

A brief note on building augmented reality models for scientific visualization

Mrudang Mathur^a, Josef M. Brozovich^b, Manuel K. Rausch^{b,c,d,*}

^a University of Texas at Austin, Department of Mechanical Engineering, 204 E Dean Keeton Street, Austin, 78712, TX, United States of America

^b University of Texas at Austin, Department of Aerospace Engineering and Engineering Mechanics, 2617 Wichita Street, Austin, 78712, TX, United States of America

^c University of Texas at Austin, Department of Biomedical Engineering, 107 W Dean Keeton Street, Austin, 78712, TX, United States of America

^d University of Texas at Austin, Oden Institute for Computational Engineering and Sciences, 201 E 24th Street, Austin, 78712, TX, United States of America

ARTICLE INFO

Keywords:

Mixed reality
Virtual reality
Digital twin
Metaverse
Finite elements

ABSTRACT

Augmented reality (AR) has revolutionized the video game industry by providing interactive, three-dimensional visualization. Interestingly, AR technology has only been sparsely used in scientific visualization. This is, at least in part, due to the significant technical challenges previously associated with creating and accessing such models. To ease access to AR for the scientific community, we introduce a novel visualization pipeline with which they can create and render AR models. We demonstrate our pipeline by means of finite element results, but note that our pipeline is generally applicable to data that may be represented through meshed surfaces. Specifically, we use two open-source software packages, ParaView and Blender. The models are then rendered through the <model-viewer> platform, which we access through Android and iOS smartphones. To demonstrate our pipeline, we build AR models from static and time-series results of finite element simulations discretized with continuum, shell, and beam elements. Moreover, we openly provide python scripts to automate this process. Thus, others may use our framework to create and render AR models for their own research and teaching activities.

1. Introduction

Scientific data are often inherently three-dimensional. Examples include imaging results or outputs from numerical methods. However, despite data's inherent three-dimensionality, our standard visualization techniques remain two-dimensional, as is the case with figures or videos. Additionally, current visualization techniques lack interactivity. In contrast, Augmented Reality (AR) models can represent the complete spatial and temporal aspects of data, are interactive in nature, and are easily accessible through smartphones. That is, AR is a next-generation visualization technique that overlays computer graphics directly into the physical space surrounding the user, thereby creating an immersive experience [1]. While originally championed by the entertainment, gaming, and computer graphics communities, AR now finds several applications in manufacturing [2], medicine [3], and education [4].

Despite the clear benefits of AR models, their adoption in research and education has been limited [5,6]. This is, at least in part, due to the use of proprietary software and hardware previously needed to create and render AR experiences, respectively [7]. Additionally, creating AR models often requires specialized training in computer graphics and

3D modeling that is not germane to most disciplines [8]. Thus, the objective of our current work is to help the scientific community to overcome some of these challenges associated with AR visualization. To do so, we introduce a novel open-source pipeline to build and render AR models of mesh-based scientific data. For demonstration purposes, we focus specifically on the visualization of finite element results but note that our work is equally applicable to many other disciplines and data types.

2. Material and methods

Towards building and rendering AR models of mesh-based data, we propose an open-source pipeline using ParaView (Kitware Inc, Clifton Park, NY), Blender (Blender Foundation BV, Amsterdam, The Netherlands), and <model-viewer> (Google Inc, Mountain View, CA), see Fig. 1. While we focus on finite element analyses in this article, our pipeline can also be used to create AR models of data from non-numerical experiments such as imaging [9,10]. Briefly, we first export our simulation results from a finite element solver, for example

* Corresponding author at: University of Texas at Austin, Department of Aerospace Engineering and Engineering Mechanics, 2617 Wichita Street, Austin, 78712, TX, United States of America.

E-mail address: manuel.rausch@utexas.edu (M.K. Rausch).

<https://doi.org/10.1016/j.finel.2022.103851>

Received 30 July 2022; Received in revised form 11 September 2022; Accepted 14 September 2022

Available online 10 October 2022

0168-874X/© 2022 Elsevier B.V. All rights reserved.

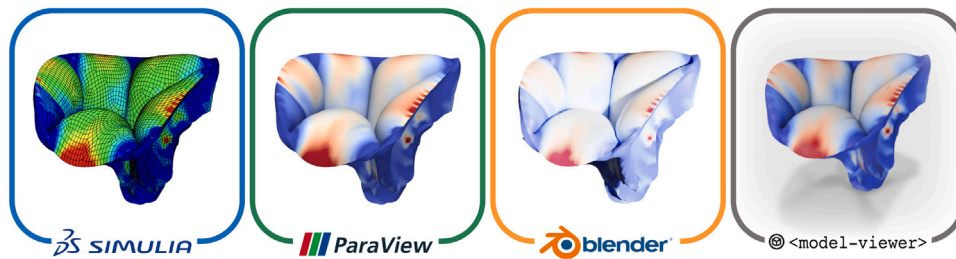


Fig. 1. Tricuspid valve finite element model rendered through novel AR visualization pipeline: We transfer simulation results from a finite element solver, Abaqus in this case, to ParaView, an open-source visualization package. In ParaView, we then specify engineering metrics to display and adjust the visualization colormap. Next, we export a surface geometry of our results to Blender, a popular open-source 3D modeling software. In Blender, we calibrate geometry scaling, translation, rotation, as well as lighting for AR visualization and animate our time-series results. Finally, we export AR models that are rendered through <model-viewer> on a smartphone.

Abaqus (v6.20-1, Dassault Systèmes, Vélizy-Villacoublay, France), to a file format compatible with ParaView. In ParaView, we then create and export a surface-only version of our simulation mesh and data. Next, we transfer these surface meshes to Blender where we calibrate model size, location, as well as orientation and create animations in case of time-series data. From Blender, we then export Android and iOS-device compatible AR assets that we host on our website using <model-viewer>. Finally, as a demonstration, we build AR models containing element types commonly used in computational mechanics: continuum elements and structural elements such as shells and beams. A detailed description of the post-processing steps is provided in the following sections. Additionally, further details on software versions and file formats used are provided in [Appendix A](#) and [Appendix B](#), respectively.

2.1. Creating a surface mesh

We first assemble our mesh-based results in the *.vtk file format. Creating *.vtk files is extensively documented in the literature and we direct readers to an excellent guide detailing this process [11]. Next, we import the *.vtk files in ParaView and calibrate the visualization to better represent engineering metrics such as stress, strain, and displacement. To this end, we filter our data and apply colormaps to our simulation mesh. We then generate a surface-only mesh of our results. Importantly, this approach can accommodate results from 3D volumetric fields as well. In such cases, we use a surface graphical object to represent data from the volumetric field at a given plane or cross-section of the simulation domain. Finally, we export the surface mesh of our model as a binary *.ply file. Please note, we embed any engineering metrics as vertex colors in these files. We provide example *.ply files for continuum, shell, and beam elements on the GitHub repository supplemental to this article. Additionally, see [Appendix C](#) for a list of filters used to create surface meshes of the aforementioned elements in ParaView.

2.2. Building AR models

To create AR models from surface meshes of our finite element results, we follow a multi-step process in Blender, as described in [Fig. 2](#). Here, we employ a common process to build static AR models for all mobile platforms. However, because Android and iOS devices use ARCore and ARKit as rendering libraries, respectively, they require distinct treatment when visualizing time-varying data. Thus, our process differs when animating our time-series data in dynamic AR models for Android and iOS devices.

2.2.1. Static models

To build a static AR model, we first import the *.ply file in Blender where we adjust model scaling, translation, and rotation. Please note, these affine transformations are applied to the deformed meshes of our scientific results to improve visualization in 3D space. To visualize any

colormaps that we add to our model in ParaView, we first create a “Material” within Blender. “Materials” are data structures in Blender that store and govern the appearance and texture of models. To our “Material”, we assign vertex colors that are embedded in the *.ply file. Next, we apply a “Decimate Modifier” to our geometry. This reduces the number of polygons in our AR assets which, in turn, reduces their storage size. Furthermore, we apply a “Solidify” modifier to shell geometries. This enhances model visibility while rendering. Additionally, we adjust lighting around our model in 3D space. Finally, we export an Android-compatible *.glb file through Blender’s native exporter. To generate an iOS-compatible *.usdz file we use the BlenderUSDZ add-on via the Blender GUI. To simplify the model creation process, we have also automated the aforementioned steps using Python. See **Supplementary Scripts 1a-c** to create static AR models of simulations with continuum, shell, and beam elements, respectively.

2.2.2. Dynamic models: Android

First, we follow the same steps used to create static models. That is, we import all geometries of our time-series data, perform affine transformations — specifically model scaling, rotation, and translation, and add the appropriate colormaps to our models in Blender. Next, we create a stop-motion animation of our time-series results. To this end, we ensure that only a single geometry is visible in each frame of our animation. Thereby, we emulate mesh motion between each animation frame. To achieve this, we dynamically resize each geometry in our dataset over the timespan of the animation. Finally, we adjust model lighting and export a *.glb file of our animation for rendering on Android devices. We provide **Supplementary Scripts 2a-c** to automatically build dynamic AR models of the continuum, shell, and beam element examples.

2.2.3. Dynamic models: iOS

To build dynamic models for iOS devices, we first import all *.ply files into Blender. Next, we adjust the scaling, translation, and rotation of our models. We then animate our time-series data using Shape Keys. To this end, Blender automatically builds a material point correspondence between each mesh in our dataset. As a result, we continuously animate the motion of each vertex in our model geometry over time. Please note, Shape Keys require the number of polygons remains constant between meshes. Thus, special attention should be paid to simulation results with self-contacting geometries to ensure a constant polygon count. Next, we modify scene lighting and export an intermediate *.usdc file of our animated model. Finally, we adjust model scaling and convert our animated model to *.usdz format in Reality Converter or usdzconvert in macOS and Windows/Linux systems, respectively. See **Supplementary Scripts 3a-c** to automate Shape Key animations in Blender.

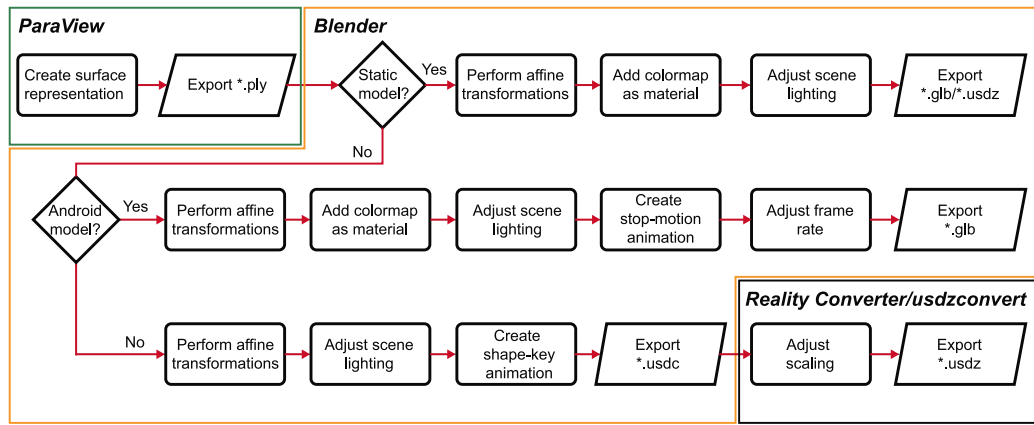


Fig. 2. Post-processing operations in Blender: We process surface geometries from ParaView in three distinct ways based on AR model type and rendering platform. To this end, we classify AR models as static models, dynamic AR models for Android devices, and dynamic AR models for iOS devices. We customize model size, position, and lighting in all cases, adjust animation timing for time series data, and add vertex colors to all models except dynamic models for iOS (see our discussion section for more detail on why we omit colors in iOS dynamic models).

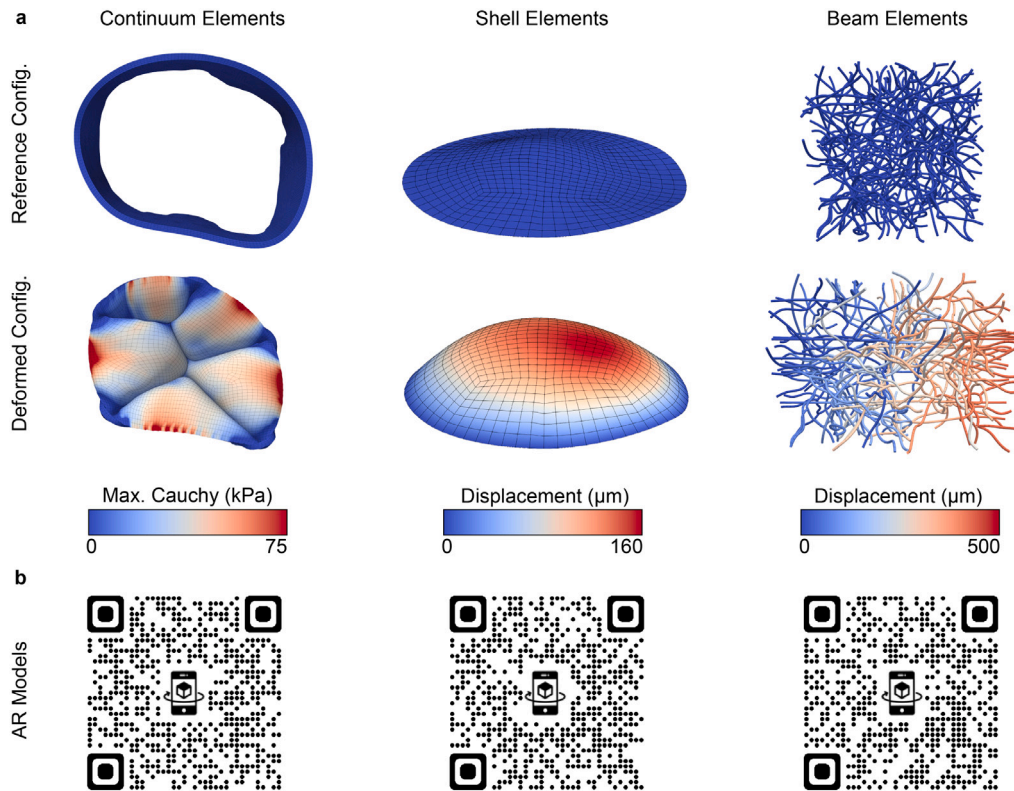


Fig. 3. AR models of finite element simulations discretized with continuum, shell, and beam elements: (a) We present simulation results in the reference and deformed configurations which are overlaid with contours of engineering metrics, such as maximum principal Cauchy stress and displacement magnitude. (b) Static and dynamic models of each simulation can be accessed through the associated QR code.

2.3. Hosting and rendering models

We use GitHub to host, access, and share our AR models. To this end, we integrate <model-viewer> with our research webpage to render AR assets. We chose <model-viewer> over other, similar, rendering platforms as it is free to use, compatible with both Android and iOS devices, and can be accessed without app download. Finally, we share links to our AR models through QR codes. This choice is motivated by the relative ease of generating QR codes and accessing them via smartphone as well as the possibility of embedding QR codes in presentations, videos, and figures.

3. Results

To demonstrate our open-source visualization pipeline, we built AR models of three distinct finite element analyses from our research group, see Fig. 3a. These include a dynamic model of the human tricuspid valve [12], a bulge inflation test of tissue from a rat's anterior tricuspid valve leaflet [13], and the uniaxial extension of an undulated fiber network. These simulations are discretized with continuum, shell, and beam elements, respectively. Here, we present images of the reference and deformed configuration of each simulation, as one would see in a scientific figure. Additionally in Fig. 3b, we provide QR codes leading to a webpage containing static and dynamic AR models of

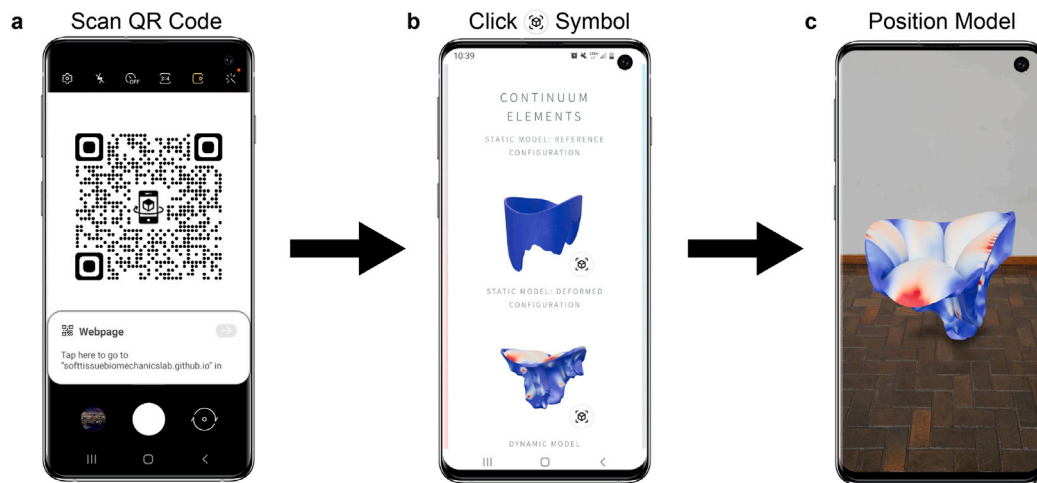


Fig. 4. Accessing AR models: To view AR models on a smartphone (a) we first scan the QR code and load the associated webpage, (b) we then click the <model-viewer> logo to enable AR, and finally (c) we position the AR model in 3D space around us.

each simulation. Readers are encouraged to scan these QR codes and view the linked AR models as described by Fig. 4. Using our open-source pipeline, we successfully build and render AR models of the aforementioned finite element analyses. Thereby, we provide scientists, in general, and mechanicians, in particular, an avenue to create and share their results in all spatial and temporal dimensions.

4. Discussion

Augmented reality represents the next frontier in personal computing and has the capacity to transform scientific visualization. In this brief note, we introduced an open-source visualization pipeline to build and render AR models from finite element simulation results. Our pipeline integrates an established and versatile tool in scientific visualization (ParaView) with a popular 3D modeling tool (Blender) to standardize and simplify the process of building AR models from scientific data. We then use an open-source AR platform (<model-viewer>) to access and render our models on Android and iOS devices. We also demonstrated our pipeline on results from finite element simulations that are spatially discretized with continuum, shell, and beam elements. Furthermore, we provide python scripts to automate the creation of those same models. Thus, we consider this pipeline to be successful in achieving our stated objectives: to democratize and simplify AR visualization of three-dimensional data by means of finite element results. That is, we eschewed the need for proprietary software, such as NVIDIA Omniverse, and expensive AR headsets, such as the Microsoft HoloLens. Moreover, through the provided python scripts, we significantly reduce the training required to use an open-source, industry-standard 3D modeling tool to create AR assets.

In addition to fulfilling these objectives, our pipeline can serve as a cornerstone for future studies integrating scientific data with mixed-, virtual-, and augmented-reality applications. Specifically concerning results from mechanical analyses, we'd like to note that our methods can be easily extended to Eulerian grids and do not have to be limited to Lagrangian analyses as used in our examples [14–17]. Furthermore, we can build AR models from isogeometric analysis results due to Blender's in-built support for NURBS [18–20]. Additionally, we may interface Blender with packages such as Unity to design virtual reality experiences [21]. Finally, when combined with image processing packages, our methods can be used to effectively visualize and interact with digital twins [22,23].

Naturally, our methods are subject to certain limitations. Firstly, our current method of creating dynamic *.usdz files precludes the use of custom colormaps. This is currently a limitation of Apple's underlying USD codebase. We anticipate that dynamic vertex colors will be added

in upcoming software releases. Secondly, our current *.usdz animation technique requires users to specify the number of times an animation should repeat. This increases the file size of the AR asset, thereby making it harder to access over slower internet connections. Moving forward, we aim to programmatically loop *.usdz animations, similar to those in *.gltf files. Additionally, by using <model-viewer> to render AR models, we require users to create both *.gltf and *.usdz models of any scientific results they have. In the future, we anticipate a greater cross-compatibility in files between Android and iOS systems, thereby reducing this burden on content creators. Finally, our pipeline is purely a visualization tool and not a computational tool. Thus, incorporating real-time prediction is a task for the future.

5. Conclusion

In conclusion, we introduced an end-to-end pipeline for building and rendering AR models from scientific, mesh-based data [24]. Importantly, our pipeline only uses open-source software packages and, thus, allows users to freely access AR models on any smartphone through the internet. Furthermore, we showcased our pipeline by building AR models of finite element analyses with three common element discretizations that are used in computational mechanics. Moreover, we provide python scripts to automatically create those models. Through this work, we hope to simplify and accelerate the adoption of AR visualization in scientific visualization. Thereby enabling researchers, educators, and students to gain a deeper understanding of complex spatio-temporal results associated with their data. Importantly, all scripts and information necessary to reproduce our work are openly available through a GitHub repository listed under “Code Availability” below.

Code availability

All supplementary scripts, surface geometries, and AR model examples are available in the GitHub repository associated with this article.

URL: https://github.com/SoftTissueBiomechanicsLab/AR_Pipeline.git

CRediT authorship contribution statement

Mrudang Mathur: Writing – original draft, Developed the code, Produced all figures and tutorials, Review. **Josef M. Brozovich:** Developed the code, Review. **Manuel K. Rausch:** Writing – original draft, Review.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Manuel K. Rausch reports financial support was provided by the National Heart Lung and Blood Institute. Manuel K. Rausch reports financial support was provided by the National Science Foundation. Manuel K. Rausch reports financial support was provided by the Office of Naval Research. Manuel K. Rausch reports financial support was provided by the American Heart Association. Mrudang Mathur reports financial support was provided by the American Heart Association. Manuel K. Rausch reports a relationship with Edwards Lifesciences Corporation that includes: speaking and lecture fees.

Data availability

All data and models are openly available through a link to a GitHub repository.

Acknowledgments

We appreciate support from the American Heart Association through an award to Dr. Rausch (18CDA34120028) and a predoctoral fellowship to Mrudang Mathur (902502), as well as the National Institutes of Health through awards to Dr. Rausch (1R21HL161832 and 1R01HL165251). Additionally, we appreciate support from the National Science Foundation through awards to Dr. Rausch (2127925, 2105175, 2046148, and 1916663) and the Office of Naval Research through an award to Dr. Rausch (N00014-22-1-2073). We would also like to thank Soham M. Mane for sharing the results of his fiber network simulations with us. Note, the opinions, findings, and conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the American Heart Association, the National Institutes of Health, the Office of Naval Research, or the National Science Foundation.

Appendix A. Software requirements

Our proposed pipeline requires the use of multiple open-source software and is, thus, subject to compatibility errors across different software versions. Therefore, we have compiled a list of stable and compatible software packages. These packages, along with their download links, are detailed in the GitHub repository associated with this article. The packages are:

- ParaView 5.10
- Blender 2.83
- BlenderUSDZ add-on
- Apple Reality Converter (for MacOS users)
- Apple usdzconvert utility (for Windows/Unix users)

Appendix B. File formats

To ensure the flexibility of our methods to varied numerical solvers, visualization packages, and mixed-reality platforms we use several industry-standard and open-source file formats in our pipeline. Specifically, we employ *.vtk files for storing and accessing numerical simulation data and *.ply files to store model geometries. Moreover, we render our AR assets as *.gltf and *.usdz files for Android and iOS devices, respectively.

Appendix C. Generating surface meshes in ParaView

To generate surface meshes for results with continuum and shell elements we use the following filters:

1. Filters > Alphabetical > Extract Surface
2. Filters > Alphabetical > Generate Surface Normals

For beam elements, we first employ the Tube filter in ParaView viz.:

1. Filters > Alphabetical > Tube
2. Filters > Alphabetical > Extract Surface
3. Filters > Alphabetical > Generate Surface Normals

References

- [1] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, M. Ivkovic, Augmented reality technologies, systems and applications, *Multimedia Tools Appl.* 51 (1) (2011) 341–377, <http://dx.doi.org/10.1007/s11042-010-0660-6>.
- [2] A. Nee, S. Ong, G. Chrysosouris, D. Mourtzis, Augmented reality applications in design and manufacturing, *CIRP Ann.* 61 (2) (2012) 657–679, <http://dx.doi.org/10.1016/j.cirp.2012.05.010>.
- [3] M.W. Chu, J. Moore, T. Peters, D. Bainbridge, D. McCarty, G.M. Guiraudon, C. Wedlake, P. Lang, M. Rajchl, M.E. Currie, R.C. Daly, B. Kiaii, Augmented reality image guidance improves navigation for beating heart mitral valve repair, *Innov. Technol. Tech. Cardiothorac. Vasc. Surg.* 7 (4) (2012) 274–281, <http://dx.doi.org/10.1097/imi.0b013e31827439ea>.
- [4] K.N. Plunkett, A simple and practical method for incorporating augmented reality into the classroom and laboratory, *J. Chem. Educ.* 96 (11) (2019) 2628–2631, <http://dx.doi.org/10.1021/acs.jchemed.9b00607>.
- [5] J. Huang, S. Ong, A. Nee, Real-time finite element structural analysis in augmented reality, *Adv. Eng. Softw.* 87 (2015) 43–56, <http://dx.doi.org/10.1016/j.advengsoft.2015.04.014>.
- [6] C. Hedenqvist, M. Romero, R. Vinuesa, Improving the learning of mechanics through augmented reality, *Technol. Knowl. Learn.* (0123456789) (2021) <http://dx.doi.org/10.1007/s10758-021-09542-1>.
- [7] C. Sutherland, K. Hashtrudi-Zaad, R. Sellens, P. Abolmaesumi, P. Mousavi, An augmented reality haptic training simulator for spinal needle procedures, *IEEE Trans. Biomed. Eng.* 60 (11) (2013) 3009–3018, <http://dx.doi.org/10.1109/TBME.2012.2236091>.
- [8] J. Huang, S. Ong, A. Nee, Visualization and interaction of finite element analysis in augmented reality, *Comput. Aided Des.* 84 (2017) 1–14, <http://dx.doi.org/10.1016/j.cad.2016.10.004>.
- [9] J. Abderezaei, A. Pionteck, I. Terem, L. Dang, M. Scadeng, P. Morgenstern, R. Shrivastava, S.J. Holdsworth, Y. Yang, M. Kurt, Development, calibration, and testing of 3D amplified MRI (aMRI) for the quantification of intrinsic brain motion, *Brain Multiph.* 2 (September 2020) (2021) 100022, <http://dx.doi.org/10.1016/j.brain.2021.100022>.
- [10] K.M. Moerman, C.A. Holt, S.L. Evans, C.K. Simms, Digital image correlation and finite element modelling as a method to determine mechanical properties of human soft tissue in vivo, *J. Biomech.* 42 (8) (2009) 1150–1153, <http://dx.doi.org/10.1016/j.jbiomech.2009.02.016>.
- [11] W. Schroeder, K. Martin, B. Lorensen, *The Visualization Toolkit, fourth ed.*, Kitware, New York, 2006.
- [12] M. Mathur, W.D. Meador, M. Malinowski, T. Jazwiec, T.A. Timek, M.K. Rausch, Texas TriValve 1.0 : a reverse-engineered, open model of the human tricuspid valve, *Eng. Comput.* (2022) <http://dx.doi.org/10.1007/s00366-022-01659-w>.
- [13] W.D. Meador, M. Mathur, S. Kakaletsis, C.-Y. Lin, M.R. Bersi, M.K. Rausch, Biomechanical phenotyping of minuscule soft tissues: An example in the rodent tricuspid valve, *Extrem. Mech. Lett.* 55 (2022) 101799, <http://dx.doi.org/10.1016/j.eml.2022.101799>.
- [14] R. Shad, A.D. Kaiser, S. Kong, R. Fong, N. Quach, C. Bowles, P. Kasinpila, Y. Shudo, J. Teuteberg, Y.J. Woo, A.L. Marsden, W. Hiesinger, Patient-specific computational fluid dynamics reveal localized flow patterns predictive of post-left ventricular assist device aortic incompetence, *Circ. Heart Fail.* 14 (7) (2021) E008034, <http://dx.doi.org/10.1161/CIRCHEARTFAILURE.120.008034>.
- [15] M.R. Pfaller, J. Pham, N.M. Wilson, D.W. Parker, A.L. Marsden, On the periodicity of cardiovascular fluid dynamics simulations, *Ann. Biomed. Eng.* 49 (12) (2021) 3574–3592, <http://dx.doi.org/10.1007/s10439-021-02796-x>, [arXiv: 2102.00107](https://arxiv.org/abs/2102.00107).

- [16] K. Menon, R. Mittal, Flow physics and dynamics of flow-induced pitch oscillations of an airfoil, *J. Fluid Mech.* 877 (McCroskey 1982) (2019) 582–613, <http://dx.doi.org/10.1017/jfm.2019.627>.
- [17] T. Berger, M. Kaliske, An arbitrary Lagrangian Eulerian formulation for tire production simulation, *Finite Elem. Anal. Des.* 204 (2022) 103742, <http://dx.doi.org/10.1016/j.finel.2022.103742>.
- [18] B. Dortdivanlioglu, A. Krischok, L. Beirão da Veiga, C. Linder, Mixed isogeometric analysis of strongly coupled diffusion in porous materials, *Internat. J. Numer. Methods Engrg.* 114 (1) (2018) 28–46, <http://dx.doi.org/10.1002/nme.5731>.
- [19] K.M. Shepherd, X.D. Gu, T.J. Hughes, Isogeometric model reconstruction of open shells via Ricci flow and quadrilateral layout-inducing energies, *Eng. Struct.* 252 (2022) 113602, <http://dx.doi.org/10.1016/j.engstruct.2021.113602>.
- [20] A. Chemin, T. Elguedj, A. Gravouil, Isogeometric local h-refinement strategy based on multigrids, *Finite Elem. Anal. Des.* 100 (2015) 77–90, <http://dx.doi.org/10.1016/j.finel.2015.02.007>.
- [21] N. Bhatia, E.A. Müller, O. Matar, A GPU accelerated lennard-jones system for immersive molecular dynamics simulations in virtual reality, in: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12191 LNCS, 2020, pp. 19–34, http://dx.doi.org/10.1007/978-3-030-49698-2_2.
- [22] R. Revetria, F. Tonelli, L. Damiani, M. Demartini, F. Bisio, N. Peruzzo, A real-time mechanical structures monitoring system based on digital twin, iot and augmented reality, in: *2019 Spring Simulation Conference*, Vol. 51, *SpringSim*, (1) IEEE, 2019, pp. 1–10, <http://dx.doi.org/10.23919/SpringSim.2019.8732917>.
- [23] E. Febrianto, L. Butler, M. Girolami, F. Cirak, Digital twinning of self-sensing structures using the statistical finite element method, 2021, pp. 1–23, [arXiv: 2103.13729](https://arxiv.org/abs/2103.13729).
- [24] C.M. Portela, J.R. Greer, D.M. Kochmann, Impact of node geometry on the effective stiffness of non-slender three-dimensional truss lattice architectures, *Extrem. Mech. Lett.* 22 (2018) 138–148, <http://dx.doi.org/10.1016/j.eml.2018.06.004>.