Diderot: A Domain-Specific Language for Visualizing FEniCS Functions

Teodoro Fields Collin[*], Hannah Morgan,
L. Ridgway Scott, John Reppy, Gordon Kindlmann, Charisee Chiw
Dept of Computer Science, University of Chicago.

Keywords: *FEniCS, Visualization, Higher Order Elements, Ray tracing, Diderot*

We describe ongoing work on extending the Diderot programming language for scientific visualization [8, 4] to handle functions produced by FEniCS. FEniCS currently works with various visualization tools [5, 6, 1, 3], and previous FEniCS conferences noted the need for better visualizations [2]. Existing tools limit the FEniCS user's ability to create high quality visualizations either because they support only approximations of higher-order elements, or because they do not support writing new visualization algorithms. For instance, Paraview can draw only piecewise linear functions, and while VisIt has a scripting language, using it to implement a new visualization algorithm involves writing a new plotting plugin, which requires knowing VisIt's API and writing C code to implement complex math. In contrast, Diderot supports higher-order elements and offers a concise and mathematically expressive language for implementing visualizations. Building on previous work [4][1], we show here how Diderot can express commonly available basic visualizations, but also supports creating new visualization methods that both leverage scientific visualization techniques and accurately depict the structure of FEM solutions. Collaboration with the FEniCS community will increase the utility and relevance of Diderot for scientific computing.

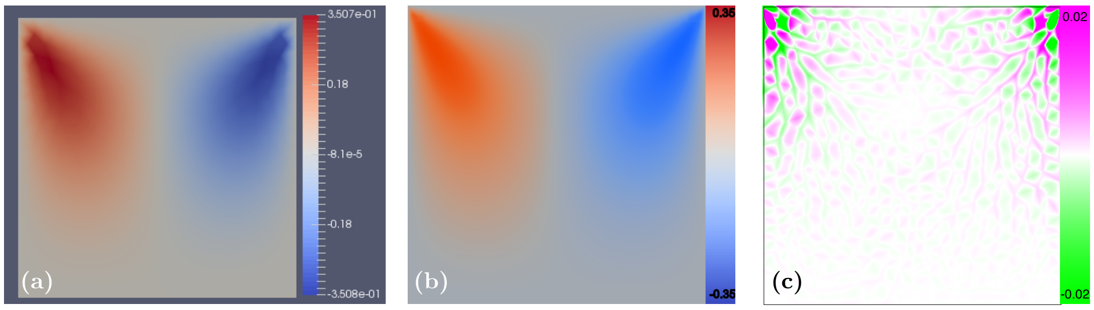

Figure 1: Paraview's plot (a) of a 2D scalar field can made more accurate in Diderot (b), and Diderot can directly visualize the difference (c), caused by Paraview's simplification to linear elements.

Basic Visualizations Fig. 1 illustrates how Paraview and Diderot handle a basic visualization method: colormapping a 2-D scalar field (a component of a 2-D lid driven cavity solution). Fig. 1(a) shows sample Paraview output, and roughly 20 lines of Diderot code[2] generated Fig. 1(b). There are in fact slight differences between the two, because Diderot is showing the true piecewise quartic function, while Paraview is showing the projection down to linear elements. With Diderot we can also visualize (Fig. 1(c)) the difference between the true result and linear approximation. Juxtaposing these results may create the impression that programming in Diderot is convenient as running "`plot`" in Dolfin. For certain forms of stock visualization (such as these colormaps), underlying Diderot programs may be wrapped within other high-level Python functions and interactive GUIs that can achieve comparable usability.

Advancing on the Basics Fig. 2 illustrates how Dolfin's "`plot`" and Diderot visualize a 2D vector field (again a lid driven cavity problem). Fig. 2(a) shows a plot of arrows drawn at each mesh vertex. Fig. 2(b) is a line integral convolution (LIC) of the vector field, created with about 50 lines of Diderot code and some post-processing. LIC more comprehensively displays the vector field structure, in a way that can be easily augmented with additional information, such as the field derivative.

Making your own visualization algorithms Fig. 1(b) and Fig. 2(b) show Diderot's ability to provide simple and more sophisticated visualizations. With its LIC plugin, Paraview can make a version of Fig. 2(b). Diderot is designed to simplify how new vis algorithms are created, when there is no existing

[*]Corresponding Author, teocollin@uchicago.edu
[1]`http://easychair.org/smart-program/FEniCS%2715/2015-06-29.html#talk:6037`
[2]See `https://github.com/Diderot-Language/examples` for code examples

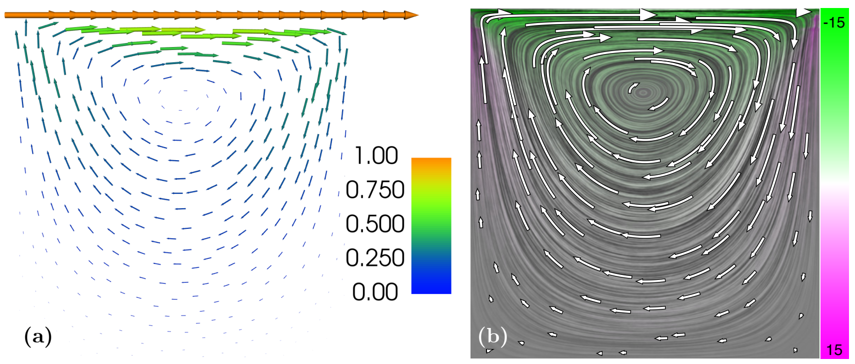

Figure 2: A 2D vector field is visualized with Dolfin's "`plot`" (a) or with line integral convolution in Diderot (b), which includes a colormap of vorticity.

plugin or other implementation. We show an example of how Diderot's concise and mathematically expressive syntax support creating new visualizations algorithms, including those that connect directly to the mathematics of FEM. Some previous research has studied visualization specific to FEM [10, 9]; we focus on the simple problem of accurately plotting a 2D scalar field as a 3D surface.

Piecewise linears can be efficiently plotted with standard polygon-based graphics (such as those used in Paraview). However, depicting the fields associated with higher-order basis functions benefits from a non-polygonal approach, such as ray-tracing, wherein the intersection of a per-pixel ray with the object of interest is computed via root-finding. Iterative root-finding may fail to converge for a variety of reasons. To avoid the distracting jagged image artifacts caused by incorrect ray-object intersections, we modified the guaranteed surface-intersection algorithm of Kalra and Barr [7], and implemented it in Diderot.

The Kalra and Barr method uses a Lipschitz constant of the scalar field derivative to find intervals that bracket exactly one root, within which numerical root finding is more reliable. The algorithm can fail to find root intervals, but the potential for failure can be reduced by picking a small Lipschitz constant.

Our modified algorithm considers the scalar field within each cell, to avoid discontinuities without missing roots, and computes the Lipschitz constant of the derivative within each cell according to knowledge of the finite element space. Figure 3 compares Dolfin's "`plot`" output to that of our new raytracer.

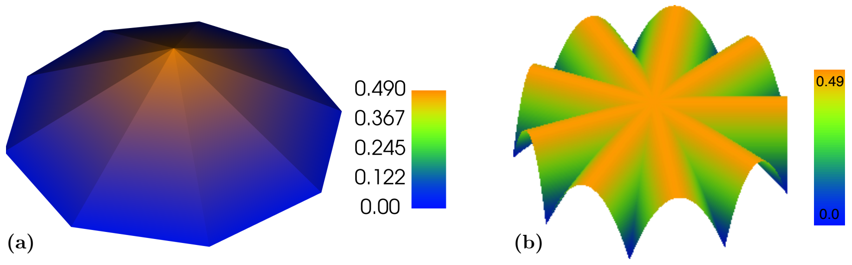

Figure 3: Dolfin (a) and Diderot (b) visualizations of the error in a solution to $\Delta u = 4$ with Dirichlet boundary conditions and higher order elements on an approximation to the unit circle. The raytracing accurately shows the shape of the error function, for which the linearly interpolated approximation gives a particularly misleading impression of the nature of the error function, and of the relationship between the approximate boundary conditions and the error.

References

[1] GLVis: Accurate finite element visualization. glvis.org.

[2] M. S. Alnæs, V. T. Fauske, and M. Ragan-Kelley. 3D visualization with FEniCS in Jupyter notebooks. In J. S. Hale, editor, *Proc. FEniCS 2017*, page 10, June 2017.

[3] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, and P. Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pages 357–372. Oct 2012.

[4] C. Chiw, G. L. Kindlmann, and J. H. Reppy. An exploration to visualize finite element data with a DSL. *CoRR*, abs/1706.05718, 2017.

[5] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Intl. J. Numerical Methods in Engineering*, 79(11):1309–1331, sep 2009.

[6] C. D. Hansen and C. R. Johnson. *Visualization Handbook*. Academic Press, 2004.

[7] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *Proc. SIGGRAPH '89*. ACM Press, 1989.

[8] G. Kindlmann, C. Chiw, N. Seltzer, L. Samuels, and J. Reppy. Diderot: a domain-specific language for portable parallel scientific visualization and image analysis. *IEEE TVCG*, 22(1):867–876, Jan. 2016.

[9] M. Meyer, B. Nelson, R. Kirby, and R. Whitaker. Particle systems for efficient and accurate high-order finite element visualization. *IEEE TVCG*, 13(5):1015–1026, Sept. 2007.

[10] B. Nelson and R. Kirby. Ray-tracing polymorphic multidomain spectral/hp elements for isosurface rendering. *IEEE TVCG*, 12(1):114–125, Jan. 2006.