Cinolib: a Generic Programming Header Only C++ Library for Processing Polygonal and Polyhedral Meshes

Marco Livesu

CNR IMATI, Genoa, Italy

Abstract. Inspired by the recent growth of computational methods for general polygonal and polyhedral meshes, this paper introduces Cinolib: a novel header only C++ library for geometry processing. Cinolib differentiates itself from similar toolkits in that it is specifically designed to support a wide set of meshes, such as triangle, quadrilateral and general polygonal surface meshes, as well as tetrahedral, hexahedral and general polyhedral volumetric meshes. At the core of the library there is a hierarchical data structure that factorizes the common properties among the various meshes, allowing tools and algorithms to operate on the widest possible set of meshes with a single implementation, thus avoiding code repetition and facilitating bug fixing and software maintenance. Cinolib is licensed with MIT, it currently counts more than 50K lines of code and, besides the core structure, already comprises a vast set of widespread tools for computer graphics and engineering.

Keywords: Mesh processing · Geometry processing · Scientific Visualization · Software

1 Introduction

Performing some computation on a geometric domain often requires a discretization of it. In computer graphics and engineering complex geometric domains are typically split into simpler elements seamlessly attached to one another, generating a so called *mesh*. For historical reasons, and because of their nice geometric properties, most of the theory and practical algorithms are tailored for meshes made of canonical elements, that is, triangles and quads for surfaces, and tetrahedra and hexahedra for volumes. However, the generation of meshes that satisfy the minimum quality requirements imposed by the analysis they will undergo is a complex problem, which may take up to 8 times the time necessary to perform the analysis itself [13]. A recent trend in literature tries to extend the analysis to general polygons and polyhedra, thus relaxing the constraints for meshing algorithms, and ultimately simplifying the meshing problem. This is for example the case of the PolySpline method [27] and the Virtual Element Method [36], which can be seen as extensions of classical finite element methods to domains containing general polygonal and polyhedral elements. Mesh processing tools offered by

the research community do not natively support this growth of computational methods for general meshes: MeshLab [6] and the underlying VCG [15], as well as ImatiSTL [2], are specifically designed for triangle meshes and besides visualization offer little to none support for other surface meshes; libIGL [14] supports both surfaces and volumes, but does not extend to general polygons or polyhedra. Its extension to general polygons, libhedra [1], allows to represent a wider class of meshes but still sacrifices flexibility to maintain a fixed memory layout, and is devoted to surfaces only, without supporting general polyhedral meshes. INRIA's GEOGRAM [16] offers a wider support for volumetric meshes, but the elementary cell elements must still be of a finite set of types (tets, hexa, prisms, cones), and more complex elements can only be achieved by clustering together multiple elementary elements. Finally, OpenVolumeMesh [3] extends to volumes the DCEL approach of OpenMesh [3] and is quite general, but the two libraries remain separated to one another, and do not offer a unified framework for the processing of surfaces and volumes.

This paper presents *Cinolib*: a C++ mesh processing library which aims to support the recent growth of computational methods for general meshes. Differently from previous geometry processing tools, Cinolib is specifically designed for general surface and volumetric meshes, and allows for an intuitive and easy to maintain code-base, where algorithms and operators are implemented once and applied to multiple mesh types, thus avoiding code repetition and facilitating bug fixing and software maintenance. Cinolib is header only, it is licensed with MIT, and is already publicly available on GitHub (`https://github.com/mlivesu/cinolib`). Using templates to define the attributes associated to each mesh element, it is both highly customizable and straightforward to compile.

2 Software description

Cinolib's flexibility is made possible by a hierarchical mesh data structure that embraces both surfaces and volumes, and is at the core of the library (Section 2.1). Overall, the library comprises around 50K lines of code, which include both the core, and a variety of widespread geometry processing tools to support mesh generation, visualization, analysis, and so forth (Section 2.2).

2.1 Architecture

In Cinolib the connectivity of each mesh is represented as a set of elements' lists and their adjacency, which are collocated in a hierarchical data structure that comprises both surfaces and volumes. As depicted in Figure 1, the hierarchy is a tree. The root summarizes all the properties that are common to *any* mesh, whereas the leaves are the actual meshes the user can create, that is: triangle meshes, quad meshes, general polygon meshes, tetrahedral meshes, hexahedral meshes, and general polyhedral meshes (Figure 2). In between there is a middle layer, which contains two nodes that summarize common properties which are

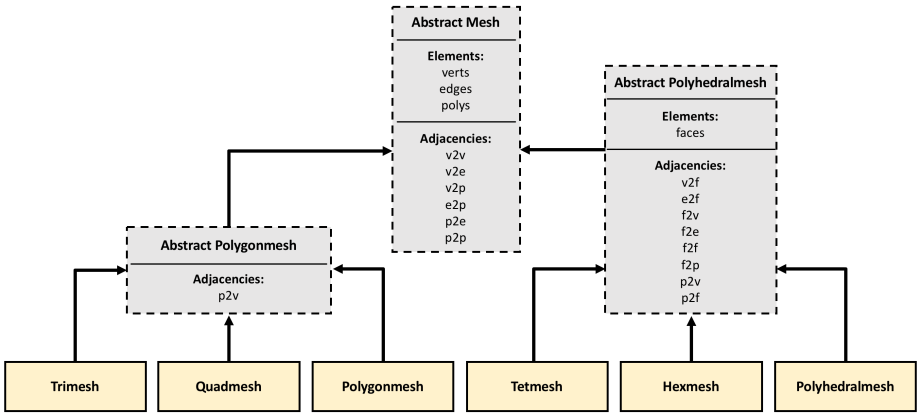

Fig. 1. The mesh hierarchy at the base of Cinolib. Yellow boxes represent the meshes available in the library; gray boxes are the abstract classes from which they inherit the structure. Elements and adjacencies that are common to both surface and volume meshes are stored at the root of the hierarchy (the *p* in AbstractMesh denotes a *polygon* for a surface mesh, and a *polyhedron* for a volume mesh).

specific to surface meshes and volumetric meshes, respectively. The philosophy of Cinolib is to make sure that algorithms operate at the highest possible level in the hierarchy, so that they can be applied to the widest set of meshes with a unique implementation. This has clear advantages, as it reduces the coding effort and greatly simplifies bug fixing and software maintenance in general. As a practical example the reader may think of the Dijkstra's algorithm for shortest path computation on a graph. If we consider the *primal* mesh, that is the graph having as nodes the mesh vertices and as arcs the mesh edges, Dijkstra operates indistinctively on surfaces and volumes. In Cinolib Dijkstra's routines operate at the root of the hierarchy, thus the same piece of code is used by all the meshes supported in the library. Let us now consider the *dual* mesh, that is the graph having one node per element, and one arc for each pair of adjacent elements. Here there is a difference between surfaces and volumes, as elements are polygons in the former, and polyhedra in the latter. To maximize compatibility between surface and volume meshes a strict naming convention is used throughout the whole library. Surface meshes are defined as lists of: verts (v), edges (e) and polys (p); volume meshes are defined as lists of: verts (v), edges (e), faces (f) and polys (p). The word polys appears both in surfaces and volumes, but denotes polygons in the former and polyhedra in the latter. This double meaning of the term polys is exploited in the hierarchy, which defines polys at the top level and allows algorithms that work on the dual mesh and only require poly to poly adjacency (p2p) to operate indistinctively on a surface or volumetric mesh. This is the case of Dijkstra's on the dual mesh, but also of methods to compute spanning trees and various clustering algorithms that start associating a label to one element and expand the cluster by conquering adjacent elements. All these algorithms

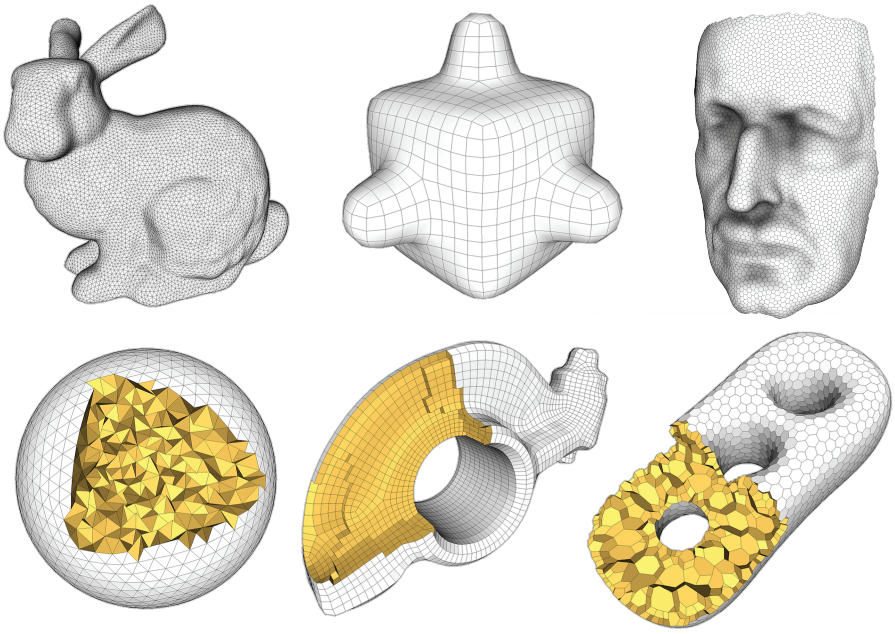

Fig. 2. From left to right, top to bottom: triangle mesh, quadmesh, general polygon mesh, tetmesh, hexmesh, general polyhedral mesh. All these meshes are supported by Cinolib.

are implemented once, and used by all the surface and volumetric meshes in the library. Similarly, all methods that apply only to surfaces (or volumes) but are not specific to a particular mesh type, are implemented at the middle level of the hierarchy, and the same code covers all the surface (or volumetric) meshes supported in the library. This is the case of various operators that allow to edit the mesh connectivity (e.g. addition/removal of mesh elements), computation of gradients and iso contours of functions encoded at mesh vertices, and so forth.

2.2 Functionalities

Besides the mesh hierarchy, Cinolib offers a variety of tools and algorithms that cover a wide specturm of ubiquitous operations in computer graphics and engineering. This section provides a non exhaustive lists of its key features. Unless stated differently, all the features are implemented directly in the code base and do not depend from external software, making the library extremely easy to compile and use

– **Discrete Differential operators:** many tools in computer graphics and engineering require solving Poisson and Laplace problems, and make extensive use of discrete differential operators defined on meshes. Cinolib provides

discretizations of the most common differential operators (gradient, divergence and laplacian) for all the meshes in the library, as well as wraps to the linear solvers of Eigen [12], which is header only and therefore as easy to compile as Cinolib. Specifically, the gradient operator supports both per vertex and per poly gradient fields [24], and it is based on the Green-Gauss method [32], which works on any type of polygons and polyhedra. The divergence operator is obtained by simply transposing the matrix of the gradient operator (see Section 3 in [18] for details), and the Laplace operator is available with uniform weights for all the meshes, and with the ubiquitous cotangent weights for triangle [26] and tetrahedral [19] meshes;

– **Mesh generation and processing:** Cinolib contains wraps to Triangle [28] and Tetgen [29] for triangle and tetrahedral mesh generation, respectively. It also provides mesh dualization to convert them into general polygonal and polyhedral meshes, as well as other utilities such as topological editing operators (e.g. add/remove polys, edge collapse, edge split), and subdivision schemes (e.g. midpoint subdivision);

– **Fields:** defining scalar or vector fields on mesh vertices and polys is useful both for mesh generation/processing and for scientific visualization (e.g. error plots). Cinolib supports per vertex scalar fields, and also allows to compute and visualize iso-lines of fields embedded on surface meshes, or iso-surfaces of fields embedded on volumetric meshes. Thanks to the aforementioned topological editing operators both iso-lines and iso-surfaces can be optionally embedded in the mesh connectivity, splitting the edges they traverse (Figure 3). Vector fields are also supported, as well as tools to process and visualize integral lines that align to them (Figure 4);

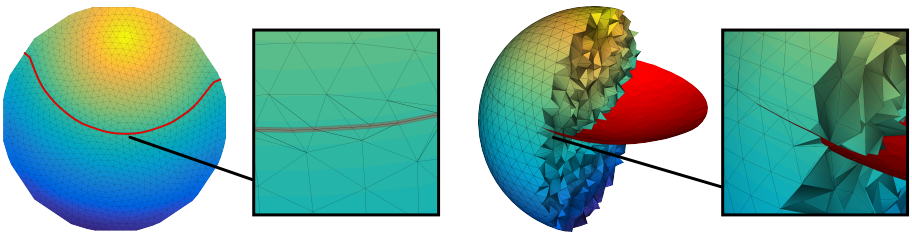

Fig. 3. Level sets of a scalar field embedded on the vertices of a surface mesh (left) and volumetric mesh (right). Curves and surfaces can be optionally embedded in the mesh connectivity by splitting the edges they traverse (closeups). This latter operation is currently supported only for triangle and tetrahedral meshes.

– **Distances and paths:** Cinolib contains various implementations of Breadth-First Search (BFS) and shortest paths computation (using Dijkstra [9]). Both

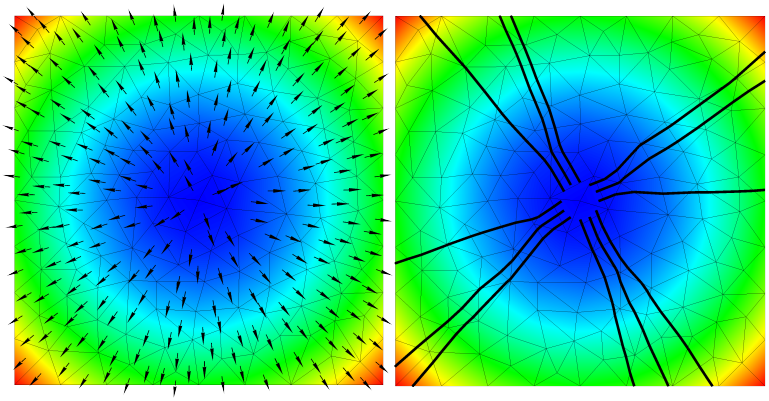

Fig. 4. Left: a scalar function embedded on the vertices of a surface mesh, and its (per poly) gradient field. Right: four bundles of integral lines that emanate from the center and align to the gradient field. The figure is taken from [24].

algorithms can run on both the primal and the dual mesh, support optional constraints (e.g. barriers), and rely on a unique implementation for all the meshes in the hierarchy. The library also contains an implementation of the heat-based geodesics [7], which also operates at the root of the mesh hierarchy and can be applied to any mesh in the library. Heat geodesics are computed solving an initial value problem rather than a boundary problem. This allows to factorize the matrix of the heat flow operator once, and then solve the geodesic problem as many times the user wishes in real time by means of a simple back-substitution (less than 0.005 seconds on the bunny shown in Figure 5, which contains 14K vertices);

- **Visualization/rendering:** various tools for rendering and visualization are offered to the user, spanning from triangulation of arbitrary polygons to plot general polygonal/polyhedral meshes, to various colors schemes and 1D/2D texture facilities. Popular color maps such as the HSV ramp (Figure 4) and Parula (Figure 3), as well as textures commonly used to show error plots and distortion maps in scientific papers (Figure 6) are also included in the library and are available for all the meshes in the hierarchy. Additionally, Cinolib provides an OpenGL canvas with trackball and perspective/orthograpic camera, and includes a slicing tool that allows to cut any mesh with axis aligned planes and show/hide only a portion of it (Figure 2). The computation of ambient occlusion [25] for realistic and more revealing rendering of complex 3D scenes is also supported (Figure7);

- **3D printing:** facilities for additive manufacturing are also included in the library. Specifically, Cinolib allows to read a set of slices and transform them into a polygon mesh that can be used for both visualization and analysis. Since support structures are typically encoded as 1D lines and their ultimate shape depends on the material and hardware used for print [20], a tool for

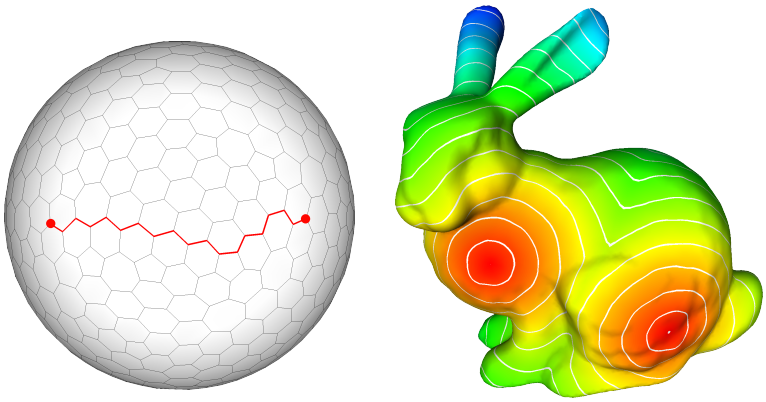

Fig. 5. Left: shortest path between two mesh vertices of a general polygonal mesh, computed with Dijkstra. Right: heat-based geodesics computed with [7].

their thickening and merging is included, so that the user can set the proper thickening radius and check how supports will look like in the actual object before the print happens (Figure 8);

– **IO:** Cinolib currently supports a wide set of popular file formats in the computer graphics and engineering community, such as OFF, OBJ and IV for surfaces, and MESH, VTU, VTK, TET for volumes. Since general polyhedral meshes are not well supported by these formats, we also added IO facilities for the HYBRID format defined by the authors of [11], as well as proposed a new format (HEDRA) for such meshes. Additionally, Cinolib can read CLI files for 3D printing facilities, and can read or write curve skeletons encoded in various ad-hoc file formats used by the authors of relevant papers in the field, such as [34,8,21,23];

– **Utilities:** additional utilities are also available, such as wraps for the min-cut and graph-cut algorithms [4]; profiling utilities to measure timings and improve performances; quality metrics for polygons (e.g. maximum inscribed circle, minimum outer circle, kernel), tetrahedra and hexahedra (implementing [33]); computation of coarse layouts for quad and hex meshes [5,30]; tools for the analysis and processing of curve-skeletons (used in [19,35,22]), as well as other features not included in this list due to space limits.

3 Illustrative Example

As discussed in Section 2.2, Cinolib provides an extensive set of tools and functionalities. Illustrating all of them is out of the scope of the article. This section

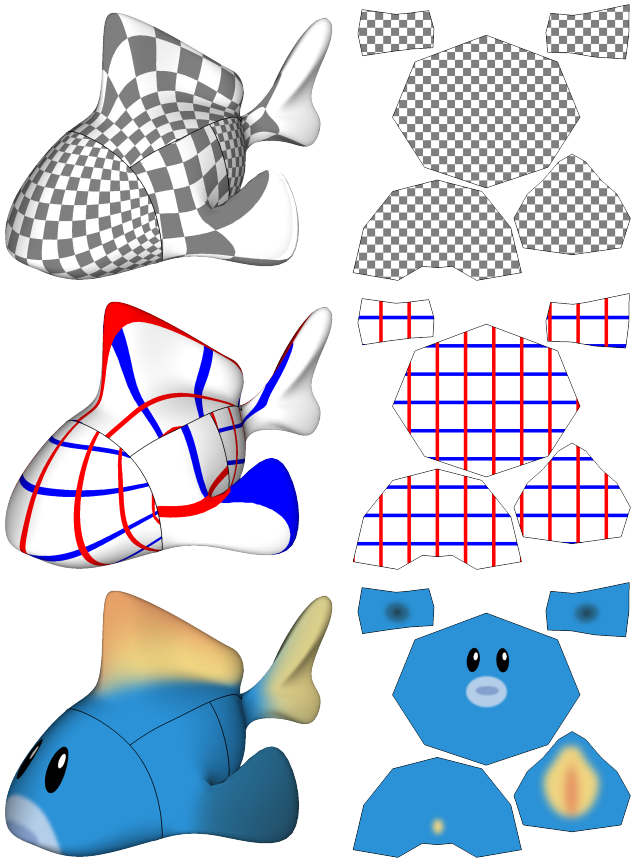

Fig. 6. Cinolib supports popular 2D textures to show distortion of UV maps, such as checkerboards (top) and iso-lines (middle), but also textures loaded from external image files (bottom).

proposes just a simple code sample that shows how to write a program to compute a harmonic field on a general polygonal mesh. Harmonic functions are at the base of many remeshing and parameterization techniques in computer graphics [10,19]. Given a set of vertices where the function reaches its extrema, computing a harmonic field f amounts to solving a Laplace equation $\Delta f = 0$, subject to Dirichlet boundary conditions on the (known a priori) function extrema. Cinolib supports the computation of harmonic fields, as well as their visualization and inspection, with tools such as color maps, iso contours, gradients and integral lines. In the following a simple program for the field computation is shown:

```
#include <cinolib/gui/qt/glcanvas.h>
#include <cinolib/meshes/meshes.h>
#include <cinolib/harmonic_map.h>
```

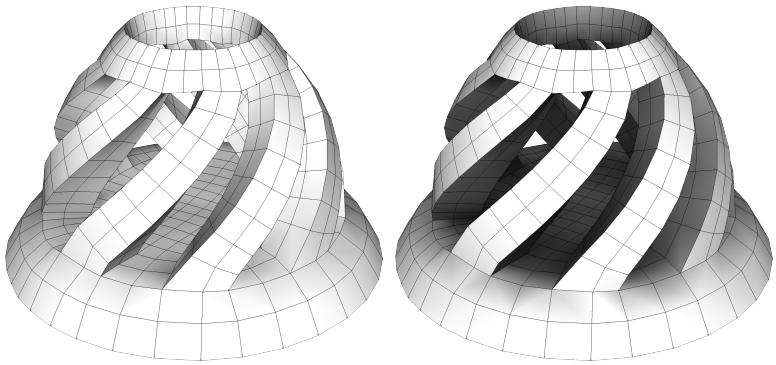

Fig. 7. Standard smooth shading may produce images of 3D scenes that are difficult to parse for the observer (left). The use of ambient occlusion produces more realistic images, where shadows help to better understand the geometry (right)

```
void main()
{
   // load a general polygon mesh from file
   cinolib::DrawablePolygonmesh<> m("./bunny.obj");

   // setup the Dirichlet boundary conditions of the field. We
   // are looking for a function that evaluates 0 at vertex v0
   // and 1 at vertex v100
   std::map<int,double> bc;
   bc[0] = 0.0;
   bc[100] = 1.0;

   // compute a harmonic field f. The Laplace operator is discretized with uniform weights
   cinolib::ScalarField f = harmonic_map(m, bc, 1, UNIFORM);

   // copy the field on the mesh (for visualization)
   f.copy_to_mesh(m);

   // create and show an OpenGL canvas showing the mesh and the field
   cinolib::GLcanvas gui;
   gui.push_obj(&m);
   gui.show();
}
```

As can be noticed from the lines above, the code is extremely concise yet intuitive. Figure 9 shows the result it produces. The routine for computation of harmonic fields, as well as many others based on discrete differential geometry operators, work at the highest level of the mesh hierarchy. Therefore, a single implementation can be re-used on many meshes, without the need to change a

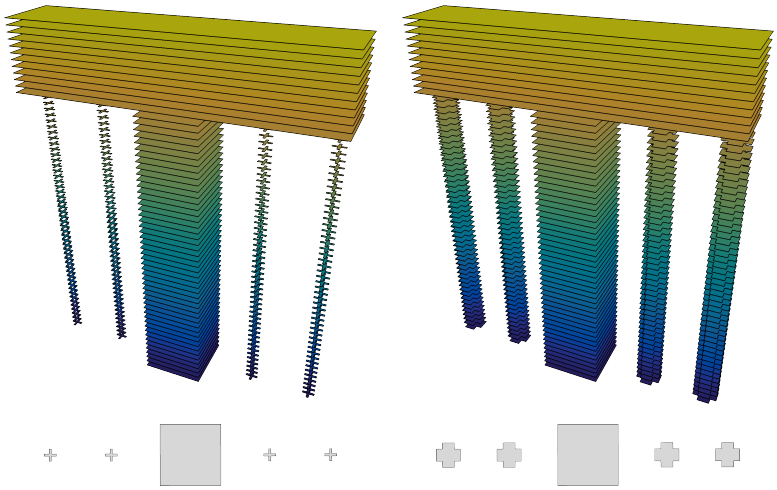

Fig. 8. Two versions of a sliced T-like shape sustained by four columns of supports. External supports are typically encoded as piece-wise linear curves, and their ultimate size and shape depends on the material and hardware used for the 3d print (e.g. the size of the filament in FDM, or the laser beam in SLM/SLS). Cinolib offers functionalities for loading and converting sliced data into polygon meshes that can be visually inspected, as well as thickening and merging of support structures, to check how they will look like in the final print. This figure is taken from [20], where these functionalities were used to convert sliced data into tetrahedral meshes for the simulation of 3d printing processes.

single line of code, if not the one where the mesh is loaded. This feature was used to produce all the results shown in [18].

4 Conclusions and future works

A new mesh processing C++ library is presented. Compared to alternative libraries, Cinolib is specifically designed to handle surface and volumetric meshes made of general polygons and polyhedra. Its hierarchical mesh data structure allows to easily develop algorithms that naturally scale to multiple mesh types, without repetitions of code occurring. Although it tries to be as efficient as possible, whenever generality and efficiency were in conflict, generality was always pursued. All in all, Cinolib is made by researchers for researchers, and is specifically designed to promote code re-usability and to easily create software prototypes that validate ideas and algorithms to be presented in scientific papers. Cinolib is licensed with MIT, and is already publicly available on GitHub (`https://github.com/mlivesu/cinolib`). We expect wide adoption from researchers in computer graphics and engineering, specifically from the ones that develop numerical methods and tools for hybrid mesh generation. The

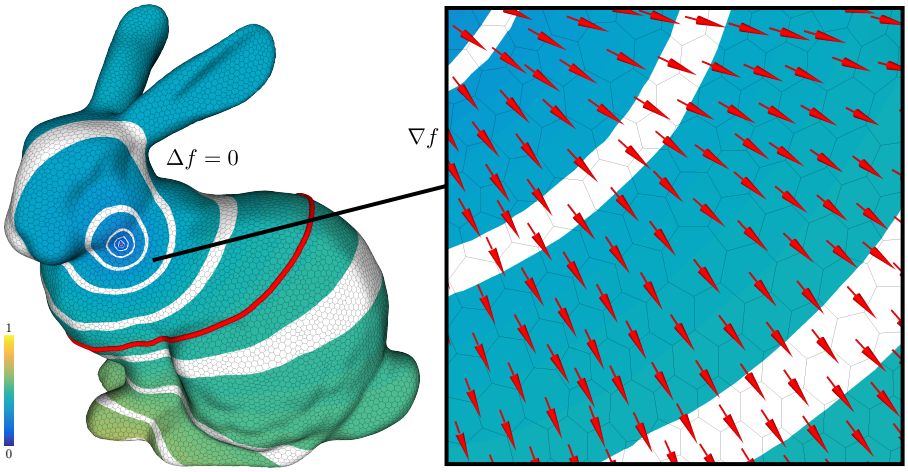

Fig. 9. Example of the solution of a Laplace problem to compute a harmonic function f on a general polygonal mesh of the Stanford Bunny. Colorization is given by the popular Parula colormap, enriched with white bands to show the level sets of the function. In thick red a iso-curve of the field is shown. The closeup shows details of the tessellation and the (normalized) gradient ∇f, computed with the Green-Gauss method, which applies to any polygon and polyhedron [32]. Right: the code used to compute the field and draw it on the canvas. The *harmonic map* routine operates at the highest level of the mesh hierarchy; substituting *DrawableTrimesh* with a different type the very same code could be used to compute similar fields on any other mesh supported by Cinolib, without changing a single line of code.

development of Cinolib is partially supported by the EU ERC Advanced Grant CHANGE which, among other things, aims to improve the Virtual Element Method for the solution of PDEs on surface and volumetric meshes made of general polygons and polyhedra.

Future works: while this article gives an overview of the library as of today, Cinolib is constantly growing and incorporating new functionalities. At the moment some of the tools exposed are supported only by simplicial meshes (e.g. tracing integral lines, or incorporating level sets into the mesh connectivity). To this end, future extensions will focus on moving such algorithms higher in the mesh hierarchy in order to extend such functionalities to all the meshes. Regarding new tools to be implemented, priority will be given to: the improvement of connectivity editing operators for general meshes (after [11]) and for quad and hex meshing (e.g. sheet/chord collapse); the implementation of a QP solver and of Lagrange multipliers for constrained optimization; the implementation of tools to create and process surface cross field and volumetric frame fields; the implementation of well-established methods for mesh deformation (e.g. [31]), and the introduction of tools for cage- and skeleton-based animation.

Acknowledgements

This work is partially supported by the EU ERC Advanced Grant CHANGE, grant agreement No.694515. We thank Keenan Crane for releasing the fish model in Figure 6 to the public domain.

References

1. libhedra: geometric processing and optimization of polygonal meshes, author = Amir Vaxman and others, note = https://github.com/avaxman/libhedra, year = 2017,
2. Attene, M.: Imatistl-fast and reliable mesh processing with a hybrid kernel. In: Transactions on Computational Science XXIX, pp. 86–96. Springer (2017)
3. Botsch, M., Steinberg, S., Bischoff, S., Kobbelt, L.: Openmesh-a generic and efficient polygon mesh data structure (2002)
4. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Transactions on pattern analysis and machine intelligence **23**(11), 1222–1239 (2001)
5. Cherchi, G., Livesu, M., Scateni, R.: Polycube simplification for coarse layouts of surfaces and volumes. Computer Graphics Forum **35**(5), 11–20 (2016). https://doi.org/10.1111/cgf.12959
6. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: an open-source mesh processing tool. In: Eurographics Italian chapter conference. vol. 2008, pp. 129–136 (2008)
7. Crane, K., Weischedel, C., Wardetzky, M.: Geodesics in heat: A new approach to computing distance based on heat flow. ACM Transactions on Graphics (TOG) **32**(5), 152 (2013)
8. Dey, T.K., Sun, J.: Defining and computing curve-skeletons with medial geodesic function. In: Proceedings of the fourth Eurographics symposium on Geometry processing. pp. 143–152. Eurographics Association (2006)
9. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische mathematik **1**(1), 269–271 (1959)
10. Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W.: Multiresolution analysis of arbitrary meshes. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. pp. 173–182. ACM (1995)
11. Gao, X., Jakob, W., Tarini, M., Panozzo, D.: Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. ACM Transactions on Graphics (TOG) **36**(4), 114 (2017)
12. Guennebaud, G., Jacob, B., et al.: Eigen v3. http://eigen.tuxfamily.org (2010)
13. Hughes, T.J., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. Computer methods in applied mechanics and engineering **194**(39-41), 4135–4195 (2005)
14. Jacobson, A., Panozzo, D., et al.: libigl: A simple C++ geometry processing library (2018), http://libigl.github.io/libigl/
15. Lab, C.I.V.C.: VCG: Visualization and computer graphics library (2004), https://github.com/cnr-isti-vclab/vcglib
16. Levy, B.: Geogram (2015)
17. Lévy, B., Petitjean, S., Ray, N., Maillot, J.: Least squares conformal maps for automatic texture atlas generation. In: ACM transactions on graphics (TOG). vol. 21, pp. 362–371. ACM (2002)

18. Livesu, M.: A heat flow relaxation scheme for n dimensional discrete hyper surfaces. Computers & Graphics **71**, 124 – 131 (2018). https://doi.org/10.1016/j.cag.2018.01.004
19. Livesu, M., Attene, M., Patane, G., Spagnuolo, M.: Explicit cylindrical maps for general tubular shapes. Computer-Aided Design **90**, 27 – 36 (2017). https://doi.org/http://dx.doi.org/10.1016/j.cad.2017.05.002, sI:SPM2017
20. Livesu, M., Cabiddu, D., Attene, M.: slice2mesh: a meshing tool for the simulation of additive manufacturing processes. Computers & Graphics **80**, 73 – 84 (2019). https://doi.org/10.1016/j.cag.2019.03.004
21. Livesu, M., Guggeri, F., Scateni, R.: Reconstructing the curve-skeletons of 3d shapes using the visual hull. IEEE Transactions on Visualization and Computer Graphics **18**(11), 1891–1901 (2012). https://doi.org/10.1109/TVCG.2012.71
22. Livesu, M., Muntoni, A., Puppo, E., Scateni, R.: Skeleton-driven adaptive hexahedral meshing of tubular shapes. Computer Graphics Forum **35**(7), 237–246 (2016). https://doi.org/10.1111/cgf.13021
23. Livesu, M., Scateni, R.: Extracting curve-skeletons from digital shapes using occluding contours. The Visual Computer **29**(9), 907–916 (2013). https://doi.org/10.1007/s00371-013-0855-8
24. Mancinelli, C., Livesu, M., Puppo, E.: A comparison of methods for gradient field estimation on simplicial meshes. Computers & Graphics **80**, 37 – 50 (2019). https://doi.org/10.1016/j.cag.2019.03.005
25. Méndez-Feliu, À., Sbert, M.: From obscurances to ambient occlusion: A survey. The Visual Computer **25**(2), 181–196 (2009)
26. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Visualization and mathematics III, pp. 35–57. Springer (2003)
27. Schneider, T., Dumas, J., Gao, X., Botsch, M., Panozzo, D., Zorin, D.: Poly-spline finite element method. CoRR **abs/1804.03245** (2018), `http://arxiv.org/abs/1804.03245`
28. Shewchuk, J.R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: Applied computational geometry towards geometric engineering, pp. 203–222. Springer (1996)
29. Si, H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. ACM Transactions on Mathematical Software (TOMS) **41**(2), 11 (2015)
30. Sorgente, T., Biasotti, S., Livesu, M., Spagnuolo, M.: Topology-driven shape chartification. Computer-Aided Geometric Design **65**, 13 – 28 (2018). https://doi.org/https://doi.org/10.1016/j.cagd.2018.07.001
31. Sorkine, O., Alexa, M.: As-rigid-as-possible surface modeling. In: Proceedings of the fifth Eurographics symposium on Geometry processing. pp. 109–116. Eurographics Association (2007)
32. Sozer, E., Brehm, C., Kiris, C.C.: Gradient calculation methods on arbitrary polyhedral unstructured meshes for cell-centered cfd solvers. In: 52nd Aerospace Sciences Meeting. p. 1440 (2014)
33. Stimpson, C., Ernst, C., Knupp, P., Pébay, P., Thompson, D.: The verdict library reference manual. Sandia National Laboratories Technical Report **9** (2007)
34. Tagliasacchi, A., Alhashim, I., Olson, M., Zhang, H.: Mean curvature skeletons. In: Computer Graphics Forum. vol. 31, pp. 1735–1744. Wiley Online Library (2012)
35. Usai, F., Livesu, M., Puppo, E., Tarini, M., Scateni, R.: Extraction of the quad layout of a triangle mesh guided by its curve skeleton. ACM Transactions on Graphics **35**(1), 6:1–6:13 (2015). https://doi.org/10.1145/2809785

36. Beirão da Veiga, L., Brezzi, F., Marini, L.D., Russo, A.: The hitchhiker's guide to the virtual element method. Mathematical models and methods in applied sciences **24**(08), 1541–1573 (2014)