    Temp = TemplateVertices([Face'; Face(1)], ☺;
    Temp = sqrt(sum(diff(Temp, 1, 1) .^ 2, 2));
    EdgeLengths(i) = max(Temp);
```

```
end

AverageLength = 1;

NumVertices = length(TemplateVertices(:, 1));

% Increasing surface mesh density ========================================

for j = 1 : 2
   NewFaces = cell(NumFaces, 1);
   NewVertices = cell(NumFaces, 1);

   for i = 1 : NumFaces
      if EdgeLengths(i) > AverageLength

         Face = TemplateFaces(i, ☺;

         %     TemplateFaces(1, ☺ = []; % delete old face to conserve memory
         NumVertices = NumVertices + 3; % update current number of vertices
         NewVertices{i, 1} = [mean(TemplateVertices(Face(1 : 2)', ☺);…
            mean(TemplateVertices(Face(2 : 3)', ☺);…
            mean(TemplateVertices([Face(3); Face(1)], ☺)];

         NewFaces{i, 1} = [Face(1), NumVertices – 2, NumVertices;…
            NumVertices – 2, Face(2), NumVertices – 1;…
            NumVertices – 1, Face(3), NumVertices;…
            NumVertices – 2, NumVertices – 1, NumVertices];
      else
         NewFaces{i, 1} = TemplateFaces(i, ☺;
      end
   end

   TemplateVertices = cat(1, TemplateVertices, NewVertices{:});
   TemplateFaces = cat(1, NewFaces{:});

   NumFaces = length(TemplateFaces(:, 1));
   NumVertices = length(TemplateVertices(:, 1));

   EdgeLengths = zeros(NumFaces, 1);
```

```
    for i = 1 : NumFaces
        Face = TemplateFaces(i, ☺;
        Temp = TemplateVertices([Face'; Face(1)], ☺;
        Temp = sqrt(sum(diff(Temp, 1, 1) .^ 2, 2));
        EdgeLengths(i) = max(Temp);
    end

    set(TemplatePlot, 'Vertices', TemplateVertices, 'Faces', TemplateFaces)
end

% Import the target geometry ==========================================
[TargetVertices, TargetFaces, ~] = import_stl_fast('L5.stl', 1);

TargetOffset3D = mean(TargetVertices);
TargetVertices = bsxfun(@minus, TargetVertices, TargetOffset3D);

TargetPlot = trimesh(TargetFaces, TargetVertices(:, 1), TargetVertices(:, 2),…
    TargetVertices(:, 3), 'Parent', Ax1, 'EdgeColor', [0, 0, 0], 'FaceColor', [1, 0, 0]);

NumFaces = length(TargetFaces(:, 1));

EdgeLengths = zeros(NumFaces, 1);

for i = 1 : NumFaces
    Face = TargetFaces(i, ☺;
    Temp = TargetVertices([Face'; Face(1)], ☺;
    Temp = sqrt(sum(diff(Temp, 1, 1) .^ 2, 2));
    EdgeLengths(i) = max(Temp);
end

NumVertices = length(TargetVertices(:, 1));

% Increasing surface mesh density =====================================
for j = 1 : 2
    NewFaces = cell(NumFaces, 1);
    NewVertices = cell(NumFaces, 1);

    for i = 1 : NumFaces
        if EdgeLengths(i) > AverageLength
```

```
            Face = TargetFaces(i, ☺;

            %     TemplateFaces(1, ☺ = []; % delete old face to conserve memory
            NumVertices = NumVertices + 3; % update current number of vertices
            NewVertices{i, 1} = [mean(TargetVertices(Face(1 : 2)', ☺);…
               mean(TargetVertices(Face(2 : 3)', ☺);…
               mean(TargetVertices([Face(3); Face(1)], ☺)];

            NewFaces{i, 1} = [Face(1), NumVertices – 2, NumVertices;…
               NumVertices – 2, Face(2), NumVertices – 1;…
               NumVertices – 1, Face(3), NumVertices;…
               NumVertices – 2, NumVertices – 1, NumVertices];
         else
            NewFaces{i, 1} = TargetFaces(i, ☺;
         end
      end

      TargetVertices = cat(1, TargetVertices, NewVertices{:});
      TargetFaces = cat(1, NewFaces{:});

      NumFaces = length(TargetFaces(:, 1));
      NumVertices = length(TargetVertices(:, 1));

      EdgeLengths = zeros(NumFaces, 1);

      for i = 1 : NumFaces
         Face = TargetFaces(i, ☺;
         Temp = TargetVertices([Face'; Face(1)], ☺;
         Temp = sqrt(sum(diff(Temp, 1, 1) .^ 2, 2));
         EdgeLengths(i) = max(Temp);
      end

      set(TargetPlot, 'Vertices', TargetVertices, 'Faces', TargetFaces)
   end

   CovarianceMatrix = TargetVertices' * TargetVertices;
   [U, ~, ~] = svd(CovarianceMatrix);
   CovarianceMatrix = [1, 0, 0; 0, 1, 0; 0, 0, 1] * U';
   [U, ~, V] = svd(CovarianceMatrix);
   SVDRotation = V * U';
```

```
TemplateVertices = TemplateVertices * SVDRotation;
BlkVertices = BlkVertices * SVDRotation;
TargetVertices = TargetVertices * SVDRotation;

% Making fine alignments between Template and Target ======================
% Creating subsets of points for rotation
TemplatePoints = randperm(length(TemplateVertices(:, 1)), 2000)';
TargetPoints = randperm(length(TargetVertices(:, 1)), 2000)';

TemplatePoints = TemplateVertices(TemplatePoints, ☺;
TargetPoints = TargetVertices(TargetPoints, ☺;

% Scaling Template to target size ==========================================
Scale = mean(sqrt(sum(TargetPoints .^ 2, 2))) / mean(sqrt(sum(TemplatePoints .^ 2, 2)));

TemplateVertices = TemplateVertices * Scale;
TemplatePoints = TemplatePoints * Scale;
BlkVertices = BlkVertices * Scale;

% Update plot
============================================================
set(TemplatePlot, 'Vertices', TemplateVertices, 'Faces', TemplateFaces);
set(BlkPlot, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2),…
    'Zdata', BlkVertices(:, 3))
set(TargetPlot, 'Vertices', TargetVertices, 'Faces', TargetFaces)

ErrorDiff = inf;
Error = inf;
Count = 0;

while ErrorDiff > 1e-5 && Count < 1000
    IDX = knnsearch(TemplatePoints, TargetPoints);
    Temp = TemplatePoints(IDX, ☺;

    CovarianceMatrix = TargetPoints' * Temp;
    [U, ~, V] = svd(CovarianceMatrix);
    RotationMatrix = V * U';

    TemplateVertices = TemplateVertices * RotationMatrix;
```

```
    TemplatePoints = TemplatePoints * RotationMatrix;
    BlkVertices = BlkVertices * RotationMatrix;

    ErrorDiff = abs(sum(sqrt(sum((TemplatePoints(IDX, ☺ – Temp) .^ 2, 2))) – Error);
    Error = sum(sqrt(sum((TemplatePoints(IDX, ☺ – Temp) .^ 2, 2)));

    set(TemplatePlot, 'Vertices', TemplateVertices, 'Faces', TemplateFaces);
    set(BlkPlot, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2),…
        'Zdata', BlkVertices(:, 3))
    drawnow
end

% for k = 1 : 3

DistanceModifier = 0.5;
% Finding Cross sections and boundary points for morphing ==================
NumPts = 20;
PointStorage = zeros(NumPts, 2);
Tolerance = 0.75;

% Angles = linspace(-pi, pi, NumPts + 1)';
Angles = linspace(2 * pi / (NumPts + 1), 2 * pi, NumPts)';
Vects = [cos(Angles), sin(Angles)];

NumSections = 150;
TemplatePoints = cell(NumSections, 1);
TargetPoints = cell(NumSections, 1);

SectionAngles = linspace(2 * pi / (NumSections + 1), 2 * pi, NumSections)';

% Fig2 = figure;
% Ax2 = axes('Parent', Fig2, 'NextPlot', 'add', 'DataAspectRatioMode', 'manual',…
%                 'PlotBoxAspectRatioMode', 'manual', 'DataAspectRatio', [1, 1, 1],
'PlotBoxAspectRatio',…
%     [4, 4, 4]);
% OutlinePlot = line('Parent', Ax2, 'Xdata', nan, 'Ydata', nan, 'Zdata', nan,…
%     'Color', [0, 0, 1]);
% SectionPlot = line('Parent', Ax2, 'Xdata', nan, 'Ydata', nan, 'Zdata', nan,…
%     'Color', [0, 0, 0], 'LineStyle', 'none', 'Marker', '.', 'MarkerSize', 15);
```

```
% Centering the bodies ==========================================================
IDX = TemplateVertices(:, 2) <= Tolerance & TemplateVertices(:, 2) >= -Tolerance;
TemplateSection = TemplateVertices(IDX, ☺;

[~, IDX] = sort(TemplateSection(:, 1), 1, 'descend');
TemplateSection = TemplateSection(IDX, ☺;
[~, IDX] = max(abs(diff(TemplateSection(:, 1), 1, 1)));

Section1 = TemplateSection(1 : IDX, ☺;
Section2 = TemplateSection(IDX + 1 : end, ☺;

Offset3D = [mean([min(Section1(:, 1)); max(Section2(:, 1))]), 0, 0];

TemplateVertices = bsxfun(@minus, TemplateVertices, Offset3D);
BlkVertices = bsxfun(@minus, BlkVertices, Offset3D);

IDX = TargetVertices(:, 2) <= Tolerance & TargetVertices(:, 2) >= -Tolerance;
TargetSection = TargetVertices(IDX, ☺;

[~, IDX] = sort(TargetSection(:, 1), 1, 'descend');
TargetSection = TargetSection(IDX, ☺;
[~, IDX] = max(abs(diff(TargetSection(:, 1), 1, 1)));

Section1 = TargetSection(1 : IDX, ☺;
Section2 = TargetSection(IDX + 1 : end, ☺;

Offset3D = [mean([min(Section1(:, 1)); max(Section2(:, 1))]), 0, 0];

TargetVertices = bsxfun(@minus, TargetVertices, Offset3D);

set(TemplatePlot, 'Vertices', TemplateVertices, 'Faces', TemplateFaces);
set(BlkPlot, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2),…
   'Zdata', BlkVertices(:, 3))
set(TargetPlot, 'Vertices', TargetVertices, 'Faces', TargetFaces)
%
================================================================================
====

for i = 1 : NumSections
   RotationMatrix = makehgtform('zrotate', SectionAngles(i));
```

```
RotationMatrix = RotationMatrix(1 : 3, 1 : 3);

TemplateVertices = TemplateVertices * RotationMatrix;
TargetVertices = TargetVertices * RotationMatrix;

IDX = TemplateVertices(:, 2) <= Tolerance & TemplateVertices(:, 2) >= -Tolerance;
TemplateSection = TemplateVertices(IDX, ☺;

% Template Cross Sections ==========================================
[~, IDX] = sort(TemplateSection(:, 1), 1, 'descend');
TemplateSection = TemplateSection(IDX, ☺;
[~, IDX] = max(abs(diff(TemplateSection(:, 1), 1, 1)));
TemplateSection = TemplateSection(1 : IDX, ☺;

% Use alpha shapes to find the boundary ============================
[~, S] = alphavol([TemplateSection(:, 1), TemplateSection(:, 3)], 10, 0);
IDX = [S.bnd(:, 1); S.bnd(1, 1)];
Outline = [TemplateSection(IDX, 1), TemplateSection(IDX, 3)];

% Dual Krigging to increase line density ===========================
ChordLength = [0; cumsum(sqrt(sum(diff(Outline, 1, 1) .^ 2, 2)))];
ChordNew = linspace(0, ChordLength(end), 200)';

Kmatrix = [bsxfun(@power, abs(bsxfun(@minus, ChordLength, ChordLength')), 3),…
   ones(length(ChordLength), 1), ChordLength;…
   [ones(1, length(ChordLength)); ChordLength'], zeros(2)];

Coeffs = Kmatrix \ [Outline; zeros(2)];

Kmatrix = [bsxfun(@power, abs(bsxfun(@minus, ChordNew, ChordLength')), 3),…
   ones(200, 1), ChordNew];
Outline = Kmatrix * Coeffs;
%
=====================================================================

% Find the centroid of the new outline =============================
Temp = [[Outline; Outline(1, ☺], circshift([Outline; Outline(1, ☺], -1, 1)];

PolygonArea = 0.5 * sum((Temp(:, 1) .* Temp(:, 4)) – (Temp(:, 3) .* Temp(:, 2)));
Cx = (1 / 6 / PolygonArea) * sum((Temp(:, 1) + Temp(:, 3)) .* ((Temp(:, 1) .* Temp(:, 4))…
```

```
    - (Temp(:, 3) .* Temp(:, 2))));
  Cy = (1 / 6 / PolygonArea) * sum((Temp(:, 2) + Temp(:, 4)) .* ((Temp(:, 1) .* Temp(:, 4))…
    - (Temp(:, 3) .* Temp(:, 2))));

  Offset2D = [Cx, Cy];

  Outline = bsxfun(@minus, Outline, Offset2D);
%   TemplateSection = bsxfun(@minus, TemplateSection(:, 2 : 3), Offset2D);
%   CrossAngles = atan2(TemplateSection(:, 2), TemplateSection(:, 1));

  % Finding radial pseudo landmark ponts from the centroid ==============
  for j = 1 : NumPts
    IDX = sum(bsxfun(@times, Outline, Vects(j, ☺), 2) > 0;
    Temp = Outline(IDX, ☺;
    NormalVect = [-Vects(j, 2), Vects(j, 1)];
    Distances = abs(sum(bsxfun(@times, Temp, NormalVect), 2));
    [~, IDX] = min(Distances);
    Distances = sum(Temp(IDX, ☺ .* Vects(j, ☺, 2);
    Distances = Distances + DistanceModifier;
    PointStorage(j, ☺ = Vects(j, ☺ * Distances;
%     IDX = CrossAngles >= Angles(j) & CrossAngles <= Angles(j + 1);
%     Temp = TemplateSection(IDX, ☺;
%     PointStorage(j, ☺ = median(Temp);
  end

  Temp = [PointStorage(:, 1) + Cx, zeros(NumPts, 1), PointStorage(:, 2) + Cy] *
RotationMatrix';

  TemplatePoints{i, 1} = Temp;

  IDX = TargetVertices(:, 2) <= Tolerance & TargetVertices(:, 2) >= -Tolerance;
  TargetSection = TargetVertices(IDX, ☺;

  % Target Cross Sections ================================================
  [~, IDX] = sort(TargetSection(:, 1), 1, 'descend');
  TargetSection = TargetSection(IDX, ☺;
  [~, IDX] = max(abs(diff(TargetSection(:, 1), 1, 1)));
  TargetSection = TargetSection(1 : IDX, ☺;

  % Use alpha shapes to find the boundary ================================
```

```
    [~, S] = alphavol([TargetSection(:, 1), TargetSection(:, 3)], 10, 0);
    IDX = [S.bnd(:, 1); S.bnd(1, 1)];
    Outline = [TargetSection(IDX, 1), TargetSection(IDX, 3)];

    % Dual Krigging to increase line density ================================
    ChordLength = [0; cumsum(sqrt(sum(diff(Outline, 1, 1) .^ 2, 2)))];
    ChordNew = linspace(0, ChordLength(end), 200)';

    Kmatrix = [bsxfun(@power, abs(bsxfun(@minus, ChordLength, ChordLength')), 3),…
       ones(length(ChordLength), 1), ChordLength;…
       [ones(1, length(ChordLength)); ChordLength'], zeros(2)];

    Coeffs = Kmatrix \ [Outline; zeros(2)];

    Kmatrix = [bsxfun(@power, abs(bsxfun(@minus, ChordNew, ChordLength')), 3),…
       ones(200, 1), ChordNew];
    Outline = Kmatrix * Coeffs;

%     set(OutlinePlot, 'Xdata', Outline(:, 1), 'Ydata', Outline(:, 2),…
%        'Zdata', zeros(200, 1))
%     set(SectionPlot, 'Xdata', TargetSection(:, 2), 'Ydata', TargetSection(:, 3),…
%        'Zdata', TargetSection(:, 1))
%     drawnow
%     pause
   %
==========================================================================

    % Find the centroid of the new outline ==================================
    Temp = [[Outline; Outline(1, ☺], circshift([Outline; Outline(1, ☺], -1, 1)];

    PolygonArea = 0.5 * sum((Temp(:, 1) .* Temp(:, 4)) – (Temp(:, 3) .* Temp(:, 2)));
    Cx = (1 / 6 / PolygonArea) * sum((Temp(:, 1) + Temp(:, 3)) .* ((Temp(:, 1) .* Temp(:, 4))…
       - (Temp(:, 3) .* Temp(:, 2))));
    Cy = (1 / 6 / PolygonArea) * sum((Temp(:, 2) + Temp(:, 4)) .* ((Temp(:, 1) .* Temp(:, 4))…
       - (Temp(:, 3) .* Temp(:, 2))));

    Offset2D = [Cx, Cy];

    Outline = bsxfun(@minus, Outline, Offset2D);
%     TargetSection = bsxfun(@minus, TargetSection(:, 2 : 3), Offset2D);
```

```
%     CrossAngles = atan2(TargetSection(:, 2), TargetSection(:, 1));

    % Finding radial pseudo landmark ponts from the centroid ==============
    for j = 1 : NumPts
        IDX = sum(bsxfun(@times, Outline, Vects(j, ☺), 2) > 0;
        Temp = Outline(IDX, ☺;
        NormalVect = [-Vects(j, 2), Vects(j, 1)];
        Distances = abs(sum(bsxfun(@times, Temp, NormalVect), 2));
        [~, IDX] = min(Distances);
        Distances = sum(Temp(IDX, ☺ .* Vects(j, ☺, 2);
        Distances = Distances + DistanceModifier;
        PointStorage(j, ☺ = Vects(j, ☺ * Distances;
%         IDX = CrossAngles >= Angles(j) & CrossAngles <= Angles(j + 1);
%         Temp = TargetSection(IDX, ☺;
%         PointStorage(j, ☺ = median(Temp);
    end

    Temp = [PointStorage(:, 1) + Cx, zeros(NumPts, 1), PointStorage(:, 2) + Cy] *
RotationMatrix';

    TargetPoints{i, 1} = Temp;

    TemplateVertices = TemplateVertices * RotationMatrix'; % rotate vertebrae back to starting
position
    TargetVertices = TargetVertices * RotationMatrix';
end

TemplatePoints = cat(1, TemplatePoints{:});
TargetPoints = cat(1, TargetPoints{:});

%                                   Eliminate                                   nans
=============================================================
SumNans = 0;
IDX = ~isnan(TargetPoints(:, 1));
SumNans = SumNans + sum(IDX == 0);
TemplatePoints = TemplatePoints(IDX, ☺;
TargetPoints = TargetPoints(IDX, ☺;

IDX = ~isnan(TemplatePoints(:, 1));
SumNans = SumNans + sum(IDX == 0);
```

```
TemplatePoints = TemplatePoints(IDX, ☺;
TargetPoints = TargetPoints(IDX, ☺;

Fig3 = figure;
Ax3 = axes('Parent', Fig3, 'NextPlot', 'add', 'DataAspectRatioMode', 'manual',…
   'PlotBoxAspectRatioMode', 'manual', 'DataAspectRatio', [1, 1, 1], 'PlotBoxAspectRatio',…
   [4, 4, 4]);
TemplatePointPlot = line('Parent', Ax3, 'Xdata', TemplatePoints(:, 1), 'Ydata',
TemplatePoints(:, 2),…
   'Zdata', TemplatePoints(:, 3), 'LineStyle', 'none', 'Marker', '.', 'MarkerSize', 15,…
   'Color', [0, 0, 0]);
TargetPointPlot = line('Parent', Ax3, 'Xdata', TargetPoints(:, 1), 'Ydata', TargetPoints(:, 2),…
   'Zdata', TargetPoints(:, 3), 'LineStyle', 'none', 'Marker', '.', 'MarkerSize', 15,…
   'Color', [1, 0, 0]);

% Morphing the Mesh
==========================================================
Kmatrix = [bsxfun(@power, bsxfun(@power, bsxfun(@minus, TemplatePoints(:, 1),…
   TemplatePoints(:, 1)'), 2) + bsxfun(@power, bsxfun(@minus, TemplatePoints(:, 2),…
   TemplatePoints(:, 2)'), 2) + bsxfun(@power, bsxfun(@minus, TemplatePoints(:, 3),…
   TemplatePoints(:, 3)'), 2), 0.5), ones(NumSections * NumPts – SumNans, 1),
TemplatePoints;…
   [ones(1, NumSections * NumPts – SumNans); TemplatePoints'], zeros(4)];

Coeffs = Kmatrix \ [TargetPoints; zeros(4, 3)];

% There are too many vertices to morph at once so they need to be broken up
% Will do the morphing in 2000 point blocks
NumPts = length(TemplateVertices(:, 1));
NumBlks = floor(NumPts / 2000) + 1;

MorphedPoints = cell(NumBlks, 1);

Count = 0;
for i = 1 : NumBlks
   Count = Count + 2000;

   if Count < NumPts

      Temp = TemplateVertices(((i – 1) * 2000) + 1 : i * 2000, ☺;
```

```
    Kmatrix = [bsxfun(@power, bsxfun(@power, bsxfun(@minus, Temp(:, 1),…
      TemplatePoints(:, 1)'), 2) + bsxfun(@power, bsxfun(@minus, Temp(:, 2),…
      TemplatePoints(:, 2)'), 2) + bsxfun(@power, bsxfun(@minus, Temp(:, 3),…
      TemplatePoints(:, 3)'), 2), 0.5), ones(2000, 1), Temp];
    Temp = Kmatrix * Coeffs;
    MorphedPoints{i, 1} = Temp;
  else

    Temp = TemplateVertices(((i – 1) * 2000) + 1 : end, ☺;
    Kmatrix = [bsxfun(@power, bsxfun(@power, bsxfun(@minus, Temp(:, 1),…
      TemplatePoints(:, 1)'), 2) + bsxfun(@power, bsxfun(@minus, Temp(:, 2),…
      TemplatePoints(:, 2)'), 2) + bsxfun(@power, bsxfun(@minus, Temp(:, 3),…
      TemplatePoints(:, 3)'), 2), 0.5), ones(length(Temp(:, 1)), 1), Temp];
    Temp = Kmatrix * Coeffs;
    MorphedPoints{i, 1} = Temp;
  end
end

TemplateVertices = cat(1, MorphedPoints{:});
set(TemplatePlot, 'Vertices', TemplateVertices)
% end
set(TargetPlot, 'FaceAlpha', 0.3)
% delete(TargetPlot)

% Morphing the blocks
==================================================

Kmatrix = [bsxfun(@power, bsxfun(@power, bsxfun(@minus, BlkVertices(:, 1),…
      TemplatePoints(:, 1)'), 2) + bsxfun(@power, bsxfun(@minus, BlkVertices(:, 2),…
      TemplatePoints(:, 2)'), 2) + bsxfun(@power, bsxfun(@minus, BlkVertices(:, 3),…
      TemplatePoints(:, 3)'), 2), 0.5), ones(length(BlkVertices(:, 1)), 1), BlkVertices];
BlkVertices = Kmatrix * Coeffs;

set(BlkPlot, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2), 'Zdata', BlkVertices(:, 3))

% Placing everything back in anatomical orientation ========================
BlkVertices = bsxfun(@plus, bsxfun(@plus, BlkVertices, Offset3D) * SVDRotation',
TargetOffset3D);
TemplateVertices = bsxfun(@plus, bsxfun(@plus, TemplateVertices, Offset3D) *
SVDRotation', TargetOffset3D);
```

```
TargetVertices = bsxfun(@plus, bsxfun(@plus, TargetVertices, Offset3D) * SVDRotation', TargetOffset3D);

set(TemplatePlot, 'Vertices', TemplateVertices)
set(TargetPlot, 'Vertices', TargetVertices);
set(BlkPlot, 'Xdata', BlkVertices(:, 1), 'Ydata', BlkVertices(:, 2), 'Zdata', BlkVertices(:, 3))


% save morphed surface mesh ==================================================
% SaveFileName = cat(2, BlkFileIDs{1}(1 : end – 4), '_Morphed.stl');
% stlwrite(SaveFileName, TemplateFaces, TemplateVertices)
```

GENERATION OF ICEM CFD CODES TO TRANSFORM 10 YO-T5 MULTI-BLOCK VERTICES TO 12 YO-T5 SCOLIOSIS VERTEBRAL GEOMETRY

```
% write ICEM CFD block morphing file ========================================
SaveFileName = cat(2, BlkFileIDs{1}(1 : end - 4), '_Morphed.txt');
TxtFileID = fopen(SaveFileName, 'w', 'n', 'US-ASCII');
fprintf(TxtFileID, 'ic_hex_place_node %d %10.5f %10.5f %10.5f\r\n',...
   [BlkNodeNumber, BlkVertices(:, 1), BlkVertices(:, 2), BlkVertices(:, 3)]');
fclose(TxtFileID);
```

```
ic_hex_place_node 2635 -45.573118 -114.519687 -103.486571 (Line#1)
ic_hex_place_node 2631 -38.429045 -115.546334 -110.712057
ic_hex_place_node 819 -35.571354 -121.552333 -98.292424
ic_hex_place_node 821 -32.285162 -124.092035 -96.953244
ic_hex_place_node 827 -37.468805 -119.744165 -100.631823
ic_hex_place_node 2637 -38.101172 -122.044484 -93.954670
ic_hex_place_node 2639 -30.245523 -126.166414 -91.092496
.
.
.
.
ic_hex_place_node 2857 -32.121088 -92.304033 -137.414340 (Line#443)
```