The Journal of Open Source Software

FURY: advanced scientific visualization

Eleftherios Garyfallidis[1], Serge Koudoro[1], Javier Guaje[1], Marc-Alex Côté[3], Soham Biswas[4], David Reagan[5], Nasim Anousheh[1], Filipi Silva[2], Geoffrey Fox[1], and FURY Contributors[6]

1 Department of Intelligent Systems Engineering, Luddy School of Informatics, Computing and Engineering, Indiana University, Bloomington, IN, USA **2** Network Science Institute, Indiana University, Bloomington, IN, USA **3** Microsoft Research, Montreal, Canada **4** Department of Computer Science and Engineering, Institute of Engineering and Management, Kolkata, India **5** Advanced Visualization Lab, University Information Technology Services, Indiana University, Bloomington, IN, USA **6** Anywhere in the Universe

DOI: 10.21105/joss.03384

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @chrishavlin
- @rougier
- @phamvanvung

Submitted: 08 April 2021
Published: 20 July 2021

Summary

Free Unified Rendering in pYthon (FURY), is a community-driven, open-source, and high-performance scientific visualization library that harnesses the graphics processing unit (GPU) for improved speed, precise interactivity, and visual clarity. FURY provides an integrated API in Python that allows UI elements and 3D graphics to be programmed together. FURY is designed to be fully interoperable with most projects of the Pythonic ecosystem that use NumPy (C. R. Harris et al., 2020) for processing numerical arrays. In addition, FURY uses core parts of VTK (Schroeder et al., 1996) and enhances them using customized shaders. FURY provides access to the latest technologies such as raytracing, signed distance functionality, physically based rendering, and collision detection for direct use in research. More importantly, FURY enables students and researchers to script their own 3D animations in Python and simulate dynamic environments.

Statement of need

The massive amount of data collected and analyzed by scientists in several disciplines requires powerful tools and techniques able to handle these whilst still managing efficiently the computational resources available. In some particular disciplines, these datasets not only are large but also encapsulate the dynamics of their environment, increasing the demand for resources. Although 3D visualization technologies are advancing quickly (Sellers & Kessenich, 2016), their sophistication and focus on non-scientific domains makes it hard for researchers to use them. In other words, most of the existing 3D visualization and computing APIs are low-level (close to the hardware) and made for professional specialist developers. Because of these issues, there is a significant barrier to many scientists and these powerful technologies are rarely deployed to everyday research practices.

Therefore, FURY is created to address this necessity of high-performance 3D scientific visualization in an easy-to-use API fully compatible with the Pythonic ecosystem.

FURY Architecture

FURY is built to be modular, scalable, and to respect software engineering principles including a well-documented codebase and unit integration testing. The framework runs in all major

The Journal of Open Source Software

operating systems including multiple Linux distributions, Windows, and macOS. Also, it can be used on the desktop and the web. The framework contains multiple interconnected engines, modules, API managers as illustrated in Figure 1.

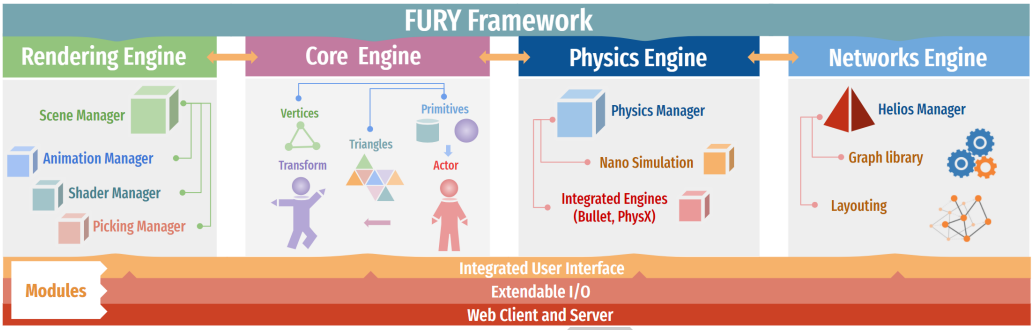

Figure 1: The FURY framework contains multiple interconnected engines to bring forward advanced visualization capabilities. Additionally, it contains an integrated user interface module and an extendable I/O module. One of the most important classes is the Scene Manager that connects the actors to the shaders, animations, and interactors for picking 3D objects. The actors are directly connected to NumPy arrays with vertices, triangles, and connectivity information that is provided by the core engine. These are then connected to the physics and networks engines.

Rendering Engine: This engine includes managers like scene, animation, shader, and picking manager. The scene manager allows the visual objects to appear on a canvas. The picking manager allows selecting specific objects in the scene. The animation manager allows users to script their own 3D animations and videos with timelines allowing objects to act in specific times. Lastly, the shader manager provides several interfaces to different elements in the OpenGL rendering pipeline. This manager allows developers to add customized shaders snippets to the existing shaders included in the API.

Core Engine: This engine contains utilities for building actors from primitives and transforming them. A primitive is an object that describes its shape and connectivity with elements such as vertices and triangles.

Physics Engine: This engine allows us to either build collision mechanisms as used in molecular dynamics or integrate well-established engines such as Bullet (Coumans & others, 2013) and NVIDIA PhysX (M. Harris, 2009).

Networks Engine: This engine allows for the creation and use of graph systems and layouts.

Integrated User Interfaces Module: FURY contains its own user interfaces. This module provides a range of UI 2D / 3D elements such as buttons, combo boxes, and file dialogues. Nevertheless, users can easily connect to other known GUIs such as Qt or IMGUI if necessary.

I/O module: FURY supports a range of file formats from the classic OBJ format to the more advanced GLTF format that can be used to describe a complete scene with many actors and animations.

Interoperability: FURY can be used together with projects such as SciPy (Virtanen et al., 2020), Matplotlib (Hunter, 2007), pandas (McKinney & others, 2010), scikit-learn (Pedregosa et al., 2011), NetworkX (Hagberg et al., 2008), PyTorch (Paszke et al., 2019) and TensorFlow (Abadi et al., 2016).

FURY's visualization API can be compared with VisPy (Campagnola et al., 2015), glumpy (Rougier, 2015), Mayavi (Ramachandran & Varoquaux, 2011), and others. VisPy and glumpy directly connect to OpenGL. FURY uses OpenGL through Python VTK, which can be advantageous because it can use the large stack of visualization algorithms available in VTK. This is similar to Mayavi, however, FURY provides an easy and efficient way to ease interaction

with 3D scientific data via integrated user interface elements and allows to reprogram the low-level shaders for the creation of stunning effects (see Figure 2) not available in VTK. Historically, FURY had also a different path than these libraries as it was originally created for heavy-duty medical visualization purposes for DIPY (Garyfallidis et al., 2014). As the project grew it spinned off as an independent project with applications across the domains of science and engineering including visualization of nanomaterials and robotics simulations.

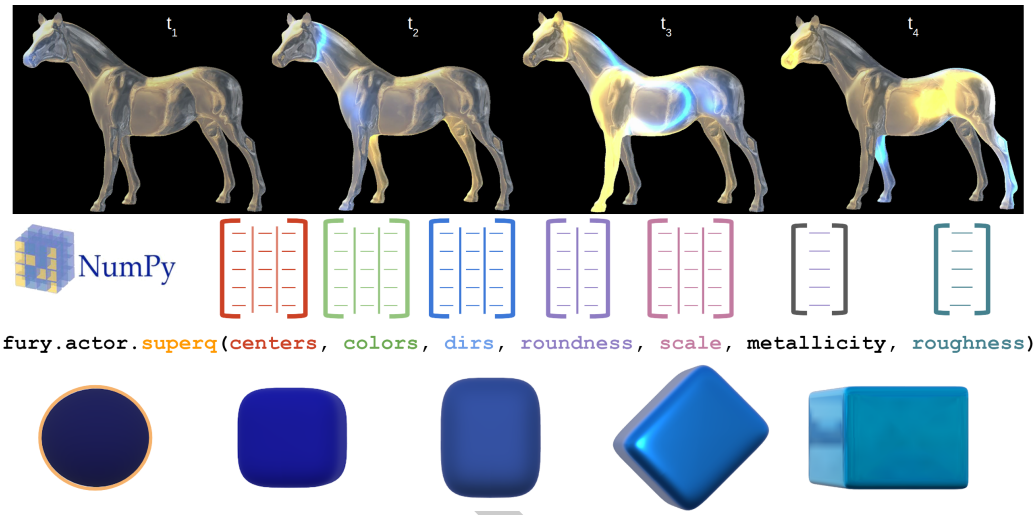

Figure 2: Top. Dynamic changes are shown as diffused waves on the surface of the horse visualized with FURY. Showing here 4 frames at 4 different time points (t1—t4). A vertex and fragment shader are used to calculate in real-time the mirroring texture and blend its colors with the blue-yellow wave. **Bottom**. In FURY we create actors that contain multiple visual objects controlled by NumPy arrays. Here an actor is generating 5 superquadrics with different properties (e.g. colors, directions, metallicity) by injecting the information as NumPy arrays in a single call. This is one of the important design choices that make FURY easier to use but also faster to render. Actors in FURY can contain many objects. The user can select any of the objects in the actor. Here the user selected the first object (spherical superquadric).

Acknowledgements

FURY is partly funded through NSF #1720625 Network for Computational Nanotechnology - Engineered nanoBIO Node (Klimeck et al., 2008).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 265–283. ISBN: 9781931971331

Campagnola, L., Klein, A., Larson, E., Rossant, C., & Rougier, N. P. (2015). VisPy: Harnessing the GPU for fast, high-level visualization. *Proceedings of the 14th Python in Science Conference*. https://doi.org/10.25080/majora-7b98e3ed-00e

Coumans, E., & others. (2013). Bullet physics library. *Open Source: Bulletphysics.org*, *15*(49), 5. http://www.bulletphysics.org

Garyfallidis, E., Brett, M., Amirbekian, B., Rokem, A., Van Der Walt, S., Descoteaux, M., & Nimmo-Smith, I. (2014). Dipy, a library for the analysis of diffusion MRI data. *Frontiers in Neuroinformatics*, *8*, 8. https://doi.org/10.3389/fninf.2014.00008

Hagberg, A., Swart, P., & S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference*, 11–15. http://conference.scipy.org/proceedings/SciPy2008/paper_2/full_text.pdf

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., & others. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harris, M. (2009). CUDA fluid simulation in NVIDIA PhysX. *SIGGRAPH Asia*.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *IEEE Annals of the History of Computing*, *9*(03), 90–95. https://doi.org/10.1109/mcse.2007.55

Klimeck, G., McLennan, M., Brophy, S. P., Adams III, G. B., & Lundstrom, M. S. (2008). Nanohub.org: Advancing education and research in nanotechnology. *Computing in Science & Engineering*, *10*(5), 17–23. https://doi.org/10.1109/MCSE.2008.120

McKinney, W., & others. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, *445*, 51–56. https://doi.org/10.25080/majora-92bf1922-00a

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., & others. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

Ramachandran, P., & Varoquaux, G. (2011). Mayavi: 3D visualization of scientific data. *Computing in Science & Engineering*, *13*(2), 40–51. https://doi.org/10.1109/mcse.2011.35

Rougier, N. P. (2015). Glumpy. *EuroScipy*. https://hal.inria.fr/hal-01217524

Schroeder, W. J., Martin, K. M., & Lorensen, W. E. (1996). The design and implementation of an object-oriented toolkit for 3D graphics and visualization. *Proceedings of Seventh Annual IEEE Visualization '96*, 93–100. https://doi.org/10.1109/VISUAL.1996.567752

Sellers, G., & Kessenich, J. (2016). *Vulkan programming guide: The official guide to learning Vulkan*. Addison-Wesley Professional. ISBN: 978-0134464541

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2