MOX-Report No. 04/2022

lifex - heart module: a high-performance simulator for the cardiac function

Africa, P.C.; Piersanti, R.; Fedele, M.; Dede', L.; Quarteroni, A.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it http://mox.polimi.it

life^{x} – heart module: a high-performance simulator for the cardiac function

Package 1: Fiber generation

Pasquale C. Africa[a], Roberto Piersanti[a], Marco Fedele[a], Luca Dede'[a], and Alfio Quarteroni[a,b]

[a]MOX, Department of Mathematics, Politecnico di Milano, Italy
[b]Institute of Mathematics, École Polytechnique Fédérale de Lausanne, Switzerland (Professor emeritus)

10 January 2022

Abstract

Modeling the whole cardiac function involves several complex multi-physics and multi-scale phenomena that are highly computationally demanding, which makes calling for simpler yet accurate, high-performance computational tools still a paramount challenge to be addressed. Despite all the efforts made by several research groups worldwide, no software has progressed as a standard reference tool for whole-heart fully-coupled cardiac simulations in the scientific community yet.

In this work we present the first publicly released package of the heart module of life^{x}, a high-performance solver for multi-physics and multi-scale problems, aimed at cardiac applications.

The goal of life^{x} is twofold. On the one side, it aims at making *in silico* experiments easily reproducible and accessible to the wider public, targeting also users with a background in medicine or bio-engineering, thanks to an extensive documentation and user guide. On the other hand, being conceived as an academic research library, life^{x} can be exploited by scientific computing experts to explore new modeling and numerical methodologies within a robust development framework.

life^{x} has been developed with a modular structure and will be released bundled in different modules/packages. This initial release includes a generator for myocardial fibers based on Laplace-Dirichlet-Rule-Based-Methods (LDRBMs). This report comes with an extensive

Official website: https://lifex.gitlab.io/
Download link: https://doi.org/10.5281/zenodo.5810269

technical and mathematical documentation to welcome new users to the core structure of a prototypical `life`^{x} application and to provide them with a possible approach to include the generated cardiac fibers into more sophisticated computational pipelines.

Contents

1 Introduction

Nowadays, many applications in astrophysics, biology, energy, engineering, environmental science, and material science demand for accurate *in silico* models matching multi-physics phenomena on a multi-scale range [1].

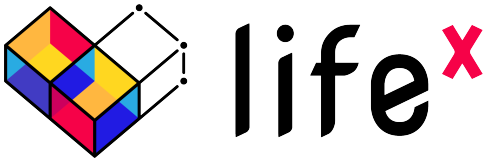

Figure 1: `life`^{x} official logo.

The human heart function is a complex system involving interacting phenomena at the molecular, cellular, tissue, and organ levels with widely varying time scales. Therefore, it is still among the most arduous modeling and computational challenges in a field where *in silico* models and experiments are essential to reproduce both physiological and pathological behaviors, which can improve the overall knowledge of the underlying biological and physical phenomena involved [2].

A satisfactory model for the whole cardiac function must be able to describe a wide range of different processes, such as the propagation of the trans-membrane potential and the flow of ionic species in the myocardium, the deformation caused by the muscles contraction, as well as the dynamics of the blood inside the heart chambers and flowing through the valves [3].

These demanding aspects make whole-heart fully-coupled simulations computationally highly expensive and call for simpler yet accurate, high-performance computational tools.

In this work we introduce `life`^{x} (pronounced /ˌlaɪfˈɛks/), a high-performance library for the solution of multi-physics and multi-scale differential problems. It is written in `C++` using the most modern programming techniques available in the `C++17` standard and is based on the `deal.II`[1] [4] finite element core. The official `life`^{x} logo is shown in fig. 1.

All the code is natively parallel and designed to run on diverse architectures, ranging from laptop computers to High Performance Computing (HPC) facilities and cloud platforms.

Despite being conceived as an academic research library in the frame of the iHEART project (see section 8), `life`^{x} is intended to provide the scientific community with a tool for real world applications that boosts the user and developer experience without sacrificing its computational efficiency and universality.

The initial development of `life`^{x} has been oriented towards a `heart` module incorporating several state-of-the-art core models for the simulation of cardiac electrophysiology, mechanics, electromechanics, blood fluid dynamics and myocardial perfusion. Such models have been recently exploited for a variety of standalone or coupled simulations both under physiological and pathological conditions (see, *e.g.*, [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]).

[1] `https://www.dealii.org/`

Since cardiac fibers drive the electric signal propagation throughout the myocardium, accurately modeling their arrangement is the essential building block for simulating cardiac electrophysiology and mechanical deformation. This initial release includes a generator for myocardial fibers based on Laplace-Dirichlet-Rule-Based-Methods (LDRBMs) [15], with application to a number of different prototypical and realistic geometries (slab models, left ventricles and left atria).

The content presented hereafter is organized as follows: section 2 describes the technical specifications of the software release associated with this documentation, *i.e.* the package content, licensing details and software/hardware requirements; in section 3 are detailed the instructions to run a generic `life`^{x} application; then, in sections 4, 5 and 6 is outlined a generic pipeline (along with guided examples) for generating cardiac fibers on different geometries, from the creation of an input mesh satisfying `life`^{x} prerequisites to the post-processing of the generated results and a possible way to integrate them into more sophisticated computational models. Finally, in section 7 we present a possible plan for future releases and developments.

2 Technical specifications

In this section we specify the package content, copyright and licensing information and software/hardware specifications required by the present `life`^{x} release.

2.1 Package content

The fiber generation package of `life`^{x} is delivered in binary form as an `AppImage`[2] executable.

This provides a universal package for `x86-64 Linux` desktop systems, without the need to deliver different distribution-specific versions. From the user's perspective, this implies an effortless *download-then-run* process, without having to manually take care of installing the proper system dependencies required.

Once the source code will be made publicly accessible, a standard *build-from-source* procedure with automatic installers will be available to make the dependencies setup tailored to the specific hardware of HPC facilities or cloud platforms.

2.2 License and third-party software

This work is copyrighted by the `life`^{x} authors and licensed under the Creative Commons Attribution Non-Commercial No-Derivatives 4.0 Interna-

[2] https://appimage.org/

tional License[3].

life^x makes use of third-party libraries. Please note that such libraries are copyrighted by their respective authors (independent of the life^x authors) and are covered by various permissive licenses.

The third-party software bundled with (in binary form), required by, copied, modified or explicitly used in life^x is listed here: Boost[4], PETSc[5], Trilinos[6], deal.II[7], VTK[8].

Some of the packages listed above, as stated by their respective authors, rely on additional third-party dependencies that may also be bundled (in binary form) with life^x, although not used directly. These dependencies include: ADOL-C[9], ARPACK-NG[10], BLACS[11], Eigen[12], FFTW[13], GLPK[14], HDF5[15], HYPRE[16], METIS[17], MUMPS[18], NetCDF[19], OpenBLAS[20], ParMETIS[21], ScaLAPACK[22], Scotch[23], SuiteSparse[24], SuperLU[25], oneTBB[26], p4est[27].

The libraries listed above are all free software and, as such, they place few restrictions on their use. However, different terms may hold. Please refer to the content of the folder doc/licenses/ for more information on license and copyright statements for these packages.

Finally, an MPI installation (such as Open MPI[28]) may also be required to successfully run life^x executables in parallel.

[3] http://creativecommons.org/licenses/by-nc-nd/4.0/
[4] https://www.boost.org/
[5] https://www.mcs.anl.gov/petsc/
[6] https://trilinos.github.io/
[7] https://www.dealii.org/
[8] https://vtk.org/
[9] https://github.com/coin-or/ADOL-C
[10] https://github.com/opencollab/arpack-ng
[11] https://www.netlib.org/blacs/
[12] https://eigen.tuxfamily.org/
[13] https://www.fftw.org/
[14] https://www.gnu.org/software/glpk/
[15] https://www.hdfgroup.org/solutions/hdf5/
[16] https://www.llnl.gov/casc/hypre/
[17] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
[18] http://mumps.enseeiht.fr/index.php?page=home
[19] https://www.unidata.ucar.edu/software/netcdf/
[20] https://www.openblas.net/
[21] http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
[22] https://www.netlib.org/scalapack/
[23] https://gitlab.inria.fr/scotch/scotch
[24] https://people.engr.tamu.edu/davis/suitesparse.html
[25] https://portal.nersc.gov/project/sparse/superlu/
[26] https://oneapi-src.github.io/oneTBB/
[27] https://www.p4est.org/
[28] https://www.open-mpi.org/

2.3 Software and hardware requirements

As an AppImage, this life^x release has been built on Debian Buster (the current oldstable version)[29] following the *"Build on old systems, run on newer systems"* paradigm[30].

Therefore, it is expected to run on (virtually) any *recent enough* x86-64 Linux distribution, assuming that a version of glibc[31] not older than 2.28 is installed.

3 How to run life^x executables

The following steps are required to run a life^x executable.

3.1 Quick start

First, download and extract the life^x release archive available at https://doi.org/10.5281/zenodo.5810269. Then[32][33]:

1. open a terminal;
2. move to the directory containing the lifex_fiber_generation-1.4.0-x86_64.AppImage file:
   ```
   cd /path/to/lifex/
   ```
3. make the AppImage executable:
   ```
   chmod +x lifex_fiber_generation-1.4.0-x86_64.AppImage
   ```
4. lifex_fiber_generation-1.4.0-x86_64.AppImage is now ready to be executed:
   ```
   ./lifex_fiber_generation-1.4.0-x86_64.AppImage [ARGS]...
   ```

Root permissions are **not** required. Please note that, in order for the above procedure to succeed, AppImage relies upon the userspace filesystem framework FUSE[34]. In case of errors, please try with the following commands:

```
./lifex_fiber_generation-1.4.0-x86_64.AppImage \
    --appimage-extract
squashfs-root/usr/bin/lifex_fiber_generation [ARGS]...
```

or refer to the AppImage troubleshooting guide[35].

[29] https://www.debian.org/releases/
[30] https://docs.appimage.org/introduction/concepts.html
[31] https://www.gnu.org/software/libc/
[32] https://docs.appimage.org/introduction/quickstart.html
[33] https://docs.appimage.org/user-guide/run-appimages.html
[34] https://www.kernel.org/doc/html/latest/filesystems/fuse.html
[35] https://docs.appimage.org/user-guide/troubleshooting/fuse.html

3.2 Step 0 - Set the parameter file

Every `life`^{x} application or example (including also the fiber generation executable described in section 3.1) defines a set of parameters that are required in order to be run. They involve problem-specific parameters (such as constitutive relations, geometry, time interval, boundary conditions, ...) as well as numerical parameters (types of linear/non-linear solvers, tolerances, maximum number of iterations, ...) or output-related options.

In case an application has sub-dependencies (such as a linear solver), also the related parameters are included.

Every application comes with a set of command line options, which can be printed using the `-h` (or `--help`) flag:

```
./executable_name -h
```

The first step before running an executable is to generate the parameter file(s) containing all the default parameter values. This is done via the `-g` (or `--generate-params`) flag:

```
./executable_name -g
```

that by default generates a parameter file named after the executable, in `.prm` format.

By default, only parameters considered *standard* are printed. The parameter file verbosity can be decreased or increased by passing an optional flag `minimal` or `full` to the `-g` flag, respectively:

```
./executable_name -g minimal
```

The parameter basename to generate can be customized with the `-f` (or `--params-filename`) option:

```
./executable_name -g -f custom_param_file.ext
```

Absolute or relative paths can be specified.

At the user's option, in order to guarantee a flexible interface to external file processing tools, the parameter file extension `ext` can be chosen among three different interchangeable file formats `prm`, `json` or `xml`, from the most human-readable to the most machine-readable.

As an example, the three parameter files listed in listings 1, 2 and 3 are semantically equivalent.

We highlight that, following with the design of the `ParameterHandler` class of `deal.II`[36], each parameter is provided with:

- a given pattern, specifying the parameter type (*e.g.* boolean, integer, floating-point number, string, list, ...) and, whenever relevant, a range of admissible values (*pattern description*);

[36] https://www.dealii.org/current/doxygen/deal.II/classParameterHandler.html

- a default value, printed in the parameter file upon generation and implicitly assumed if the user omits a custom value;
- the *actual* value, possibly overriding the default one;
- a documentation string;
- a global index.

All of these features, combined to a runtime check for the correctness of each parameter, make the code syntactically and semantically robust with respect to possible errors or typos introduced by mistake.

```
# Listing of Parameters
# ---------------------
subsection Fiber generation
  subsection Mesh and space discretization

    # Specify whether the input mesh has hexahedral or
    # tetrahedral elements. Available options are: Hex | Tet.
    set Element type           = Hex

    # Number of global mesh refinement steps applied to the
    initial grid (Hex only).
    set Number of refinements = 0

    subsection File
      # Mesh file.
      set Filename       =

      # Mesh scaling factor: 1e-3 => from [mm] to [m].
      set Scaling factor = 1e-3
    end
  end

  subsection Output
    # Enable/disable output.
    set Enable output = true

    # Output file.
    set Filename      = fibers
  end
end
```

Listing 1: PRM parameter file format.

```
{
  "Fiber_20generation": {
    "Mesh_20and_20space_20discretization": {
      "File": {
        "Filename": {
          "value": "",
          "default_value": "",
```

```
          "documentation": "Mesh file.",
          "pattern": "0",
          "pattern_description": "[FileName (Type: input)]"
        },
        "Scaling_20factor": {
          "value": "1",
          "default_value": "1e-3",
          "documentation": "Mesh scaling factor: 1e-3 => from [
    mm] to [m].",
          "pattern": "1",
          "pattern_description": "[Double 0...MAX_DOUBLE (
    inclusive)]"
        }
      },
      "Element_20type": {
        "value": "Hex",
        "default_value": "Hex",
        "documentation": "Specify whether the input mesh has
    hexahedral or tetrahedral elements. Available options are:
    Hex | Tet.",
        "pattern": "2",
        "pattern_description": "[Selection Hex|Tet ]"
      },
      "Number_20of_20refinements": {
        "value": "0",
        "default_value": "0",
        "documentation": "Number of global mesh refinement
    steps applied to the initial grid (Hex only).",
        "pattern": "3",
        "pattern_description": "[Integer range 0...2147483647 (
    inclusive)]"
      }
    },
    "Output": {
      "Enable_20output": {
        "value": "true",
        "default_value": "true",
        "documentation": "Enable\/disable output.",
        "pattern": "4",
        "pattern_description": "[Bool]"
      },
      "Filename": {
        "value": "fibers",
        "default_value": "fibers",
        "documentation": "Output file.",
        "pattern": "5",
        "pattern_description": "[FileName (Type: output)]"
      }
    }
  }
}
```

Listing 2: JSON parameter file format.

```
<?xml version="1.0" encoding="utf-8"?>
<ParameterHandler>
  <Fiber_20generation>
    <Mesh_20and_20space_20discretization>
      <File>
        <Filename>
          <value/>
          <default_value/>
          <documentation>Mesh file.</documentation>
          <pattern>0</pattern>
          <pattern_description>[FileName (Type: input)]</
    pattern_description>
        </Filename>
        <Scaling_20factor>
          <value>1</value>
          <default_value>1</default_value>
          <documentation>Mesh scaling factor: 1e-3 =&amp;gt; from [
    mm] to [m].</documentation>
          <pattern>1</pattern>
          <pattern_description>[Double 0...MAX_DOUBLE (
    inclusive)]</pattern_description>
        </Scaling_20factor>
      </File>
      <Element_20type>
        <value>Hex</value>
        <default_value>Hex</default_value>
        <documentation>Specify whether the input mesh has
    hexahedral or tetrahedral elements. Available options are:
    Hex | Tet.</documentation>
        <pattern>2</pattern>
        <pattern_description>[Selection Hex|Tet ]</
    pattern_description>
      </Element_20type>
      <Number_20of_20refinements>
        <value>0</value>
        <default_value>0</default_value>
        <documentation>Number of global mesh refinement steps
    applied to the initial grid (Hex only).</documentation>
        <pattern>3</pattern>
        <pattern_description>[Integer range 0...2147483647 (
    inclusive)]</pattern_description>
      </Number_20of_20refinements>
    </Mesh_20and_20space_20discretization>
    <Output>
      <Enable_20output>
        <value>true</value>
        <default_value>true</default_value>
        <documentation>Enable/disable output.</documentation>
        <pattern>4</pattern>
        <pattern_description>[Bool]</pattern_description>
      </Enable_20output>
      <Filename>
        <value>fibers</value>
```

```
        <default_value>fibers</default_value>
        <documentation>Output file.</documentation>
        <pattern>5</pattern>
        <pattern_description>[FileName (Type: output)]</
    pattern_description>
      </Filename>
    </Output>
  </Fiber_20generation>
</ParameterHandler>
```

Listing 3: XML parameter file format.

Once generated, the user can modify, copy, move or rename the parameter file depending on their needs.

3.3 Step 1 - Run

To run an executable, the `-g` flag has simply to be omitted whereas the `-f` option is used to specify the parameter file to be *read* (as opposed to *written*, in generation mode), *e.g.*:

```
./executable_name -f custom_param_file.ext [option...]
```

If no `-f` flag is provided, a file named `executable_name.prm` is assumed to be available in the directory where the executable is run from.

The path to the directory where all the app output files will be saved to can be selected via the `-o` (or `--output-directory`) flag:

```
./executable_name -o ./results/
```

If the specified directory does not already exist, it will be created. By default, the current working directory is used.

Absolute or relative paths can be specified for both the input parameter file and the output directory.

3.3.1 Parallel run

To run an app in parallel, use the `mpirun` or `mpiexec` wrapper commands (which may vary depending on the MPI implementation available on your machine) *e.g.*:

```
mpirun -n <N_PROCS> ./executable_name [option...]
```

where `<N_PROCS>` is the desired number of parallel processes to run on.

As a rule of thumb, 10000 to 100000 degrees of freedom per process should lead to the best performance.

3.3.2 Dry run and parameter file conversion

Upon running, a parameter log file is automatically generated in the output directory, that can be used later to retrieve which parameters had been used for a specific run.

By default, `log_params.ext` will be used as its filename. This can be changed via the `-l` (or `--log-file`) flag, *e.g.*:

```
./executable_name -l my_log_file.ext [option...]
```

The extension is not mandatory: if unspecified, the same extension as the input parameter file will be used.

If the **dry run** option is enabled via the `-d` (or `--dry-run`) flag, the execution terminates right after the parameter log file generation. This has a two-fold purpose:

1. checking the correctness of the parameters being declared and parsed *before* running the actual simulation (if any of the parameters does not match the specified pattern or has a wrong name or has not been declared in a given subsection then a runtime exception is thrown);
2. **converting** a parameter file between two different formats/extensions. For example, the following command converts `input.xml` to `output.json`:

```
./executable_name -f input.xml -d -l output.json [option
    ...]
```

4 Input data

Additional input data (scripts, meshes and parameter files) associated with the guided examples described below can be downloaded from the release archive https://doi.org/10.5281/zenodo.5810269.

`life`[x] is designed to support both hexahedral and tetrahedral labelled meshes in the widely used `*.msh` format, see fig. 2. This type of mesh can be generated by a variety of mesh generation software (*e.g.* `gmsh`[37], `netgen`[38], `vmtk`[39] and `meshtools`[40]). Otherwise, other mesh-format types can be converted in `*.msh` using for example the open-source library `meshio`[41].

In this getting started guide, we provide different ready-to-use meshes, namely

[37] http://gmsh.info
[38] https://ngsolve.org/
[39] (http://www.vmtk.org)
[40] https://bitbucket.org/aneic/meshtool/src/master/
[41] https://pypi.org/project/meshio/

- a set of four idealized geometries consisting of a ventricular slab, a spherical slab, an idealized based left ventricle and an idealized left atrium, see fig. 3(a-d);

- two realistic geometries composed by a left ventricle and a left atrium, see fig. 3(e-f).

The idealized meshes have been generated using the built-in CAD engine of `gmsh`, an open-source 3D finite element mesh generator, starting from the corresponding `gmsh` geometrical models (represented by `*.geo` files, also provided) defined using their boundary representation, where a volume is bounded by a set of surfaces. For details about the geometrical definition of a model geometry we refer to the online documentation of `gmsh`[42].

In order to perform the mesh generation, starting from the geometrical files provided in this tutorial, the following command can be run in a terminal:

```
gmsh geometry.geo -clscale s -o mesh.msh -save
```

where `geometry.geo` is the geometrical file model, `mesh.msh` is the output mesh file, which will be provided as an input to a `life`[x] app, and `s` $\in (0, 1]$ is the mesh element size factor. To produce a coarser (finer) mesh the `clscale` factor can be reduced (increased).

The realistic left ventricle and left atrium have been produced starting from the open-source meshes adopted in [16] (for the left atrium[43]) and in [17] (for the left ventricle[44]) and using the Vascular Modelling Toolkit (`vmtk`) software [18] along with the semi-automatic meshing tools[45] recently proposed in [19].

5 Fiber generation

In this section, we describe the working principles of the fiber generation application of `life`[x]. First, we briefly recall the LDRBMs that stand behind the myocardial fiber generation (section 5.1) and then we present several examples where we detail about the parameter settings (section 5.2).

5.1 Underlying methods

Rule-Based-Methods (RBMs) represent a commonly used strategy to include cardiac fibers in computational models. A particular class of such methods is known as LDRBMs since they rely on the solution of Laplace problems. In

[42] https://gmsh.info/doc/texinfo/gmsh.html
[43] https://doi.org/10.18742/RDM01-289
[44] https://doi.org/10.5281/zenodo.3890034
[45] https://github.com/marco-fedele/vmtk

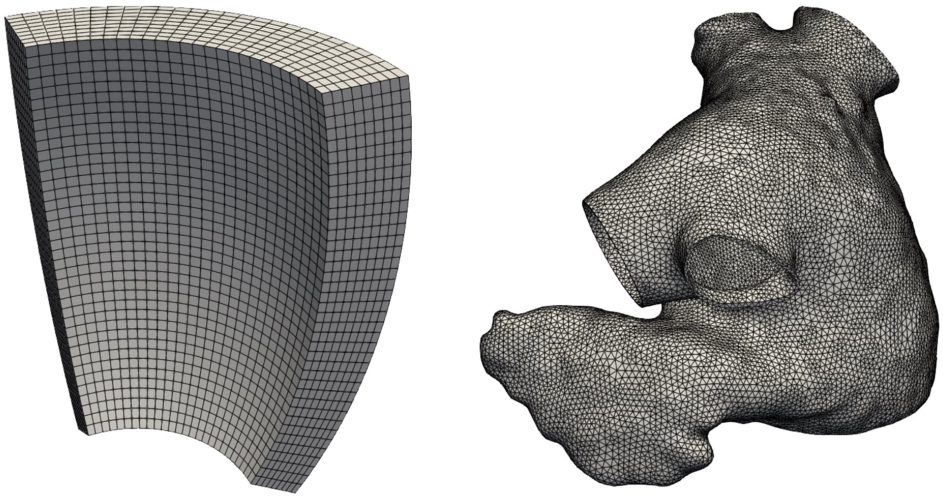

Figure 2: (a) Hexahedral mesh of a ventricular slab; (b) Tetrahedral mesh of a realistic left atrium [16].

this getting started guide, we present LDRBMs for (ventricular and spherical) slabs, (based and complete) left ventricular and left atrial geometries. For further details about the LDRBMs presented here see [15].

The following common steps are the building block of all LDRBMs.

1. **Labelled mesh:** Provide a labelled mesh of the domain Ω to define specific partition of the boundary $\partial\Omega$ as

$$\partial\Omega = \Gamma_{\text{epi}} \cup \Gamma_{\text{endo}} \cup \Gamma_{\text{base}} \cup \Gamma_{\text{apex}},$$

where Γ_{endo} is the endocardium, Γ_{epi} is the epicardium, Γ_{base} is the basal plane and Γ_{apex} is the apex.

Epicardium and endocardium: for the ventricular slab geometry Γ_{endo} and Γ_{epi} are the lateral walls of the slab, see fig. 3(a); for the spherical slab, the ventricular and atrial geometries Γ_{endo} and Γ_{epi} are the internal and external surfaces, see fig. 3(b-f).

Basal plane and apex: for the ventricular slab geometry Γ_{base} and Γ_{apex} are the top and bottom surfaces, respectively (see fig. 3(a)); for the spherical slab, Γ_{base} and Γ_{apex} are selected as the north and south pole points of the epicardial sphere (see fig. 3(b)); for the based ventricular geometry Γ_{base} is an artificial basal plane located well below the cardiac valves (see fig. 3(d)), while for the complete ventricular geometry Γ_{base} is split into Γ_{mv} and Γ_{av},

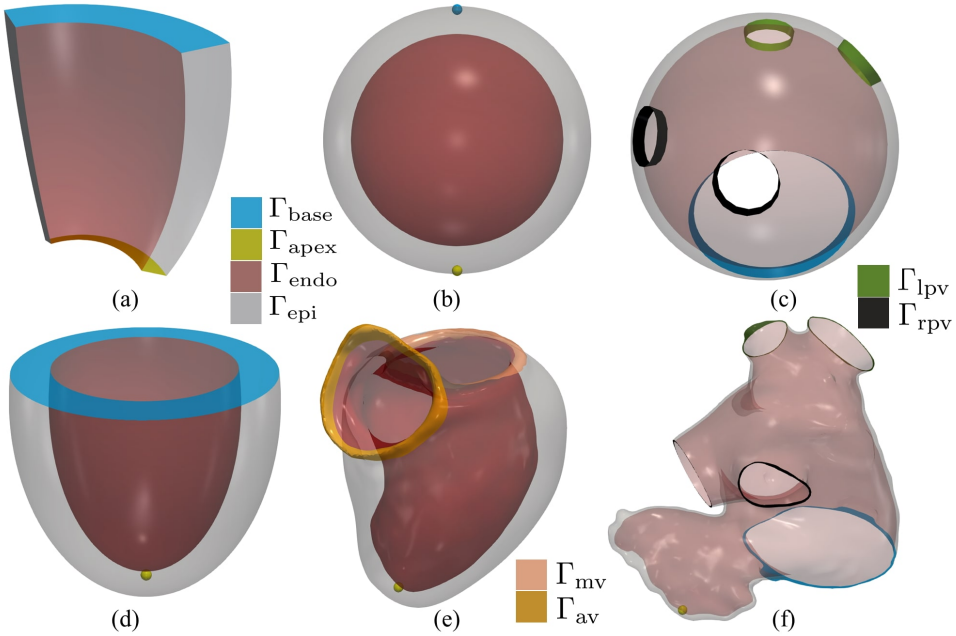

Figure 3: Labelled meshes. (a) Ventricular slab; (b) Spherical slab; (c) Idealized left atrium; (d) Idealized based left ventricle; (e) Realistic complete left ventricle; (f) Realistic left atrium.

representing the mitral and aortic valve rings, respectively (see fig. 3(e)); for the ventricular geometries Γ_{apex} is selected as the epicardial point furthest from the ventricular base (see fig. 3(d-e)); for the atrial geometry Γ_{base} is the mitral valve ring and Γ_{apex} represents the apex of the left atrial appendage (see fig. 3(c-f));

Atrial pulmonary rings: the atrial geometry type also requires the definition of the boundary labels for the left Γ_{lpv} and right Γ_{rpv} pulmonary vein rings, see fig. 3(c-f).

2. **Transmural direction:** A transmural distance ϕ is defined to compute the distance of the epicardium from the endocardium, by means of the following Laplace-Dirichlet problem:

$$\begin{cases} -\Delta\phi = 0, & \text{in } \Omega, \\ \phi = 1, & \text{on } \Gamma_{\text{epi}}, \\ \phi = 0, & \text{on } \Gamma_{\text{endo}}, \\ \nabla\phi \cdot \mathbf{n} = 0, & \text{on } \partial\Omega \setminus (\Gamma_{\text{endo}} \cup \Gamma_{\text{epi}}). \end{cases} \tag{1}$$

Then, the transmural distance gradient $\nabla\phi$ is used to build the unit

transmural direction:

$$\widehat{\boldsymbol{e}}_t = \frac{\nabla\phi}{\|\nabla\phi\|}.$$

3. **Normal direction:** A normal (or apico-basal) direction $\boldsymbol{k}$ (which is directed from the apex towards the base) is introduced and used to build the unit normal direction $\widehat{\boldsymbol{e}}_n$:

$$\widehat{\boldsymbol{e}}_n = \frac{\boldsymbol{k} - (\boldsymbol{k}\cdot\widehat{\boldsymbol{e}}_t)\widehat{\boldsymbol{e}}_t}{\|\boldsymbol{k} - (\boldsymbol{k}\cdot\widehat{\boldsymbol{e}}_t)\widehat{\boldsymbol{e}}_t\|}.$$

The normal direction $\boldsymbol{k}$ can be computed following one of these approaches (see also fig. 4):

Rossi-Lassila et al. (RL) approach [20]: $\boldsymbol{k}$ is defined as the vector $\mathbf{n}_{\text{base}}$, *i.e.* the outward normal to the basal plane, that is $\boldsymbol{k} = \mathbf{n}_{\text{base}}$.

Bayer-Trayanova et. al (BT) approach [21]: $\boldsymbol{k}$ is the gradient of the solution ψ ($\boldsymbol{k} = \nabla\psi$), which can be obtained by solving the following Laplace-Dirichlet problem:

$$\begin{cases} -\Delta\psi = 0, & \text{in } \Omega, \\ \psi = 1, & \text{on } \Gamma_{\text{base}}, \\ \psi = 0, & \text{on } \Gamma_{\text{apex}}, \\ \nabla\psi\cdot\mathbf{n} = 0, & \text{on } \partial\Omega \setminus (\Gamma_{\text{base}} \cup \Gamma_{\text{apex}}). \end{cases} \tag{2}$$

Doste et al. approach [22]: $\boldsymbol{k}$ is a weighted sum of the apico-basal ($\nabla\psi_{\text{ab}}$) and apico-outflow-tract ($\nabla\psi_{\text{ot}}$) directions, obtained using an interpolation function w:

$$\boldsymbol{k} = w\nabla\psi_{\text{ab}} + (1-w)\nabla\psi_{\text{ot}},$$

where ψ_{ab} and ψ_{ot} are obtained by solving Laplace-Dirichlet problems in the form of (2) where $\Gamma_{\text{base}} = \Gamma_{\text{mv}}$ (for ψ_{ab}) and $\Gamma_{\text{base}} = \Gamma_{\text{av}}$ (for ψ_{ot}), respectively. Moreover, the interpolation function w is obtained by solving:

$$\begin{cases} -\Delta\omega = 0, & \text{in } \Omega, \\ \omega = 1, & \text{on } \Gamma_{\text{mv}} \cup \Gamma_{\text{apex}}, \\ \omega = 0, & \text{on } \Gamma_{\text{av}}, \\ \nabla\omega\cdot\mathbf{n} = 0, & \text{on } \partial\Omega \setminus (\Gamma_{\text{av}} \cup \Gamma_{\text{mv}} \cup \Gamma_{\text{apex}}). \end{cases}$$

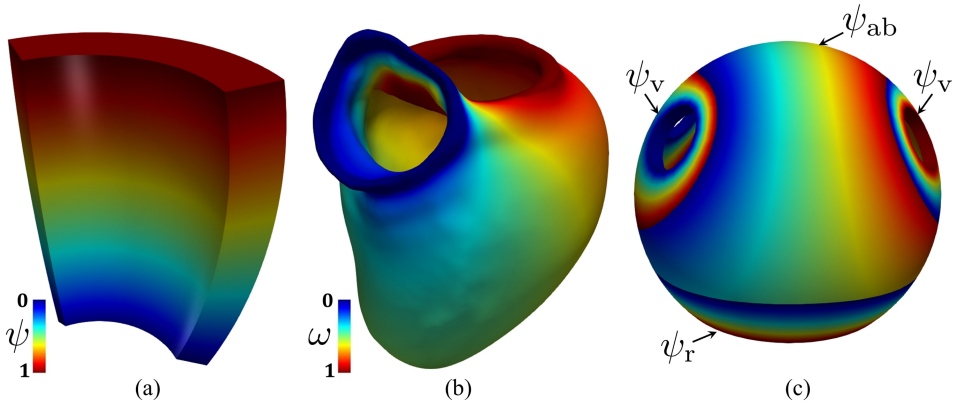

Figure 4: Different types of normal distances: (a) Bayer-Trayanova et al. approach [21]; (b) Doste et al. approach [22]; (c) Piersanti et al. approach [15].

Piersanti et al. approach [15]: for each point in Ω, a unique normal direction $\boldsymbol{k}$ is selected among the gradient of several normal directions $\boldsymbol{k} = \nabla\psi_{\mathrm{i}}$ ($\mathrm{i} = \mathrm{ab}, \mathrm{v}, \mathrm{r}$), where ψ_{i} are obtained by solving the following Laplace-Dirichlet problem

$$\begin{cases} -\Delta\psi_{\mathrm{i}} = 0, & \text{in } \Omega, \\ \psi_{\mathrm{i}} = \chi_{\mathrm{a}}, & \text{on } \Gamma_{\mathrm{a}}, \\ \psi_{\mathrm{i}} = \chi_{\mathrm{b}}, & \text{on } \Gamma_{\mathrm{b}}, \\ \nabla\psi_i \cdot \mathbf{n} = 0, & \text{on } \partial\Omega \setminus (\Gamma_{\mathrm{a}} \cup \Gamma_{\mathrm{b}}). \end{cases} \tag{3}$$

See [15] for further details about the selection procedure $\boldsymbol{k} = \nabla\psi_{\mathrm{i}}$ and for the specific choices of χ_{a}, χ_{b}, Γ_{a} and Γ_{b} in problem (2) made for ψ_{i} ($i = \mathrm{ab}, \mathrm{v}, \mathrm{r}$).

The BT approach is used in the ventricular and spherical slab geometry types, see also fig. 4(a). The RL and BT approaches can be adopted in the based ventricular geometry (by setting either `Algorithm type` equal to `BT` or `RL` in the parameter file, respectively), while the Doste approach is used in the complete ventricular geometry, see also fig. 4(b). Finally, the Piersanti approach is employed for the atrial geometry, see also fig. 4(c).

4. **Local coordinate system:** For each point of the domain an orthonormal local coordinate axial system is defined by $\widehat{\boldsymbol{e}}_t$, $\widehat{\boldsymbol{e}}_n$ and the unit longitudinal direction $\widehat{\boldsymbol{e}}_l$ (orthogonal to the previous ones), as shown

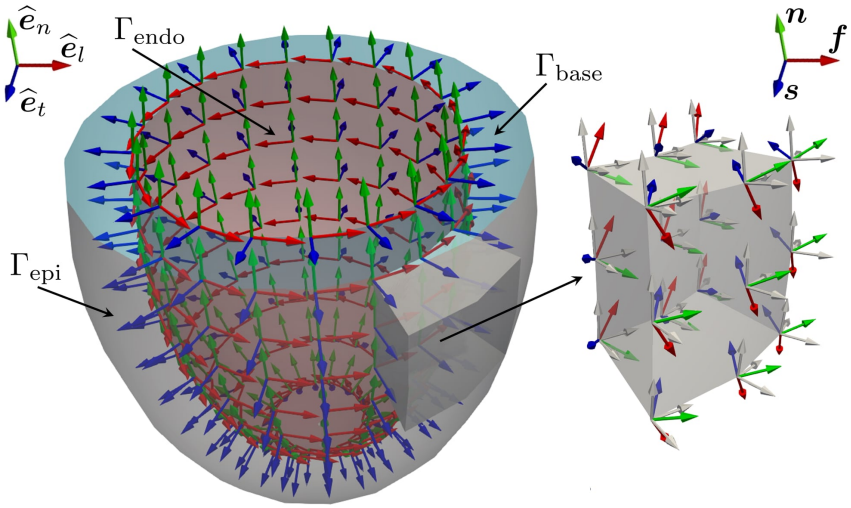

Figure 5: Representation of the three directions employed by a LDRBM for an idealized ventricular domain. Only directions on the endocardium Γ_{endo} are represented. In blue: unit transmural direction, $\widehat{e}_t$; In green: unit normal direction, $\widehat{e}_n$; In red: unit longitudinal direction, $\widehat{e}_l$. Right: zoom on a slab of the left ventricular myocardium showing the three final myofibers orientations $\boldsymbol{f}$, $\boldsymbol{s}$ and $\boldsymbol{n}$.

in fig. 5:

$$Q = [\widehat{e}_l, \widehat{e}_n, \widehat{e}_t] = \begin{cases} \widehat{e}_t = \dfrac{\nabla \phi}{\|\nabla \phi\|}, \\ \widehat{e}_n = \dfrac{\boldsymbol{k} - (\boldsymbol{k} \cdot \widehat{e}_t)\widehat{e}_t}{\|\boldsymbol{k} - (\boldsymbol{k} \cdot \widehat{e}_t)\widehat{e}_t\|}, \\ \widehat{e}_l = \widehat{e}_n \times \widehat{e}_t. \end{cases} \tag{4}$$

5. **Axis rotation:** The reference frame is rotated with the purpose of defining the myofibers orientation: $\boldsymbol{f}$ the fiber direction, $\boldsymbol{n}$ the sheet-normal direction and $\boldsymbol{s}$ the sheet direction. Specifically, $\widehat{e}_l$ rotates counter-clockwise around $\widehat{e}_t$ by the helical angle α, whereas the transmural direction $\widehat{e}_t$ is rotated counter-clockwise around $\widehat{e}_l$ by the sheetlet angle β, see fig. 5:

$$[\widehat{e}_l, \widehat{e}_n, \widehat{e}_t] \longrightarrow [\boldsymbol{f}, \boldsymbol{n}, \boldsymbol{s}],$$

The rotation angles follow the linear relationships:

$$\alpha(\phi) = \alpha_{\text{endo}}(1-\phi) + \alpha_{\text{epi}}\phi, \qquad \beta(\phi) = \beta_{\text{endo}}(1-\phi) + \beta_{\text{epi}}\phi,$$

where α_{endo}, α_{epi}, β_{endo}, β_{epi} are suitable helical and sheetlet rotation angles on the epicardium and endocardium (specifying in the parameter file `alpha epi`, `alpha endo`, `beta epi`, `beta endo`). Moreover,

for the complete ventricular geometry it is possible to set specific fiber and sheet angle rotations in the outflow tract (OT) region (*i.e.* around the aortic valve ring) by specifying `alpha epi OT`, `alpha endo OT`, `beta epi OT`, `beta endo OT`). Finally, for the atrial geometry type, no transmural variation in the myofibers direction is prescribed and the three unit directions correspond to the final myofibers directions $[\widehat{\boldsymbol{e}}_l, \widehat{\boldsymbol{e}}_n, \widehat{\boldsymbol{e}}_t] = [\boldsymbol{f}, \boldsymbol{n}, \boldsymbol{s}]$.

5.2 Setting the parameter files

A parameter file for fiber generation is characterized by a common section named `Mesh and space discretization`. Select `Element type = Tet` for tetrahedral meshes or `Element type = Hex` for hexahedral meshes. Finally, specify in `FE space degree` the degree of the (piecewise continuous) polynomial FE space used to solve the Laplace-Dirichlet problems described above. Finally, we remark that `life`[x] internally treats all physical quantities as if they are provided in SI units: therefore, a `Scaling factor` can be set in order to convert the input mesh from a given unit of measurement (*e.g.* if the mesh coordinates are provided in millimeters then `Scaling factor` must be set equal to `1e-3`).

The `Geometry type` parameters enables to specify the kind of geometry provided in input, in order to apply the proper LDRBM algorithm among the ones described above. Once specified, parameters related to the specific algorithm and geometry will be parsed from a subsection named after the value of `Geometry type`.

All parameters missing from the parameter file will take their default value, which is hard-coded.

```
subsection Mesh and space discretization
  set Element type     = Tet
  set FE space degree = 1
  set Geometry type    = Slab

  subsection File
    set Filename        = /path/to/mesh/slab.msh
    set Scaling factor = 1e-3
  end
end
```

5.2.1 Slab fibers

Specify `Geometry type = Slab` to prescribe fibers in slab geometries and the path of the input mesh file in `Filename`.

Ventricular slab For a ventricular slab geometry `Sphere slab = false` must be set. Specify the label for the top (`Tags base up`) and bottom

(`Tags base down`) surfaces of the slab, as well as the epicardium (`Tags epi`) and endocardium (`Tags endo`) ones. Finally, prescribe the helical and sheetlet rotation angles at the epicardium and endocardium using the corresponding `alpha epi`, `alpha endo`, `beta epi`, `beta endo` parameters.

```
subsection Slab
  set Sphere slab = false

  set Tags base up   =  50
  set Tags base down =  60
  set Tags epi       =  10
  set Tags endo      =  20
  set alpha epi      = -60
  set alpha endo     =  60
  set beta epi       =  45
  set beta endo      = -45
end
```

Spherical slab For a spherical slab geometry set `Sphere slab = true`. Specify the epicardial coordinates (x, y, z) of the north (`North pole`) and south (`South pole`) poles of the sphere of the slab, and the labels of the endocardium (`Tags endo`) and epicardium (`Tags epi`). Finally, prescribe the helical and sheetlet rotation angles at the epicardium and endocardium in `alpha epi`, `alpha endo`, `beta epi`, `beta endo`. A fiber architecture for the sphere slab with radial fiber $\boldsymbol{f}$ can be prescribed by setting in the parameter file `Sphere with radial fibers = true`. This consists of exchanging the sheet direction $\boldsymbol{s}$ with the fiber direction $\boldsymbol{f}$. Instead, a tangential (to the epicardial and endocardial surfaces) fiber field $\boldsymbol{f}$ is assigned when `Sphere with radial fibers = false`.

```
subsection Slab
  set Sphere slab                = true
  set Sphere with radial fibers = true

  set North pole = 0 0 0.025
  set South pole = 0 0 -0.025

  set Tags epi  = 10
  set Tags endo = 20

  set alpha epi  = 0
  set alpha endo = 0
  set beta epi   = 0
  set beta endo  = 0
end
```

5.2.2 Left ventricular fibers

Specify `Geometry type = Left ventricle` to prescribe fibers in a based left ventricular geometry, or `Geometry type = Left ventricle complete` to prescribe fibers in complete left ventricular geometry. Other mesh-related parameters have the same meaning as described above.

```
subsection Mesh and space discretization
  set Element type     = Tet
  set FE space degree = 1
  set Geometry type    = Left ventricle complete

  subsection File
    set Filename        = /path/to/mesh/ventricle.msh
    set Scaling factor = 1e-3
  end
end
```

Based left ventricle Specify the labels for the basal plane (`Tags base`), the epicardium (`Tags epi`) and endocardium (`Tags endo`) of the ventricle. Select `RL` or `BT` approach in `Algorithm type`. Prescribe the helical and sheetlet rotation angles at the epicardium and endocardium in `alpha epi`, `alpha endo`, `beta epi`, `beta endo`. Finally, for the `RL` approach specify the outward normal vector to the basal plane in `Normal to base`, while for the `BT` approach prescribe the apex epicardial coordinates (x, y, z) (`Apex`) of the ventricle.

```
subsection Left ventricle
  set Tags base = 50
  set Tags epi  = 10
  set Tags endo = 20

  set Algorithm type = BT

  set alpha epi  = -60
  set alpha endo =  60
  set beta epi   =  20
  set beta endo  = -20

  subsection RL
    set Normal to base = 0 0 1
  end

  subsection BT
    set Apex = 0 0 0.0601846
  end
end
```

Complete left ventricle Specify the labels for the mitral (`Tags MV`) and aortic (`Tags AV`) valve rings, the epicardium (`Tags epi`) and endocardium (`Tags endo`) of the ventricle. Prescribe the apex epicardial coordinates (x, y, z) in `Apex`. Finally, define the helical and sheetlet rotation angles at the epicardium and endocardium in `alpha epi`, `alpha endo`, `beta epi`, `beta endo`. A specific helical and sheetlet rotation angles around the outflow tract of the left ventricle (*i.e.* the mitral valve ring) can be specified by setting `alpha epi OT`, `alpha endo OT`, `beta epi OT`, `beta endo OT`.

```
subsection Left ventricle complete
  set Tags MV    = 50
  set Tags AV    = 60
  set Tags epi   = 10
  set Tags endo  = 20

  set Apex = 0.0692 0.0710 0.3522

  set alpha epi  = -60
  set alpha endo =  60
  set beta epi   =  20
  set beta endo  = -20

  set alpha epi OT  = 0
  set alpha endo OT = 90
  set beta epi OT   = 0
  set beta endo OT  = 0
end
```

5.2.3 Left atrial fibers

Specify `Geometry type = Left atrium` to prescribe fibers in a left atrial geometry. Other mesh-related parameters have the same meaning as described above.

```
subsection Mesh and space discretization
  set Element type    = Tet
  set FE space degree = 1
  set Geometry type   = Left atrium

  subsection File
    set Filename       = /path/to/mesh/atrium.msh
    set Scaling factor = 1e-3
  end
end
```

Idealized left atrium Select `Appendage = false` to prescribe fibers in the hollow sphere geometry. Specify the labels for the mitral valve ring (`Tags MV`), the right (`Tags RPV`) and left (`Tags LPV`) pulmonary veins rings, the epicardium (`Tags epi`) and endocardium (`Tags endo`) of the idealized

atrium. Finally, select the dimension of the atrial bundles: `Tau bundle MV` for the mitral valve bundle; `Tau bundle LPV` and `Tau bundle RPV` for the left and right pulmonary valves rings bundle.

```
subsection Left atrium
  set Appendage = false

  set Tags epi  = 30
  set Tags endo = 10
  set Tags RPV  = 20
  set Tags LPV  = 50
  set Tags MV   = 40

  set Tau bundle MV  = 0.65
  set Tau bundle LPV = 0.85
  set Tau bundle RPV = 0.15
end
```

Realistic left atrium Select `Appendage = true` to prescribe fibers in a realistic left atrial geometry. Specify the labels for the mitral valve ring (`Tags MV`), the right (`Tags RPV`) and left (`Tags LPV`) pulmonary veins rings, the epicardium (`Tags epi`) and endocardium (`Tags endo`) of the idealized atrium. Prescribe in `Apex` the epicardial coordinates (x, y, z) for the apex of the atrial appendage. Finally, select the dimension of the atrial bundles: for the mitral valve bundle `Tau bundle MV`; for the left and right pulmonary valves rings bundle `Tau bundle LPV` and `Tau bundle RPV`, respectively.

```
subsection Left atrium
  set Appendage = true

  set Apex = 83.868 16.369 45.989

  set Tags epi  = 30
  set Tags endo = 10
  set Tags RPV  = 20
  set Tags LPV  = 50
  set Tags MV   = 40

  set Tau bundle MV  = 0.60
  set Tau bundle LPV = 0.90
  set Tau bundle RPV = 0.10
end
```

6 Output and visualization

To enable the output select `Enable output = true` and specify the corresponding output filename in `Filename`. This produces a `XDMF` schema file `output_filename.xdmf` (wrapped around a same-named `HDF5` output file

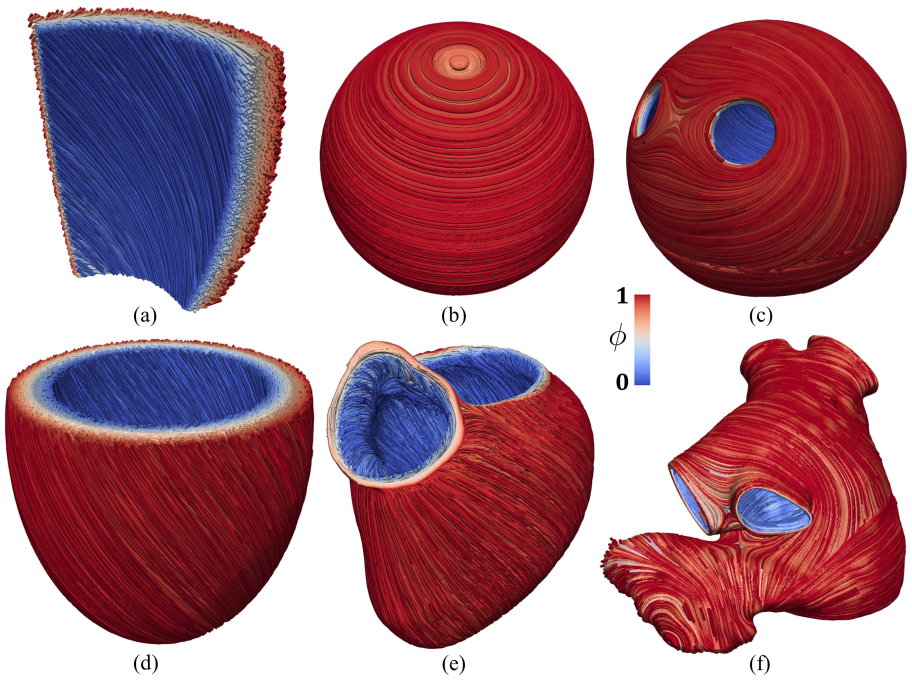

Figure 6: Fiber field $\boldsymbol{f}$ visualized in streamlines. (a) Ventricular slab; (b) Spherical slab with circumferential fibers; (c) Idealized left atrium; (d) Idealized based left ventricle; (e) Realistic complete left ventricle; (f) Realistic left atrium.

output_filename.h5) that can be visualized in ParaView[46], an open-source multi-platform data analysis and visualization application. Specifically, the streamtracer and the tube ParaView filters can be applied in sequence to visualize the fiber fields, see fig. 6.

```
subsection Output
  set Enable output = true
  set Filename      = fibers
end
```

Moreover, the HDF5 file format guarantees that the output can easily be further post-processed, not only for visualization purposes but rather to be fed as an input to more sophisticated computational pipelines.

7 Future perspectives

The content of this release is published on the official website https://lifex.gitlab.io/: we encourage users to interact with the lifex development community via the issue tracker[47] of our public website repository. Any kind of curiosity, question, bug report or suggestion is welcome!

As anticipated in section 1, the development of lifex has been initially oriented towards a heart module incorporating packages for the simulation of cardiac electrophysiology, mechanics, electromechanics, blood fluid dynamics and myocardial perfusion models.

In the near future, the deployment of lifex will follow two lines:

- more packages and modules will be successively published in binary form, starting from an advanced solver for cardiac electrophysiology and other solvers from the *heart* module;
- in the meantime, the source code associated with lifex core and with previous binary releases will be gradually made publicly available under an open-source license.

In the long run, also additional modules unrelated to the cardiac function are expected to be included within lifex.

News and announcements about lifex will be posted to the official website https://lifex.gitlab.io/. Stay tuned!

8 Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation pro-

[46] https://www.paraview.org
[47] https://gitlab.com/lifex/lifex.gitlab.io/-/issues

gramme (grant agreement No 740132, iHEART - An Integrated Heart Model for the simulation of the cardiac function, P.I. Prof. A. Quarteroni).

References

[1] D. Groen, S. J. Zasada, and P. V. Coveney. "Survey of Multiscale and Multiphysics Applications and Communities". In: *Computing in Science Engineering* 16.2 (2014), pp. 34–43. ISSN: 1558-366X. DOI: 10.1109/MCSE.2013.47.

[2] A. Quarteroni, L. Dede', A. Manzoni, and C. Vergara. *Mathematical Modelling of the Human Cardiovascular System: Data, Numerical Approximation, Clinical Applications*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2019. DOI: 10.1017/9781108616096.

[3] A. Quarteroni, T. Lassila, S. Rossi, and R. Ruiz-Baier. "Integrated Heart—Coupling multiscale and multiphysics models for the simulation of the cardiac function". In: *Computer Methods in Applied Mechanics and Engineering* 314 (2017). Special Issue on Biological Systems Dedicated to William S. Klug, pp. 345–407. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2016.05.031.

[4] D. Arndt, W. Bangerth, B. Blais, M. Fehling, R. Gassmöller, T. Heister, L. Heltai, U. Köcher, M. Kronbichler, M. Maier, P. Munch, J. P. Pelteret, S. Proell, S. Konrad, B. Turcksin, D. Wells, and J. Zhang. "The `deal.II` Library, Version 9.3". In: *Journal of Numerical Mathematics* 29.3 (2021), pp. 171–186. DOI: 10.1515/jnma-2021-0081.

[5] F. Regazzoni, M. Salvador, P. C. Africa, M. Fedele, L. Dede', and A. Quarteroni. *A cardiac electromechanics model coupled with a lumped parameters model for closed-loop blood circulation. Part I: model derivation*. 2020. arXiv: 2011.15040 [math.NA].

[6] F. Regazzoni, M. Salvador, P. C. Africa, M. Fedele, L. Dede', and A. Quarteroni. *A cardiac electromechanics model coupled with a lumped parameters model for closed-loop blood circulation. Part II: numerical approximation*. 2020. arXiv: 2011.15051 [math.NA].

[7] M. Salvador, M. Fedele, P. C. Africa, E. Sung, L. Dede', A. Prakosa, J. Chrispin, N. Trayanova, and A. Quarteroni. "Electromechanical modeling of human ventricles with ischemic cardiomyopathy: numerical simulations in sinus rhythm and under arrhythmia". In: *Computers in Biology and Medicine* 136 (2021), p. 104674. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2021.104674.

[8] F. Regazzoni and A. Quarteroni. "Accelerating the convergence to a limit cycle in 3D cardiac electromechanical simulations through a data-driven 0D emulator". In: *Computers in Biology and Medicine* 135 (2021), p. 104641. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2021.104641.

[9] A. Zingaro, I. Fumagalli, L. Dede', M. Fedele, P. C. Africa, A. F. Corno, and A. Quarteroni. *A multiscale CFD model of blood flow in the human left heart coupled with a lumped-parameter model of the cardiovascular system*. 2021. arXiv: 2110.02114 [math.NA].

[10] M. Salvador, F. Regazzoni, S. Pagani, L. Dede', N. Trayanova, and A. Quarteroni. "The role of mechano-electric feedbacks and hemodynamic coupling in scar-related ventricular tachycardia". In: *Computers in Biology and Medicine* (2022), p. 105203.

[11] I. Fumagalli, P. Vitullo, R. Scrofani, and C. Vergara. "Image-based computational hemodynamics analysis of systolic obstruction in hypertrophic cardiomyopathy". In: *medRxiv* (2021). DOI: 10.1101/2021.06.02.21258207.

[12] S. Stella, C. Vergara, M. Maines, D. Catanzariti, P. C. Africa, C. Dematté, M. Centonze, F. Nobile, M. Del Greco, and A. Quarteroni. "Integration of activation maps of epicardial veins in computational cardiac electrophysiology". In: *Computers in Biology and Medicine* 127 (2020), p. 104047. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2020.104047.

[13] L. Dede', F. Regazzoni, C. Vergara, P. Zunino, M. Guglielmo, R. Scrofani, L. Fusini, C. Cogliati, G. Pontone, and A. Quarteroni. "Modeling the cardiac response to hemodynamic changes associated with COVID-19: a computational study". In: *Mathematical Biosciences and Engineering* 18.4 (2021), pp. 3364–3383.

[14] R. Piersanti, F. Regazzoni, M. Salvador, A. Corno, L. Dede', C. Vergara, and A. Quarteroni. *3D-0D closed-loop model for the simulation of cardiac biventricular electromechanics*. 2021. arXiv: 2108.01907 [math.NA].

[15] R. Piersanti, P. C. Africa, M. Fedele, C. Vergara, L. Dede', A. F. Corno, and A. Quarteroni. "Modeling cardiac muscle fibers in ventricular and atrial electrophysiology simulations". In: *Computer Methods in Applied Mechanics and Engineering* 373 (2021), p. 113468.

[16] T. E. Fastl, C. Tobon-Gomez, A. Crozier, J. Whitaker, R. Rajani, K. P. McCarthy, D. Sanchez-Quintana, S. Y. Ho, M. D. O'Neill, G. Plank, et al. "Personalized computational modeling of left atrial geometry and transmural myofiber architecture". In: *Medical Image Analysis* (2018).

[17] M. Strocchi, C. M. Augustin, M. A. F. Gsell, E. Karabelas, A. Neic, K. Gillette, O. Razeghi, A. J. Prassl, E. J. Vigmond, J. M. Behar, J. Gould, B. Sidhu, C. A. Rinaldi, M. J. Bishop, G. Plank, and S. A. Niederer. "A publicly available virtual cohort of four-chamber heart meshes for cardiac electro-mechanics simulations". In: *PLOS ONE* 15 (June 2020), pp. 1–26. DOI: 10.1371/journal.pone.0235145.

[18] L. Antiga and D. A. Steinman. "The vascular modeling toolkit". In: (2008). URL: http://www.vmtk.org.

[19] M. Fedele and A. Quarteroni. "Polygonal surface processing and mesh generation tools for the numerical simulation of the cardiac function". In: *International Journal for Numerical Methods in Biomedical Engineering* 37.4 (2021), e3435.

[20] S. Rossi, T. Lassila, R. Ruiz-Baier, A. Sequeira, and A. Quarteroni. "Thermodynamically consistent orthotropic activation model capturing ventricular systolic wall thickening in cardiac electromechanics". In: *European Journal of Mechanics-A/Solids* 48 (2014), pp. 129–142.

[21] J. D. Bayer, R. C. Blake, G. Plank, and N. Trayanova. "A novel rule-based algorithm for assigning myocardial fiber orientation to computational heart models". In: *Annals of Biomedical Engineering* 40.10 (2012), pp. 2243–2254.

[22] R. Doste, D. Soto-Iglesias, G. Bernardino, A. Alcaine, R. Sebastian, S. Giffard-Roisin, M. Sermesant, A. Berruezo, D. Sanchez-Quintana, and O. Camara. "A rule-based method to model myocardial fiber orientation in cardiac biventricular geometries with outflow tracts". In: *International Journal for Numerical Methods in Biomedical Engineering* 35.4 (2019), e3185.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

02/2022 Antonietti, P.F.; Scacchi, S.; Vacca, G.; Verani, M.
C^1-VEM for some variants of the Cahn-Hilliard equation: a numerical exploration

03/2022 Giacomini, M.; Perotto, S.
Anisotropic mesh adaptation for region-based segmentation accounting for image spatial information

01/2022 Gavazzoni, M.; Ferro, N.; Perotto, S.; Foletti, S.
Multi-physics inverse homogenization for the design of innovative cellular materials: application to thermo-mechanical problems

95/2021 Di Gregorio, S.; Vergara, C.; Montino Pelagi, G.; Baggiano, A.; Zunino, P.; Guglielmo, M.; Fu
Prediction of myocardial blood flow under stress conditions by means of a computational model

93/2021 Parolini, N.; Dede', L; Ardenghi, G.; Quarteroni, A.
Modelling the COVID-19 epidemic and the vaccination campaign in Italy by the SUIHTER model

92/2021 Antonietti, P.F.; Manzini, G.; Scacchi, S.; Verani, M.
On arbitrarily regular conforming virtual element methods for elliptic partial differential equations

94/2021 Antonietti, P.F.; Berrone, S.; Busetto, M.; Verani, M.
Agglomeration-based geometric multigrid schemes for the Virtual Element Method

90/2021 Hernandez, V.M.; Paolucci, R.; Mazzieri, I.
3D numerical modeling of ground motion in the Valley of Mexico: a case study from the Mw3.2 earthquake of July 17, 2019

87/2021 Both, J.W.; Barnafi, N.A.; Radu, F.A.; Zunino, P.; Quarteroni, A.
Iterative splitting schemes for a soft material poromechanics model

84/2021 Torti, A.; Galvani, M.; Urbano, V.; Arena, M.; Azzone, G.; Secchi, P.; Vantini, S.
Analysing transportation system reliability: the case study of the metro system of Milan