MatGeom: A toolbox for geometry processing with MATLAB.

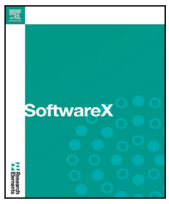
David Legland

SoftwareX 29 (2025) 101984

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

MatGeom: A toolbox for geometry processing with MATLAB

David Legland

INRAE, UR BIA, 44316 Nantes, France
INRAE, PROBE Research Infrastructure, BIBS Facility, 44316 Nantes, France

ARTICLE INFO

Keywords:
Geometry processing
Geometric computing
Polygon
Polygon mesh
Geometry

ABSTRACT

MatGeom (for "MATLAB Geometry") is a pure MATLAB library for geometry processing in two and three dimensions, that aims at facilitating the processing and analysis of scientific data. It provides a collection of functions for the manipulation of common 2D and 3D geometries such as points, lines, ellipses, polygons, or polygon meshes. Functions allow for combining together geometries (intersections, mutual distances, projections, fitting to a set of points), evaluating quantitative features (area, volume, curvatures, orientations), or drawing with various options. The library is fully documented: user manual, code comments, function headers, and demonstration scripts.

Code metadata

Current code version	1.2.8
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00469
Permanent link to Reproducible Capsule	For example: https://zenodo.org/records/12800469
Legal Code License	BSD 2-Clause "Simplified" License.
Code versioning system used	git
Software code languages, tools, and services used	MATLAB
Compilation requirements, operating environments & dependencies	No compilation required (pure MATLAB library), and no dependency to third-party libraries.
If available Link to developer documentation/manual	https://github.com/mattools/matGeom
Support email for questions	david.legland(at)inrae.fr

1. Motivation and significance

The processing of geometric data is a recurrent task in many scientific domains, either when analyzing data obtained from imaging (e.g. microscopy, tomography) or laser scanning, or when modeling and fitting of geometric models to data.

A large number of geometry processing libraries have been developed over the years. The most generic one is certainly CGAL, that provides a high level of abstraction and a high level of control on the precision of computation results [1]. Other significant libraries comprise libigl [2], GeometricTools [3], or the Wykobi library [4], to name only a few. These libraries are proposed for the C++ language, making them powerful for building end-user applications. They are however not easy to integrate into common data analysis workflows that are usually written in interpreted language such as MATLAB, Python or R. Moreover, the C++ language is relatively more complex than the above-mentioned languages, requiring higher programming skills, and often larger development time.

Besides integrated software, geometric data are often analyzed within interpreted platforms such as MATLAB (or its open-source counter-part octave), Python, or R. Within the MATLAB environment, a large number of geometry processing toolboxes have been proposed. Many of them have been developed for specific data structures, such as 2D or 3D polygon meshes [5–7]. In particular, the gptoolbox provides a large number of state-of-the-art methods for the processing of 3D polygon meshes [6]. The ACME library is a 3D geometry library written in C++, that can be compiled for MATLAB [8]. It provides several algorithms for 3D computational geometry, in particular for the computation of collisions and intersections, and focusses on the management of a large number of elements. It is however limited to a

E-mail address: david.legland@inrae.fr.

https://doi.org/10.1016/j.softx.2024.101984
Received 29 August 2024; Received in revised form 11 November 2024; Accepted 15 November 2024
Available online 26 November 2024

small number of geometric primitives. The GIBBON toolbox is an open-source MATLAB toolbox, that provides a large number of geometry visualization and processing tools [9]. It focusses on Computer Aided Design (CAD), mesh generation from 3D volume images, and Finite Element Modeling of 3D structures represented as polygon meshes.

As a complement to more specialized toolboxes, the MatGeom library aims at providing a simple yet versatile library for the processing of geometry data within the MATLAB environment, in two or three dimensions, without requiring external dependencies nor compilation procedure. It was designed with the objective of quickly providing solutions to common problems in the analysis, editing and modeling of geometric data, by considering a larger variety of geometric primitives (ellipses, ellipsoids, cylinders...), and providing a large number of utility functions not necessarily provided within other toolboxes. It also aims at leveraging the need for computer-science related skills, and making it simple to extend by developing specific scripts or functions. Moreover, the use of pure MATLAB language makes its de facto available within the open-source Octave software [10].

2. Software description

2.1. Software functionalities

Features include the creation of geometries of various types, the combination of basic geometries (intersection, projection, distance), or the determination of an equivalent geometry from a collection of points (e.g. computation of convex hull, or fitting an equivalent circle, ellipse or ellipsoid).

The MatGeom library was designed with the aim to keep a low level of complexity, to facilitate both its compatibility with other software and its appropriation by users. It relies on numeric arrays for representing geometric data, and on functions to perform geometric computations.

2.1.1. Array representation of geometries

Each geometric primitive is represented as a numeric row vector. For example, 2D or 3D points or vectors are represented using a 1×2 or 1×3 array containing their coordinates, ellipses are represented as 1×5 array corresponding to center, minor and major radius, and orientation values, etc. Collections of elementary geometries are easily represented as two-dimensional numeric arrays concatenating representation of each geometry. More complex geometries such as polygons or polygon meshes are based on $n \times 2$ or $n \times 3$ numeric array containing vertex coordinates, eventually combined with topological information (a face vertex index array). This simplicity allows simple interoperability with native MATLAB functions or other libraries, and makes it easy to export geometry data into text files.

2.1.2. Function-based library

Geometry processing is performed via MATLAB functions that follow specific naming conventions. Processing functions start with a verb describing the action, and post-concatenate the type of the geometry expected as input: "resamplePolygon", "drawEllipse", "smoothMesh", etc.

Geometries may also be quantified through descriptive features such as perimeter, surface area, curvatures, dihedral angles, etc. The functions that compute a descriptive feature usually start with the name of the type of the geometry, followed by the name of the feature, for example: "polygonArea", "ellipsePerimeter", "meshCurvatures"...

Each primitive is associated to a "drawXXX" function to facilitate its graphical representation, using classical plotting options from MATLAB.

2.1.3. Library documentation

All operations are described in a user manual, illustrated with simple examples in order to facilitate re-usability. In addition, each function is fully documented, most of them providing a description of function syntax together with a running sample code, and a list of related functions.

It is expected that the simplicity of the library make it easy to use by a non computer-scientist user, easy to interconnect with more specialized toolboxes, and easy to extend by developing specific processing scripts or functions.

2.2. Software architecture

The MatGeom library is organized into several modules, that gather functions operating on similar data structures. The geom2d and the geom3d modules provide the core features of geometric operations in the 2D or 3D Euclidean spaces. The polygons2d and the meshes3d provides a collection of functions for geometry processing of 2D polygons and polylines, and of 3D polygon meshes. The graphs module is devoted to the manipulation of "geometric graphs", where vertices correspond to 2D or 3D points, and edges to adjacencies between vertices.

2.2.1. Module geom2d

The geom2d module provides functions for processing points, vectors, linear shapes (including straight lines, line segments, or rays). It also manages smooth curves such as circles or ellipses, or utility geometries such as bounding boxes. Several functions have been generalized to manage inputs with any dimensionality. It also manages geometric (affine) transforms, by providing functions for creating transforms, for transforming geometries, or fitting registration transforms.

2.2.2. Module polygons2d

The polygons2d module gathers the functions operating on polygons and polylines represented as a list of vertices. Polygons or polylines are represented by $n_p \times 2$ arrays containing the coordinates of the n_p vertices defining the geometry. Multiple polygons can be represented by a cell array, each cell containing the vertex coordinates of one of the polygon rings.

The polygons2d module comprises edition functions (like resampling, smoothing, or reversing a polygon), computation of intersections with a linear geometry or another polygon, the computation of measures (perimeter, area, normal angles) or derived geometries (centroid), or more complex operations such as triangulation or skeletonization of a polygon.

2.2.3. Module geom3d

The geom3d module is the equivalent of the geom2d module for the 3D Euclidean space. It provides functions for processing 3D points, vectors, linear shapes (including 3D lines and planes) or smooth surface geometries such as ellipsoids, cylinders, or torus.

As for the geom2d module, geometries are represented using row vectors. Functions allow for computing distances, angles, or intersections between geometries, computing projection of points, or computing position of points within geometries.

A large number of functions are provided for the management of 3D affine transforms, either to create elementary transforms, to combine them, to transform geometries, or to convert between the different representations: rotation matrix, Euler angles, axis-angle... The transformation corresponding to the rigid registration of two 3D point sets can be computed by using the Iterated Closest Point algorithm.

2.2.4. Module meshes3d

The meshes3d module allows for managing and processing 3D polygonal meshes. Several types of mesh are supported: trimesh, quadmesh, or more generic meshes with variable number of vertices per face. Polygon mesh processing require to consider both the location of the vertices (as a $n_p \times 3$ numeric array) and the topology of the mesh. The topology of meshes is represented by a $n_f \times 3$ (for triangular meshes) or $n_f \times 4$ (for quadrangular meshes) array containing the indices of vertices corresponding to each of the n_f faces. Functions operating on meshes expect either a pair of input arguments, or a MATLAB structure containing at least the two fields "vertices" and "faces".

The library provides functions to read and write meshes from files in common file formats, or to generate polygonal models approximating smooth surfaces (spheres, cylinders, torus). A large number of mesh processing operations have been implemented, comprising global mesh processing (mesh smoothing, triangulation, recursive subdivision), interactions with other geometries (intersections with lines or planes, clipping, distance to points), computation of related geometries (face normals, bounding box), or quantification of geometrical features (surface area, curvature map, edge dihedral angle).

2.2.5. The graphs module

The graphs module is devoted to the manipulation of "geometric graphs". Geometric graphs are defined within MatGeom as traditional graphs (a set of edges defines the adjacencies between a set of vertices), with the additional assumption that vertices are associated to a 2D or 3D position. The topology of graphs is represented by a $n_e \times 2$ array containing indices of adjacent vertices, with n_e being the number of edges in the graph.

Geometric graphs can be useful for representing geometric features obtained from images such as the boundary of a binary image, or the adjacency graph of regions within a segmented image. They also can be used as basis for more complex operations on polygons or polygonal meshes, such as the computation of polygon skeleton.

2.3. Comparison with other software

In comparison with other software and libraries, MatGeom aims at providing a generic and simple to use library, that can be used in conjunction with more specialized libraries.

The target audience of MatGeom is more the data analyst than a specialist of geometry processing. For example, if the question is to explore geometric data and comparing them with fitted primitives, then MatGeom may be useful. If the performance is an issue, for example when manipulating polygon or meshes with very large number of vertices, or if the user needs to create more specialized data structure, then the use of the reference library CGAL, or of a more specialized software, may be more relevant. The use of the CGAL library within MATLAB however requires integration of compiled C++ code, which is not always a trivial task.

Compared to the Acme library, the MatGeom library provides a larger number of geometric primitives, and provides also 2D geometry processing tools. If the number of elements is very large, then the Acme library is expected to be more efficient.

When working with 3D for polygon meshes, the gptoolbox provides a larger number of geometry processing tools than MatGeom and relies on well-established libraries such as CGAL or libigl. The MatGeom library can easily be used as a complement, either to combine meshes with other geometric primitives (planes, ellipsoids...), or to facilitate the visualization of computation results.

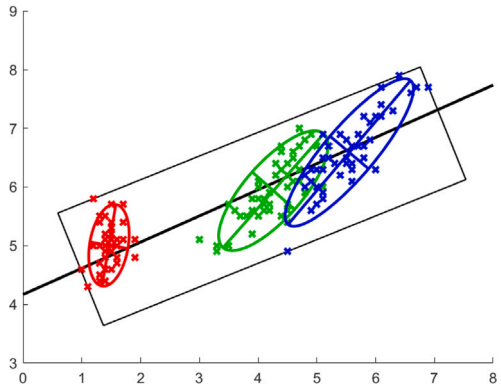


Fig. 1. Analysis of the geometry of a collection of points. Computation of the fitting line, of the oriented bounding box, and of the equivalent ellipse of three subsets of the collection.

Listing 1 : Using the geom2d module for data analysis.

```
% load data
str = load('fisheriris');
data = str.meas;
% create point collection
pts = data(:, [3 1]);

% display
figure; axis equal; hold on; axis([0 8 3 9]);
drawPoint(pts, 'bx');

% Fit line on the whole collection
line = fitLine(pts);
drawLine(line, 'color', 'k', 'linewidth', 2);

% Draw oriented box
obox = orientedBox(pts);
drawOrientedBox(obox, 'color', 'k', 'linewidth', 1);

% Process by species
[labels,  ~, inds]= unique(str.species);
colors = [1 0 0; 0 0.8 0; 0 0 1];
for i = 1:3
    pts_i = pts(inds == i, :);
    drawPoint(pts_i, 'marker', 'x', 'color', colors(i,:), '
        linewidth', 2);
    elli = equivalentEllipse(pts_i);
    drawEllipse(elli, 'color', colors(i,:), 'linewidth', 2)
    drawEllipseAxes(elli, 'color', colors(i,:), 'linewidth',
        2)
end
```

3. Illustrative examples

3.1. Data exploration

The listing 1 illustrates some features of the geom2d module to display point data sets, retrieve geometric primitives that summarizes them, and provide graphical display. The result is shown on Fig. 1.

3.2. Polygon processing

The listing 2 demonstrates the usage of the library for more complex processing on polygon data. The sample data represent the contour of

Listing 2 : Polygon processing example script.

```
% read polygon data as a numeric N-by-2 array
poly = load('leaf_poly.txt');

% display the polygon using basic color option
figure;axis equal;hold on;axis([0 600 0 400]);
drawPolygon(poly, 'k');

% compute polygon bounding box
poly2 = boundingBox(poly);
drawBox(poly2, 'k');

% compute convex hull of polygon vertices
poly3 = convexHull(poly);
drawPolygon(poly3,'LineWidth',2,'Color','k');

% apply smoothing to the original polygon.
poly4 = smoothPolygon(poly, 51);
drawPolygon(poly4,'Color','b','LineWidth',2);

% compute a simplified version of the polygon
poly5 = simplifyPolygon(poly, 20);
drawPolygon(poly5,'Color','r','LineWidth',2);
drawVertices(poly5, 'Color', 'k',...
    'Marker', 's', 'MarkerFaceColor', 'w');

% compute intersections with an arbitrary line
line = createLine([0 250], [600 350]);
drawLine(line, 'k');
inters = intersectLinePolygon(line, poly5);
drawPoint(inters,'Color','r','Marker','o',...
    'MarkerFaceColor','w','LineWidth', 2);
```

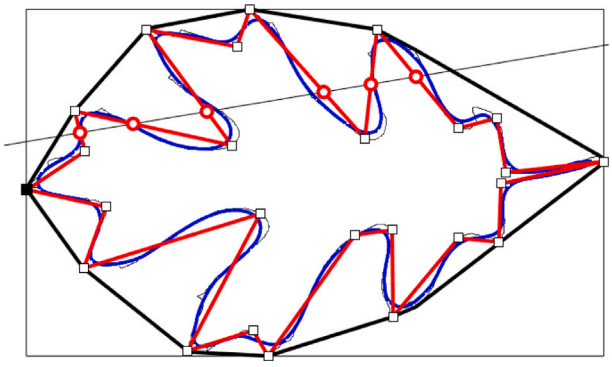


Fig. 2. Demonstration of various geometry operations on a polygon. Thin black curve: original polygon. Blue curve: smoothing of the polygon. Red curve: polygonal simplification of the polygon. Thick black curve: computation of the convex hull. Thin black box: bounding box. Red dots: intersection with an arbitrary line. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

a leaf, stored as a collection of pairs of coordinates. The result is shown in Fig. 2.

3.3. Mesh processing

The meshes3d module provides functions for the manipulation of 3D surface meshes, also known as polygonal meshes. The library support triangular meshes, quadrangular meshes, as well as meshes with arbitrary number of face vertices for some functions.

Listing 3 : Mesh processing example script.

```
% read sample mesh and display mesh using equal-scale axes
mesh = readMesh('bunny_F1k.ply');
figure; hold on; axis equal; view(3);
drawMesh(mesh, 'faceColor', [.7 .7 .7]);
axis([-8 10 -6 8 -8 10]); view(15, 20);

% Display face normals and centroids
normals = meshFaceNormals(mesh);
centroids = meshFaceCentroids(mesh);
figure; hold on; axis equal; view(3);
drawMesh(mesh, 'faceColor', [.7 .7 .7]);
axis([-8 10 -6 8 -8 10]); view(15, 20);
drawArrow3d(centroids, normals);

% Compute distance point mesh
point = [8 -3 8;2 -5 8;-6 -4 -4];
[dist, proj] = distancePointMesh(point, mesh);
% also compute a distance map for a vertical slice
    intersecting the mesh
lx = linspace(-8, 10, 181); lz = linspace(-8, 10, 181);
[x, z] = meshgrid(lx, lz); y = ones(size(x)) * 3;
pts = [x(:) y(:) z(:)];
dists = distancePointMesh(pts, mesh);
distMap = reshape(dists, size(x));
% display mesh
figure; hold on; axis equal; view(3);
drawMesh(mesh, 'faceColor', [.7 .7 .7]);
axis([-8 10 -6 8 -8 10]);view(15, 20);
% display the distance map
surf(x, y, z, distMap, 'linestyle', 'none');
% display point-to-mesh distances
drawPoint3d(point, 'ko');
drawPoint3d(proj, 'k*');
drawEdge3d([point proj], 'color', 'k', 'linewidth', 2)

% Compute the two main curvatures on each vertex of the mesh
[k1, k2] = meshCurvatures(mesh.vertices, mesh.faces);
figure; hold on; axis equal; view(3);
drawMesh(mesh, 'VertexColor', k1 .* k2);
axis([-8 10 -6 8 -8 10]); view(15, 20);
set(gca, 'clim', [-0.01 0.01]);
colormap jet;

% Clip mesh with a plane
plane = createPlane([0 0 0], [-5 5 3]);
[v2, f2] = clipMeshByPlane(mesh, plane);
figure; hold on; axis equal; view(3);
drawMesh(v2, f2, 'faceColor', [.7 .7 .7]);
axis([-8 10 -6 8 -8 10]); view(15, 20);
% also draw the boundary
boundary = meshBoundary(v2, f2);
drawPolygon3d(boundary, 'linewidth', 2, 'color', 'm');
```

The listing 3 illustrates a selection of mesh processing features available within the MatGeom library. Some of their results are shown on Fig. 3.

4. Impact

The development of the library originally started as a need for validating the development of image processing and analysis algorithms

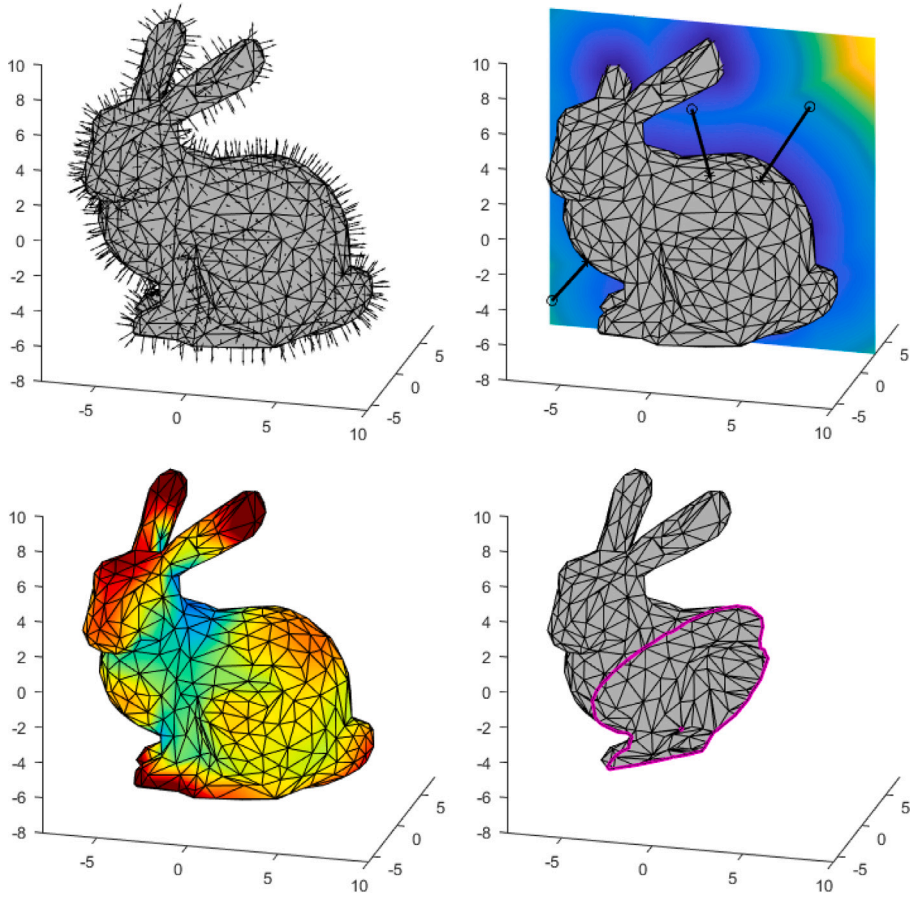


Fig. 3. Demonstration of various mesh processing operations on the standard "Stanford Bunny" triangle mesh. Computation of face normal vectors and centroids. Computation of distance between 3d points and mesh. Computation of Gaussian curvature for each vertex of the mesh. Clipping of the mesh with a plane and extraction of mesh boundary.

[11,12], and for the modeling of biological structures [13]. The initial set of functions was later used and expanded to develop image analysis workflows in the context of plant sciences, either from two-dimensional images [14,15], or from 3D volumetric images [16].

The library was furthermore used by other people, mostly for research in biology: study of tissue morphogenesis in *Drosophila*[17,18], of space perception within insects [19], or of anatomical structure of animal organs [20–22]. It was also used for medical research in neurosurgery [23], for the modeling of neuronal activity [24], or for modeling torsion angles of biological molecules [25]. Besides applications in biology, the MatGeom library was also used to analyze coatings properties of sprays [26], to simulate retinal optic flow during natural locomotion [27], or for applications in robotics [28,29].

Some features of the MatGeom library have also been integrated into other software. Examples include the *Microscopy Image Browser*, for the reconstruction of 3D biological structures from serial sections [30], *FoCa*, a planning system for the study of proton radiotherapy [31], or *DICE*, an application for the quantification of fractures within granular materials [32]. These integrations can be seen as a positive marker of the facility to use the MatGeom library in interaction with other tools.

5. Conclusions

Using a well established code base, the MatGeom library provides geometry processing features that made it useful for a variety of applications, ranging from data analysis to modeling. MatGeom can be used in conjunction with other MATLAB libraries, and is open to public contributions.

A number of improvements can be envisioned. The management of a larger variety of primitives (like Bezier curves, NURBS, or curves and surfaces defined by a parametric equation) could facilitate the modeling from complex data. From a performance point of view, the introduction of efficient data structures could improve the management of a larger number of elements, or of geometries based on a large number of vertices. The use of parallel computation seems also promising [33].

The management of a large number of functions (more than 500 in the last version of the library) can raise some difficulty both from a user point of view (to quickly identify the right function for the right task) and from a developer point of view (to maintain a consistent set of functions interacting together). As MATLAB provides Object-Oriented programming, its use could simplify the global organization of the library, by encapsulating features related to a given family of geometries (e.g. ellipsoids) into a specific class. A difficulty would be to maintain a low level of complexity for the library user, while providing new features and consolidating existing ones.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Many people have contributed to the library, through bug reportings, push requests, or proposals for enhancements, and are greatly acknowledged. In particular, Sven Holcombe and GitHub user oqilipo have contributed several functions, and deserve particular acknowledgments.

References

[1] The CGAL Project. CGAL user and reference manual. 5th ed.. CGAL Editorial Board; 2023.

[2] Jacobson A, Panozzo D, et al. Libigl: A simple C++ geometry processing library. 2018, https://libigl.github.io/.

[3] Eberly D. Geometric Tools Engine, version 5, https://github.com/davideberly/GeometricTools.

[4] Partow A. Wykobi, C++ Computational Geometry Library, version 0.0.5, http://www.wykobi.com/index.html.

[5] Peyre G. Toolbox Graph, MATLAB central file exchange. 2009, https://www.mathworks.com/matlabcentral/fileexchange/5355-toolbox-graph, (Last retrieved 23 August 2024).

[6] Jacobson A, et al. gptoolbox: Geometry processing toolbox. 2024, http://github.com/alecjacobson/gptoolbox.

[7] Engwirda D. Locally-optimal delaunay-refinement and optimisation-based mesh generation [Ph.D. thesis], School of Mathematics and Statistics, The University of Sydney; 2014.

[8] Stocco D, Bertolazzi E. Acme: A small 3D geometry library. SoftwareX 2021;16:100845. http://dx.doi.org/10.1016/j.softx.2021.100845.

[9] Moerman KM. GIBBON: The geometry and image-based bioengineering add-on. J Open Source Softw 2018;3(22):506. http://dx.doi.org/10.21105/joss.00506.

[10] Eaton JW, Bateman D, Hauberg S, Wehbring R. GNU Octave version 9.2.0 manual: a high-level interactive language for numerical computations. 2024, https://www.gnu.org/software/octave/doc/v9.2.0/.

[11] Legland D, Kiêu K, Devaux M-F. Computation of Minkowski measures on 2D and 3D binary images. Image Anal Stereol 2007;26(6):83–92. http://dx.doi.org/10.5566/ias.v26.p83-92.

[12] Lehmann G, Legland D. Efficient n-dimensional surface estimation using crofton formula and run-length encoding. Tech. rep, 2012.

[13] Legland D, Devaux M-F, Kiêu K, Bouchet B. Stereological estimation for layered structures based on slabs perpendicular to a surface. J Microsc 2008;232(1):44–55. http://dx.doi.org/10.1111/j.1365-2818.2008.02080.x, submitted.

[14] Legland D, Devaux M-F, Guillon F. Statistical mapping of maize bundle intensity at the stem scale using spatial normalisation of replicated images. PLoS ONE 2014;9(3):e90673. http://dx.doi.org/10.1371/journal.pone.0090673.

[15] Schaefer E, Belcram K, Uyttewaal M, Duroc Y, Goussot M, Legland D, et al. The preprophase band of microtubules controls the robustness of division orientation in plants. Science 2017;356(6334):186–9. http://dx.doi.org/10.1126/science.aal3016.

[16] Le TDQ, Alvarado C, Girousse C, Legland D, Chateigner-Boutin A-L. Use of X-ray micro computed tomography imaging to analyze the morphology of wheat grain through its development. Plant Methods 2019;15(1):84.

[17] Supatto W, McMahon A, Fraser SE, Stathopoulos A. Quantitative imaging of collective cell migration during drosophila gastrulation: multiphoton microscopy and computational analysis. Nat Protoc 2009;4:1397–412. http://dx.doi.org/10.1038/nprot.2009.130.

[18] Eritano AS, Bromley CL, Bolea Albero A, Schütz L, Wen F-L, Takeda M, et al. Tissue-scale mechanical coupling reduces morphogenetic noise to ensure precision during epithelial folding. Dev Cell 2020;53(2):212–28. http://dx.doi.org/10.1016/j.devcel.2020.02.012.

[19] Dürr V, Schilling M. Transfer of spatial contact information among limbs and the notion of peripersonal space in insects. Front Comput Neurosci 2018;12. http://dx.doi.org/10.3389/fncom.2018.00101.

[20] Le Garrec J-F, Domínguez JN, Desgrange A, Ivanovitch KD, Raphaël E, Bangham JA, et al. Predictive model of asymmetric morphogenesis from 3D reconstructions of mouse heart looping dynamics. eLife 2017;6:e28951. http://dx.doi.org/10.7554/eLife.28951.

[21] Sijilmassi O, López Alonso J-M, Del Río Sevilla A, del Carmen Barrio Asensio M. Multifractal analysis of embryonic eye structures from female mice with dietary folic acid deficiency. Part I: fractal dimension, lacunarity, divergence, and multifractal spectrum. Chaos Solitons Fractals 2020;138:109885. http://dx.doi.org/10.1016/j.chaos.2020.109885.

[22] Hrncirik F, Roberts IV, Swords C, Christopher PJ, Chhabu A, Gee AH, et al. Impact of scala tympani geometry on insertion forces during implantation. Biosensors 2022;12(11). http://dx.doi.org/10.3390/bios12110999.

[23] Hirt L, Thies KA, Ojemann S, Abosch A, Darwin ML, Thompson JA, Kern DS. Case series investigating the differences between stimulation of rostral zona incerta region in isolation or in conjunction with the subthalamic nucleus on acute clinical effects for parkinson's disease. Interdiscipl Neurosurg 2022;29:101553. http://dx.doi.org/10.1016/j.inat.2022.101553.

[24] Zhang Y, Chen Y, Wang T, Cui H. Neural geometry from mixed sensorimotor selectivity for predictive sensorimotor control. 2024, http://dx.doi.org/10.7554/elife.100064.1.

[25] Tikhonov DA, Kulikova LI, Efimov AV. Analysis of torsion angles between helical axes in pairs of helices in protein molecules. Math Biol Bioinformat 2018;12(2):398–410. http://dx.doi.org/10.17537/2017.12.398.

[26] Katranidis V, Kamnis S, Gu S. Prediction of coating properties of thermally sprayed WC–co on complex geometries. J Therm Spray Tech 2018;27:1025–37. http://dx.doi.org/10.1007/s11666-018-0739-6.

[27] Matthis JS, Muller KS, Bonnen KL, Hayhoe MM. Retinal optic flow during natural locomotion. Plos ONE 2022;18(2):e1009575. http://dx.doi.org/10.1371/journal.pcbi.1009575.

[28] Kurz T, Eberhard P, Henninger C, Schiehlen W. From neweul to neweul-M2: symbolical equations of motion for multibody system analysis and synthesis. Multibody Syst Dyn 2010;24:25–41. http://dx.doi.org/10.1007/s11044-010-9187-x.

[29] Gonzalez R, Mahulea C, Kloetzer M. A matlab-based interactive simulator for mobile robotics. In: 2015 IEEE international conference on automation science and engineering (CASE), Gothenburg, Sweden. 2015, p. 310–5. http://dx.doi.org/10.1109/CoASE.2015.7294097.

[30] Belevich M, andJoensuu Ilya, Kumar D, Vihinen H, Jokitalo E. Microscopy Image Browser: A platform for segmentation and analysis of multidimensional datasets. PLoS Biol 2016;14(1):1–13. http://dx.doi.org/10.1371/journal.pbio.1002340.

[31] Sánchez-Parcerisa D, Kondrla M, Shaindlin A, Carabe A. FoCa: a modular treatment planning system for proton radiotherapy with research and educational purposes. Phys Med Biol 2014;59:7341. http://dx.doi.org/10.1088/0031-9155/59/23/7341.

[32] Menegoni N, Giordan D, Inama R, Perotti C. DICE: An open-source MATLAB application for quantification and parametrization of digital outcrop model-based fracture datasets. J Rock Mech Geotech Eng 2023;15(5):1090–110. http://dx.doi.org/10.1016/j.jrmge.2022.09.011.

[33] Wang Y, Yesantharao R, Yu S, Dhulipala L, Gu Y, Shun J. ParGeo: A library for parallel computational geometry. In: 30th annual European symposium on algorithms (ESA 2022). Leibniz international proceedings in informatics (lIPIcs), vol. 244. Schloss Dagstuhl – Leibniz-Zentrum für Informatik; 2022, p. 88:1–88:19. http://dx.doi.org/10.1371/journal.pbio.1002340.