Package: SurfaceMesh (via r-universe)

November 18, 2026

Type Package

Title 3D Surface Meshes Based on 'CGAL'

Version 0.1.2

Maintainer Daniel Wollschlaeger <wollschlaeger@uni-mainz.de>

Description Processing of 3D surface meshes based on the 'C++' library 'CGAL' (Computational Geometry Algorithms Library, <https://www.cgal.org/>). Reads and writes common file formats for meshes and polygon soups. Supports mesh repair, remeshing, subdivision, Boolean operations on bounded volumes, outlier removal, surface reconstruction, (optimal) bounding box, bounding sphere, bounding ellipsoid, convex hull, vertex normals, surface distance metrics (such as Hausdorff), volume-overlap similarity measures JSC (Jaccard Similarity Coefficient) and DSC (Dice Similarity Coefficient). Adapts code from packages 'Boov' and 'cgalMeshes' developed by Stephane Laurent.

License GPL-3

URL https://github.com/dwoll/SurfaceMesh/

BugReports https://github.com/dwoll/SurfaceMesh/issues/

Depends R (>= 4.0.0)

Imports Rcpp (>= 1.0.9), rgl

Suggests knitr, rmarkdown, misc3d

LinkingTo BH, Rcpp, RcppCGAL (>= 6.2.1.1), RcppEigen

Encoding UTF-8

LazyData true

SystemRequirements C++17, gmp, mpfr, pandoc (>=1.14)

SystemRequirementsNote 'MPFR' (MP Floating-Point Reliable Library, https://www.mpfr.org/), 'GMP' (GNU Multiple Precision Library, https://gmplib.org/), Pandoc for vignette building

VignetteBuilder knitr, rmarkdown

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Daniel Wollschlaeger [aut, cre], Stephane Laurent [aut], The CGAL Project [cph]

Config/pak/sysreqs libgmp3-dev libmpfr-dev

Repository https://cran.r-universe.dev

Date/Publication 2026-11-17 13:30:02 UTC

RemoteUrl https://github.com/cran/SurfaceMesh

RemoteRef HEAD

RemoteSha 66d9c47fde7582cc96280310f7329a0af059ea4f

Contents

addVertexNormals . . . 4
alphaWrap . . . 5
boolDifference . . . 6
boolIntersection . . . 7
boolUnion . . . 9
dataCyclide . . . 10
dataEntzenStar . . . 11
dataHeart1 . . . 11
dataHeart2 . . . 12
dataHopfTorus . . . 12
dataICN5Deight . . . 13
dataOloid . . . 14
dataOrthoCircle . . . 14
dataPentaPrism . . . 15
dataSeptuaginta . . . 16
dataSolidMobiusStrip . . . 16
dataSphere . . . 17
dataSpiderCage . . . 18
dataToroHelix . . . 18
dataTorus . . . 19
dataTruncIcosahedron . . . 19
dataTubularKnot . . . 20
doesBoundVolume . . . 21
doesSelfIntersect . . . 22
fillBoundaryHoles . . . 23
getArea . . . 24
getAverageSpacing . . . 25
getBoundingBox . . . 26
getBoundingEll . . . 27
getBoundingSphere . . . 28
getCentroid . . . 29
getConvexHull . . . 30

getDCOM 31
getDistance 32
getHausdorff 33
getJSCDSC 34
getNormalsFun 35
getSurfaceDist 36
getVertexNormals 38
getVolume 39
hasGarbage 40
isClosed 40
isOutwardOriented 41
isQuad 42
isTriangle 43
isValid 43
makeMesh 44
makeMeshValid 46
meshEllipsoid 48
meshIsoCuboid 49
meshSphere 50
orientToBoundVolume 50
plotEdges 52
readMesh 53
reconstructAFS 54
reconstructPoisson 55
reconstructSSS 57
remeshACVD 58
remeshDelaunay 60
remeshIsotropic 61
remeshSimplify 63
remeshSmoothAngle 64
remeshSmoothShape 66
remeshSmoothTangentRelax 67
removeOutliers 68
removeSelfIntersections 69
reverseFaceOrientations 71
samplePoints 72
setVertexNormals 73
subdivide 74
toRGL 76
triangulateMesh 77
writeMesh 78

Index **79**

addVertexNormals *Add vertex normals to a mesh*

Description

Add normal vectors to a given 3D surface mesh. Currently, only vertex normals are supported.

Usage

```
addVertexNormals(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh <- makeMesh(dataToroHelix, normals=FALSE)
mesh
mesh_vn <- addVertexNormals(mesh)
mesh_vn

mfrow3d(1, 2)
view3d(0, 30, zoom=0.8)
shade3d(toRGL(mesh), col="gray")
next3d()
view3d(0, 30, zoom=0.8)
shade3d(toRGL(mesh_vn), col="gray")
```

alphaWrap *3D alpha wrapping*

Description

Reconstruction of a 3D surface mesh from a cloud of 3D points by alpha wrapping.

Usage

```
alphaWrap(x, alphaRel, offsetRel, normals = FALSE)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
alphaRel	Relative alpha parameter. The actual alpha parameter (see details) is defined as the length of the diagonal of the bounding box of the point cloud divided by the relative alpha parameter. Increase for a more detailed mesh.
offsetRel	Relative offset. The actual offset parameter (see details) is defined as the length of the diagonal of the bounding box of the point cloud divided by the relative offset parameter. Increase for output that is closer to input mesh.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Alpha_wrap_3/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See also reconstructAFS, reconstructSSS, reconstructPoisson for other surface reconstruction methods.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh        <- dataHeart1
mesh_rgl    <- toRGL(mesh)
mesh_aw     <- alphaWrap(mesh,
                         alphaRel =10,
                         offsetRel=300)
mesh_aw_rgl <- toRGL(mesh_aw)

mfrow3d(1, 2)
view3d(0, -90, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(0, -90, zoom=0.7)
shade3d(mesh_aw_rgl, col="gray")
```

---

boolDifference *Difference between two meshes*

Description

Compute the difference between two 3D surface meshes (`mesh1 - mesh2`).

Usage

```
boolDifference(
  mesh1,
  mesh2,
  repairSoup = TRUE,
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

`mesh1`	A mesh, either given as a `mesh3d` object from package **rgl**, or as a `CGALmesh` object, i.e., the output of `makeMesh`.
`mesh2`	A mesh, either given as a `mesh3d` object from package **rgl**, or as a `CGALmesh` object, i.e., the output of `makeMesh`.
`repairSoup`	Boolean. Clean the input meshes (merging duplicated vertices, duplicated faces, removing isolated vertices)? Set to `FALSE` if you know the meshes are clean to gain some speed.
`normals`	Boolean. Return vertex normals of the output mesh?
`verbose`	Boolean. Print out messages about mesh processing?

Details

See https://doc.cgal.org/latest/PMP_Boolean_operations/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See also boolIntersection and boolUnion for other Boolean mesh operations.

Examples

```
library(SurfaceMesh)
library(rgl)
# mesh one: a cube
mesh1_rgl <- cube3d() # (from the rgl package)

# mesh two: another cube
mesh2_rgl <- translate3d(cube3d(), 1, 1, 0)

# compute the difference
mesh_d <- boolDifference(mesh1_rgl, mesh2_rgl)

# plot
mesh_d_rgl <- toRGL(mesh_d)
open3d(windowRect=c(50, 50, 562, 562))
shade3d(mesh1_rgl,  color="yellow", alpha=0.2)
shade3d(mesh2_rgl,  color="cyan",   alpha=0.2)
shade3d(mesh_d_rgl, color="red")
plotEdges(mesh_d[["vertices"]],
          edges            =mesh_d[["edgesExterior"]],
          edgesAsTubes     =TRUE,
          verticesAsSpheres=TRUE)
```

---

boolIntersection *Intersection of meshes*

Description

Compute the intersection of the given 3D surface meshes.

Usage

```
boolIntersection(x, repairSoup = TRUE, normals = FALSE, verbose = FALSE)
```

Arguments

x	A list of meshes, each being either a mesh3d object from package **rgl**, or a CGALmesh object, i.e., the output of makeMesh.
repairSoup	Boolean. Clean the meshes (merging duplicated vertices, duplicated faces, removing isolated vertices)? Set to FALSE if you know the meshes are clean, to gain some speed.
normals	Boolean. Return vertex normals of the output mesh?
verbose	Boolean. Print out messages about mesh processing?

Details

See https://doc.cgal.org/latest/PMP_Boolean_operations/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See also boolUnion and boolDifference for other Boolean mesh operations.

Examples

```
library(SurfaceMesh)
library(rgl)
# mesh one: truncated icosahedron; we triangulate it
mesh1 <- makeMesh(dataTruncIcosahedron,
                  triangulate=TRUE,
                  normals    = FALSE)

# mesh two: a cube
mesh2_rgl <- translate3d(cube3d(), 2, 0, 0)
mesh2     <- makeMesh(mesh2_rgl,
                      triangulate=TRUE,
                      normals    =FALSE)

# compute the intersection
mesh_i <- boolIntersection(list(mesh1, mesh2))

# plot
mesh1_rgl  <- toRGL(mesh1)
mesh_i_rgl <- toRGL(mesh_i)
open3d(windowRect=c(50, 50, 562, 562))
shade3d(mesh1_rgl,  color="yellow", alpha=0.2)
shade3d(mesh2_rgl,  color="cyan",   alpha=0.2)
shade3d(mesh_i_rgl, color="red")
```

```
plotEdges(mesh_i[["vertices"]],
          edges            =mesh_i[["edgesExterior"]],
          edgesAsTubes     =FALSE,
          lwd              =3,
          verticesAsSpheres=FALSE)
```

`boolUnion` *Union of meshes*

Description

Compute the union of the given meshes.

Usage

```
boolUnion(x, repairSoup = TRUE, normals = FALSE, verbose = FALSE)
```

Arguments

`x`	A list of meshes, each being either a `mesh3d` object from package **rgl**, or a `CGALmesh` object, i.e., the output of `makeMesh`.
`repairSoup`	Boolean. Clean the input meshes (merging duplicated vertices, duplicated faces, removing isolated vertices)? Set to `FALSE` if you know the meshes are clean to gain some speed.
`normals`	Boolean. Return vertex normals of the output mesh?
`verbose`	Boolean. Print out messages about mesh processing?

Details

See https://doc.cgal.org/latest/PMP_Boolean_operations/ for details.

Value

A `CGALmesh` object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See also `boolIntersection` and `boolDifference` for other Boolean mesh operations.

Examples

```
library(SurfaceMesh)
library(rgl)
# mesh one: a cube
mesh1_rgl <- cube3d() # (from the rgl package)

# mesh two: another cube
mesh2_rgl <- translate3d(cube3d(), 1, 1, 1)

# compute the union
mesh_u <- boolUnion(list(mesh1_rgl, mesh2_rgl))

# plot
mesh_u_rgl <- toRGL(mesh_u)
open3d(windowRect=c(50, 50, 562, 562))
shade3d(mesh_u_rgl, color="red")
plotEdges(mesh_u[["vertices"]],
          edges            =mesh_u[["edgesExterior"]],
          edgesAsTubes     =TRUE,
          verticesAsSpheres=TRUE)
```

---

dataCyclide *Mesh of a Dupin cyclide*

Description

An object of class `mesh3d` from package **rgl** representing a Dupin cyclide.

Usage

```
dataCyclide
```

Format

A list with components `vb`, `it`.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "Cyclide.R",           package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
```

```
view3d(0, -35, zoom=0.8)
shade3d(dataCyclide, color="orangered")
```

---

dataEntzenStar *Mesh of an Entzensberger star*

Description

An object of class mesh3d from package **rgl** representing an Entzensberger star.

Usage

```
dataEntzenStar
```

Format

A list with components vb, it.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "EntzenStar.R",        package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-55, -20, zoom=0.8)
shade3d(dataEntzenStar, color="steelblue2")
```

---

dataHeart1 *Mesh of a human heart from a CT scan*

Description

An object of class CGALmesh representing a human heart based on a CT scan.

Usage

```
dataHeart1
```

Format

A CGALmesh object.

Examples

```
library(SurfaceMesh)
library(rgl)
view3d(-5, -90, zoom=0.8)
shade3d(toRGL(dataHeart1), color="tomato")
```

dataHeart2 *Mesh of a human heart from a CT scan*

Description

An object of class CGALmesh representing a human heart based on a CT scan.

Usage

```
dataHeart2
```

Format

A CGALmesh object.

Examples

```
library(SurfaceMesh)
library(rgl)
view3d(-5, -90, zoom=0.8)
shade3d(toRGL(dataHeart2), color="gray")
```

dataHopfTorus *Mesh of a Hopf torus*

Description

A CGALmesh object, i.e., the output of makeMesh representing a Hopf torus.

Usage

```
dataHopfTorus
```

Format

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "HopfTorus.R",         package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(35, -20, zoom=0.8)
shade3d(toRGL(dataHopfTorus), color="darkred")
```

`dataICN5Deight` *Mesh of an ICN5D eight*

Description

An object of class `mesh3d` from package **rgl** representing an ICN5D eight, a shape presented by user ICN5D on the Higher Space Forum.

Usage

```
dataICN5Deight
```

Format

A list with components `vb`, `it`, and `normals`.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "ICN5Deight.R",        package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(35, -20, zoom=0.8)
shade3d(dataICN5Deight, color="violetred")
```

dataOloid *Mesh of an oloid*

Description

A CGALmesh object, i.e., the output of makeMesh representing an oloid.

Usage

```
dataOloid
```

Format

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "Oloid.R",             package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(20, 15, zoom=0.8)
shade3d(toRGL(dataOloid), color="darkviolet")
```

dataOrthoCircle *Mesh of an ortho circle*

Description

An object of class mesh3d from package **rgl** representing an Ortho Circle.

Usage

```
dataOrthoCircle
```

Format

A list with components vb, it, and normals.

Details

https://paulbourke.net/geometry/orthocircle/

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "OrthoCircle.R",       package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-15, -10, zoom=0.8)
shade3d(dataOrthoCircle, color="darkolivegreen4")
```

---

dataPentaPrism *Mesh of a pentagrammic prism*

Description

A list representing a pentagrammic prism, giving the vertices and the faces; it has 20 vertices, 10 triangular faces, 10 rectangular faces and two pentagonal faces.

Usage

```
dataPentaPrism
```

Format

A list with components `vertices`, `faces`.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "PentaPrism.R",        package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
mPentaPrism <- makeMesh(dataPentaPrism, triangulate=TRUE)
view3d(-15, -15, zoom=0.8)
shade3d(toRGL(mPentaPrism), color="forestgreen")
```

dataSeptuaginta *Mesh of a septuaginta*

Description

An object of class mesh3d from package **rgl** representing a septuaginta, i.e. a polyhedron with 72 faces like a sphere based on the drawing by Leonardo da Vinci.

Usage

```
dataSeptuaginta
```

Format

A list with components vb, it, and ib.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "Septuaginta.R",       package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-10, -20, zoom=0.8)
shade3d(dataSeptuaginta, color="orangered")
```

dataSolidMobiusStrip *Mesh of a solid Mobius strip*

Description

An object of class mesh3d from package **rgl**, representing a solid Mobius Strip.

Usage

```
dataSolidMobiusStrip
```

Format

A list with components vb, it, and normals.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "SolidMobiusStrip.R",  package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-10, -20, zoom=0.8)
shade3d(dataSolidMobiusStrip, color="darkred")
```

---

dataSphere *Mesh of a sphere*

Description

An object of class `mesh3d` from package **rgl** representing a sphere

Usage

```
dataSphere
```

Format

A list with components `vb`, `it`, and `normals`.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "Sphere.R",            package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(15, -20, zoom=0.8)
shade3d(dataSphere, color="darkgoldenrod")
```

dataSpiderCage *Mesh of a spider cage*

Description

An object of class mesh3d from package **rgl** representing a Spider Cage.

Usage

```
dataSpiderCage
```

Format

A list with components vb, it, and normals.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "SpiderCage.R",        package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-25, -45, zoom=0.8)
shade3d(dataSpiderCage, color="mediumseagreen")
```

dataToroHelix *Mesh of a toroidal helix*

Description

An object of class mesh3d from package **rgl** representing a toroidal helix.

Usage

```
dataToroHelix
```

Format

A list with components vb, it.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "ToroHelix.R",         package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(15, -20, zoom=0.8)
shade3d(dataToroHelix, color="violetred")
```

---

dataTorus *Mesh of a torus*

Description

An object of class `mesh3d` from package **rgl** representing a torus.

Usage

```
dataTorus
```

Format

A list with components `vb`, `it`.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "Torus.R",             package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(15, -20, zoom=0.8)
shade3d(dataTorus, color="green")
```

---

dataTruncIcosahedron *Mesh of the truncated icosahedron*

Description

A list giving the vertices and the faces of a truncated icosahedron. There are some hexagonal faces and some pentagonal faces.

Usage

```
dataTruncIcosahedron
```

Format

A list with two components: `vertices` and `faces`.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "TruncIcosahedron.R",  package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
mTruncIcosahedron <- makeMesh(dataTruncIcosahedron, triangulate=TRUE)
view3d(-15, -10, zoom=0.8)
shade3d(toRGL(mTruncIcosahedron), color="wheat2")
```

---

dataTubularKnot *Mesh of a tubular knot*

Description

An object of class `mesh3d` from package **rgl** representing a tubular knot.

Usage

```
dataTubularKnot
```

Format

A list with components `vb`, `it`.

Author(s)

Originally developed by Stephane Laurent.

Examples

```
## The following files were used to generate this mesh.
system.file("data-raw", "TubularKnot.R",       package="SurfaceMesh")
system.file("data-raw", "00_GenerateMeshes.R", package="SurfaceMesh")

library(SurfaceMesh)
library(rgl)
view3d(-15, -15, zoom=0.8)
shade3d(dataTubularKnot, color="royalblue3")
```

doesBoundVolume *Does mesh bound a volume?*

Description

Does given 3D surface mesh bound a volume?

Usage

```
doesBoundVolume(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

TRUE or FALSE.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See orientToBoundVolume for orienting the mesh to bound a volume, and getVolume for the volume bounded by the mesh.

Examples

```
library(SurfaceMesh)
vertices <- rbind(
    c(0, 0, 0),
    c(2, 2, 0),
    c(2, 0, 2),
    c(0, 2, 2),
    c(3, 3, 3),
    c(5, 5, 3),
    c(5, 3, 5),
    c(3, 5, 5))

faces <- rbind(
    c(3, 2, 1),
    c(3, 4, 2),
    c(1, 2, 4),
    c(4, 3, 1),
    c(5, 6, 7),
    c(6, 8, 7),
    c(8, 6, 5),
    c(5, 7, 8))
```

```
mesh <- makeMesh(vertices, faces=faces)
doesBoundVolume(mesh)    # FALSE
getVolume(mesh)          # NA

mesh_bv <- orientToBoundVolume(mesh)
doesBoundVolume(mesh_bv) # TRUE
getVolume(mesh_bv)
```

`doesSelfIntersect` *Does mesh self intersect?*

Description

Does the given 3D surface mesh self intersect?

Usage

```
doesSelfIntersect(x)
```

Arguments

x	A `CGALmesh` object, i.e., the output of `makeMesh`.

Value

`TRUE` or `FALSE`.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `removeSelfIntersections` for removing self-intersections.

Examples

```
library(SurfaceMesh)
verts1 <- vapply(c(0, 2, 4, 1, 3),
                 function(i) { c(cos(2*i*pi/5), sin(2*i*pi/5), 0.3) },
                 numeric(3L))

verts2 <- vapply(c(0, 2, 4, 1, 3),
                 function(i) { c(cos(2*i*pi/5), sin(2*i*pi/5), -0.3) },
                 numeric(3L))

vertices    <- t(cbind(verts1, verts2))
pentagramms <- rbind(1L:5L, 6L:10L)
```

```
rectangles <- rbind(
    c(1L, 2L, 7L, 6L),
    c(2L, 3L, 8L, 7L),
    c(3L, 4L, 9L, 8L),
    c(4L, 5L, 10L, 9L),
    c(5L, 1L, 6L, 10L))

faces <- list(
    pentagramms[1L, ],
    pentagramms[2L, ],
    rectangles[1L, ],
    rectangles[2L, ],
    rectangles[3L, ],
    rectangles[4L, ],
    rectangles[5L, ])

mesh <- makeMesh(vertices, faces)
doesSelfIntersect(mesh)
```

`fillBoundaryHoles` *Fill boundary holes*

Description

Try to fill boundary holes in a 3D surface mesh.

Usage

```
fillBoundaryHoles(
  x,
  fairHole = TRUE,
  maxNumHoles = 10L,
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`.
`fairHole`	Boolean. Use CGAL `triangulate_refine_and_fair_hole()` (`TRUE`) or `triangulate_and_refine_ho` (`FALSE`)?
`maxNumHoles`	`integer`. Maximum number of holes to be filled.
`normals`	Boolean. Return vertex normals?
`verbose`	Boolean. Print out messages about mesh processing?

Details

See https://www.cgal.org/2025/06/13/autorefine-and-snap/ and https://doc.cgal.org/latest/PMP_Mesh_repair/ for details. If faces are not already triangle, the mesh is first triangulated.

Value

CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See removeSelfIntersections for removing self-intersections.

Examples

```
library(SurfaceMesh)
library(rgl)
f_mesh <- system.file("extdata", "dataCornerHole.off", package="SurfaceMesh")
mesh   <- makeMesh(f_mesh, triangulate=TRUE, verbose=TRUE)
mesh_rgl  <- toRGL(mesh)
mesh_fill <- fillBoundaryHoles(mesh, verbose=TRUE)
mesh_fill_rgl <- toRGL(mesh_fill)

mfrow3d(1, 2)
view3d(70, 25, zoom=0.8)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(70, 25, zoom=0.8)
shade3d(mesh_fill_rgl, col="gray")
```

---

getArea *Get mesh area*

Description

Get the surface area of a given 3D surface mesh.

Usage

```
getArea(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

numeric. The mesh area.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See getVolume for the volume bounded by the mesh.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
getArea(mesh)
```

---

getAverageSpacing *Compute average spacing of input points*

Description

Computes the average spacing of all input points to their nNeighbors nearest neighbor points. This provides an order of a point set density, useful in a surface reconstruction pipeline.

Usage

```
getAverageSpacing(x, nNeighbors = 6L)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 3 points.
nNeighbors	integer. How many neighboring points to consider for distance calculation.

Details

See https://doc.cgal.org/latest/Point_set_processing_3/ for details.

Value

The average spacing.

See Also

removeOutliers, reconstructPoisson, reconstructSSS, alphaWrap

Examples

```
library(SurfaceMesh)
library(rgl)

pts <- matrix(rnorm(120L), ncol=3)
getAverageSpacing(pts, nNeighbors=6L)
```

`getBoundingBox` *Get bounding box (axis-parallel or oriented)*

Description

Get the axis-parallel or optimal (oriented) bounding box of a 3D surface mesh.

Usage

```
getBoundingBox(
  x,
  oriented = FALSE,
  out = c("CGALmesh", "rgl", "Points"),
  triangulate = FALSE,
  normals = FALSE
)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`, a `mesh3d` object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
`oriented`	Boolean. Get the optimal (oriented) bounding box?
`out`	`character`. Type of return value. One of `"Points"` for a matrix giving the corner points, `"CGALmesh"` for a `CGALmesh` object, and `"rgl"` for a `mesh3d` object from package **rgl**.
`triangulate`	Boolean. Triangulate the faces of the bounding box?
`normals`	Boolean. Return vertex normals?

Value

A `CGALmesh` object, a `mesh3d` object, or a matrix with the corner points as rows.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See getConvexHull for the convex hull, getBoundingSphere for the bounding sphere, getBoundingEll for the bounding ellipse.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh1     <- makeMesh(dataTubularKnot)
mesh1_rgl <- toRGL(mesh1)
bb1_rgl   <- getBoundingBox(mesh1[["vertices"]], out="rgl")
open3d(windowRect=50 + c(0, 0, 400, 400))
view3d(0, 30, zoom=0.9)
shade3d(mesh1_rgl, color="seashell")
wire3d(bb1_rgl)

mesh2     <- dataHeart1
mesh2_rgl <- toRGL(mesh2)
bb2_rgl   <- getBoundingBox(mesh2[["vertices"]], oriented=TRUE, out="rgl")
open3d(windowRect=50 + c(0, 0, 400, 400))
shade3d(mesh2_rgl, color="seashell")
wire3d(bb2_rgl)
```

---

getBoundingEll *Bounding ellipsoid of a set of 3D points*

Description

Get the approximate bounding ellipsoid of a given set of 3D points.

Usage

```
getBoundingEll(
  x,
  out = c("CGALmesh", "rgl", "CtrRadDir"),
  nIter = 3L,
  eps = 0.01,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.

out	character. Type of return value. One of "CtrRadDir" for a list giving the sphere's center, semi-axis lengths, and semi-axis directions, "CGALmesh" for a CGALmesh object, and "rgl" for a mesh3d object from package **rgl**.
nIter	integer. Number of iterations (the mesh is obtained by iteratively subdividing the faces of an icosahedron).
eps	numeric. For the approximation ratio 1 + eps. See details.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Bounding_volumes/ for details.

Value

A list with components "center", "lengths", and "directions"; a CGALmesh object; or a mesh3d object - depending on argument out.

See Also

See getBoundingBox for the bounding box, getBoundingSphere for the bounding sphere, getConvexHull for the convex hull.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh     <- makeMesh(dataHeart1)
mesh_rgl <- toRGL(mesh)
bell_rgl <- getBoundingEll(mesh, out="rgl", eps=0.01)

shade3d(mesh_rgl, color="lightgreen")
wire3d(bell_rgl)
```

---

getBoundingSphere *Bounding sphere of a set of 3D points*

Description

Get the bounding sphere of a given set of 3D points.

Usage

```
getBoundingSphere(
  x,
  out = c("CGALmesh", "rgl", "CtrRad"),
  nIter = 3L,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
out	character. Type of return value. One of "CtrRad" for a list giving the sphere's center and radius, "CGALmesh" for a CGALmesh object, and "rgl" for a mesh3d object from package **rgl**.
nIter	integer. Number of iterations (the mesh is obtained by iteratively subdividing the faces of an icosahedron)
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Bounding_volumes/ for details.

Value

A list with components "center" and "radius", a CGALmesh object; or a mesh3d object - depending on argument out.

See Also

See getBoundingBox for the bounding box, getBoundingEll for the bounding ellipse, getConvexHull for the convex hull.

Examples

```
library(SurfaceMesh)
library(rgl)
bs_rgl <- getBoundingSphere(dataToroHelix, out="rgl")
shade3d(dataToroHelix, color="plum2")
wire3d(bs_rgl)
```

---

getCentroid *Get mesh centroid (center of mass)*

Description

Get the centroid (center of mass) of a given 3D surface mesh.

Usage

```
getCentroid(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be triangle or must be able to be made triangle.

Value

numeric. 3-vector with the coordinates of the mesh centroid.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
getCentroid(mesh)
```

---

getConvexHull *Convex hull of a set of 3D points*

Description

Get the convex hull of a given set of 3D points using the quickhull algorithm.

Usage

```
getConvexHull(x, normals = FALSE)
```

Arguments

x	numeric matrix with 3 columns with one point per row.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Convex_hull_3/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See getBoundingBox for the bounding box, getBoundingSphere for the bounding sphere, getBoundingEll for the bounding ellipse.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh     <- makeMesh(dataHopfTorus)
mesh_rgl <- toRGL(mesh)
hull     <- getConvexHull(mesh[["vertices"]])
hull_rgl <- toRGL(hull)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
wire3d(mesh_rgl)
next3d()
wire3d(hull_rgl)
```

---

getDCOM *Euclidean distance between center of mass of two meshes*

Description

Calculates the Euclidean distance between the two respective centers of mass of two 3D surface meshes.

Usage

```
getDCOM(mesh1, mesh2)
```

Arguments

mesh1	A CGALmesh object, i.e., the output of makeMesh.
mesh2	A CGALmesh object, i.e., the output of makeMesh.

Value

The Euclidean distance between the two respective centers of mass.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See getCentroid for calculating the center of mass.

Examples

```
library(SurfaceMesh)
getDCOM(dataHeart1, dataHeart2)
```

`getDistance` *Get distance from points to a mesh*

Description

Calculate the Euclidean distance of a set of points to a given 3D surface mesh.

Usage

```
getDistance(x, points)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`. The mesh must be triangle or must be able to be made triangle.
`points`	`numeric` matrix with 3 columns with one point per row.

Value

`numeric` vector: The distance of each point in `points` to the mesh `x`.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
mesh   <- makeMesh(dataPentaPrism, triangulate=TRUE)
points <- matrix(2*runif(3*4), ncol=3)
getDistance(mesh, points)
```

getHausdorff *Hausdorff distance between two meshes (full or quantile)*

Description

Hausdorff distance between two 3D surface meshes. The approximate distance, the distance estimate with a given error bound, or the quantile Hausdorff distance, e.g., HD95.

Usage

```
getHausdorff(mesh1, mesh2, symmetric = TRUE, errBound, p, ...)
```

Arguments

mesh1	A CGALmesh object, i.e., the output of makeMesh.
mesh2	A CGALmesh object, i.e., the output of makeMesh.
symmetric	Boolean. Report the symmetric Hausdorff distance?
errBound	A positive number. If provided, upper bound on the error of the Hausdorff distance estimate. If missing, and p is also missing, the approximate distance is returned.
p	A number in $[0, 1]$. The quantile probability, defaults to 0.95. If provided and errBound is missing, the quantile Hausdorff distance is returned.
...	Options for random sampling passed to getSurfaceDist. See there fore details. Relevant for the approximate and quantile distance.

Details

The quantile Hausdorff distance approximates the Hausdorff distance using a quantile (often the 95th percentile, i.e. "HD95") of the distances from a random sample of points on one mesh to the other mesh, instead of their maximum. This is less sensitive to small, isolated outlying regions than the full Hausdorff distance.

See https://metrics-reloaded.dkfz.de/metric-library/hd and https://metrics-reloaded.dkfz.de/metric-library/xhd for more information on the (quantile) Hausdorff distance. See https://doc.cgal.org/latest/Polygon_mesh_processing/index.html#PMPDistance for implementation details.

Value

A number. For the apprixmate and quantile distance, the algorithm uses random sampling, and thus the result can vary.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `getSurfaceDist` for other surface distance metrics, and `getJSCDSC` for volume-overlap based mesh similarity metrics (JSC, DSC).

Examples

```
library(SurfaceMesh)
## approximate symmetric Hausdorff distance
getHausdorff(dataHeart1, dataHeart2, symmetric=TRUE)

## estimate with error bound
getHausdorff(dataHeart1, dataHeart2, symmetric=TRUE, errBound=0.01)

## quantile Hausdorff distance
getHausdorff(dataHeart1, dataHeart2, symmetric=TRUE, p=0.95)
```

---

getJSCDSC *Jaccard Similarity Coefficient and Dice Similarity Coefficient*

Description

Compute the Jaccard Similarity Coefficient (JSC, also known as Intersection over Union, IoU) and the Dice Similarity Coefficient (DSC) for the respective volumes defined by two 3D surface meshes.

Usage

```
getJSCDSC(mesh1, mesh2, repairSoup = TRUE, verbose = FALSE)
```

Arguments

`mesh1`	Either a `mesh3d` object from package **rgl**, or a `CGALmesh` object, i.e., the output of `makeMesh`.
`mesh2`	Either a `mesh3d` object from package **rgl**, or a `CGALmesh` object, i.e., the output of `makeMesh`.
`repairSoup`	Boolean. Clean the input meshes (merging duplicated vertices, duplicated faces, removing isolated vertices)? Set to `FALSE` if you know the meshes are clean to gain some speed.
`verbose`	Boolean. Print out messages about mesh processing?

Details

See https://metrics-reloaded.dkfz.de/metric-library/dsc and https://metrics-reloaded.dkfz.de/metric-library/intersection_over_union for details.

Value

A `list` with five components: `"Vol1"`, `"Vol2"`, `"VolI"`, `"VolU"`, `"JSC"` (IoU) and `"DSC"`.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See also boolUnion, boolIntersection, and boolDifference for Boolean mesh operations.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh1_rgl <- cube3d()    # (from the rgl package)
mesh2_rgl <- translate3d(cube3d(), 1, 1, 1)
getJSCDSC(mesh1_rgl, mesh2_rgl)
```

---

getNormalsFun *Function to generate normals for a point cloud*

Description

Returns a function which estimates normals for a 3D point cloud.

Usage

```
getNormalsFun(x, method = c("PCA", "Jet"))
```

Arguments

x	integer. Number of neighbors used to estimate the normals.
method	One of "PCA" to estimate the normal direction at each point by linear least squares fitting of a plane over its nearest neighbors, or "Jet" to estimate the normal direction at each point by fitting a jet surface over its nearest neighbors. See details.

Details

See https://doc.cgal.org/latest/Point_set_processing_3/ for details. The getNormalsFun function is intended to be used in the reconstructPoisson function. If you want to use it for another purpose, be careful because the function it returns does not check the matrix it takes as argument.

Value

A function which takes one argument: a numeric matrix with three columns, each row represents a point, and the function returns a matrix of the same size as the input matrix, with each row giving one unit normal per point.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh     <- dataHeart1
mesh_rgl <- toRGL(mesh)
fun      <- getNormalsFun(6)
mesh_psr <- reconstructPoisson(mesh[["vertices"]],
                               normalsFun=fun,
                               smAngle=10,
                               smRadius=1.5,
                               smDistance=0.3)

mesh_psr_rgl <- toRGL(mesh_psr)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
wire3d(mesh_rgl)
next3d()
wire3d(mesh_psr_rgl)
```

---

getSurfaceDist *Several distance metrics between two meshes*

Description

Quantile Hausdorff distance (HD), average symmetric surface distance (ASSD), and root mean squared error (RMSE) for the surface distance between two 3D surface meshes.

Usage

```
getSurfaceDist(
  mesh1,
  mesh2,
  returnDists = FALSE,
  symmetric = TRUE,
  p = 0.95,
  method = c("random", "grid", "mc"),
  sampleVerts = TRUE,
  sampleEdges = TRUE,
  sampleFaces = TRUE,
  gridSpacing = NULL,
  ptsOnEdges = NULL,
  ptsOnFaces = NULL,
```

```
  ptsPerDist = NULL,
  ptsPerEdge = NULL,
  ptsPerArea = NULL,
  ptsPerFace = NULL
)
```

Arguments

mesh1	A CGALmesh object, i.e., the output of makeMesh.
mesh2	A CGALmesh object, i.e., the output of makeMesh.
returnDists	Boolean. Return the sampled distances from mesh 1 to 2, and from 2 to 1?
symmetric	Boolean. Report the symmetric quantile Hausdorff distance?
p	A number in $[0, 1]$. The quantile probability, defaults to 0.95.
method	character. "random" for random uniform sampling, "grid" for grid sampling, "mc" for Monte Carlo sampling. See details.
sampleVerts	Boolean. Do sample vertices?
sampleEdges	Boolean. Do sample edges?
sampleFaces	Boolean. Do sample faces?
gridSpacing	numeric. The grid spacing for method="grid".
ptsOnEdges	integer. For the random sampling method as the number of points to pick exclusively on edges. If missing, the number of edges is used.
ptsOnFaces	integer. For the random sampling method as the number of points to pick on the surface. If missing, the number of vertices is used.
ptsPerDist	numeric. Points per distance unit. Used for method="random" and "mc" to respectively determine the total number of points on edges and the number of points per edge.
ptsPerEdge	integer. Points per edge. Used for method="mc" as the number of points per edge to pick.
ptsPerArea	numeric. Points per area unit. Used for method="random" and "mc" to respectively determine the total number of points inside faces and the number of points per face.
ptsPerFace	integer. Points per face. Used for method="mc" as the number of points per face to pick.

Details

See getHausdorff for details on the quantile Hausdorff distance, https://metrics-reloaded.dkfz.de/metric-library/assd for ASSD. Inspired by http://vcglib.net/metro.html. For details on sampling options, see https://doc.cgal.org/latest/Polygon_mesh_processing/group__PMP__distance__grp.html.

Value

A list with components "HDq" (quantile Hausdorff distance), "ASSD" (average symmetric surface distance), "RMSE" (root mean squared error). For returnDists=TRUE also "dists12" and "dists21". The algorithm uses random sampling, so the result can vary.

See Also

See getHausdorff for the (approximate) Hausdorff distance, and getJSCDSC for volume-overlap based mesh similarity metrics (JSC, DSC).

Examples

```
library(SurfaceMesh)
getSurfaceDist(dataHeart1, dataHeart2, p=0.95, ptsOnFaces=1000L)
```

---

getVertexNormals *Get vertex normals of a mesh*

Description

Get the vertex normals of a 3D surface mesh.

Usage

```
getVertexNormals(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Details

Note that this function is not really necessary as the vertex normals are simply stored in component "normals" of the CGALmesh object on the R side. However, importing vertex normals to the C++ / CGAL side, and then exporting them to the R side is a test case for working with CGAL property maps that may be useful in the future.

Value

The numeric matrix of normals attached to the vertices of the mesh. NULL if there are no vertex normals.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

setVertexNormals

Examples

```
library(SurfaceMesh)
mesh <- makeMeshValid(dataSphere, normals=TRUE)
vn   <- getVertexNormals(mesh)
head(vn)
```

---

getVolume *Get mesh volume*

Description

Get the volume bounded by a closed 3D surface mesh.

Usage

```
getVolume(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

numeric. The mesh volume - if mesh bounds a volume.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See getArea for the mesh area.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
getVolume(mesh)
```

hasGarbage *Does mesh have garbage?*

Description

Check whether the given 3D surface mesh has garbage, i.e., there is a mismatch between faces and vertices.

Usage

```
hasGarbage(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

TRUE or FALSE.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See isValid for checking if the mesh is valid.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism)
hasGarbage(mesh)
```

isClosed *Is mesh closed?*

Description

Check whether the given 3D surface mesh is closed (watertight).

Usage

```
isClosed(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

TRUE or FALSE.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See fillBoundaryHoles for a function to fill holes.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
isClosed(mesh)
```

---

isOutwardOriented *Is mesh outward oriented?*

Description

Check whether the given 3D surface mesh has a positive (outward) orientation.

Usage

```
isOutwardOriented(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

TRUE or FALSE.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

reverseFaceOrientations, orientToBoundVolume.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
isOutwardOriented(mesh)
```

`isQuad` *Is mesh a quad mesh?*

Description

Check whether the given 3D surface mesh is a quad mesh.

Usage

```
isQuad(x)
```

Arguments

`x`	A `list` with components `vertices` and `faces`, e.g., a `CGALmesh` object, i.e., the output of `makeMesh`.

Value

`TRUE` or `FALSE`.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `isTriangle` to check whether mesh is a triangle mesh.

Examples

```
library(SurfaceMesh)
f_mesh <- system.file("extdata", "dataCorner.off", package="SurfaceMesh")
mesh   <- makeMesh(f_mesh)
isQuad(mesh)
```

isTriangle *Is mesh a triangle mesh?*

Description

Check whether the given 3D surface mesh is a triangle mesh.

Usage

```
isTriangle(x)
```

Arguments

x	A `list` with components `vertices` and `faces`, e.g., a `CGALmesh` object, i.e., the output of `makeMesh`.

Value

`TRUE` or `FALSE`.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `isQuad` to check whether mesh is a quad mesh.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=FALSE)
isTriangle(mesh)
```

isValid *Is mesh a valid mesh?*

Description

Check whether the given mesh is a valid 3D surface mesh.

Usage

```
isValid(x)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.

Value

TRUE or FALSE.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See hasGarbage to check whether mesh has garbage.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism)
isValid(mesh)
```

---

makeMesh *Make a 3D mesh*

Description

Make a 3D surface mesh from an input file, from an existing **rgl** mesh3d object, or from given vertices and faces. The mesh is optionally cleaned: duplicated vertices or faces are merged, and isolated vertices are removed. The returned faces are coherently oriented, normals are computed if requested, and triangulation is performed if requested.

Usage

```
makeMesh(
  x,
  faces,
  triangulate = FALSE,
  repairSoup = TRUE,
  removeIntersections = FALSE,
  removeMethod = c("auto", "auto_snap"),
  fillHoles = FALSE,
  fairHole = FALSE,
  maxNumHoles = 5L,
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

x	One of three options: 1) A numeric matrix with 3 columns providing the coordinates of the vertices of the mesh. 2) Either a list containing the components vertices and faces, or a mesh3d object from package **rgl**. 3) A filename to read a mesh file as in readMesh.
faces	When x is a numeric matrix with the vertices: Either an integer matrix (each row provides the vertex indices of the corresponding face) or a list of integer vectors, each one providing the vertex indices of the corresponding face.
triangulate	Boolean. Triangulate the faces? Ignored if faces are already triangle.
repairSoup	Boolean. Clean the mesh (merging duplicated vertices and duplicated faces, removing isolated vertices)?
removeIntersections	Boolean. Try to remove self-intersections?
removeMethod	character. Method for removing self-intersections. One of "auto" (for auto-refine) and "auto_snap" (auto-refine with iterative snap).
fillHoles	Boolean. Try to fill boundary holes?
fairHole	Boolean. Use CGAL triangulate_refine_and_fair_hole() (TRUE) or triangulate_and_refine_ho (FALSE) when filling holes?
maxNumHoles	integer. Maximum number of holes to be filled. May be 0.
normals	Boolean. Compute vertex normals?
verbose	Boolean. Print out messages about mesh processing?

Details

CAVE: Y-axis direction is front-back in **rgl** (OpenGL) convention but down-up in CGAL convention. This is ignored here when importing a mesh3d object.

Value

A list of class CGALmesh giving the vertices, the edges, the faces of the mesh, the exterior edges, the exterior vertices, and optionally the vertex normals.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See plotEdges for details about the edges returned by this function. See makeMeshValid for a similar function that assumes that the input defines a valid mesh, and does not perform mesh repair to gain some speed. See toRGL for conversion to class mesh3d from package **rgl**.

Examples

```
library(SurfaceMesh)
library(rgl)
## make mesh based on file
f_mesh    <- system.file("extdata", "dataHeart3.ply", package="SurfaceMesh")
mesh1     <- makeMesh(f_mesh, verbose=TRUE)
mesh1_rgl <- toRGL(mesh1)

open3d(windowRect=c(50, 50, 562, 562))
wire3d(mesh1_rgl)

## make mesh from vertices and faces
## tetrahedron with ill-oriented faces
vertices <- rbind(
  c(-1, -1, -1),
  c(1, 1, -1),
  c(1, -1, 1),
  c(-1, 1, 1))

faces <- rbind(
  c(1, 2, 3),
  c(3, 4, 2),
  c(4, 2, 1),
  c(4, 3, 1))

mesh2 <- makeMesh(vertices, faces, verbose=TRUE)
mesh2_rgl <- toRGL(mesh2)
open3d(windowRect=c(50, 50, 562, 562))
wire3d(mesh2_rgl)

## illustration of the `triangulate` option
## the faces of the truncated icosahedron are hexagonal or pentagonal:
dataTruncIcosahedron[["faces"]]
# so we triangulate them:
mesh3     <- makeMesh(dataTruncIcosahedron, triangulate=TRUE)
mesh3_rgl <- toRGL(mesh3)
open3d(windowRect=c(50, 50, 562, 562))
wire3d(mesh3_rgl)
```

---

makeMeshValid *Make a 3D mesh assuming valid input*

Description

Make a 3D surface mesh from an input file, from an existing **rgl** mesh3d object, or from given vertices and faces - assuming that the input defines a valid mesh. Omitted validity checks save some processing time. The returned faces are coherently oriented (if possible), normals are computed if requested, triangulation is performed if requested.

Usage

```
makeMeshValid(
  x,
  faces,
  soup = FALSE,
  triangulate = FALSE,
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

x	One of three options: 1) A numeric matrix with three columns providing the coordinates of the vertices of the mesh. 2) Either a list containing the components vertices and faces, or a **rgl** mesh (i.e. a mesh3d object). 3) A filename to read a mesh file as in readMesh.
faces	When x is a numeric matrix with the vertices: Either an integer matrix (each row provides the vertex indices of the corresponding face) or a list of integer vectors, each one providing the vertex indices of the corresponding face.
soup	Boolean. Assume a polygon soup? If FALSE, assume correctly oriented faces - which is a bit faster.
triangulate	Boolean. Triangulate the faces? Ignored if faces are already triangle.
normals	Boolean. Compute vertex normals?
verbose	Boolean. Print out messages about mesh processing?

Details

CAVE: Y-axis direction is front-back in **rgl** (OpenGL) convention but down-up in CGAL convention. This is ignored here when importing a mesh3d object.

Value

A list of class CGALmesh giving the vertices, the edges, the faces of the mesh, the exterior edges, the exterior vertices, and optionally the normals.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See plotEdges for details about the edges returned by this function. See makeMesh for a similar function that does not assume that the input defines a valid mesh, and may perform mesh repair. See toRGL for conversion to class mesh3d from package **rgl**.

Examples

```
library(SurfaceMesh)
library(rgl)
## mesh from file
f_mesh    <- system.file("extdata", "dataCorner.off", package="SurfaceMesh")
mesh1     <- makeMeshValid(f_mesh, soup=TRUE, triangulate=TRUE)
mesh1_rgl <- toRGL(mesh1)

open3d(windowRect=c(50, 50, 562, 562))
wire3d(mesh1_rgl)

## mesh from vertices, faces
head(dataPentaPrism[["vertices"]])
head(dataPentaPrism[["faces"]])
mesh2     <- makeMeshValid(dataPentaPrism, soup=TRUE, triangulate=TRUE)
mesh2_rgl <- toRGL(mesh2)
open3d(windowRect=c(50, 50, 562, 562))
wire3d(mesh2_rgl)
```

`meshEllipsoid` *Ellipsoid mesh*

Description

Generate a triangle surface smesh of a 3D ellipsoid.

Usage

```
meshEllipsoid(
  ctr = c(0, 0, 0),
  r = c(1, 1, 1),
  dirs = diag(nrow = 3L),
  euler,
  nIter = 3L
)
```

Arguments

`ctr`	`numeric`. Vector with the x, y, z-coordinates of the centroid.
`r`	`numeric`. Vector with the 3 semi-axis lengths.
`dirs`	`numeric`. 3-by-3 matrix. Columns are the unit direction vectors of the ellipsoid's semi-axes. CAVE: Y-axis direction is assumed to be front-back, corresponding to **rgl** (OpenGL) convention.
`euler`	`numeric`. Euler rotation angles phi, theta, psi. If given, takes precedence over `dirs`.
`nIter`	`integer`. Number of iterations (the mesh is obtained by iteratively subdividing the faces of an icosahedron).

Value

A mesh3d object.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh_rgl <- meshEllipsoid(c(0, 0, 0), r=c(1, 0.5, 0.25))
wire3d(mesh_rgl)
```

---

meshIsoCuboid *Iso-oriented cuboid mesh*

Description

Generate a surface mesh of an iso-oriented cuboid, i.e. with edges parallel to the axes.

Usage

```
meshIsoCuboid(lo, up)
```

Arguments

lo	numeric 3-vector. Lower left front corner, a point whose coordinates must all be lower than those of the upper corner.
up	numeric 3-vector. Upper right back corner, a point whose coordinates must all be higher than those of the lower corner.

Details

CAVE: Y-axis direction is assumed to be front-back, corresponding to **rgl** (OpenGL) convention.

Value

A **rgl** mesh, i.e. a mesh3d object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
lo <- c(-4, -2, -1)
up <- c( 4,  2,  1)
cuboid_rgl <- meshIsoCuboid(lo, up)
wire3d(cuboid_rgl)
```

meshSphere *Sphere mesh*

Description

Generate a triangle surface mesh of a 3D sphere.

Usage

```
meshSphere(ctr = c(0, 0, 0), r = 1, nIter = 3L)
```

Arguments

ctr	numeric. Vector with the x, y, z-coordinates of the centroid.
r	numeric. Radius.
nIter	integer. Number of iterations (the mesh is obtained by iteratively subdividing the faces of an icosahedron).

Value

A mesh3d object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh_rgl <- meshSphere(c(0, 0, 0), r=2)
wire3d(mesh_rgl)
```

orientToBoundVolume *Orient mesh to bound a volume*

Description

Orient a given 3D surface mesh to bound a volume.

Usage

```
orientToBoundVolume(x, normals = FALSE)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
normals	Boolean. Return vertex normals?

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

doesBoundVolume to check whether mesh bounds a volume and getVolume to get the volume bounded by the mesh.

Examples

```
library(SurfaceMesh)

## Two tetraeders
## one with positive orientation
## one with negative orientation
vertices <- rbind(
    c(0, 0, 0),
    c(2, 2, 0),
    c(2, 0, 2),
    c(0, 2, 2),
    c(3, 3, 3),
    c(5, 5, 3),
    c(5, 3, 5),
    c(3, 5, 5))

faces <- rbind(
    c(3, 2, 1),
    c(3, 4, 2),
    c(1, 2, 4),
    c(4, 3, 1),
    c(5, 6, 7),
    c(6, 8, 7),
    c(8, 6, 5),
    c(5, 7, 8))

mesh <- makeMesh(vertices, faces=faces)
doesBoundVolume(mesh)    # FALSE
getVolume(mesh)          # NA

mesh_bv <- orientToBoundVolume(mesh)
doesBoundVolume(mesh_bv) # TRUE
getVolume(mesh_bv)
```

plotEdges *Plot some edges*

Description

Plot the given edges with functions from package **rgl**.

Usage

```
plotEdges(
  x,
  edges,
  color = "black",
  lwd = 2,
  edgesAsTubes = TRUE,
  tubesRadius = 0.03,
  verticesAsSpheres = TRUE,
  only,
  spheresRadius = 0.05,
  spheresColor = color
)
```

Arguments

x	A matrix with 3 columns giving the coordinates of the vertices.
edges	integer matrix with two columns giving the edges by pairs of vertex indices.
color	character. Color for the edges.
lwd	Positive number. Line width. Ignored if edgesAsTubes=TRUE.
edgesAsTubes	Boolean. Draw edges as tubes?
tubesRadius	Positive number. Radius of the tubes when edgesAsTubes=TRUE.
verticesAsSpheres	Boolean. Draw vertices as spheres?
only	integer vector with the indices of the vertices to plot as spheres. If missing, all vertices are plotted as spheres.
spheresRadius	The radius of the spheres when verticesAsSpheres=TRUE.
spheresColor	character. Color of the spheres when verticesAsSpheres=TRUE.

Value

No value.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
# triangulate and plot the pentagrammic prism mesh
mesh     <- makeMesh(dataPentaPrism, triangulate=TRUE)
mesh_rgl <- toRGL(mesh)
open3d(windowRect=c(50, 50, 562, 562), zoom=0.9)
shade3d(mesh_rgl, color="navy")
plotEdges(mesh[["vertices"]],
          mesh[["edgesExterior"]],
          color        ="gold",
          tubesRadius  =0.02,
          spheresRadius=0.02)
```

`readMesh` *Read a mesh file*

Description

Read mesh vertices and faces from a file.

Usage

```
readMesh(x, method = c("soup", "mesh"), normals = FALSE, verbose = FALSE)
```

Arguments

`x`	Path to the mesh file. Supported formats are `stl`, `ply`, `obj`, `off`, `ts`.
`method`	Either `"soup"` when the file is a polygon soup, or `"mesh"` when the file is known to be a valid mesh with correct face orientations.
`normals`	Boolean. Return vertex normals for `method="mesh"`?
`verbose`	Boolean. Print out messages about mesh processing?

Value

For `method="soup"`: A list with two components: `vertices`, a numeric matrix with three columns, and `faces`, either a list of integer vectors or, in the case if all faces have the same number of sides, an integer matrix. For `method="mesh"`: A list of class `CGALmesh` giving the vertices, the edges, the faces of the mesh, the exterior edges, the exterior vertices and optionally the normals.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `makeMesh` and `makeMeshValid` for creating a `CGALmesh` object.

Examples

```
library(SurfaceMesh)
library(rgl)
f_ply    <- system.file("extdata", "dataHeart3.ply", package="SurfaceMesh")
mesh_vf  <- readMesh(f_ply, method="soup")
mesh     <- makeMesh(mesh_vf, normals=TRUE)
mesh_rgl <- toRGL(mesh)
open3d(windowRect=c(50, 50, 562, 562))
view3d(0, 0, zoom=0.8)
shade3d(mesh_rgl, color="palevioletred")
```

`reconstructAFS` *Advancing front surface reconstruction*

Description

Advancing front surface reconstruction of a 3D surface mesh from a cloud of 3D points.

Usage

```
reconstructAFS(x, nNeighbors, repairSoup = TRUE, normals = FALSE)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`, a `mesh3d` object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
`nNeighbors`	Optional integer >= 2. If specified, the point cloud is smoothed before the reconstruction, using this integer as the number of neighbors for the smoothing. Note that this smoothing preprocessing relocates the points and then should not be used if the points have been sampled without noise on the surface.
`repairSoup`	Boolean. Clean the mesh (merging duplicated vertices, duplicated faces, removing isolated vertices)?
`normals`	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Advancing_front_surface_reconstruction/ for details.

Value

A `CGALmesh` object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

removeOutliers, reconstructSSS, reconstructPoisson, alphaWrap

Examples

```
library(SurfaceMesh)
library(rgl)

mesh         <- dataHeart1
mesh_rgl     <- toRGL(mesh)
mesh_afs     <- reconstructAFS(mesh[["vertices"]], nNeighbors=30L)
mesh_afs_rgl <- toRGL(mesh_afs)

mfrow3d(1, 2)
view3d(0, -90, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(0, -90, zoom=0.7)
shade3d(mesh_afs_rgl, col="gray")
```

---

reconstructPoisson *Poisson surface reconstruction*

Description

Poisson reconstruction of a 3D surface mesh from a cloud of 3D points.

Usage

```
reconstructPoisson(
  x,
  normalsFun = getNormalsFun(6L),
  spacing,
  smAngle = 20,
  smRadius = 30,
  smDistance = 0.375,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
normalsFun	A function that accepts a matrix with 3D points and uses it to return a matrix with normals. Such a function is returned by getNormalsFun.

spacing	Optional size parameter. Smaller values increase the precision of the output mesh at the cost of higher computation time. If missing, an average spacing is chosen as a reasonable default value.
smAngle	Bound for the minimum facet angle in degrees.
smRadius	Relative bound for the radius of the surface Delaunay balls.
smDistance	Relative bound for the center-center distances.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Poisson_surface_reconstruction_3/ for details. Performs well if the aim is to approximate a noisy point cloud with a smooth surface. Not appropriate if the surface is expected to interpolate the input points.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

removeOutliers, reconstructAFS, reconstructSSS, alphaWrap

Examples

```
library(SurfaceMesh)
library(rgl)

mesh     <- makeMesh(dataSeptuaginta)
mesh_rgl <- toRGL(mesh)
mesh_psr <- reconstructPoisson(mesh[["vertices"]],
                               normalsFun=getNormalsFun(6L),
                               smAngle=20,
                               smRadius=1,
                               smDistance=20)
mesh_psr_rgl <- toRGL(mesh_psr)

mfrow3d(1, 2)
shade3d(mesh_rgl, col="gray")
next3d()
shade3d(mesh_psr_rgl, col="gray")
```

`reconstructSSS` *Scale-space surface reconstruction*

Description

Scale-space surface reconstruction of a 3D surface mesh from a cloud of 3D points.

Usage

```
reconstructSSS(
  x,
  nScaleIter = 1L,
  nNeighbors = 12L,
  nSamples = 300L,
  separateShells = FALSE,
  forceManifold = TRUE,
  borderAngle = 45,
  repairSoup = TRUE,
  normals = FALSE
)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`, a `mesh3d` object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 4 points.
`nScaleIter`	Positive integer. Number of iterations used to increase the scale.
`nNeighbors`	Positive integer. Number of neighbors used to smooth the point cloud.
`nSamples`	Positive integer. Number of samples used to smooth the point cloud.
`separateShells`	Boolean. Separate the shells?
`forceManifold`	Boolean. Force a manifold output mesh?
`borderAngle`	Bound on the angle in degrees used to detect border edges.
`repairSoup`	Boolean. Clean the mesh (merging duplicated vertices, duplicated faces, removing isolated vertices)?
`normals`	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Scale_space_reconstruction_3/ for details. A good choice if the input point cloud is noisy but the user still wants the surface to pass exactly through the points.

Value

A `CGALmesh` object or a `mesh3d` object from package **rgl**.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

removeOutliers, reconstructAFS, reconstructPoisson, alphaWrap

Examples

```
library(SurfaceMesh)
library(rgl)

mesh     <- dataHeart1
mesh_rgl <- toRGL(mesh)
mesh_sss <- reconstructSSS(mesh[["vertices"]],
                           nScaleIter   =2L,
                           nNeighbors   =12L,
                           forceManifold=TRUE)
mesh_sss_rgl <- toRGL(mesh_sss)

mfrow3d(1, 2)
view3d(0, -90, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(0, -90, zoom=0.7)
shade3d(mesh_sss_rgl, col="gray")
```

---

remeshACVD *Approximated centroidal Voronoi diagram remeshing*

Description

Approximated centroidal Voronoi diagram remeshing (ACVD) uses clustering on polygonal meshes as to approximate a centroidal Voronoi diagram construction. Includes triangulation if mesh is not already triangle.

Usage

```
remeshACVD(
  x,
  method = c("uniform_iso", "uniform_iso_qem", "uniform_qem", "adapt_iso"),
  nVerts = 500L,
  vertCountRatio = 0.1,
  gradationFactor = 2,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be a single connected component. The mesh must be triangle or be able to be made triangle.
method	character. "uniform_iso" for uniform isotropic ACVD, "uniform_iso_qem" for uniform isotropic ACVD with post-processing QEM optimization, "uniform_qem" for uniform ACVD QEM energy minimization, "adapt_iso" for adaptive isotropic ACVD.
nVerts	Positive integer. Lower bound on the target number of vertices in the output mesh
vertCountRatio	Positive number. Ratio used when input mesh does not have enough vertices compared to nVerts. The number of vertices of the input mesh should at least be vertCountRatio times nVerts If not, the mesh will first be subdivided appropriately.
gradationFactor	Positive number. For method="adapt_iso". A factor used to gradate the weights of the vertices based on their curvature values. The larger the value is, the more the curvature will impact the distribution of output vertices on the surface.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details.

Value

A CGALmesh object.

See Also

See remeshDelaunay, remeshIsotropic, remeshSimplify, remeshSmoothShape, remeshSmoothAngle, remeshSmoothTangentRelax for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- makeMeshValid(dataOloid)
mesh_rgl   <- toRGL(mesh)
mesh_d     <- remeshACVD(mesh, method="uniform_iso_qem", nVerts=100L)
mesh_d_rgl <- toRGL(mesh_d)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 30, zoom=0.8)
wire3d(mesh_rgl)
next3d()
view3d(0, 30, zoom=0.8)
wire3d(mesh_d_rgl)
```

remeshDelaunay *Surface Delaunay remeshing*

Description

Remeshing algorithm based on Delaunay refinement of a restricted Delaunay triangulation. Includes triangulation if mesh is not already triangle.

Usage

```
remeshDelaunay(
  x,
  edgeSize,
  facetAngle = 0,
  facetDist = 0.5,
  protectConstraints = TRUE,
  dihedralAngle = 60,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be triangle or be able to be made triangle.
edgeSize	Positive number: Upper bound for the lengths of curve edges. Has to be set to a positive value when protectConstraints=TRUE.
facetAngle	Positive number: Lower bound for the angles (in degrees) of the surface mesh facets.
facetDist	Positive number. Upper bound for the distance between the facet circumcenter and the center of its surface Delaunay ball.
protectConstraints	Boolean. Protect sharp edges. See dihedralAngle.
dihedralAngle	Positive number. Constrain edges with a dihedral angle over given value to protect sharp edges.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See remeshACVD, remeshIsotropic, remeshSimplify, remeshSmoothShape, remeshSmoothAngle, remeshSmoothTangentRelax for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- makeMesh(dataPentaPrism, triangulate=TRUE)
mesh_rgl   <- toRGL(mesh)
mesh_d     <- remeshDelaunay(mesh,
                             edgeSize=0.2,
                             facetDist=0.2,
                             protectConstraints=TRUE)
mesh_d_rgl <- toRGL(mesh_d)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 30, zoom=0.8)
wire3d(mesh_rgl)
next3d()
view3d(0, 30, zoom=0.8)
wire3d(mesh_d_rgl)
```

---

remeshIsotropic *Isotropic remeshing*

Description

Isotropic remeshing of a triangular 3D surface mesh.

Usage

```
remeshIsotropic(
  x,
  method = c("uniform", "adaptive"),
  tol = 0.001,
  edgeMin = 0.001,
  edgeMax,
  targetEdgeLen,
  nIter = 1L,
  nRelaxSteps = 1L,
```

```
  protectConstraints = TRUE,
  dihedralAngle = 60,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
method	character. Either "uniform" for uniform sizing field (all triangle edges are targeted to have equal lengths) or "adaptive" for adaptive sizing field (triangle edge lengths depend on the local curvature).
tol	Positive number for method="adaptive". Error tolerance. See details.
edgeMin	Positive number for method="adaptive". Minimum edge length. See details.
edgeMax	Positive number for method="adaptive". Maximum edge length. See details.
targetEdgeLen	Positive number for method="uniform": The target edge length of the remeshed mesh.
nIter	Positive integer: Number of iterations.
nRelaxSteps	Positive integer: Number of relaxation steps.
protectConstraints	Boolean. Protect constraints? See details.
dihedralAngle	Positive number. Constrain edges with a dihedral angle over given value to protect sharp edges.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See remeshACVD, remeshDelaunay, remeshSimplify, remeshSmoothShape, remeshSmoothAngle, remeshSmoothTangentRelax for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh         <- makeMesh(dataTruncIcosahedron, triangulate=TRUE)
mesh_rgl     <- toRGL(mesh)
mesh_rem     <- remeshIsotropic(mesh, targetEdgeLen=0.7)
mesh_rem_rgl <- toRGL(mesh_rem)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
wire3d(mesh_rgl)
next3d()
wire3d(mesh_rem_rgl)
```

---

remeshSimplify *Triangulated surface mesh simplification*

Description

Simplification of a triangular 3D surface mesh.

Usage

```
remeshSimplify(
  x,
  repairSoup = TRUE,
  repairMesh = TRUE,
  method = c("LT-R", "LT-C", "LT-BNCF", "GH"),
  stopUeRatio = 0.1,
  stopUeCount,
  policyGH = c("CP", "CT", "PP", "PT", "PL"),
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
repairSoup	Boolean. Clean the input mesh (merging duplicated vertices and duplicated faces, removing isolated vertices)?
repairMesh	Boolean. Try to remove self-intersections from output mesh?
method	character. Cost and placement strategy. One of "LT-R" (Lindstrom-Turk using undirected edge ratio as stop predicate), "LT-C" (Lindstrom-Turk using undirected edge count as stop predicate), "LT-BNCF" (Lindstrom-Turk, applying bounded normal change filter), and "GH" (Garland-Heckbert).

stopUeRatio	Positive number. Stop predicate. For method="LT-R", "GH". Ratio of undirected edges relative to initial count. Lower -> coarser.
stopUeCount	integer. Stop predicate. For method="LT-C", "LT-BNCF". Count of undirected edges left. If missing, set to (half of all edges) - 1. Lower -> coarser.
policyGH	character. For method="GH". Simplification strategy. One of "CP" (classic plane), "CT" (classic tri), "PP" (prob plane), "PT" (prob tri), and "PL" (plane and line).
normals	Boolean. Return vertex normals?
verbose	Boolean. Print out messages about mesh processing?

Details

See https://doc.cgal.org/latest/Surface_mesh_simplification/ for details.

Value

A CGALmesh object.

See Also

See remeshACVD, remeshDelaunay, remeshIsotropic, remeshSmoothShape, remeshSmoothAngle, remeshSmoothTangentRelax for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- makeMesh(dataHopfTorus, triangulate=TRUE)
mesh_rgl   <- toRGL(mesh)
mesh_s     <- remeshSimplify(mesh, method="LT-R", stopUeRatio=0.1)
mesh_s_rgl <- toRGL(mesh_s)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
wire3d(mesh_rgl)
next3d()
wire3d(mesh_s_rgl)
```

---

remeshSmoothAngle *Smoothing by angle optimization*

Description

Smoothes a 3D surface mesh by angle optimization. Includes triangulation if mesh is not already triangle.

Usage

```
remeshSmoothAngle(
  x,
  dihedralAngle = 60,
  nIter = 1L,
  useSafeConstr = FALSE,
  doProject = TRUE,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be triangle or be able to be made triangle.
dihedralAngle	Positive number. Constrain edges with a dihedral angle over given value to preserve sharp edges.
nIter	Positive integer: Number of iterations.
useSafeConstr	Boolean. Use safety constraints for moving vertices?
doProject	Boolean. Project points onto initial surface after each iteration?
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details. Note that CGAL angle_and_area_smoothing() supports smoothing by area optimization as well. However, this feature requires the external Ceres library (https://github.com/ceres-solver/ceres-solver) and thus is currently not enabled here.

Value

A CGALmesh object.

See Also

See remeshACVD, remeshDelaunay, remeshIsotropic, remeshSimplify, remeshSmoothShape, remeshSmoothTangentRelax for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- makeMesh(dataSeptuaginta, triangulate=TRUE)
mesh_rgl   <- toRGL(mesh)
mesh_s     <- remeshSmoothAngle(mesh, nIter=5L)
mesh_s_rgl <- toRGL(mesh_s)

mfrow3d(1, 2)
```

```
view3d(45, -45, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(45, -45, zoom=0.7)
shade3d(mesh_s_rgl, col="gray")
```

`remeshSmoothShape` *Smooth shape*

Description

Smooths the overall shape of a 3D surface mesh by using the mean curvature flow. Includes triangulation if mesh is not already triangle.

Usage

```
remeshSmoothShape(x, faces, nIter = 1L, time = 0.001, normals = FALSE)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`. The mesh must be triangle or be able to be made triangle.
`faces`	Indices of the faces to be smoothed. If missing, the whole mesh is smoothed.
`nIter`	Positive `integer`: Number of iterations.
`time`	Positive number: A time step that corresponds to the speed by which the surface is smoothed (the larger the faster). typical values lie between `1e-6` and `1`. See details.
`normals`	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details.

Value

A `CGALmesh` object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See `remeshACVD`, `remeshDelaunay`, `remeshIsotropic`, `remeshSimplify`, `remeshSmoothAngle`, `remeshSmoothTangentRelax` for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- dataHeart1
mesh_rgl   <- toRGL(mesh)
mesh_s     <- remeshSmoothShape(mesh, nIter=5L, time=1)
mesh_s_rgl <- toRGL(mesh_s)

mfrow3d(1, 2)
view3d(0, -90, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(0, -90, zoom=0.7)
shade3d(mesh_s_rgl, col="gray")
```

---

remeshSmoothTangentRelax

Smoothing by tangential relaxation

Description

Iterative area-based smoothing of a 3D surface mesh by tangential relaxation. Includes triangulation if mesh is not already triangle.

Usage

```
remeshSmoothTangentRelax(x, nIter = 1L, relaxConstr = FALSE, normals = FALSE)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be triangle or be able to be made triangle.
nIter	Positive integer: Number of iterations.
relaxConstr	Boolean. If TRUE, the end vertices of the edges set as constrained and boundary edges move along the constrained polylines they belong to.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/PMP_Remeshing/ for details.

Value

A CGALmesh object.

See Also

See remeshACVD, remeshDelaunay, remeshIsotropic, remeshSimplify, remeshSmoothShape, remeshSmoothAngle for other remeshing operations.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh       <- makeMesh(dataSeptuaginta, triangulate=TRUE)
mesh_rgl   <- toRGL(mesh)
mesh_s     <- remeshSmoothTangentRelax(mesh, nIter=5L)
mesh_s_rgl <- toRGL(mesh_s)

mfrow3d(1, 2)
view3d(45, -45, zoom=0.7)
shade3d(mesh_rgl, col="gray")
next3d()
view3d(45, -45, zoom=0.7)
shade3d(mesh_s_rgl, col="gray")
```

---

removeOutliers *Remove outliers from 3D point cloud*

Description

Remove outliers from 3D point cloud based on their distance to neighboring points.

Usage

```
removeOutliers(x, nNeighbors = 6L, threshPerc, threshDst)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh, a mesh3d object from package **rgl**, or a numeric matrix with 3 columns which stores the point coordinates, one point per row, and at least 3 points.
nNeighbors	integer. How many neighboring points to consider for distance calculation.
threshPerc	numeric. The maximum percentage of points to remove. If specified, threshDst should not be specified, otherwise threshDst takes precedence.
threshDst	numeric. The minimum distance for a point to be considered as outlier. If missing and threshPerc is missing as well, twice the average spacing is chosen as a reasonable default value.

Details

See https://doc.cgal.org/latest/Point_set_processing_3/ for details.

Value

A numeric matrix of points with outliers removed.

See Also

reconstructAFS, reconstructPoisson, reconstructSSS, alphaWrap

Examples

```
library(SurfaceMesh)
library(rgl)

ptsBefore   <- matrix(rnorm(120L, mean=0, sd=1), ncol=3L)
(avgSpacing <- getAverageSpacing(ptsBefore, nNeighbors=6L))
## contaminate
idx             <- sample(seq_len(nrow(ptsBefore)), 4L, replace=FALSE)
ptsBefore[idx, ] <- matrix(rnorm(12L, mean=10, sd=1), ncol=3L)
ptsAfter        <- removeOutliers(ptsBefore, nNeighbors=6L)
dim(ptsBefore)
dim(ptsAfter)
```

---

removeSelfIntersections

Remove self intersections

Description

Try to remove self intersections of a 3D surface mesh.

Usage

```
removeSelfIntersections(
  x,
  method = c("auto", "auto_snap"),
  normals = FALSE,
  verbose = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
method	character. One of "auto" (for auto-refine) and "auto_snap" (auto-refine with iterative snap). See details.
normals	Boolean. Return vertex normals?
verbose	Boolean. Print out messages about mesh processing?

Details

See https://www.cgal.org/2025/06/13/autorefine-and-snap/ and https://doc.cgal.org/latest/PMP_Mesh_repair/ for details. If faces are not triangle, the mesh is triangulated. Not guaranteed to work for every input mesh.

Value

CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

See fillBoundaryHoles for filling boundary holes.

Examples

```
library(SurfaceMesh)
verts1 <- vapply(c(0, 2, 4, 1, 3),
                 function(i) { c(cos(2*i*pi/5), sin(2*i*pi/5), 0.3) },
                 numeric(3L))

verts2 <- vapply(c(0, 2, 4, 1, 3),
                 function(i) { c(cos(2*i*pi/5), sin(2*i*pi/5), -0.3) },
                 numeric(3L))

vertices    <- t(cbind(verts1, verts2))
pentagramms <- rbind(1L:5L, 6L:10L)

rectangles <- rbind(
    c(1L, 2L, 7L, 6L),
    c(2L, 3L, 8L, 7L),
    c(3L, 4L, 9L, 8L),
    c(4L, 5L, 10L, 9L),
    c(5L, 1L, 6L, 10L))

faces <- list(
    pentagramms[1L, ],
    pentagramms[2L, ],
    rectangles[1L, ],
    rectangles[2L, ],
    rectangles[3L, ],
    rectangles[4L, ],
    rectangles[5L, ])

mesh <- makeMesh(vertices, faces)
doesSelfIntersect(mesh)

mesh_nsi <- removeSelfIntersections(mesh)
getVolume(mesh_nsi)
```

reverseFaceOrientations *Reverse orientation of mesh faces*

Description

For the faces of a 3D surface mesh, reverses the order of the vertices along the face boundary.

Usage

```
reverseFaceOrientations(x, faces, normals = FALSE)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
faces	Indices of the faces to select for reversing orientation. If missing, the orientation of all faces is reversed. Cave: The resulting mesh may be invalid if faces have inconsistent orientation.
normals	Boolean. Return vertex normals?

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

isOutwardOriented, doesBoundVolume, orientToBoundVolume.

Examples

```
library(SurfaceMesh)
mesh <- makeMeshValid(dataTorus)
doesBoundVolume(mesh)
getVolume(mesh)  # positive

mesh_rfo <- reverseFaceOrientations(mesh)
doesBoundVolume(mesh_rfo)
getVolume(mesh_rfo)  # negative

## invalid mesh from inconsistent face orientations
mesh_invalid <- reverseFaceOrientations(mesh, faces=501:1000)
```

`samplePoints` *Sample vertices on mesh*

Description

Random sampling of vertices on a given 3D triangle surface mesh.

Usage

```
samplePoints(
  x,
  method = c("random", "grid", "mc"),
  sampleVerts = TRUE,
  sampleEdges = TRUE,
  sampleFaces = TRUE,
  gridSpacing = NULL,
  ptsOnEdges = NULL,
  ptsOnFaces = NULL,
  ptsPerDist = NULL,
  ptsPerEdge = NULL,
  ptsPerArea = NULL,
  ptsPerFace = NULL
)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`.
`method`	`character`. `"random"` for random uniform sampling, `"grid"` for grid sampling, `"mc"` for Monte Carlo sampling. See details.
`sampleVerts`	Boolean. Do sample vertices?
`sampleEdges`	Boolean. Do sample edges?
`sampleFaces`	Boolean. Do sample faces?
`gridSpacing`	`numeric`. The grid spacing for `method="grid"`.
`ptsOnEdges`	`integer`. For the random sampling method as the number of points to pick exclusively on edges. If missing, the number of edges is used.
`ptsOnFaces`	`integer`. For the random sampling method as the number of points to pick on the surface. If missing, the number of vertices is used.
`ptsPerDist`	`numeric`. Points per distance unit. Used for `method="random"` and `"mc"` to respectively determine the total number of points on edges and the number of points per edge.
`ptsPerEdge`	`integer`. Points per edge. Used for `method="mc"` as the number of points per edge to pick.
`ptsPerArea`	`numeric`. Points per area unit. Used for `method="random"` and `"mc"` to respectively determine the total number of points inside faces and the number of points per face.

ptsPerFace	integer. Points per face. Used for method="mc" as the number of points per face to pick.

Details

For details on sampling options, see https://doc.cgal.org/latest/Polygon_mesh_processing/group__PMP__distance__grp.html.

Value

A n x 3 numeric matrix containing the sampled vertices.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE)
samplePoints(mesh, ptsOnFaces=2L)
```

---

setVertexNormals *Assign normal vectors to mesh vertices*

Description

Assign given per-vertex normal vectors to a 3D surface mesh.

Usage

```
setVertexNormals(x, normals, tol = 1e-08)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh.
normals	A numeric matrix providing unit normal vectors with three columns and as many rows as the number of vertices.
tol	numeric. Tolerance for checking that normals are unit length.

Details

Note: This function is currently not doing anything except adding a list component normals to the CGALmesh object after setting the property map on the C++ / CGAL side, and then exporting it back to R. This is a test case for working with CGAL property maps that may be useful in the future.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

See Also

getVertexNormals

Examples

```
library(SurfaceMesh)
mesh <- makeMesh(dataPentaPrism, triangulate=TRUE, normals=FALSE)

## random unit normal vectors
nVerts  <- nrow(mesh[["vertices"]])
nrmls0  <- matrix(runif(nVerts*3), ncol=3)
lens    <- sqrt(rowSums(nrmls0^2))
nrmls   <- diag(1/lens) %*% nrmls0
mesh_vn <- setVertexNormals(mesh, nrmls)
head(mesh_vn[["normals"]])
head(nrmls)
```

---

subdivide *Subdivision and deformation*

Description

Performs the Catmull-Clark, Doo-Sabin, Loop, and Sqrt3 subdivision and deformation of a 3D surface mesh. Includes triangulation if mesh is not already triangle.

Usage

```
subdivide(
  x,
  method = c("CC", "DS", "Loop", "Sqrt3"),
  nIter = 1L,
  triangulate = TRUE,
  normals = FALSE
)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. The mesh must be triangle or be able to be made triangle.
method	character. One of "CC" (Catmull-Clark), "DS" (Doo-Sabin), "Loop" (Loop), "Sqrt3" (Sqrt3).
nIter	Positive integer: Number of iterations.
triangulate	Boolean. For method="DS". Triangulate resulting mesh? For other methods, result is automatically triangulated.
normals	Boolean. Return vertex normals?

Details

See https://doc.cgal.org/latest/Subdivision_method_3/ for details.

Value

A CGALmesh object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)

mesh        <- makeMesh(dataPentaPrism, triangulate=TRUE)
mesh_rgl    <- toRGL(mesh)
mesh_cc     <- subdivide(mesh, method="CC", nIter=2L)
mesh_ds     <- subdivide(mesh, method="DS", nIter=2L)
mesh_cc_rgl <- toRGL(mesh_cc)
mesh_ds_rgl <- toRGL(mesh_ds)

open3d(windowRect=50 + c(0, 0, 1200, 400))
mfrow3d(1, 3)
wire3d(mesh_rgl)
next3d()
wire3d(mesh_cc_rgl)
next3d()
wire3d(mesh_ds_rgl)
```

toRGL *Conversion to 'rgl' mesh*

Description

Converts a CGALmesh object (the output of the makeMesh function) to a mesh3d object from package **rgl**.

Usage

```
toRGL(x, ...)
```

Arguments

x	A CGALmesh object, i.e., the output of makeMesh. In order to be convertible to a mesh3d object from package **rgl**, its faces must have at most four sides.
...	Arguments passed to mesh3d.

Details

CAVE: Y-axis direction is front-back in **rgl** (OpenGL) convention but down-up in CGAL convention. This is ignored here.

Value

A mesh3d object from package **rgl**.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
mesh     <- makeMesh(dataTruncIcosahedron, triangulate=TRUE)
mesh_rgl <- toRGL(mesh, segments=t(mesh[["edges"]]))
open3d(windowRect=c(50, 50, 562, 562), zoom=0.9)
wire3d(mesh_rgl, color="darkred")
```

`triangulateMesh` *Triangulate mesh*

Description

Triangulate a given 3D surface mesh.

Usage

```
triangulateMesh(x, normals = FALSE)
```

Arguments

`x`	A `CGALmesh` object, i.e., the output of `makeMesh`.
`normals`	Boolean. Return vertex normals?

Value

A `CGALmesh` object.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Examples

```
library(SurfaceMesh)
library(rgl)
## quad mesh
f_mesh   <- system.file("extdata", "dataCorner.off", package="SurfaceMesh")
mesh     <- makeMesh(f_mesh)
mesh_rgl <- toRGL(mesh)
isTriangle(mesh)

mesh_tri     <- triangulateMesh(mesh)
mesh_tri_rgl <- toRGL(mesh_tri)

open3d(windowRect=50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
wire3d(mesh_rgl)
next3d()
wire3d(mesh_tri_rgl)
```

writeMesh *Export mesh to a file*

Description

Export a 3D surface mesh to a file.

Usage

```
writeMesh(
  x,
  filename,
  normals = FALSE,
  precision = 17L,
  binary = FALSE,
  verbose = FALSE
)
```

Arguments

x	A mesh given as CGALmesh object, i.e., the output of makeMesh. Or a mesh3d object from package **rgl**. Or a polygon soup given as a list containing (at least) the components vertices and faces.
filename	Name of the output file. Supported file formats are stl, ply, obj, off, ts.
normals	Boolean. For x being a CGALmesh object which has a property map v:normal: Write vertex normals? Only supported for file formats off, ply. TODO Currently not implemented.
precision	Positive integer. Number of decimal digits for the vertex coordinates.
binary	Boolean. For file types ply, stl: Write a binary file (instead of a plain text ASCII file)?
verbose	Boolean. Print out messages about mesh processing?

Value

No value. Just generates the file.

Author(s)

Originally developed by Stephane Laurent, adapted by Daniel Wollschlaeger.

Index

∗ **datasets**
dataCyclide, 10
dataEntzenStar, 11
dataHeart1, 11
dataHeart2, 12
dataHopfTorus, 12
dataICN5Deight, 13
dataOloid, 14
dataOrthoCircle, 14
dataPentaPrism, 15
dataSeptuaginta, 16
dataSolidMobiusStrip, 16
dataSphere, 17
dataSpiderCage, 18
dataToroHelix, 18
dataTorus, 19
dataTruncIcosahedron, 19
dataTubularKnot, 20

addVertexNormals, 4
alphaWrap, 5, *25*, *55, 56*, *58*, *69*

boolDifference, 6, *8, 9*, *35*
boolIntersection, 7, *7*, *9*, *35*
boolUnion, *7, 8*, 9, *35*

dataCyclide, 10
dataEntzenStar, 11
dataHeart1, 11
dataHeart2, 12
dataHopfTorus, 12
dataICN5Deight, 13
dataOloid, 14
dataOrthoCircle, 14
dataPentaPrism, 15
dataSeptuaginta, 16
dataSolidMobiusStrip, 16
dataSphere, 17
dataSpiderCage, 18
dataToroHelix, 18
dataTorus, 19
dataTruncIcosahedron, 19
dataTubularKnot, 20
doesBoundVolume, 21, *51*, *71*
doesSelfIntersect, 22

fillBoundaryHoles, 23, *41*, *70*

getArea, 24, *39*
getAverageSpacing, 25
getBoundingBox, 26, *28, 29*, *31*
getBoundingEll, 27, *27*, *29*, *31*
getBoundingSphere, *27*, 28, *28*, *31*
getCentroid, 29, *31*
getConvexHull, *27–29*, 30
getDCOM, 31
getDistance, 32
getHausdorff, 33, *37, 38*
getJSCDSC, 34, *34*, *38*
getNormalsFun, 35, *55*
getSurfaceDist, *33, 34*, 36
getVertexNormals, 38, *74*
getVolume, *21*, *25*, 39, *51*

hasGarbage, 40, *44*

isClosed, 40
isOutwardOriented, 41, *71*
isQuad, 42, *43*
isTriangle, *42*, 43
isValid, *40*, 43

makeMesh, *4–6*, *8, 9*, *12*, *14*, *21–27*, *29–34*, *37–43*, 44, *44*, *47*, *51*, *53–55*, *57*, *59, 60*, *62, 63*, *65–69*, *71–73*, *75–78*
makeMeshValid, *45*, 46, *53*
mesh3d, *5, 6*, *8–11*, *13, 14*, *16–20*, *25–29*, *34*, *44–47*, *49, 50*, *54, 55*, *57*, *68*, *76*, *78*
meshEllipsoid, 48
meshIsoCuboid, 49
meshSphere, 50

orientToBoundVolume, *21*, *41*, 50, *71*

plotEdges, *45*, *47*, 52

readMesh, *45*, *47*, 53
reconstructAFS, *5*, 54, *56*, *58*, *69*
reconstructPoisson, *5*, *25*, *35*, 55, *55*, *58*, *69*
reconstructSSS, *5*, *25*, *55, 56*, 57, *69*
remeshACVD, 58, *61, 62*, *64–66*, *68*
remeshDelaunay, *59*, 60, *62*, *64–66*, *68*
remeshIsotropic, *59*, 61, *61*, *64–66*, *68*
remeshSimplify, *59*, *61, 62*, 63, *65, 66*, *68*
remeshSmoothAngle, *59*, *61, 62*, 64, *64*, *66*, *68*
remeshSmoothShape, *59*, *61, 62*, *64, 65*, 66, *68*
remeshSmoothTangentRelax, *59*, *61, 62*, *64–66*, 67
removeOutliers, *25*, *55, 56*, *58*, 68
removeSelfIntersections, *22*, *24*, 69
reverseFaceOrientations, *41*, 71

samplePoints, 72
setVertexNormals, *38*, 73
subdivide, 74

toRGL, *45*, *47*, 76
triangulateMesh, 77

writeMesh, 78