Acrossim – Technical report

Title

Acrossim: A toolkit for cross-platform integration of CFD simulation data in computer graphics

Authors

Serkan Solmaz and Tom Van Gerven

Department of Chemical Engineering, KU Leuven, Celestijnenlaan 200F, B-3001 Leuven, Belgium

Abstract

Acrossim is a versatile toolkit designed to seamlessly integrate computational fluid dynamics (CFD) simulation data into game engines. This modular, component-based toolkit incorporates data from widely used CFD solvers in engineering. The primary focus of Acrossim lies in its lightweight, end-to-end, and automated approach, making it a cost-effective and freely available resource. The toolkit aims to encourage the development of cross-platform user applications, such as virtual reality, utilizing CFD simulation. This approach allows for easy consumption of CFD data in user-friendly digital environments, providing an attractive alternative to traditional engineering simulation software.

Keywords:

Computational fluid dynamics; engineering; visualization; augmented reality; virtual reality; game engines

1 Introduction

Computational fluid dynamics (CFD) simulations are essential for engineering design, analysis, and decision-making. It provides practical solutions to fluid flow and multiphysics problems that can be difficult, expensive, or impossible to study with handbooks and experiments. Non-experts can benefit from meaningful simulation data for case-specific applications without having a deep insight into simulation workflows. Nonetheless, CFD simulations require sophisticated and computationally intensive operations to be handled by a competent analyst to perform simulations and extract meaningful data. In addition, non-experts can still struggle to comprehend post-processing results because of the expert-centric environment of the data postprocessing software [1].

Personal user devices are ubiquitous in everyday life and day-by-day invading workplaces. Notably, augmented and virtual reality (AR/VR) applications have received much attention due to cognitive and behavioral advantages [2], [3]. CFD simulations in AR/VR applications can be a user-friendly medium to make challenging content accessible and interactive for non-experts such as students [4], [5]. Despite the increasing interest in this integration, literature mostly proposed stand-alone, platform-specific, computationally demanding, nonreplicable, and manually integrated data processing methodologies [1], [6]. A prototyping toolkit is still required to provide a malleable solution for open research and stimulate the development of cross-platform user applications with CFD simulations.

This study aims at developing a modular component-based toolkit to integrate CFD simulation data into game engines within an optimized and automated data processing approach. The provided toolkit helps developers effortlessly bring any simulation content to end-user devices.

2 Background, motivation and significance

CFD simulation data is inherently composed of large, complex, and also transient datasets. Only a part of this data is processed to obtain meaningful results with dedicated data processing algorithms available in post-processing software. The processed CFD data can be usually categorized as follows: visual representations, graphs & charts, and text data. An extract-based data processing approach can diminish the data size of CFD post-processing to reasonable sizes [7]. Scholars have studied the integration of extract-based CFD data with cross-platform development tools based on two fundamental approaches.

On the one hand, native CFD data is processed via post-processing software and subsequently exported to text files. Following, the intended CFD data is reproduced in cross-platform environments using dedicated post-processing algorithms implemented by developers [8]–[11]. This approach brought technical drawbacks in the design and operation of user applications. First of all, CFD post-processing algorithms should be implemented in the game engines to reproduce CFD data from text files. While this results in additional workloads to developers during the development and design processes; more powerful user hardware is required for data reproduction. In addition, in case another game engine with a different programming language is alternatively employed, all post-processing algorithms should be transformed into another. Moreover, some researchers exercised the integration of the Visualization Toolkit (VTK) to directly use dedicated post-processing algorithms inside game engines [12]–[14]. It is observed that some challenges in the integration of VTK with game engines remain unsolved requiring wrappers to communicate different programming languages.

On the other hand, native CFD data is processed via post-processing software and exported to suitable file formats such as 3D, image and text. The exported data are then exchanged to the formats that are readable by game engines. This methodology points out a lightweight approach to employ any end-user devices [15]. Researchers proposed different data processing pipelines coupling CFD post-processors and 3D computer graphics software to produce 3D data formats readable by game engines such as VRML, FBX and 3DS [6], [16]–[18]. The data processing pipelines proposed in these studies are so diverse by means of data formats, software and data processing performance. Hence, it is practically inappropriate to make a comparison to figure out the optimal workflow.

The literature study revealed that cross-platform user applications developed with CFD simulation data are generally hardcoded, computationally demanding, platform-specific, and standalone. Also, the integration of CFD data into game engines is handled manually by developers [1], [6], [19]. Besides, most of the available studies only focused on one type of data; therefore, it is still ambiguous how to integrate 3D, image and, text altogether [6].

Previously, we proposed a data processing methodology to the above-mentioned problems targeting lightweight, interoperable, end-to-end, and free-to-use utilities [20]. Our study highlighted inclusive guidance on the integration of CFD data with game engines through a modular component-oriented data processing pipeline. A thorough investigation was carried out to examine optimized and automated data processing options between CFD solvers and game engines. A qualitative assessment was performed to scrutinize data format, size, processing time, quality, automation, and management. The study introduced an extract-based approach to adapt big CFD data into cross-platform user applications through manageable data bundles. We mostly aimed at exploring potential data processing solutions to build up a reliable methodology to integrate CFD data into game engines to develop AR/VR applications. Several code snippets and experimental tools were delivered alongside our previous study.

In the present study, as a follow-up to our previous work [20], we develop a software toolkit called Acrossim. This toolkit enables developers to directly implement a reliable and tested data processing software into their workflow. It facilitates the rapid integration of CFD simulation results into cross-platform environments in a time- and cost-effective manner, without requiring powerful end-user hardware and hours of data preparation. Beyond the expertise of CFD specialists, Acrossim empowers individuals in engineering education, design, analysis, communication, and decision-making to democratize access to CFD simulations.

3 Software Framework

3.1 Software Architecture

The toolkit processes CFD data from multiple CFD software to integrate simulation results with various formats into game engines. A modular component-based software structure is acquired by blending already available resources. It is developed with Python in an Anaconda environment and compiled with ParaView and Blender libraries. Standard Python libraries are also utilized to embed file-handling operations to control input and output (I/O) functionalities.

The toolkit comprises the following components: input, modules, processors, output, and storage. A detailed description of its schematic architecture is illustrated in Figure 1. It is noteworthy that CFD solvers give input files in various formats, thereby requiring custom data-handling features. To tackle these hindrances, the toolkit proposes distinguished modules for each CFD software within supplementary additions in the processors. Each module can serve as a standalone provider for a specific CFD solver. Native CFD datasets and ParaView states are two input files to appropriately run the toolkit. The prior can be in any format as long as it is imported in ParaView. The latter is only one time created to provide a baseline of CFD post-processing data. Besides, each module has three main processors based on data formats; tessellated, image, and text. Processors utilize state files to automatically integrate native CFD data into target output files.

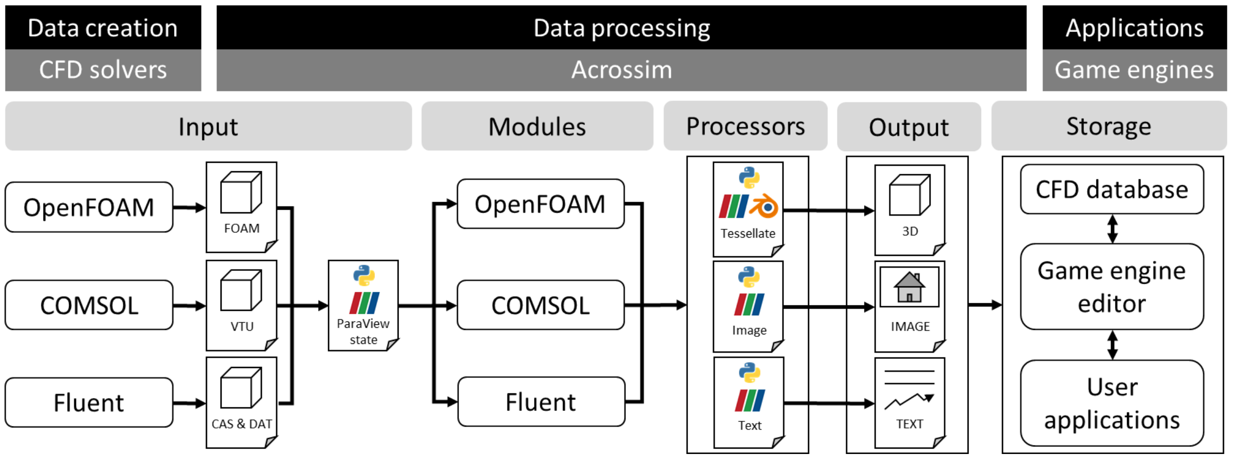

Figure 1 The schematic architecture of the toolkit with its components. For each input data fed by CFD software, the toolkit facilitates a dedicated data processing pipeline governed by a module to eventually output data in three main formats. More interestingly, this approach enables an co-simulation environment for the combined use of CFD data from various solves.

ParaView is an open-source scientific data visualization software that enables extraordinary features to post-process CFD data and create data extracts for third-party software. It comes with a Python application programming interface (API) making the integration feasible in Acrossim. Data export options in ParaView are not compatible with game engines by means of visual representations. Therefore, Blender, a 3D computer graphics software, is introduced in the data processing pipeline for tessellate processors. Blender is an open-source software popular among visual artists to create digital artifacts including 3D models. It gives support to 3D tessellated data formats that can be read by game engines. Blender is written in Python including an extensive Python API. This enables a seamless connection between ParaView and Blender to prepare visual CFD data for game engines, as well as recuperating geometric features along the data processing pipeline. It is significant to highlight that image and text data exported from ParaView can be directly imported into game engines without any processing. Both ParaView and game engines give support to the most common image and text formats.

CFD simulations are generally performed with either remote or cloud-based solutions due to the heavy calculations. Post-processing of simulation data can also result in computationally intensive workflows that can be demanding for end-user devices. Acrossim serves as a means for developers to integrate lightweight CFD data into game engines. Above all, a fully automated routine of Acrossim may facilitate a content delivery network remotely linking CFD solver with end-user applications. This connection can open gates for two-way coupled interactive systems with content delivery networks (CDN), thereby remotely streaming any CFD content to end-user devices and digital twins.

3.2 Software Functionalities

In this section, we identify functionalities provided by Acrossim and its modules.

3.2.1 Toolkit

OpenFOAM, COMSOL, and Ansys Fluent are CFD solvers implemented in the data processing pipeline. Based on a previous study that qualitatively investigated the integration of CFD data between CFD solvers and game engines [20], Acrossim proposes an automated data processing pipeline considering three major data formats; tessellated, image and text. Modules are highly customizable mediums to edit and implement user-specific demands. Data produced by Acrossim can be directly utilized in Unity and Unreal game engines, through which miscellaneous cross-platform applications such as mobile games and AR/VR experiences can be readily built.

3.2.2 Data processing modules and processors

The toolkit produces data in tessellated, image and text formats through the processors exclusively coded for each. Figure 2 depicts a generic flowchart of data processing in processors. By default, processors ask for a number of timesteps from native CFD data. Data processing performance is tracked by means of time and data size for tessellate processor where heavy calculations are performed. Neither image nor text processor does include a performance test. Several metadata are created alongside final outputs. An option is set in processors to clean redundant metadata. Each processor specifically requires a directory to store final data. Processors incrementally output log files to keep track of actions taken while running. Each output is named with timesteps of native simulation data.

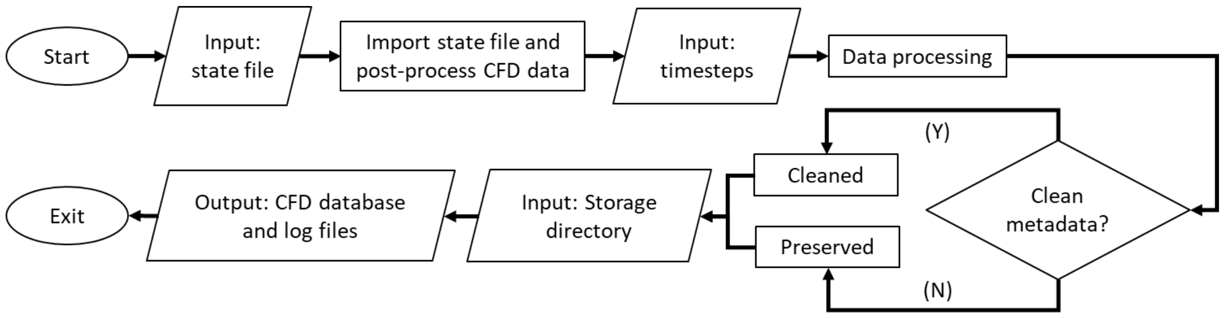

Figure 2 The generic data processing algorithm of Acrossim providing an end-to-end data processing pipeline between native CFD data from CFD solvers and CFD data extract to be utilized in game engines. Thanks to the modular structure of the toolkit, developers can easily manipulate the algorithm for use case specific requirements.

Tessellate processor: Data formats to visually represent 2D and 3D CFD data – for instance, contours, streamlines, and other filters – are directly derived from native CFD data in CFD post-processors. Game engines give support on tessellated 3D models other than data formats used in computer-aided design (CAD) and CFD post-processing software. The tessellate processor executes a pipeline through ParaView and Blender. It extracts visual CFD data in X3D format with ParaView. The data is then imported into Blender and converted to FBX format, which is qualitatively suitable and readable by game engines. Model geometry and mesh data can also be integrated with the tessellate processor. Alternatives can be deployed by only changing export data formats in the tessellate processor upon data formats supported by ParaView and Blender. More information on this can be found in the previous research [20].

Image and text processors: Visual CFD data is generally accompanied by color legends, charts & graphs, and text data. ParaView and game engines support the most commonly utilized formats for image (e.g., JPEG, PNG, EPS) and text (e.g., CSV, TXT) data.

OpenFOAM module: This module includes three processors as baselines for other modules. Tessellated, image and text data are directly processed from native CFD data produced by OpenFOAM. It merely requires a reference ParaView state file to perform the data processing.

COMSOL module: Data from COMSOL is exported to VTU format in order to import CFD data with ParaView and implement it in Acrossim. COMSOL module utilizes the same processors as OpenFOAM with one exception in the tessellate one. COMSOL can export CFD visual extract in VTU format as an asset of VTK legacy, thus enabling data processing with ParaView. It was observed that VTU encodes only simulation data rather than geometric features. Hence, the format does exclude all geometric features while extracting data from COMSOL, for instance, extrusion of streamline tubes from splines. To cope with this complication, tessellate processor in COMSOL module is extended using a bevel modifier in Blender to reconstruct 3D data from splines and curves. This extension loops with an “if conditioner” in the tessellate processor. It merely applies to streamlines, glyphs, and similar visual data.

Fluent modules: Fluent saves CFD data with a single CAS file where simulation settings are stored, and also DAT files for each written timestep. ParaView cannot inherit the same system to manage transient Fluent data. Each DAT file should be accompanied by a CAS file to be properly imported into ParaView. In order to automatically handle this, we wrote “fluenttoparaview.py” script through which CAS files are created and subsequently named for each DAT file. It automatically detects and converts Fluent native CFD data into formats that can be read by ParaView. No input timesteps are necessary for processors. All native data collected under the toolkit directory are directly processed.

4 Implementation and Empirical Results

This section details the implementation of Acrossim. An Anaconda environment with ParaView v5.8 and Blender v2.79 packages should be created to run processors. The environment can be effortlessly compiled by running “Acrossim.yml” file in Anaconda. A Dell laptop with Intel(R) Core(TM) i7-8850U CPU @ 2.6GHz - 32 GB with Windows 10 was employed to run modules. Further details are available in the online documentation with reference files and tutorials on GitHub: https://sersolmaz.github.io/Acrossim/.

4.1 OpenFOAM module

An example of implementation was given with OpenFOAM module. The module comprises baseline processors of Acrossim. COMSOL and Fluent modules basically inherit these processors with minor inclusions by only means of data preparation and recuperation. A CFD study was conducted utilizing “pitzDaily” tutorial from OpenFOAM Foundation. The native CFD data, processors, and reference state files should be documented under the same directory. Two data processing cases were performed to assess the implementation of the tessellate processor for 2D velocity contour and 3D velocity vector, including 5 timesteps for each, as shown in Figure 3. The case with 2D velocity contour was processed in 22 s whereas the data size is reduced 13.7 times. The case with the 3D velocity vector was performed in 46 s and 13 times reduction in data size. The results are consistent with previous work in which an optimal data processing pipeline was proposed while preserving data quality [20]. The integrated CFD data were stored in “CFD_database” folder created under the working directory. Data processing can be further speeded up with parallel programming utilities available for ParaView and Blender.

4.2 COMSOL module

VTU export from COMSOL does not include geometrical features, thereby eliminating volumetric data for streamlines and glyphs post-processing. Without data recuperation, streamline data in VTU format cannot be imported into game engines. Aside from data exchange as in the tessellate processor, Blender can also be utilized to recuperate and reconstruct geometric data from VTU file. This feature comes in tessellate processor of COMSOL module with “qStreamline” command which activates the bevel modifier in Blender to construct 3D data from curves and splines. Due care must be exercised to tune the depth and resolution of the geometry model in terms of data quality, size, and processing time.

Furthermore, despite the fact that transient COMSOL VTU savings were collected in the same directory to automatically process data through Acrossim, the process failed to import multiple timesteps in ParaView. This highlighted the need for a data merger missed in the working directory. ParaView enables default merging options to bundle transient VTU files from the same simulation. Dragging all VTU files in ParaView, a PVD file is produced which should be saved only one time. This PVD file is utilized in the reference state to automatically include all transient data in the data processing workflow.

4.3 Fluent module

In order to import Fluent CAS & DAT files in ParaView, we developed “fluenttoparaview.py” script which decompresses CAS & DAT files, creates a CAS for each DAT file, and subsequently renames to create CAS & DAT pairs for each timestep. This enables Acrossim to integrate Fluent data into the data processing pipeline. The “fluenttoparaview.py” code only runs one time to prepare native Fluent data for ParaView. The processors inherit “casedic” and “number_casedic” from “fluenttoparaview.py” to automatically update the reference state files and import them into ParaView. The main difference in processers of Fluent modules is that each timestep encompasses a CAS file instead of having a single CAS file for all DAT files. The processors include all available data in a row in the working directory, instead of requiring input timesteps. In practice, this doesn’t bring any differences for output files compared to other modules. It only relates to the internal handling of Fluent data in Acrossim.

5 Illustrative Examples

The main advantage of Acrossim is that the toolkit provides an inclusive workflow for developers who are keen to create user-friendly applications with CFD simulation data. From education to public dissemination of scientific results, entry-level can be plausibly eased. Here we demonstrate an example to show the potential of the toolkit to create user applications with integrated CFD data. A virtual reality user application developed with Unity 2019.2.18f1 is shown in Figure 3. The integrated CFD data were adapted from the implementation presented in Section 4.1. OpenFOAM module. The data bundled in “CFD_database” was moved to the asset directory of the project in Unity editor. It should also be noted that given the scope of the present work we only provide digital tools to integrate CFD data into cross-platform environments. Hence, except for guidance, no digital resource is given in the development of the user application with Unity.

Unity is a cross-platform game engine available free of charge. The software enables developers to create cross-platform applications such 2D/3D mobile games and AR/VR environments. Its asset store provides miscellaneous – both free and paid – materials to create functional user environments. Developers utilize Unity game engine editor to create digital experiences through malleable and easily customized features. Each CFD data is represented with either a game object or material in Unity. By default, all simulation data are simultaneously streamed in Unity. In order to create animations from transient simulation data, developers can simply use animator controllers in animation systems to provide functionalities to create specific animations without coding. Once an animation clip is developed with game objects, it can be renewed automatically changing the data in the existing directory. Moreover, Unity is written in C# and comes with an API. Integrated CFD data in text format can be implemented in the user application to create interactive numerical and text data.

Acrossim is a highly customizable toolkit that can serve any user-defined demands. Either ParaView state files or relevant processors can be modified to obtain specific processing workflows. For instance, in order to change color legend without creating a new state file, the user can go into the existing state file and manipulate the relevant object to change color codes. This requires an understanding of the ParaView Python API. The trace option in ParaView dynamically illustrates each action taken by the user with a graphical user interface (GUI) in ParaView. Hence users can intuitively decode ParaView API by keeping track of traces. Similarly, available documentation of ParaView API can aid in finding the target object to process user-defined needs. This may enable remote data processing through connected systems for specific use cases.

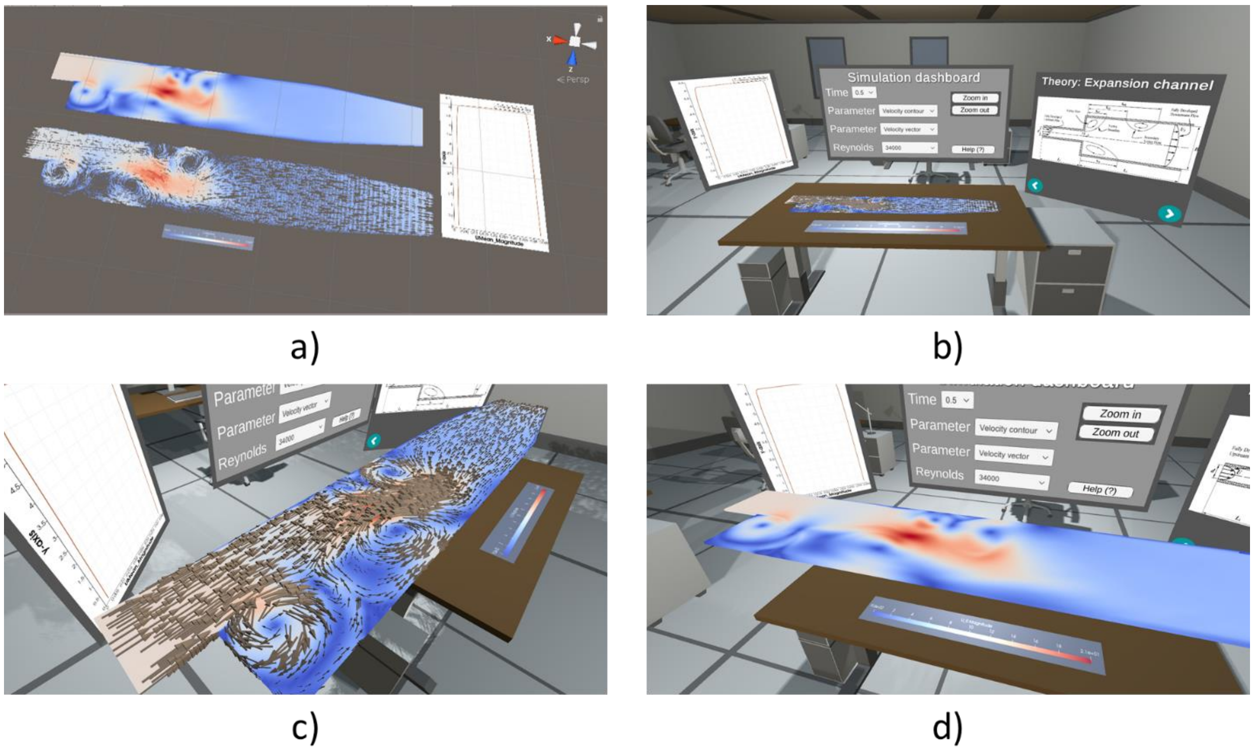

Figure 3 A user application with CFD data developed via Unity; a) integrated CFD data in Unity editor; b) a scene from VR environment; c) CFD data in VR environment; d) CFD data and GUI in VR environment. This is an illustrative example to highlight the potential of game engines with CFD data to enable user-friendly visualization environments.

Whilst Unity inherently builds standalone applications by default, it also enables maintaining connected systems within complementary CDNs and cloud service providers to remotely update data in user applications. Acrossim is originally developed concerning the emergency of remote and cloud-based data processing approaches. This makes the CFD data interoperable and any user device employable.

6 Conclusions

CFD simulations are genuinely multidimensional and require supportive information for CFD non-expert interactions. Cross-platform applications developed with game engines can open gates for highly customizable user applications for any digital medium. Acrossim, an open-source data processing toolkit, integrates complex CFD simulation data into game engines to increase the utility of CFD simulation results in broader domains. Notably, for educational practices, the toolkit may lead to supplementary learning cycles with accurately produced content and toward learning CFD simulations.

The toolkit, written in Python, is versatile and adaptable maintaining integration among multiple CFD solvers and game engines. The component-based modular structure of Acrossim makes the software open for updates and new workflows, bringing an inclusive long-lasting toolkit for cross-platform integrations of CFD applications. The data processing and integration capabilities of Acrossim will be an easy-to-use tool for researchers and practitioners who are keen to integrate CFD simulations into user-friendly accessible mediums.

Acknowledgements
This project has received funding from the European Union's EU Framework Programme for Research and Innovation Horizon 2020 under Grant Agreement 812716. This publication reflects only the authors' view exempting the community from any liability. Project website: https://charming-etn.eu/.

References

[1] S. Su *et al.*, "Virtual and Augmented Reality Applications to Support Data Analysis and Assessment of Science and Engineering," *Comput. Sci. Eng.*, vol. 22, no. 3, pp. 27–39, May 2020, doi: 10.1109/MCSE.2020.2971188.
[2] A. Suh and J. Prophet, "The state of immersive technology research: A literature analysis," *Comput. Hum. Behav.*, vol. 86, pp. 77–90, Sep. 2018, doi: 10.1016/j.chb.2018.04.019.
[3] S.-Y. Lee, J.-G. Kim, R.-W. Kim, U.-H. Yeo, and I.-B. Lee, "Development of three-dimensional visualisation technology of aerodynamic environment in fattening pig house using CFD and VR technology," *Comput. Electron. Agric.*, vol. 194, p. 106709, Mar. 2022, doi: 10.1016/j.compag.2022.106709.
[4] K. Takrouri, E. Causton, and B. Simpson, "AR Technologies in Engineering Education: Applications, Potential, and Limitations," *Digital*, vol. 2, no. 2, pp. 171–190, May 2022, doi: 10.3390/digital2020011.
[5] Y. Wei, J. Orlosky, and T. Mashita, "Visualization and Manipulation of Air Conditioner Flow via Touch Screen," in *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, Lisbon, Portugal: IEEE, Mar. 2021, pp. 430–431. doi: 10.1109/VRW52623.2021.00097.
[6] J. Yan, K. Kensek, K. Konis, and D. Noble, "CFD Visualization in a Virtual Reality Environment Using Building Information Modeling Tools," *Buildings*, vol. 10, no. 12, p. 21, 2020, doi: https://doi.org/10.3390/buildings10120229.
[7] A. C. Kirby, Z. Yang, D. J. Mavriplis, E. P. Duque, and B. J. Whitlock, "Visualization and Data Analytics Challenges of Large-Scale High-Fidelity Numerical Simulations of Wind Energy Applications," in *2018 AIAA Aerospace Sciences Meeting*, Kissimmee, Florida: American Institute of Aeronautics and Astronautics, Jan. 2018. doi: 10.2514/6.2018-1171.
[8] M. Berger and V. Cristie, "CFD Post-processing in Unity3D," *Procedia Comput. Sci.*, vol. 51, pp. 2913–2922, 2015, doi: 10.1016/j.procs.2015.05.476.
[9] J. Yao, Y. Lin, Y. Zhao, C. Yan, C. Li, and P. F. Yuan, "Augmented reality technology based wind environment visualization," in *Learning, Adapting and Prototyping - Proceedings of the 23rd CAADRIA Conference*, 2018, pp. 369–377.
[10] Y. Yu, M. Duan, C. Sun, Z. Zhong, and H. Liu, "A virtual reality simulation for coordination and interaction based on dynamics calculation," *Ships Offshore Struct.*, vol. 12, no. 6, pp. 873–884, Aug. 2017, doi: 10.1080/17445302.2017.1293762.

[11] M. Wang, N. Férey, F. Magoulès, and P. Bourdot, "Interactive Simulation for easy Decision-making in Fluid Dynamics," *Eurographics 2021 - Short Pap.*, p. 4 pages, 2021, doi: 10.2312/EGS.20211022.
[12] J. Huang, S. K. Ong, and A. Y.-C. Nee, "An approach for augmented learning of finite element analysis," *Comput. Appl. Eng. Educ.*, Jun. 2019, doi: 10.1002/cae.22125.
[13] M. Kim, S. Yi, D. Jung, S. Park, and D. Seo, "Augmented-Reality Visualization of Aerodynamics Simulation in Sustainable Cloud Computing," *Sustainability*, vol. 10, no. 5, p. 1362, Apr. 2018, doi: 10.3390/su10051362.
[14] G. Wheeler *et al.*, "Virtual interaction and visualisation of 3D medical imaging data with VTK and Unity," *Healthc. Technol. Lett.*, vol. 5, no. 5, pp. 148–153, Oct. 2018, doi: 10.1049/htl.2018.5064.
[15] S. Baheri Islami, M. Wesolowski, W. Revell, and X. Chen, "Virtual Reality Visualization of CFD Simulated Blood Flow in Cerebral Aneurysms Treated with Flow Diverter Stents," *Appl. Sci.*, vol. 11, no. 17, p. 8082, Aug. 2021, doi: 10.3390/app11178082.
[16] T. M. Wasfy and A. K. Noor, "Visualization of CFD results in immersive virtual environments," *Adv. Eng. Softw.*, vol. 32, no. 9, pp. 717–730, Sep. 2001, doi: 10.1016/S0965-9978(01)00020-5.
[17] T. Fukuda, K. Yokoi, N. Yabuki, and A. Motamedi, "An indoor thermal environment design system for renovation using augmented reality," *J. Comput. Des. Eng.*, Jun. 2018, doi: 10.1016/j.jcde.2018.05.007.
[18] L. Bergonzi, G. Colombo, D. Redaelli, and M. Lorusso, "An Augmented Reality Approach to Visualize Biomedical Images," *Comput.-Aided Des. Appl.*, vol. 16, no. 6, pp. 1195–1208, Mar. 2019, doi: 10.14733/cadaps.2019.1195-1208.
[19] W. Li, A. Nee, and S. Ong, "A State-of-the-Art Review of Augmented Reality in Engineering Analysis and Simulation," *Multimodal Technol. Interact.*, vol. 1, no. 3, p. 17, Sep. 2017, doi: 10.3390/mti1030017.
[20] S. Solmaz and T. Van Gerven, "Automated integration of extract-based CFD results with AR/VR in engineering education for practitioners," *Multimed. Tools Appl.*, Feb. 2021, doi: 10.1007/s11042-021-10621-9.

Required Metadata: Current code version

Table – Code metadata

Nr	Code metadata description	*Please fill in this column*
C1	Current Code version	*v1.0*
C2	Permanent link to code / repository used of this code version	*https://github.com/sersolmaz/Acrossim*
C3	Legal Code License	*GPL-3.0*
C4	Code Versioning system used	*none*
C5	Software Code Language used	*Python*
C6	Compilation dependencies	*Anaconda, ParaView, Blender, and Python packages*
C7	If available Link to developer documentation / manual	*https://sersolmaz.github.io/Acrossim/*
C8	Support email for questions	*serknslmz@gmail.com*