MatGeom library user manual

D. Legland

June 8, 2020

Abstract *The MatGeom library provides a collection of functions for geometric computing within the Matab environment. It is organised in several modules, devoted to generic computations in 2D or 3D, polylines and polygons operators, 3D meshes operators, or geometric graphs operators. Many plotting functions are provided to facilitate the graphical representation of computation results. The library is provided with a large amount of user help: code comments, function headers, demonstration scripts...*

Contents

1 Introduction

MatGeom is a library for geometric computing with Matlab in 2D and 3D. It contains several hundreds of functions for the creation and manipulation of 2D and 3D shapes such as point sets, lines, polygons, 3D meshes, ellipses...

The official homepage for the project is hosted on GitHub[1]. A starting help is provided in the MatGeom wiki.

The library is organised into several modules:

geom2d General functions in euclidean plane

polygons2d Functions operating on point lists

graphs Manipulation of geometric graphs

polynomialCurves2d Representation of smooth polynomial curves

geom3d General functions in 3D euclidean space

meshes3d Manipulation of 3D polygonal meshes

[1] http://github.com/mattools/matGeom

2 Module geom2d

The geom2d module of the MatGeom library allows to process geometric planar shapes such as point sets, edges, straight lines, bounding boxes, conics (circles and ellipses)... Most functions works for planar shapes, but some ones have been extended to 3D or to any dimension. Other modules provide additional functions for specific shapes: polygons2d, graphs, polynomialCurves2d.

Contents

2.1 Points and vectors

Both points and vectors are defined by their two cartesian coordinates, stored into a row vector of 2 elements:

```
pt = [x y];
vect = [vx vy];
```

Point sets and vector sets are stored in a matrix with two columns, one for the x-coordinate, one for the y-coordinate:

```
pts = [x1 y1 ; x2 y2 ; x3 y3];
vectList = [vx1 vy1 ; vx2 vy2 ; vx3 vy3];
```

2.1.1 Points

General functions operating on points.

points2d
Description of functions operating on points.

midPoint
Middle point of two points or of an edge.

circumCenter
Circumcenter of three points.

isCounterClockwise
Computes relative orientation of 3 points.

polarPoint
Creates a point from polar coordinates (rho + theta).

angle2Points
Computes horizontal angle between 2 points.

angle3Points
Computes oriented angle made by 3 points.

distancePoints
Computes distance between two points.

transformPoint
Applies an affine transform to a point or a point array.

drawPoint
Draws the point on the axis.

2.1.2 Point Sets

The following listings provides an overview of some functions operating on point sets. The result is shown on Figure 2.1.

```
% generate random data
rng(42); pts = randn([100 2]) * 15 + 50;
% compute derived shapes
centro = centroid(pts); bbox = boundingBox(pts);
elli = equivalentEllipse(pts); hull = convexHull(pts);
% display shapes
figure; hold on; axis([0 100 0 100]);
drawPoint(pts, 'color', 'k', 'marker', 'o', 'linewidth', 2);
drawPoint(centPts, 'color', 'b', 'marker', '*', 'linewidth', 2, 'MarkerSize', 10);
drawBox(bbox, 'color', [0 0 .7], 'linewidth', 2);
drawEllipse(elli, 'color', [.7 0 0], 'linewidth', 2);
drawPolygon(hull, 'color', [0 .7 0], 'linewidth', 2);
legend({'Points', 'Centroid', 'BoundingBox', 'Equiv. Ellipse', 'Conv. Hull'}, 'Location', 'NorthEast');
```

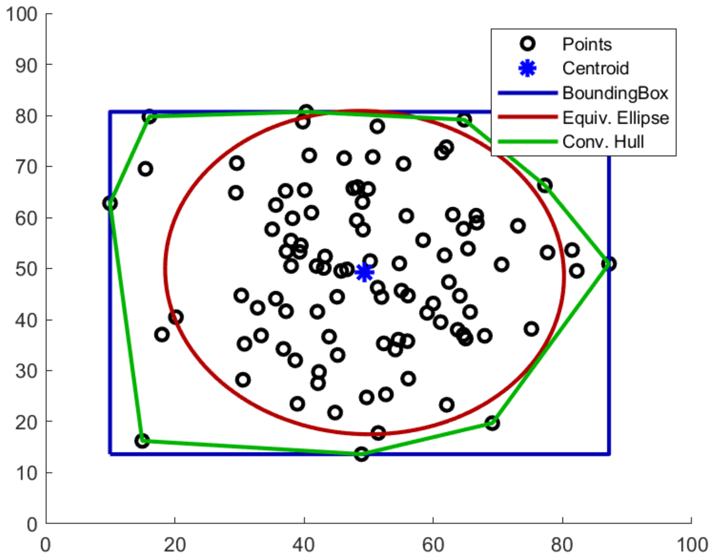


Figure 2.1: *Generation of a random point set and computation of geometric derived shapes.*

clipPoints
Clips a set of points by a box.

centroid
Computes centroid (center of mass) of a set of points.

boundingBox
Bounding box of a set of points.

principalAxes
Principal axes of a set of ND points, returned as a centroid, a rotation matrix, and optionally a scaling factor. See also EquivalentEllipse and EquivalentEllipsoid.

angleSort
Sorts points in the plane according to their angle to origin.

findClosestPoint
Finds index of closest point in an array.

minDistancePoints
Minimal distance between several points.

mergeClosePoints
Merges points that are closer than a given distance.

hausdorffDistance
Hausdorff distance between two point sets.

nndist
Nearest-neighbor distances of each point in a set.

2.1.3 Vectors

General functions operating on vectors.

vectors2d
Description of functions operating on plane vectors.

createVector
Creates a vector from two points.

vectorNorm
Computes norm of a vector, or of a set of vectors.

vectorAngle
Angle of a vector, or between 2 vectors.

normalizeVector
Normalizes a vector to have norm equal to 1.

isPerpendicular
Checks orthogonality of two vectors.

isParallel
Checks parallelism of two vectors.

transformVector
Transforms a vector with an affine transform.

rotateVector
Rotates a vector by a given angle.

2.1.4 Various drawing functions

Some functions allow to draw less standard objects.

drawVector
Draws vector at a given position.

drawArrow
Draws an arrow on the current axis.

drawLabels
Draws labels at specified positions.

drawShape
Draws various types of shapes (circles, polygons...).

2.2 Linear shapes

Linear shapes encompass three kinds of shapes:

straight lines are infinite in ech direction

line segments, or edges correspond to the set of points between two extremity points

rays emanate from a point, and are unbounded in one direction

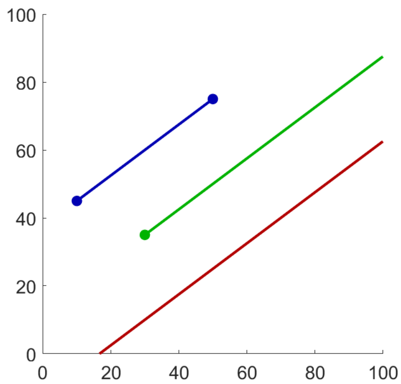


Figure 2.2: *Three examples of linear shapes: line segment, ray, and straight line. The ray and the line are clipped by the axis bounds.*

An example of each of these three shapes is represented on Fig. 2.2.

2.2.1 Straight lines

Straight lines are infinite in each direction.

lines2d
Description of functions operating on planar lines.

createLine
Creates a straight line from 2 points, or from other inputs.

medianLine
Creates a median line between two points.

cartesianLine
Creates a straight line from cartesian equation coefficients.

orthogonalLine
Creates a line orthogonal to another one through a point.

parallelLine
Creates a line parallel to another one.

intersectLines
Returns all intersection points of N lines in 2D.

lineAngle
Computes angle between two straight lines.

linePosition
Position of a point on a line.

lineFit
Fits a straight line to a set of points.

clipLine
Clips a line with a box.

reverseLine
Returns same line but with opposite orientation.

transformLine
Transforms a line with an affine transform.

drawLine
Draws a straight line clipped by the current axis.

2.2.2 Edges (line segments between 2 points)

Line segments correspond to the set of points between two extremity points. The term "edge" is used interchangeably with line segment.

edges2d
Description of functions operating on planar edges.

createEdge
Creates an edge between two points, or from a line.

edgeToLine
Converts an edge to a straight line.

edgeAngle
Returns the horizontal angle of edge.

edgeLength
Returns the length of an edge.

parallelEdge
Create a new edge parallel to another edge.

centeredEdgeToEdge
Converts a centered edge to a two-points edge.

midPoint
Middle point of two points or of an edge.

edgePosition
Returns the position of a point on an edge.

clipEdge
Clips an edge with a rectangular box.

reverseEdge
Interverts the source and target vertices of edge.

intersectEdges
Returns all intersections between two sets of edges.

intersectLineEdge
Returns the intersection between a line and an edge.

transformEdge
Transforms an edge with an affine transform.

edgeToPolyline
Converts an edge to a polyline with a given number of segments.

drawEdge
Draws an edge given by 2 points.

drawCenteredEdge
Draws an edge centered on a point.

2.2.3 Rays

Rays emanate from a point, and are unbounded in one direction.

rays2d
Description of functions operating on planar rays.

createRay
Creates a ray (half-line), from various inputs.

bisector
Returns the bisector of two lines, or 3 points.

clipRay
Clips a ray with a box.

drawRay
Draws a ray on the current axis.

2.2.4 Relations between points and lines

These functions determine relative position of a point (or an array of points) and a linear shape.

distancePointEdge
Minimum distance between a point and an edge.

distancePointLine
Minimum distance between a point and a line.

projPointOnLine
Projects of a point orthogonally onto a line.

pointOnLine
Creates a point on a line at a given position on the line.

isPointOnLine
Tests if a point belongs to a line.

isPointOnEdge
Tests if a point belongs to an edge.

isPointOnRay
Tests if a point belongs to a ray.

isLeftOriented
Tests if a point is on the left side of a line.

2.3 Smooth curves

Smooth curves comprises simple curves such as circles, ellipses, or other conics, as well as more generic curves such as Bezier curves.

2.3.1 Circles

Several function operate on circles. Circles are represented by a 1-by-3 array [xc yc r], where xc and yc denote the circle center and r denotes the circle radius. Figure 2.3 presents the results of the computation obtained in the following script.

```
% construction of circum circle to three points
pA = [30 20]; pB = [80 40]; pC = [20 70];
circ = circumCircle(pA, pB, pC);
% polygon discretisation
poly = circleToPolygon(circ, 12);
% line-ciercle intersection
line = [60 70 5 2];
inters = intersectLineCircle(line, circ);
```

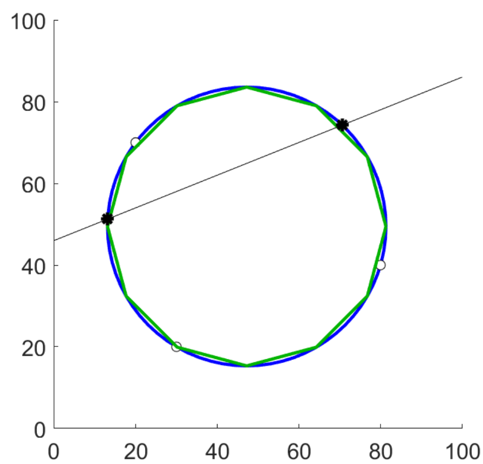


Figure 2.3: *Construction of a circle from 3 points, discretization into a polygon, and computation of its intersection with a line.*

circles2d
Description of functions operating on circles.

createCircle
Creates a circle from 2 or 3 points.

createDirectedCircle
Creates a directed circle.

intersectCircles
Computes the intersection points of two circles.

intersectLineCircle
Compute the intersection point(s) of a line and a circle.

circleToPolygon
Converts a circle into a series of points.

circleArcToPolyline
Converts a circle arc into a series of points.

isPointInCircle
Tests if a point is located inside a given circle.

isPointOnCircle
Tests if a point is located on a given circle.

enclosingCircle
Finds the minimum circle enclosing a set of points.

circumCircle
Circumscribed circle of three points.

radicalAxis
Computes the radical axis (or radical line) of 2 circles

drawCircle
Draws a circle on the current axis.

drawCircleArc
Draws a circle arc on the current axis.

2.3.2 Ellipses and Parabola

Ellipses are represented by a 1-by-5 array [xc yc a b theta], where xc and yc denote the ellipse center, a and b denote the lengths of the semi axes, and theta denotes the orientation of the first principal axis.

ellipses2d
Description of functions operating on ellipses.

equivalentEllipse
Computes the equivalent ellipse with same moments up to the second order as a set of points.

isPointInEllipse
Checks if a point is located inside a given ellipse.

ellipsePerimeter
Computes the perimeter of an ellipse.

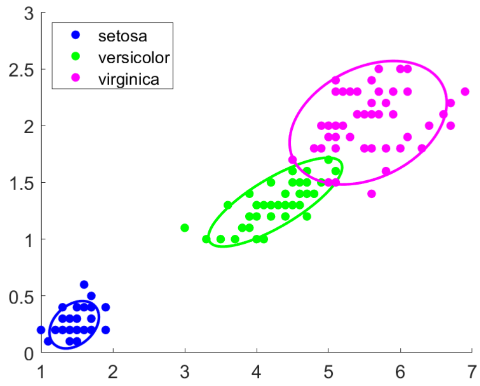


Figure 2.4: *Computation of equivalent ellipses to represent variance of groups within Fisher iris dataset.*

ellipseToPolygon

Converts an ellipse into a series of points.

drawEllipse

Draws an ellipse on the current axis.

drawEllipseArc

Draws an ellipse arc on the current axis.

drawParabola

Draws a parabola on the current axis.

A small example for ellipses is given in following script.

```
load fisherIris;
figure; hold on; set(gca, 'fontsize', 14);
colors = {'b', 'g', 'm'};
hi = zeros(1, 3);
for i = 1:3
    pts = meas((1:50)+(i-1) * 50, 3:4);
    hi(i) = drawPoint(pts, 'Marker', 'o', 'Color', colors{i}, 'MarkerFaceColor', colors{i});
    drawEllipse(equivalentEllipse(pts), 'Color', colors{i}, 'LineWidth', 2);
end
legend(hi, species([1 51 101]), 'Location', 'NorthWest');
```

2.3.3 Splines

Spline curves are a convenient way to represent a large family of curves with a few number control points.

cubicBezierToPolyline
Computes an approximated polyline from Bezier curve control points, specifying the number of vertices.

drawBezierCurve
Draws a cubic bezier curve defined by 4 control points (Fig. 2.5).

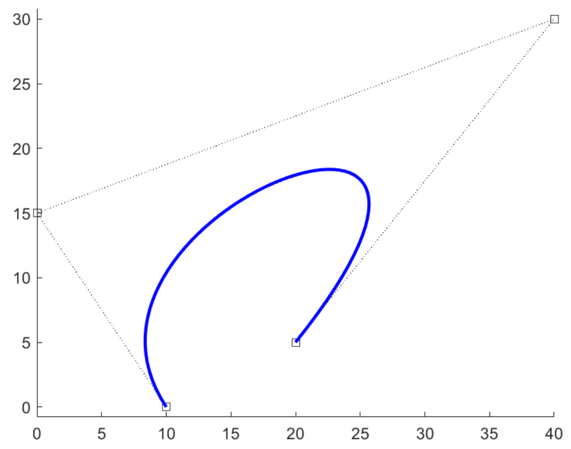


Figure 2.5: *Bezier Curve through four points*

2.4 Polygonal shapes

This sections concerns simple polygons such as triangles, rectangles, and boxes. For polygons with an arbitrary number of vertices, see the Section 3.

2.4.1 Boxes

Boxes are used to represent bounds. They are represented by a four-element row vector containing the minimum and maximum coordinate along each dimension.

```
box = [xmin xmax ymin ymax];
```

boxes2d
Description of functions operating on bounding boxes.

intersectBoxes
Intersection of two bounding boxes.

mergeBoxes
Merges two boxes, by computing their greatest extent.

randomPointInBox
Generates random point within a box.

boxToRect
Converts box data to rectangle data.

boxToPolygon
Converts a bounding box to a square polygon.

drawBox
Draws a box defined by coordinate extents.

2.4.2 Triangles

Triangles are simply represented by a 3×2 array of three points.

isPointInTriangle
Tests if a point is located inside a triangle.

triangleArea
Computes the signed area of a triangle.

2.4.3 Rectangles

A rectangle is represented by the coordinates of the upper-left vertex, and by the dimensions of the rectangle.

```
rect = [x0 y0 sizeX sizeY];
```

rectToPolygon
Converts a rectangle into a polygon (set of vertices).

rectToBox
Converts rectangle data to box data.

drawRect
Draws rectangle on the current axis.

orientedBox
Minimum-width oriented bounding box of a set of points.

orientedBoxToPolygon
Converts an oriented box to a polygon (set of vertices).

drawOrientedBox
Draws centered oriented rectangle.

2.5 Transforms and angles

2.5.1 Geometric transforms

The MatGeom library contains various functions for manipulation of geometric transforms. Most of them consider affine transforms in the plane, that can be represented by a 3-by-3 matrix in homogeneous coordinates:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{xx} & m_{xy} & t_x \\ m_{yx} & m_{yy} & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2.1}$$

where the m_{ij} correspond to the linear part of the transform (rotations, scaling, shear...) and the t_i correspond to the translation part. Note that in MatGeom, while points are represented as 1×2 row vectors, the transform matrix is represented as in eq. 2.1.

transforms2d
Description of functions operating on transforms.

createTranslation
Creates the 3-by-3 matrix of a translation.

createRotation
Creates the 3-by-3 matrix of a rotation by an angle θ, corresponding to the following transform matrix:

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Example:

```
>> createRotation(pi/6)
ans =
    0.8660   -0.5000         0
    0.5000    0.8660         0
         0         0    1.0000
```

createRotation90

Matrix of a rotation around the origin by multiples of 90 degrees. Matrix values are therefore only 0 or ±1. Example:

```
>> createRotation90(1)
ans =
     0    -1     0
     1     0     0
     0     0     1
```

createScaling

Creates the 3-by-3 matrix of a scaling in 2 dimensions.

$$R_\theta = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

createHomothecy

Creates the the 3-by-3 matrix of an homothetic transform.

createBasisTransform

Computes matrix for transforming a basis into another basis.

createLineReflection

Creates the the 3-by-3 matrix of a line reflection.

fitAffineTransform2d

Fits an affine transform using two point sets.

registerICP

Fits an affine transform by using Iterative Closest Point (ICP) algorithm.

polynomialTransform2d

Applies a polynomial transform to a set of points.

fitPolynomialTransform2d

Coefficients of polynomial transform between two point sets.

2.5.2 Angles

Angles are expressed in radians, counter-clockwise, with 0 corresponding to the horizontal direction. Many functions consider angles within the $[0; 2\pi)$ domain.

angles2d
Description of functions for manipulating angles.

normalizeAngle
Normalizes an angle value within the $[0; 2\pi)$ domain.

angleAbsDiff
Absolute difference between two angles.

angleDiff
Difference between two angles.

2.6 Grids and tessellations

This sections presents functions used to generate less common geometric objects such as grids.

squareGrid
Generates equally spaces points in plane.

hexagonalGrid
Generates hexagonal grid of points in the plane.

triangleGrid
Generates triangular grid of points in the plane.

crackPattern
Creates a (bounded) crack pattern tessellation.

crackPattern2
Creates a (bounded) crack pattern tessellation.

3 Module polygons2d

The **polygons2d** module contains functions operating on shapes composed of a vertex list, like polygons or polylines.

Contents

3.1 Data representation

3.1.1 Definitions

A **polyline** is the curve defined by a series of vertices. A polyline can be either **closed** or **open**, depending on whether the last vertex is connected to the first one or not. This can be given as an option is some functions in the module.

A **polygon** is the planar domain delimited by a closed polyline. We sometimes want to consider '**complex polygons**', whose boundary is composed of several disjoint domains. The domain enclosed by a single closed polyline is called '**simple polygon**'. Its boundary is called a '**linear ring**'.

We call **curve** a polyline with many vertices, such that the polyline can be considered as a discrete approximation of a "real" curve.

3.1.2 Data structures

A simple polygon or polyline is represented by a N-by-2 array, each row of the array representing the coordinates of a vertex. Simple polygons are assumed to be closed, so there is no need to repeat the first vertex at the end.

As both polygons and polylines can be represented by a list of vertex coordinates, some functions also consider the vertex list itself. Such functions are prefixed by 'pointSet'. Also, many functions prefixed by 'polygon' or 'polyline' works also on the other type of shape.

For multiple-connected polygons, the different connected boundaries are separated by a row [NaN NaN]. For some functions, the orientation of the polygon can be relevant: CCW stands for 'Counter-Clockwise' (positive orientation), CW stands for 'Clockwise'.

3.1.3 Parametrization

Polylines and polygons are parametrized in the following way:

- the i-th vertex is located at position i-1
- points of the i-th edge have positions ranging linearly from i-1 to i

The parametrization domain for an open polyline is from 0 to $N_v - 1$, and from 0 to N_v for a closed polyline (in the latter case, positions 0 and Nv correspond to the same point).

Example:

```
% Simple polygon:
P1 = [1 1;2 1;2 2;1 2];
drawPolygon(P1);
axis([0 5 0 5]);
% Multiple polygon:
P2 = [10 10;40 10;40 40;10 40;NaN NaN;20 20;20 30;30 30;30 20];
figure; drawPolygon(P2); axis([0 50 0 50]);
```

3.2 Basic operations

3.2.1 Extracting sub-elements

These functions allow to extract specific elements or subsets of a polyline or a polygon.

polygonLoops
Divides a possibly self-intersecting polygon into a set of simple loops.

polygonPoint
Extracts a point from a polygon.

polygonSubcurve
Extracts a portion of a polygon.

polygonEdges
Returns the edges of a simple or multiple polygon.

polygonVertices
Extracts all vertices of a (multi-)polygon.

polylinePoint
Extracts a point from a polyline and a position.

polylineSubcurve
Extracts a portion of a polyline.

3.2.2 Measures

Some functions to compute area, perimeter, or more complex geometric measures on polygons.

polygonBounds
Computes the bounding box of a polygon.

polylineLength
Returns the length of a polyline given as a list of points.

polygonLength
Perimeter of a polygon.

polylineCentroid
Computes centroid of a curve defined by a series of points.

polygonCentroid
Computes the centroid (center of mass) of a polygon.

polygonArea
Computes the signed area of a polygon.

polygonEquivalentEllipse
Computes the ellipse with the same moments as the polygon.

polygonSecondAreaMoments
Computes second-order area moments of a polygon.

polygonNormalAngle
Computes the normal angle at a vertex of the polygon.

polygonOuterNormal
Outer normal vector for a given vertex(ices).

3.3 Smoothing and filtering

Simplification of a polygon or a polyline.

resamplePolyline
Distributes N points equally spaced on a polyline.

resamplePolylineByLength
Resamples a polyline with a fixed sampling step.

resamplePolygon
Distributes N points equally spaced on a polygon.

resamplePolygonByLength
Resamples a polygon with a fixed sampling step.

densifyPolygon
Adds several points on each edge of the polygon.

smoothPolygon
Smooths a polygon using local averaging.

simplifyPolygon
Simplifies a polygon by using Douglas-Peucker algorithm ((Douglas and Peucker, 1973).

3.4 Global processing

More complex operations on polygons.

expandPolygon
Expands a polygon by a given (signed) distance.

triangulatePolygon
Computes a triangulation of the polygon.

polygonSkeletonGraph
Computes the skeleton of a polygon with a dense distribution of vertices, using algorithm from Ogniewicz and Kübler (1995).

medialAxisConvex
Computes the medial axis of a convex polygon (not fully functional).

polygonSymmetryAxis
Tries to identify symmetry axis of polygon.

4 Module graphs

The aim of this module is to provide functions to easily create, modify and display geometric graphs (geometric in a sense the nodes are associated to geometric position in 2D or 3D).

Contents

4.1 Data representation

The Graph module provides functionnalities for the processing of geometric graphs. Graphs are defined by a set of nodes (or vertices), and a relation operator that defines which nodes are neighbors. Geometric graphs associate each node to a position, as a 2D or 3D point.

Graphs are represented by two variables:

nodes which contains coordinates of each vertex

edges which contains indices of start and end vertex.

These two information can be manipulated individually, or be fields of a structure.

Some graph functions consider adjacency list, as a cell array where each cell contains the indices of the neighbor vertices.

Others arrays may sometimes be used:

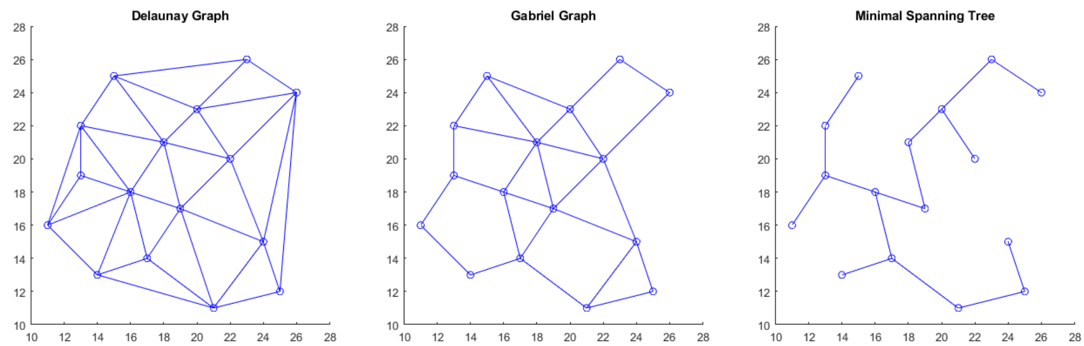
faces which contains indices of vertices of each face (either a double array, or a cell array)

cells which contains indices of faces of each cell.

4.2 Graph creation

4.2.1 Create graphs from point sets

The library contains several functions to generate classical graphs from a set of points. Some of them are illustrated on Figure 4.1.



(a) Delaunay Graph **(b)** Gabriel Graph **(c)** Delaunay Graph

Figure 4.1: *Several graphs generated from a simple set of points.*

delaunayGraph

Graph associated to Delaunay triangulation[1] of input points (Fig. 4.1-a).

[1]https://en.wikipedia.org/wiki/Delaunay_triangulation

euclideanMST
Build the euclidean minimal spanning tree (MST) of a set of points. The minimal spanning tree is the graph with the smallest total length of edges that connect all the nodes of the graph (Fig. 4.1-c).

prim_mst
Computes the minimal spanning tree by using Prim's algorithm.

knnGraph
Create the k-nearest neighbors graph of a set of points.

relativeNeighborhoodGraph
Computes the Relative Neighborhood Graph (RNG) of a set of points. The RNG[2] connects two points by an edge whenever there does not exist any third point that is closer to candidate points than they are to each other.

gabrielGraph
Computes the Gabriel Graph of a set of points. Gabriel Graph[3] connects points if the disc formed by the diameter of the two points does not contain any other point from the set (Fig. 4.1-b).

4.2.2 Voronoi Graphs

Voronoi diagrams are a fundamental data structure in geometry (Aurenhammer, 1991). Several functions are provided to generate graphs corresponding to Voronoi diagram of a set of points. In particular, Centroidal Voronoi Diagrams (CVD), or Centroidal Voronoi Tesselations (CVT), correspond to the case where the germs of the diagram are located on the centroids of the Voronoi polygons.

voronoi2d
Computes a voronoi diagram as a graph structure.

boundedVoronoi2d
Returns a bounded voronoi diagram as a graph structure.

centroidalVoronoi2d
Centroidal Voronoi tesselation within a polygon.

centroidalVoronoi2d_MC
Centroidal Voronoi tesselation by Monte-Carlo.

boundedCentroidalVoronoi2d
Create a 2D Centroidal Voronoi Tesselation in a box.

cvtUpdate
Updates the germs of a CVT with given points.

[2] https://en.wikipedia.org/wiki/Relative_neighborhood_graph
[3] https://en.wikipedia.org/wiki/Gabriel_graph

cvtIterate
Updates the germs of a CVT using random points with given density.

4.2.3 Create graph from images

Some functions allows to generate graphs from a (usually binary) 2D or 3D image. In most cases, node positions correspond to pixels or voxels of the original image.

imageGraph
Create equivalent graph of a binary image.

boundaryGraph
Get boundary of image as a graph.

gcontour2d
Creates contour graph of a 2D binary image.

gcontour3d
Create contour graph of a 3D binary image.

4.3 Operators on graphs

4.3.1 Graph processing (general applications)

adjacencyListToEdges
Convert an adjacency list to an edge array.

pruneGraph
Remove all edges with a terminal vertex.

mergeGraphs
Merge two graphs, by adding nodes, edges and faces lists.

grMergeNodes
Merge two (or more) nodes in a graph.

grMergeMultipleNodes
Simplify a graph by merging multiple nodes.

grMergeMultipleEdges
Remove all edges sharing the same extremities.

grSimplifyBranches
Replace branches of a graph by single edges.

4.3.2 Filtering operations on Graph

These functions adapt morphological operators to operate on graphs data stucture. An array of values associated to the vertices must be provided to the functions. The new values are returned as result.

grMean
Compute mean from neighbours.

grMedian
Compute median from neighbours.

grDilate
Morphological dilation on graph.

grErode
Morphological erosion on graph.

grClose
Morphological closing on graph.

grOpen
Morphological opening on graph.

4.3.3 Geodesic and shortest path operations

grShortestPath
Find a shortest path between two nodes in the graph.

grPropagateDistance
Propagates distances from a vertex to other vertices.

grVertexEccentricity
Eccentricity of vertices in the graph.

graphDiameter
Diameter of a graph.

graphPeripheralVertices
Peripheral vertices of a graph.

graphCenter
Center of a graph.

graphRadius
Radius of a graph.

grFindGeodesicPath
Find a geodesic path between two nodes in the graph.

grFindMaximalLengthPath
Find a path that maximizes sum of edge weights.

4.3.4 Operations for geometric graphs

grEdgeLengths
Compute length of edges in a geometric graph.

grMergeNodeClusters
Merge cluster of connected nodes in a graph.

grMergeNodesMedian
Replace several nodes by their median coordinate.

clipGraph
Clip a graph with a rectangular area.

clipGraphPolygon
Clip a graph with a polygon.

clipMesh2dPolygon
Clip a planar mesh with a polygon.

addSquareFace
Add a (square) face defined from its vertices to a graph.

grFaceToPolygon
Compute the polygon corresponding to a graph face.

graph2Contours
Convert a graph to a set of contour curves.

4.3.5 Graph management (low level operations)

Some functions for removing elements from a graph by maintaining the consistency of the informations.

grRemoveNode
Remove a node in a graph.

grRemoveNodes
Remove several nodes in a graph.

grRemoveEdge
Remove an edge in a graph.

grRemoveEdges
Remove several edges from a graph.

4.4 Graph information

Several functions to obtain quantitative information about a graph.

grNodeDegree
Degree of a node in a (undirected) graph.

grNodeInnerDegree
Inner degree of a node in a graph.

grNodeOuterDegree
Outer degree of a node in a graph.

grAdjacentNodes
Find list of nodes adjacent to a given node.

grAdjacentEdges
Find list of edges adjacent to a given node.

grOppositeNode
Return opposite node in an edge.

grLabel
Associate a label to each connected component of the graph.

4.5 Display

Display a graph, or specific elements of a graph.

drawGraph
Draw a graph, given as a set of vertices and edges.

drawGraphEdges
Draw edges of a graph.

fillGraphFaces
Fill faces of a graph with specified color.

drawDigraph
Draw a directed graph, given as a set of vertices and edges.

drawDirectedEdges
Draw edges with arrow indicating direction.

drawEdgeLabels
Draw values associated to graph edges.

drawNodeLabels
Draw values associated to graph nodes.

drawSquareMesh
Draw a 3D square mesh given as a graph.

patchGraph
Transform 3D graph (mesh) into a patch handle.

4.6 Reading and writing graphs

Read and write graphs from text files using simple format.

4.6.1 Format

An example of graph is given in the following listing.

```
# graph
# nodes
5 2
10 10
20 10
10 20
20 20
27 15
# edges
6
1 2
1 3
2 4
2 5
3 4
4 5
```

Lines starting with a dash are comments. The first part of the file describes the nodes. It starts with a line containing the number of nodes, and the dimensionality of the graph (usually 2 or 3). Then the coordinates of the nodes follow.

The second part of the file describes the edges. It start with a line containing the number of edges. Then the index of source and target vertices of each edge follow. Vertex indices are 1-indexed.

4.6.2 Functions

readGraph
Read a graph from a text file.

writeGraph
Write a graph to an ascii file.

5 Module geom3d

The geom3d module allows to create, manipulate, transform, and visualize geometrical 3D primitives, such as points, lines, planes, polyhedra, circles and spheres.

Contents

5.1 Points and Vectors

Both points and vectors are represented by a 1-by-3 array of coordinates:

```
point = [x0 y0 z0];
vector = [dx dy dz];
```

Arrays of points or vectors are represented by N-by-3 arrays of coordinates.

5.1.1 Points

midPoint3d
Middle point of two 3D points or of a 3D edge.

isCoplanar
Tests input points for coplanarity in 3-space.

transformPoint3d
Transform a point with a 3D affine transform.

distancePoints3d
Computes euclidean distance between pairs of 3D Points.

clipPoints3d
Clips a set of points by a box or other 3d shapes.

drawPoint3d
Draws 3D point on the current axis.

5.1.2 3D Vectors

transformVector3d
Transform a vector with a 3D affine transform.

normalizeVector3d
Normalizes a 3D vector to have norm equal to 1.

vectorNorm3d
Norm of a 3D vector or of set of 3D vectors.

hypot3
Length of a 3D vector, or diagonal length of a cuboidal 3D box.

crossProduct3d
Vector cross product, faster than inbuilt MATLAB cross.

vectorAngle3d
Angle between two 3D vectors.

isParallel3d
Checks parallelism of two 3D vectors.

isPerpendicular3d
Checks orthogonality of two 3D vectors.

drawVector3d
Draws vector at a given position.

5.1.3 Boxes

3D bounding boxes are used to determine physical extent of 3D geometries, or to clip geometries.

```
box = [xmin xmax ymin ymax zmin zmax];
```

5.2 Linear shapes

Linear shapes comprise straight lines, edges (line segments), and rays (half-lines).

A 3D Line is represented by a 3D point (its origin) and a 3D vector (its direction):

```
LINE = [X0 Y0 Z0 DX DY DZ];
```

A 3D ray is represented the same way as a line.

A 3D edge is represented by the coordinates of its extremities:

```
EDGE = [X1 Y1 Z1 X2 Y2 Z2];
```

5.2.1 Creation

createLine3d
Creates a line with various inputs.

fitLine3d
Fits a 3D line to a set of points.

parallelLine3d
Creates 3D line parallel to another one.

transformLine3d
Transforms a 3D line with a 3D affine transform.

reverseLine3d
Returns the same 3D line but with opposite orientation.

5.2.2 Relations with points

distancePointLine3d
Euclidean distance between 3D point and line.

isPointOnLine3d
Tests if a 3D point belongs to a 3D line.

projPointOnLine3d
Projects a 3D point orthogonally onto a 3D line.

distancePointEdge3d
Minimum distance between a 3D point and a 3D edge.

linePosition3d
Returns the position of a 3D point projected on a 3D line.

5.2.3 Clipping and conversion

This functions compute the intersection of a linear geometry with a 3D bounding box.

clipLine3d
Clips a line with a box and return an edge.

clipEdge3d
Clips a 3D edge with a cuboid box.

clipRay3d
Clip a 3D ray with a box and return a 3D edge.

5.2.4 Utility functions

distanceLines3d
Minimal distance between two 3D lines.

edgeToLine3d
Converts a 3D edge to a 3D straight line.

midPoint3d
Middle point of two 3D points or of a 3D edge.

5.2.5 Drawing

Drawing functions for linear geometries, performing clipping with the bounding box correspondingto the current figure axes.

drawLine3d
Draws a 3D line clipped by the current axes.

drawEdge3d
Draws 3D edge in the current axes.

drawRay3d
Draw a 3D ray on the current axis.

5.3 Planes

Planes are represented by a 3D point (the plane origin) and 2 direction vectors, which should not be colinear.

```
PLANE = [X0 Y0 Z0 DX1 DY1 DZ1 DX2 DY2 DZ2];
```

The plane origin and direction vectors can be accessed by using array indexing:

```
plane = ...
origin = plane(1,1:3);
v1 = plane(1, 4:6);
v2 = plane(1, 7:9);
```

5.3.1 Creation and transformations

createPlane
Creates a plane in parametrized form.

medianPlane
Creates a plane in the middle of 2 points.

fitPlane
Fits a 3D plane to a set of points.

normalizePlane
Normalizes parametric representation of a plane.

parallelPlane
Parallel to a plane through a point or at a given distance.

reversePlane
Returns the same 3D plane but with opposite orientation.

transformPlane3d
Transforms a 3D plane with a 3D affine transform.

5.3.2 Computing intersections

intersectPlanes
Returns the intersection line between 2 planes in space.

intersectThreePlanes
Returns the intersection point between 3 planes in space.

intersectLinePlane
Intersection point between a 3D line and a plane.

intersectEdgePlane
Returns intersection point between a plane and a edge.

planesBisector
Bisector plane between two other planes.

5.3.3 Point positions

planePosition
Computes the position of a point on a plane.

planePoint
Computes the 3D position of a point in a plane.

projPointOnPlane
Returns the orthogonal projection of a point on a plane.

distancePointPlane
Signed distance betwen 3D point and plane.

isBelowPlane
Tests whether a point is below or above a plane.

projLineOnPlane
Returns the orthogonal projection of a line on a plane.

5.3.4 Measures

planeNormal
Computes the normal to a plane.

isPlane
Checks if input is a plane.

dihedralAngle
Computes the dihedral angle between 2 planes.

5.3.5 Drawing

drawPlane3d
Draws a plane clipped by the current axes.

5.4 Polygons and polylines

These functions operate on 3D polygons and polylines that are not necessarily embedded into a plane. Both are represented by $N \times 3$ array of vertex coordinates.

5.4.1 Computing intersections

intersectLinePolygon3d
Intersection point of a 3D line and a 3D polygon.

intersectRayPolygon3d
Intersection point of a 3D ray and a 3D polygon.

clipConvexPolygon3dHP
Clips a convex 3D polygon with Half-space.

5.4.2 Measurements

polygonCentroid3d
Centroid (or center of mass) of a polygon.

polygonArea3d
Area of a 3D polygon.

polygon3dNormalAngle
Normal angle at a vertex of the 3D polygon.

5.4.3 Drawing functions

drawPolygon3d
Draws a 3D polygon specified by a list of vertex coords.

drawPolyline3d
Draws a 3D polyline specified by a list of vertex coords.

fillPolygon3d
Fills a 3D polygon specified by a list of vertex coords.

5.4.4 3D Triangles

triangleArea3d
Area of a 3D triangle.

distancePointTriangle3d
Minimum distance between a 3D point and a 3D triangle.

intersectLineTriangle3d
Intersection point of a 3D line and a 3D triangle.

5.5 3D circles and ellipses

fitCircle3d
Fits a 3D circle to a set of points.

circle3dPosition
Returns the angular position of a point on a 3D circle.

circle3dPoint
Coordinates of a point on a 3D circle from its position.

circle3dOrigin
Returns the first point of a 3D circle.

drawCircle3d
Draws a 3D circle.

drawCircleArc3d
Draws a 3D circle arc.

drawEllipse3d
Draws a 3D ellipse.

5.6 Spheres and smooth surfaces

5.6.1 Spheres

Spheres are defined by a center and a radius.

```
sphere = [x0 y0 z0 R]
```

5.6.1.1 Creation and intersections

createSphere
Creates a sphere passing through 4 points.

intersectLineSphere
Returns the intersection points between a line and a sphere.

intersectPlaneSphere
Returns the intersection circle between a plane and a sphere.

5.6.1.2 Drawing functions

Several functions are provided to draw spheres, or geometries defined over a sphere.

drawSphere
Draws a sphere as a mesh.

drawSphericalEdge
Draws an edge on the surface of a sphere.

drawSphericalTriangle
Draws a triangle on a sphere.

fillSphericalTriangle
Fills a triangle on a sphere.

drawSphericalPolygon
Draws a spherical polygon.

fillSphericalPolygon
Fills a spherical polygon.

sphericalVoronoiDomain
Computes a spherical voronoi domain.

5.6.2 Ellipsoids

Ellipsoids are a generalization of spheres, that are defined by a center, three radius, and three Euler angles (see Section 5.8.1).

```
Elli = [x0 y0 z0 RA RB RC PHI THETA PSI]
```

equivalentEllipsoid
Computes the ellipsoid with the same moments up to the second order as the given set of 3D points (Fig. 5.1). Note that it **does not** correspond to the inertia ellipsoid as defined with mechanical conventions.

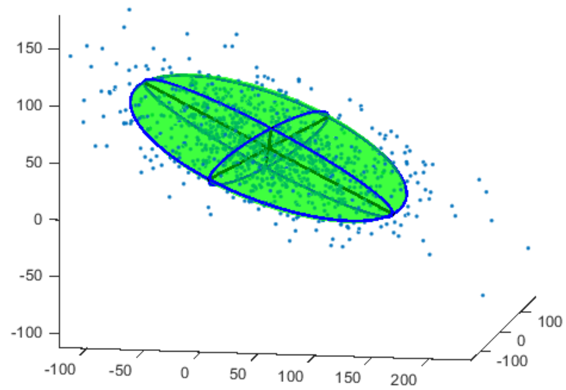


Figure 5.1: *Equivalent ellipsoid of a point cloud*

fitEllipse3d
Fits a 3D ellipse to a set of points.

ellipsoidSurfaceArea
Computes an approximation of the surface area of an ellipsoid from the semi-axis lengths. The approximation formula is given by:

$$S \sim 4\pi \cdot \left(\frac{1}{3} \left(a^p \cdot b^p + a^p \cdot c^p + b^p \cdot c^p \right) \right)^{1/p}$$

with $p = 1.6075$. The resulting error should be less than 1.061%.

oblateSurfaceArea
Approximated surface area of an oblate ellipsoid.

prolateSurfaceArea
Approximated surface area of a prolate ellipsoid.

drawEllipsoid
Draws a 3D ellipsoid.

5.6.3 Cylinders

A cylinder is defined by two end-points and a radius. It is represented as a 1×7 row vector (3 values for each endpoint, and 1 value for the radius).

```
Cylinder = [X1 Y1 Z1 X2 Y2 Z2 R];
```

cylinderSurfaceArea
Computes the surface area of a cylinder.

intersectLineCylinder
Computes the intersection points between a line and a cylinder.

drawCylinder
Draws a cylinder.

drawEllipseCylinder
Draws a cylinder with ellipse cross-section.

5.6.4 Other smooth surfaces

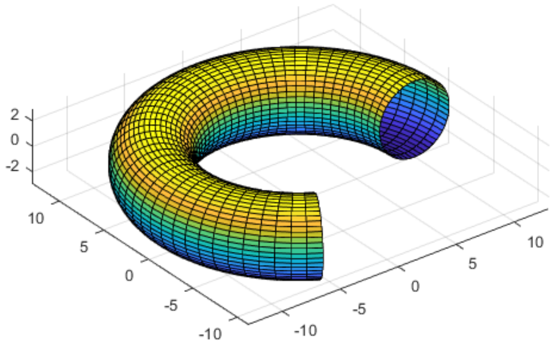


Figure 5.2: *3D revolution surface*

revolutionSurface
Creates a surface of revolution from a planar curve.

surfaceCurvature
Curvature on a surface from angle and principal curvatures.

drawTorus
Draws a torus (3D ring).

drawSurfPatch
Draws a 3D surface patch, with 2 parametrized surfaces.

5.7 3D Transforms

Transforms in 3D space are represented as 4-by-4 matrices in homogeneous coordinates:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{5.1}$$

5.7.1 Basic transforms

createTranslation3d

Creates the 4x4 matrix of a 3D translation.

$$T(\boldsymbol{u}) = \begin{bmatrix} 1 & 0 & 0 & u_x \\ 0 & 1 & 0 & u_y \\ 0 & 0 & 1 & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

createScaling3d

Creates the 4x4 matrix of a 3D scaling.

createRotationOx

Creates the 4x4 matrix of a 3D rotation around x-axis.

$$R_X(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

createRotationOy

Creates the 4x4 matrix of a 3D rotation around y-axis.

$$R_Y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

createRotationOz

Creates the 4x4 matrix of a 3D rotation around z-axis.

$$R_Z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.7.2 Euler Angles and basis transforms

createBasisTransform3d
Computes matrix for transforming a basis into another basis.

eulerAnglesToRotation3d
Converts 3D Euler angles to 3D rotation matrix.

rotation3dToEulerAngles
Extracts Euler angles from a rotation matrix.

createRotation3dLineAngle
Creates rotation around a line by an angle theta.

rotation3dAxisAndAngle
Determines axis and angle of a 3D rotation matrix.

createRotationVector3d
Calculates the rotation between two vectors.

createRotationVectorPoint3d
Calculates the rotation between two vectors.

5.7.3 Utility functions

recenterTransform3d
Changes the fixed point of an affine 3D transform.

composeTransforms3d
Concatenates several space transformations.

5.8 Angles and coordinate systems

5.8.1 3D Angles

Euler Angles are defined as follow:

PHI is the azimut, i.e. the angle of the projection on horizontal plane with the Ox axis, with value beween 0 and 180 degrees.

THETA is the latitude, i.e. the angle with the Oz axis, with value between -90 and +90 degrees.

PSI is the 'roll', i.e. the rotation around the (PHI, THETA) direction, with value in degrees

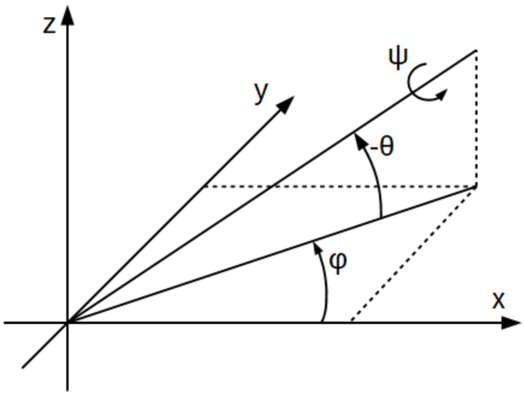


Figure 5.3: *Definition of 3D angles.*

anglePoints3d
Computes angle between three 3D points.

sphericalAngle
Computes angle between points on the sphere.

angleSort3d
Sorts 3D coplanar points according to their angles in plane.

randomAngle3d
Returns a 3D angle uniformly distributed on unit sphere.

5.8.2 Coordinate transforms

sph2cart2
Converts spherical coordinates to cartesian coordinates.

cart2sph2
Converts cartesian coordinates to spherical coordinates.

cart2sph2d
Converts cartesian coordinates to spherical coordinates in degrees.

sph2cart2d
Converts spherical coordinates to cartesian coordinates in degrees.

cart2cyl
Converts cartesian to cylindrical coordinates.

cyl2cart
Converts cylindrical to cartesian coordinates.

5.9 Other drawing functions

drawGrid3d
Draws a 3D grid on the current axis.

drawAxis3d
Draws a coordinate system and an origin.

drawAxisCube
Draws a colored cube representing axis orientation.

drawCube
Draws a 3D centered cube, eventually rotated.

drawCuboid
Draws a 3D cuboid, eventually rotated.

6 Module meshes3d

The meshes3d module provides functions for the manipulation of 3D surface meshes. Meshes can be composed of triangular faces ("tri-mesh"), or have faces with variable number of vertices.

Contents

6.1 Mesh representation

A 3D surface mesh is represented by (at least) two arrays:

vertices a $N_v \times 3$ array of double containing coordinates of the N_v vertices

faces an array containing the vertex indices for each face. For triangular meshes, faces are stored as a $N_f \times 3$ array. For generic meshes with faces with variable vertex number, faces is stored as a cell array, each cell containing the array of vertex indices for corresponding face.

Some functions may require or return additionnal data:

edges an additional array that contains the source and target vertex of each edge

6.2 Display functions

The library includes several functions to quickly display a mesh. Input arguments usually comprise `vertices` and `faces` arrays, but a mesh structure may sometimes be passed as well.

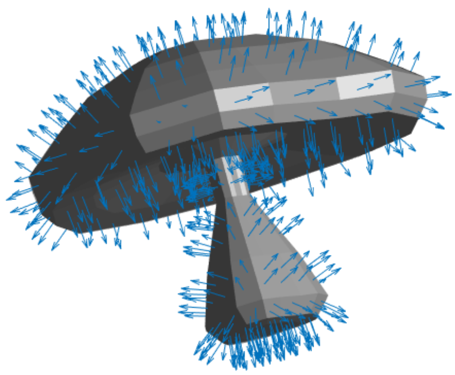


Figure 6.1: *Representation of a 3D polygonal mesh together with the face normals.*

drawMesh

Draws a 3D mesh defined by vertex and face arrays (Fig. 6.1).

fillMeshFaces

Fills the faces of a mesh with the specified colors.

drawFaceNormals

Draws normal vector of each face in a mesh (Fig. 6.1).

6.3 Creation of meshes

The library contains many functions for generating polygonal meshes corresponding to classical polyhedra, such as platonic solids. It also provides facilities for converting from smooth surfaces.

6.3.1 Platonic solids

Several functions allows creation of meshes representing classical polyhedra. The results are typically of the form `[v,f]`, or `[v,e,f]`, where `v` is the array of vertex coordinates, `f` is the array of face vertex indices, and `e` is the array of edge vertex indices.

createCube
Creates a 3D mesh representing the unit cube (Fig. 6.2-a).

createOctahedron
Creates a 3D mesh representing an octahedron (Fig. 6.2-b).

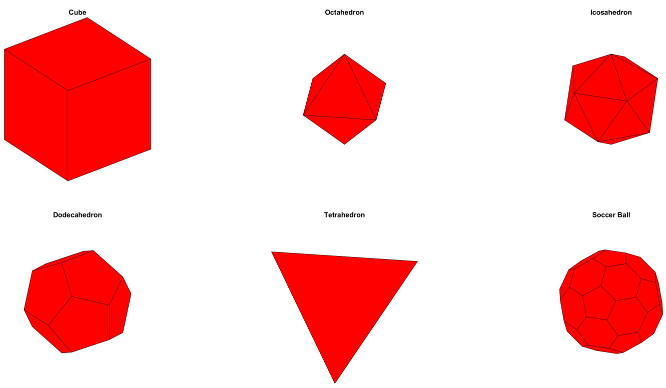


Figure 6.2: *The five platonic solids and a soccer ball represented as 3D meshes.*

createIcosahedron
Creates a 3D mesh representing an Icosahedron (Fig. 6.2-c).

createDodecahedron
Creates a 3D mesh representing a dodecahedron (Fig. 6.2-d).

createTetrahedron
Creates a 3D mesh representing a tetrahedron (Fig. 6.2-e).

6.3.2 Other classical polyhedra

Other classical (non platonic) polyhedra can be easily generated.

createSoccerBall

Creates a 3D mesh representing a soccer ball (Fig. 6.2-f). It can be seen as a truncated icosahedron.

createCubeOctahedron

Creates a 3D mesh representing a cube-octahedron (Fig. 6.3-a).

createTetrakaidecahedron

Creates a 3D mesh representing a tetrakaidecahedron (Fig. 6.3-b). It can be seen as a truncated tetraedra.

createRhombododecahedron

Creates a 3D mesh representing a rhombododecahedron (Fig. 6.3-c). This mesh is composed of twelve identical faces, but vertices do not all have the same number of vertices.

createStellatedMesh

Replaces each face of a mesh by a pyramid.

createDurerPolyhedron

Creates a mesh corresponding to the polyhedron represented in Durer's "Melancholia" (Fig. 6.3-d).

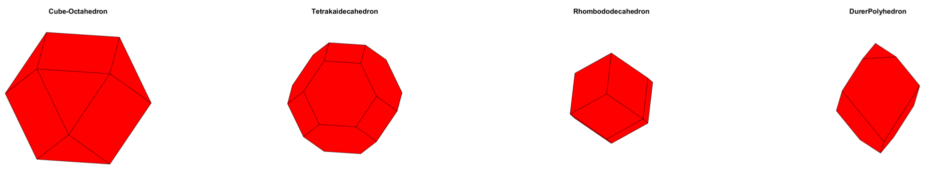


Figure 6.3: *Additional polyhedra that can be generated from the "meshes3d" module.*

6.3.3 Conversion from smooth surface models

It is often convenient to convert a geometrical 3D model (cylinder, ellipsoid...) with known parameters into a discretized version represented by a mesh.

cylinderMesh
Creates a 3D mesh representing a cylinder.

sphereMesh
Creates a 3D mesh representing a sphere.

ellipsoidMesh
Converts a 3D ellipsoid to face-vertex mesh representation.

torusMesh
Creates a 3D mesh representing a torus.

curveToMesh
Creates a mesh surrounding a 3D curve (Fig. 6.4).

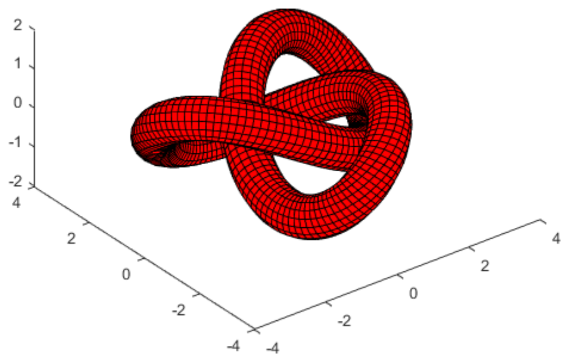


Figure 6.4: *Application of the curveToMesh function*

6.3.4 Other creation functions

boxToMesh
Converts a box into a quad mesh with the same size.

surfToMesh
Converts surface grids into face-vertex mesh.

triangulateCurvePair
Computes triangulation between a pair of 3D open curves.

triangulatePolygonPair
Computes triangulation between a pair of 3D closed curves.

minConvexHull
Returns the unique minimal convex hull of a set of 3D points. It consists in merging the triangular coplanar faces of the convex hull, resulting in a mesh composed of polygonal faces with various numbers of vertices.

createMengerSponge
Creates a cube with an inside cross removed. Can be used to test algorithms on meshes with complex topology.

steinerPolytope
Creates a steiner polytope from a set of vectors. Example (See Fig. 6.5):

```
vecList = [1 0 0; 0 1 0; 0 0 1; 1 1 1];
[v, f] = steinerPolytope(vecList);
figure; drawMesh(v, f);
```

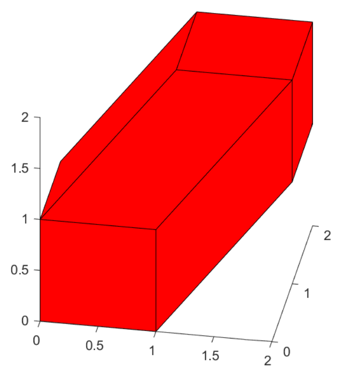


Figure 6.5: *Computation of the Steiner polytope obtained from four 3D vectors.*

6.4 Operations on meshes

Most functions in this section transform a mesh into another mesh, or into another geometric data structure.

6.4.1 Filtering of meshes

Several functions allows for smoothing or simplifying meshes.

smoothMesh

Smoothes mesh by replacing each vertex by the average of its neighbors .

```
[V2, F2] = smoothMesh(V, F);
```

meshVertexClustering

Simplifies a mesh using vertex clustering.

```
[V2, F2] = meshVertexClustering(V, F, SPACING);
```

concatenateMeshes

Concatenates multiple meshes.

```
[V, F] = concatenateMeshes(V1, F1, V2, F2);
```

splitMesh

Returns the connected components of a mesh.

```
meshes = splitMesh(vertices, faces);
```

subdivideMesh

Subdivides each face of the mesh.

```
[v2, f2] = subdivideMesh(v, f, nDivs);
```

triangulateFaces

Converts face array to an array of triangular faces.

```
[v, f] = createCube;
f2 = triangulateFaces(f);
```

mergeCoplanarFaces

Merges coplanar faces of a polyhedral mesh.

```
[v2, f2] = mergeCoplanarFaces(v, f, tol);
```

meshComplement

Reverses the normal of each face in the mesh.

6.4.2 Intersection and clipping

Can identify and select elements of the mesh that intersect other primitives, or that are contained within a region.

intersectLineMesh3d
Intersection points of a 3D line with a mesh.

intersectPlaneMesh
Computes the polygons resulting from plane-mesh intersection.

polyhedronSlice
Intersects a convex polyhedron with a plane.

clipMeshVertices
Clips vertices of a surface mesh and remove outer faces.

clipConvexPolyhedronHP
Clips a convex polyhedron by a plane.

cutMeshByPlane
Cuts a mesh by a plane. Example:

```
% create triangulated mesh, and a plane from origin and normal vector
[v, f] = createSoccerBall; f = triangulateFaces(f);
plane = createPlane([-0.2 0 0], [-1 0 -1]);
% split the different parts of the mesh
[above, inside, below] = cutMeshByPlane(mesh, plane);
% draw the different parts
figure('color','w'); axis equal; hold on; view(3)
drawMesh(above, 'FaceColor', 'r'); drawMesh(inside, 'FaceColor', 'g'); drawMesh(below, 'FaceColor', 'b');
drawPlane3d(plane, 'FaceAlpha',.7)
```

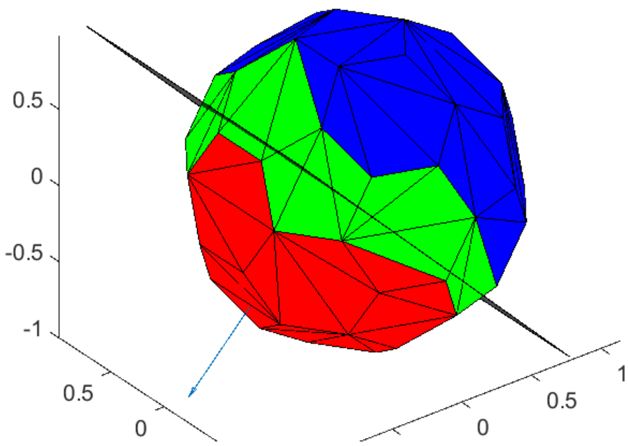


Figure 6.6: *Illustration of the "cutMeshByPlane" function.*

6.4.3 Generic operations

averageMesh
Computes an average mesh from a list of meshes.

6.4.4 Cleanup meshes

Some functions for removing topological inconsistencies and trying to obtain a manifold mesh.

trimMesh
Reduces the memory footprint of a polygonal mesh by removing vertices that are not referenced by any face, and recomputing indices of remaining vertices.

isManifoldMesh
Checks whether the input mesh may be considered as manifold. A mesh is a manifold if all edges are connected to either two or one faces. Border edges should also form a 3D linear ring.

ensureManifoldMesh
Applies several simplification to obtain a manifold mesh.

removeDuplicateFaces
Removes duplicate faces in a face array.

removeMeshEars
Removes vertices that are connected to only one face.

removeInvalidBorderFaces
Removes faces whose edges are connected to 3, 3, and 1 faces.

collapseEdgesWithManyFaces
Removes mesh edges adjacent to more than two faces.

6.4.5 Mesh basic edition

Some low-level functions to modify a mesh.

removeMeshVertices
Removes vertices and associated faces from a mesh.

mergeMeshVertices
Merges two vertices and removes eventual degenerated faces.

removeMeshFaces
Removes faces from a mesh by face indices.

6.5 Measures on meshes

6.5.1 Query functions

Low level functions for investigating topology of meshes.

meshFace
Returns the vertex indices of a face in a mesh.

meshFaceEdges
Computes edge indices of each face.

meshFaceNumber
Returns the number of faces in this mesh.

meshEdges
Computes array of edge vertex indices from face array.

meshEdgeFaces
Computes index of faces adjacent to each edge of a mesh.

trimeshEdgeFaces
Computes index of faces adjacent to each edge of a triangular mesh.

meshFaceAdjacency
Computes adjacency list of face around each face.

meshAdjacencyMatrix
Computes the adjacency matrix of a mesh from set of faces.

checkMeshAdjacentFaces
Checks if adjacent faces of a mesh have similar orientation.

6.5.2 Geometric measures

Several functions allows to measure 3D intrinsic volumes, corresponding to volume, surface area, Euler number, or mean breadth (proportionnal to the integral of mean curvature along mesh). Some functions are dedicated to specific mesh types.

meshSurfaceArea
Surface area of a polyhedral mesh.

trimeshSurfaceArea
Surface area of a triangular mesh.

meshFaceAreas
Surface area of each face of a mesh.

meshVolume
Volume of the space enclosed by a polygonal mesh.

meshEdgeLength
Lengths of edges of a polygonal or polyhedral mesh.

meshDihedralAngles
Dihedral angle at edges of a polyhedal mesh.

meshFacePolygons
Returns the set of polygons that constitutes a mesh.

polyhedronCentroid
Computes the centroid of a 3D convex polyhedron.

tetrahedronVolume
Signed volume of a tetrahedron.

polyhedronNormalAngle
Computes the normal angle at a vertex of a 3D polyhedron.

polyhedronMeanBreadth
Mean breadth of a convex polyhedron.

trimeshMeanBreadth
Mean breadth of a triangular mesh.

6.5.3 Point positions

Describes the relative position of a 3D points with respect to the input mesh.

isPointInMesh
Checks if a point is inside a 3D mesh.

distancePointMesh
Shortest distance between a (3D) point and a triangle mesh.

6.6 Reading and writing meshes

The template of functions for reading (or writing) meshes is readMesh_XXX (or writeMesh_XXX), where XXX corresponds to the format used.

6.6.1 OFF format

readMesh_off
Reads mesh data stored in OFF format.

writeMesh_off
Writes a mesh into a text file in OFF format.

6.6.2 Polygon format

The “Polygon File Format”, or “Stanford triangle format”, is more general and more widely used than the OFF format. Partial support is provided by MatGeom.

readMesh_ply
Reads mesh data stored in PLY (Stanford triangle) format.

writeMesh_ply
Writes a mesh into a text file in PLY format.

Bibliography

Aurenhammer, F. (1991). Voronoi diagram - a study of a fondamental geometric data structure. *ACM Computing surveys*, 23(3):345–405.

Douglas and Peucker (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122.

Ogniewicz, R. L. and Kübler, O. (1995). Hierarchic Voronoi Skeletons. *Pattern Recognition*, 28(3):343 – 359.