Package 'MeshesOperations'

August 12, 2022

Type Package

Title Operations on 3D Meshes

Version 0.1.0

Author Stéphane Laurent

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description Uses the 'CGAL' C++ library to perform operations on 3D meshes: Boolean operations (intersection, union, difference), Minkowski sum, smoothing, clipping, decomposition into convex parts.

License GPL-3

URL https://github.com/stla/MeshesOperations

BugReports https://github.com/stla/MeshesOperations/issues

Depends R (>= 3.1.0)

Imports data.table, gmp, Rcpp (>= 1.0.7), rgl, Rvcg

Suggests misc3d, rmarchingcubes, randomcoloR

LinkingTo Rcpp, RcppCGAL, RcppEigen, BH

Encoding UTF-8

LazyData true

LazyDataCompression bzip2

RoxygenNote 7.2.1

SystemRequirements C++ 14, gmp, mpfr

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-08-12 14:20:05 UTC

R topics documented:

`clipMesh` *Clip a mesh*

Description

Clip a mesh to the volume bounded by another mesh.

Usage

```
clipMesh(mesh, clipper, clipVolume = TRUE, normals = FALSE)
```

Arguments

mesh	a mesh given either as a list containing (at least) the fields vertices and faces, otherwise a **rgl** mesh (i.e. a mesh3d object)
clipper	a mesh given either as a list containing (at least) the fields vertices and faces, otherwise a **rgl** mesh (i.e. a mesh3d object)
clipVolume	Boolean, whether the clipping has to be done on the volume bounded by mesh rather than on its surface (i.e. mesh will be kept closed if it is closed)
normals	Boolean, whether to compute the vertex normals of the output mesh

Value

A triangle mesh represented as the output of the Mesh function.

Note

If clipVolume=TRUE, the mesh to be clipped (mesh) must be without self-intersection.

Examples

```
# cube clipped to sphere
library(MeshesOperations)
library(rgl)
mesh    <- cube3d()
clipper <- sphereMesh(r= sqrt(2))
clippedMesh <- clipMesh(mesh, clipper)
open3d(windowRect = c(50, 50, 562, 562))
view3d(zoom = 0.9)
shade3d(toRGL(clippedMesh), color = "purple")

# Barth sextic ####
library(MeshesOperations)
library(rgl)
library(rmarchingcubes)
# isosurface function
gold <- (1+sqrt(5))/2
f <- function(x,y,z){
x2 <- x*x; y2 <- y*y; z2 <- z*z
4*(gold^2*x2-y2)*(gold^2*y2-z2)*(gold^2*z2-x2) -
(1+2*gold)*(x2+y2+z2-1)^2
}
# grid
n <- 200L
x <- y <- z <- seq(-sqrt(3), sqrt(3), length.out = n)
g <- expand.grid(X = x, Y = y, Z = z)
# calculate voxel
voxel <- array(with(g, f(X, Y, Z)), dim = c(n, n, n))
# calculate isosurface
contour_shape <- contour3d(
griddata = voxel, level = 0, x = x, y = y, z = z
)
```

```
# make rgl mesh (plotted later)
mesh <- tmesh3d(
vertices = t(contour_shape[["vertices"]]),
indices  = t(contour_shape[["triangles"]]),
normals  = contour_shape[["normals"]],
homogeneous = FALSE
)
# clip to sphere of radius sqrt(3)
clipper <- sphereMesh(r = sqrt(3))
clippedMesh <- clipMesh(mesh, clipper, clipVolume = FALSE, normals = TRUE)
# plot
open3d(windowRect = c(50, 50, 950, 500))
mfrow3d(1, 2)
view3d(zoom = 0.8)
shade3d(mesh, color = "darkred")
next3d()
view3d(zoom = 0.8)
shade3d(toRGL(clippedMesh), color = "darkred")
```

`connectedComponents` *Connected components of a 3D mesh*

Description

Computes the connected components of a 3D mesh; for each returned component, its faces are coherently oriented, its normals are computed if desired, and it is triangulated if desired.

Usage

```
connectedComponents(
  vertices,
  faces,
  mesh = NULL,
  triangulate = FALSE,
  clean = FALSE,
  normals = FALSE,
  numbersType = "double"
)
```

Arguments

`vertices`	a numeric matrix with three columns, or a `bigq` matrix with three columns if `numbersType="gmp"`
`faces`	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
`mesh`	if not `NULL`, this argument takes precedence over `vertices` and `faces`, and must be either a list containing the fields `vertices` and `faces` (objects as described above), otherwise a **rgl** mesh (i.e. a `mesh3d` object)

`triangulate`	Boolean, whether to triangulate the faces
`clean`	Boolean, whether to clean the mesh (merging duplicated vertices, duplicated faces, removed isolated vertices)
`normals`	Boolean, whether to compute the normals
`numbersType`	the type of the numbers used in C++ for the computations; must be one of `"double"`, `"lazyExact"` (a type provided by CGAL for exact computations), or `"gmp"` (exact computations with rational numbers); using exact computations can improve the detection of the exterior edges

Value

A list of meshes, the connected components, each one being represented as the output of the `Mesh` function.

Examples

```
library(MeshesOperations)
library(rgl)

# a tetrahedron with ill-oriented faces ####
vertices1 <- rbind(
  c(-1, -1, -1),
  c( 1,  1, -1),
  c( 1, -1,  1),
  c(-1,  1,  1)
)
faces1 <- rbind(
  c(1, 2, 3),
  c(3, 4, 2),
  c(4, 2, 1),
  c(4, 3, 1)
)
# same tetrahedron translated ####
vertices2 <- vertices1 + 3
# merge the two tetrahedra ####
vertices <- rbind(vertices1, vertices2)
faces <- rbind(faces1, faces1 + 4)

# now run the `connectedComponents` function ####
meshes <- connectedComponents(vertices, faces, normals = FALSE)
mesh1 <- meshes[[1]]; mesh2 <- meshes[[2]]
# plot
tmesh1 <- toRGL(mesh1)
tmesh2 <- toRGL(mesh2)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(tmesh1, color = "green", back = "culled")
shade3d(tmesh2, color = "red", back = "culled")
```

convexParts *Decomposition into convex parts*

Description

Decomposition of a mesh into convex parts.

Usage

```
convexParts(vertices, faces, mesh = NULL, triangulate = TRUE)
```

Arguments

vertices	a numeric matrix with three columns, or a bigq matrix with three columns if numbersType="gmp"
faces	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
mesh	if not NULL, this argument takes precedence over vertices and faces, and must be either a list containing the fields vertices and faces (objects as described above), otherwise a **rgl** mesh (i.e. a mesh3d object)
triangulate	Boolean, whether to triangulate the convex parts

Value

A list of cgalMesh lists, each corresponding to a convex part.

Examples

```
# a non-convex polyhedron ####
library(MeshesOperations)
library(rgl)
library(randomcoloR)
meshes <- convexParts(mesh = NonConvexPolyhedron)
ncp <- length(meshes)
colors <- randomColor(ncp, hue = "random", luminosity = "bright")
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.8)
for(i in seq_len(ncp)){
  shade3d(toRGL(meshes[[i]]), color = colors[i])
}
plotEdges(
  NonConvexPolyhedron[["vertices"]],
  NonConvexPolyhedron[["edges"]]
)

# pentagrammic prism ####
library(MeshesOperations)
library(rgl)
```

```
library(randomcoloR)
meshes <- convexParts(mesh = pentagrammicPrism)
ncp <- length(meshes)
colors <- randomColor(ncp, hue = "random", luminosity = "bright")
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.8)
for(i in seq_len(ncp)){
  shade3d(toRGL(meshes[[i]]), color = colors[i])
}
plotEdges(
  pentagrammicPrism[["vertices"]],
  pentagrammicPrism[["edges"]],
  tubesRadius = 0.01,
  spheresRadius = 0.02
)
```

`cyclideMesh` *Cyclide mesh*

Description

Triangle mesh of a Dupin cyclide.

Usage

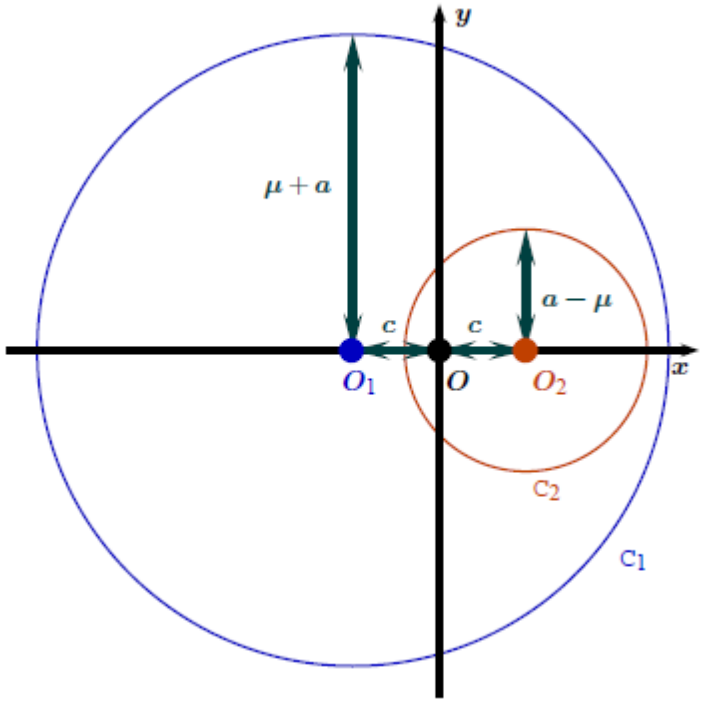
```
cyclideMesh(a, c, mu, nu = 90L, nv = 40L, rgl = TRUE)
```

Arguments

`a, c, mu`	cyclide parameters, positive numbers such that `c < mu < a`
`nu, nv`	numbers of subdivisions, integers (at least 3)
`rgl`	Boolean, whether to return a **rgl** mesh

Details

The Dupin cyclide in the plane $z=0$:



Value

A triangle **rgl** mesh (class `mesh3d`) if `rgl=TRUE`, otherwise a `cgalMesh` list (vertices, faces, and normals).

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- cyclideMesh(a = 97, c = 32, mu = 57)
sphere <- sphereMesh(x = 32, y = 0, z = 0, r = 40)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "chartreuse")
wire3d(mesh)
shade3d(sphere, color = "red")
wire3d(sphere)
```

`distancesToMesh` *Distance to a mesh*

Description

Computes the distances from given points to a mesh.

Usage

```
distancesToMesh(mesh, points)
```

Arguments

mesh	a mesh given either as a list containing (at least) the fields vertices and faces, otherwise a **rgl** mesh (i.e. a mesh3d object)
points	either one point given as a numeric vector or several points given as a numeric matrix with three columns

Value

A numeric vector providing the distances between the given point(s) to the mesh.

Examples

```
# cube example ####
library(MeshesOperations)
mesh <- rgl::cube3d()
points <- rbind(
  c(0, 0, 0),
  c(1, 1, 1)
)
distancesToMesh(mesh, points) # should be 1 and 0

# cyclide example ####
library(MeshesOperations)
a <- 100; c <- 30; mu <- 80
mesh <- cyclideMesh(a, c, mu, nu = 100L, nv = 100L)
O2 <- c(c, 0, 0)
# should be a - mu = 20 (see ?cyclideMesh):
distancesToMesh(mesh, O2)
```

HopfTorusMesh *Hopf torus mesh*

Description

Triangle mesh of a Hopf torus.

Usage

```
HopfTorusMesh(nlobes = 3, A = 0.44, alpha = NULL, nu, nv, rgl = TRUE)
```

Arguments

nlobes	number of lobes of the Hopf torus, a positive integr
A	parameter of the Hopf torus, number strictly between 0 and pi/2
alpha	if not NULL, this is the exponent of a modified stereographic projection, a positive number; otherwise the ordinary stereographic projection is used
nu, nv	numbers of subdivisions, integers (at least 3)
rgl	Boolean, whether to return a **rgl** mesh

Value

A triangle **rgl** mesh (class mesh3d) if rgl=TRUE, otherwise a cgalMesh list (vertices, faces, and normals).

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- HopfTorusMesh(nu = 90, nv = 90)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "forestgreen")
wire3d(mesh)
mesh <- HopfTorusMesh(nu = 90, nv = 90, alpha = 1.5)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "yellowgreen")
wire3d(mesh)
```

---

isotropicRemesh *Isotropic remeshing*

Description

Isotropically remesh a mesh.

Usage

```
isotropicRemesh(
  vertices,
  faces,
  mesh = NULL,
  targetEdgeLength,
  iterations = 1,
  relaxSteps = 1,
  normals = FALSE
)
```

Arguments

`vertices`	a numeric matrix with three columns
`faces`	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
`mesh`	if not `NULL`, this argument takes precedence over `vertices` and `faces`, and must be either a list containing the fields `vertices` and `faces` (objects as described above), otherwise a **rgl** mesh (i.e. a `mesh3d` object)
`targetEdgeLength`	positive number, the target edge length of the remeshed mesh
`iterations`	number of iterations, a positive integer
`relaxSteps`	number of relaxation steps, a positive integer
`normals`	Boolean, whether to compute the vertex normals of the output mesh

Value

A triangle mesh represented as the output of the `Mesh` function.

Examples

```
library(MeshesOperations)
library(rgl)

theta <- seq(0, 2*pi, length.out = 16)
torus <- cylinder3d(
  cbind(cos(theta), sin(theta), 0),
  radius = 0.4, closed = TRUE
)

mesh <- isotropicRemesh(
  mesh = torus,
  targetEdgeLength = 0.3,
  iterations = 3
)
rglmesh <- toRGL(mesh)

open3d(windowRect = c(50, 50, 950, 500))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.8)
wire3d(torus)
next3d()
view3d(0, 0, zoom = 0.8)
wire3d(rglmesh)
```

Mesh *Make a 3D mesh*

Description

Make a 3D mesh from given vertices and faces; the returned faces are coherently oriented, normals are computed if desired, and triangulation is performed if desired.

Usage

```
Mesh(
  vertices,
  faces,
  mesh = NULL,
  triangulate = FALSE,
  clean = FALSE,
  normals = FALSE,
  numbersType = "double"
)
```

Arguments

vertices	a numeric matrix with three columns, or a bigq matrix with three columns if numbersType="gmp"
faces	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
mesh	if not NULL, this argument takes precedence over vertices and faces, and must be either a list containing the fields vertices and faces (objects as described above), otherwise a **rgl** mesh (i.e. a mesh3d object)
triangulate	Boolean, whether to triangulate the faces; if TRUE, it is highly recommended to use an exact type of numbers, i.e. numbersType="lazyExact" or numbersType="gmp"
clean	Boolean, whether to clean the mesh (merging duplicated vertices, duplicated faces, removed isolated vertices)
normals	Boolean, whether to compute the normals
numbersType	the type of the numbers used in C++ for the computations; must be one of "double", "lazyExact" (a type provided by CGAL for exact computations), or "gmp" (exact computations with rational numbers); using exact computations can improve the detection of the exterior edges

Value

A list giving the vertices, the edges, the faces of the mesh, the exterior edges, the exterior vertices and optionally the normals. This list has two additional components edges0 and normals0 if triangulate=TRUE, giving the edges and the normals before the triangulation, unless the mesh is already triangulated, in which case the triangulate option is ignored.

Examples

```
library(MeshesOperations)
library(rgl)

# a 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)
)

# plot the tetrahedron, hiding the back of the faces
# then some faces do not appear, as their orientation is not correct
tmesh1 <- tmesh3d(
  vertices = t(vertices),
  indices = t(faces),
  homogeneous = FALSE
)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(tmesh1, color = "green", back = "cull")

# now run the `Mesh` function
mesh2 <- Mesh(vertices, faces, normals = FALSE)
# plot the tetrahedron, hiding the back of the faces
# then all faces appear now
tmesh2 <- toRGL(mesh2)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(tmesh2, color = "blue", back = "cull")

# illustration of the `clean` option ####
# we construct a mesh with a lot of duplicated vertices
library(misc3d) # to compute a mesh of an isosurface
a <- 0.94; mu <- 0.56; c <- 0.34 # cyclide parameters
f <- function(x, y, z, a, c, mu){ # implicit equation of the cyclide
  b <- sqrt(a^2 - c^2)
  (x^2 + y^2 + z^2 - mu^2 + b^2)^2 - 4*(a*x - c*mu)^2 - 4*b^2*y^2
}
x <- seq(-c - mu - a, abs(mu - c) + a, length.out = 45)
y <- seq(-mu - a, mu + a, length.out = 45)
z <- seq(-mu - c, mu + c, length.out = 30)
g <- expand.grid(x = x, y = y, z = z)
voxel <- array(with(g, f(x, y, z, a, c, mu)), c(45, 45, 30))
cont <- computeContour3d(voxel, level = 0, x = x, y = y, z = z)
ids <- matrix(1:nrow(cont), ncol = 3, byrow = TRUE)
# run the `Mesh` function with `clean=TRUE`
```

```
mesh <- Mesh(cont, ids, clean = TRUE, normals = TRUE)
# plot the cyclide
tmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
shade3d(tmesh, color = "green")

# illustration of the `triangulate` option ####
# the faces of the truncated icosahedron are hexagonal or pentagonal:
truncatedIcosahedron[["faces"]]
# so we triangulate them:
mesh <- Mesh(
  mesh = truncatedIcosahedron,
  triangulate = TRUE, normals = FALSE,
  numbersType = "lazyExact"
)
# now we can plot the truncated icosahedron
tmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
shade3d(tmesh, color = "orange")
```

`meshArea` *Mesh area*

Description

Computes the surface area a mesh.

Usage

```
meshArea(mesh)
```

Arguments

`mesh`	a mesh given either as a list containing (at least) the two fields `vertices` (numeric matrix with three columns) and `faces` (integer matrix or list of integer vectors), otherwise as a **rgl** mesh (i.e. a `mesh3d` object)

Value

A number, the surface area of the mesh.

Examples

```
library(MeshesOperations)
R <- 4; r <- 2
mesh <- torusMesh(R, r)
meshArea(mesh)
# true area of the torus:
4 * pi^2 * R * r
```

MeshesDifference *Meshes difference*

Description

Computes the difference between two meshes.

Usage

```
MeshesDifference(
  mesh1,
  mesh2,
  clean = FALSE,
  normals = FALSE,
  numbersType = "double"
)
```

Arguments

mesh1, mesh2	two meshes, each being either a **rgl** mesh, or as a list with (at least) two fields: vertices and faces; the vertices matrix must have the bigq class if numbersType="gmp", otherwise it must be numeric
clean	Boolean, whether to clean the input mesh (merging duplicated vertices, duplicated faces, removing isolated vertices) as well as the output mesh
normals	Boolean, whether to return the per-vertex normals of the output mesh
numbersType	the type of the numbers used in C++ for the computations; must be one of "double", "lazyExact" (a type provided by CGAL for exact computations), or "gmp" (exact computations with rational numbers); of course using exact computations is slower but more accurate

Value

A triangle mesh given as a list with fields vertices, faces, edges, exteriorEdges, gmpvertices if numbersType="gmp", and normals if normals=TRUE.

Examples

```
library(MeshesOperations)
library(rgl)

# mesh one: a cube
cube1 <- cube3d() # (from the rgl package)
mesh1 <-
  list(vertices = t(cube1[["vb"]][-4L, ]), faces = t(cube1[["ib"]]))

# mesh two: another cube
cube2 <- translate3d( # (from the rgl package)
```

```
  cube3d(), 1, 1, 0
)
mesh2 <-
  list(vertices = t(cube2[["vb"]][-4L, ]), faces = t(cube2[["ib"]]))

# compute the difference
differ <- MeshesDifference(mesh1, mesh2)

# plot
rgldiffer <- toRGL(differ)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(cube1, color = "yellow", alpha = 0.2)
shade3d(cube2, color = "cyan", alpha = 0.2)
shade3d(rgldiffer, color = "red")
plotEdges(
  vertices = differ[["vertices"]], edges = differ[["exteriorEdges"]],
  edgesAsTubes = TRUE, verticesAsSpheres = TRUE
)
```

`MeshesIntersection` *Meshes intersection*

Description

Computes the intersection of the given meshes.

Usage

```
MeshesIntersection(
  meshes,
  clean = FALSE,
  normals = FALSE,
  numbersType = "double"
)
```

Arguments

`meshes`	a list of meshes, each being either a **rgl** mesh, or as a list with (at least) two fields: `vertices` and `faces`; the `vertices` matrix must have the `bigq` class if `numbersType="gmp"`, otherwise it must be numeric
`clean`	Boolean, whether to clean the input meshes (merging duplicated vertices, duplicated faces, removing isolated vertices) as well as the output mesh
`normals`	Boolean, whether to return the per-vertex normals of the output mesh
`numbersType`	the type of the numbers used in C++ for the computations; must be one of `"double"`, `"lazyExact"` (a type provided by CGAL for exact computations), or `"gmp"` (exact computations with rational numbers); of course using exact computations is slower but more accurate

Value

A triangle mesh given as a list with fields `vertices`, `faces`, `edges`, `exteriorEdges`, `gmpvertices` if `numbersType="gmp"`, and `normals` if `normals=TRUE`.

Examples

```
library(MeshesOperations)
library(rgl)

# mesh one: truncated icosahedron; we triangulate it for plotting
mesh1 <- Mesh(
  mesh = truncatedIcosahedron,
  triangulate = TRUE, normals = FALSE,
  numbersType = "lazyExact"
)

# mesh two: a cube
cube <- translate3d( # (from the rgl package)
  cube3d(), 2, 0, 0
)
mesh2 <-
  list(vertices = t(cube[["vb"]][-4L, ]), faces = t(cube[["ib"]]))

# compute the intersection
inter <- MeshesIntersection(list(mesh1, mesh2))

# plot
rglmesh1 <- toRGL(mesh1)
rglinter <- toRGL(inter)
open3d(windowRect = c(50, 50, 562, 562))
shade3d(rglmesh1, color = "yellow", alpha = 0.2)
shade3d(cube, color = "cyan", alpha = 0.2)
shade3d(rglinter, color = "red")
plotEdges(
  vertices = inter[["vertices"]], edges = inter[["exteriorEdges"]],
  edgesAsTubes = FALSE, lwd = 3, verticesAsSpheres = FALSE
)

# other example, with 'gmp' rational numbers ####
library(MeshesOperations)
library(gmp)
library(rgl)

cube <- cube3d()

rglmesh1 <- cube
mesh1 <-
  list(vertices = t(cube[["vb"]][-4L, ]), faces = t(cube[["ib"]]))
mesh1[["vertices"]] <- as.bigq(mesh1[["vertices"]])

rotMatrix <- t(cbind( # pi/3 around a great diagonal
  as.bigq(c(2, -1, 2), c(3, 3, 3)),
```

```
  as.bigq(c(2, 2, -1), c(3, 3, 3)),
  as.bigq(c(-1, 2, 2), c(3, 3, 3))
))
mesh2 <-
  list(vertices = t(cube[["vb"]][-4L, ]), faces = t(cube[["ib"]]))
mesh2[["vertices"]] <- as.bigq(mesh2[["vertices"]]) %*% rotMatrix
rglmesh2 <- rotate3d(cube, pi/3, 1, 1, 1)

inter <- MeshesIntersection(list(mesh1, mesh2), numbersType = "gmp")
# perfect vertices:
inter[["gmpVertices"]]
rglinter <- toRGL(inter)

open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
bg3d("#363940")
shade3d(rglmesh1, color = "yellow", alpha = 0.2)
shade3d(rglmesh2, color = "orange", alpha = 0.2)
shade3d(rglinter, color = "hotpink")
plotEdges(
  inter[["vertices"]], inter[["exteriorEdges"]],
  only = inter[["exteriorVertices"]],
  color = "firebrick",
  tubesRadius = 0.05, spheresRadius = 0.07
)
```

`MeshesUnion` *Meshes union*

Description

Computes the union of the given meshes.

Usage

```
MeshesUnion(meshes, clean = FALSE, normals = FALSE, numbersType = "double")
```

Arguments

`meshes`	a list of two or more meshes, each being either a **rgl** mesh, or as a list with (at least) two fields: `vertices` and `faces`; the `vertices` matrix must have the `bigq` class if `numbersType="gmp"`, otherwise it must be numeric
`clean`	Boolean, whether to clean the input meshes (merging duplicated vertices, duplicated faces, removed isolated vertices) as well as the output mesh
`normals`	Boolean, whether to return the per-vertex normals of the output mesh
`numbersType`	the type of the numbers used in C++ for the computations; must be one of `"double"`, `"lazyExact"` (a type provided by CGAL for exact computations), or `"gmp"` (exact computations with rational numbers); of course using exact computations is slower but more accurate

Value

A triangle mesh given as a list with fields `vertices`, `faces`, `edges`, `exteriorEdges`, `gmpvertices` if `numbersType="gmp"`, and `normals` if `normals=TRUE`.

Examples

```
library(MeshesOperations)
library(rgl)

# mesh one: a cube
mesh1 <- cube3d() # (from the rgl package)

# mesh two: another cube
mesh2 <- translate3d( # (from the rgl package)
  cube3d(), 1, 1, 1
)

# compute the union
umesh <- MeshesUnion(list(mesh1, mesh2))

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

---

meshVolume *Mesh volume*

Description

Computes the volume bounded by a mesh.

Usage

```
meshVolume(mesh)
```

Arguments

`mesh`	a mesh given either as a list containing (at least) the two fields `vertices` (numeric matrix with three columns) and `faces` (integer matrix or list of integer vectors), otherwise as a **rgl** mesh (i.e. a `mesh3d` object)

Value

A number, the volume bounded by the mesh.

Examples

```
library(MeshesOperations)
R <- 4; r <- 2
mesh <- torusMesh(R, r)
meshVolume(mesh)
# true volume of the torus:
2 * pi^2 * R * r^2
```

`MinkowskiSum` *Minkowski sum of two meshes*

Description

Returns the mesh defined as the Minkowski sum of the two input meshes.

Usage

```
MinkowskiSum(mesh1, mesh2, triangulate = TRUE, normals = FALSE)
```

Arguments

`mesh1, mesh2`	two meshes, each one given either as a list containing (at least) the two fields `vertices` (numeric matrix with three columns) and `faces` (integer matrix or list of integer vectors), otherwise a **rgl** mesh (i.e. a `mesh3d` object)
`triangulate`	Boolean, whether to triangulate the output mesh (note that it is not necessarily triangle when the two input meshes are triangle)
`normals`	Boolean, whether to compute the vertex normals of the output mesh

Value

A mesh represented as the output of the `Mesh` function.

Examples

```
# example 1: octahedron + sphere
library(MeshesOperations)
library(rgl)
mesh1 <- octahedron3d()
mesh2 <- sphereMesh(iterations = 2L)
mesh <- MinkowskiSum(mesh1, mesh2, normals = TRUE)
rglmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562))
view3d(30, 30, zoom = 0.8)
shade3d(rglmesh, color = "maroon")

# example2: truncated icosahedron + tetrahedron
library(MeshesOperations)
library(rgl)
```

```
# mesh 1
mesh1 <- truncatedIcosahedron
# mesh 2: regular tetrahedron
a <- 1 / sqrt(3)
vertices <- rbind(
c( a, -a, -a),
c( a,  a,  a),
c(-a, -a,  a),
c(-a,  a, -a)
)
faces <- rbind(
c(1L, 2L, 3L),
c(3L, 2L, 4L),
c(4L, 2L, 1L),
c(1L, 3L, 4L)
)
mesh2 <- list(vertices = vertices, faces = faces)
# sum
mesh <- MinkowskiSum(mesh1, mesh2, normals = FALSE)
# plot
rglmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562))
view3d(30, 30, zoom = 0.8)
shade3d(rglmesh, color = "navy")
plotEdges(mesh[["vertices"]], mesh[["edges0"]], color = "yellow")
```

NonConvexPolyhedron *A mesh of a non-convex polyhedron*

Description

A cgalMesh list representing a non-convex polyhedron with 14 vertices and 24 triangular faces.

Usage

```
NonConvexPolyhedron
```

Format

A cgalMesh list (vertices, edges, faces).

octahedraCompound *Compound of five octahedra*

Description

Five octahedra in a pretty configuration. Each octahedron is centered at the origin.

Usage

```
octahedraCompound
```

Format

A list with three fields: the field `meshes` is a list of five elements, each one representing an octahedron by a list with two elements, the vertices and the faces; the field `rglmeshes` is the list of the five corresponding **rgl** meshes; the field `gmpmeshes` is the same as `meshes` except that the vertices are **gmp** rational numbers.

pentagrammicPrism *A mesh of a pentagrammic prism*

Description

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

Usage

```
pentagrammicPrism
```

Format

A `cgalMesh` list (`vertices`, `edges`, `faces`).

plotEdges *Plot some edges*

Description

Plot the given edges with **rgl**.

Usage

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

Arguments

vertices	a three-columns matrix giving the coordinates of the vertices
edges	a two-columns integer matrix giving the edges by pairs of vertex indices
color	a color for the edges
lwd	line width, a positive number, ignored if edgesAsTubes=TRUE
edgesAsTubes	Boolean, whether to draw the edges as tubes
tubesRadius	the radius of the tubes when edgesAsTubes=TRUE
verticesAsSpheres	Boolean, whether to draw the vertices as spheres
only	integer vector made of the indices of the vertices you want to plot (as spheres), or NULL to plot all vertices
spheresRadius	the radius of the spheres when verticesAsSpheres=TRUE
spheresColor	the color of the spheres when verticesAsSpheres=TRUE

Value

No value.

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- Mesh(
  mesh = truncatedIcosahedron,
  triangulate = TRUE, normals = FALSE,
  numbersType = "lazyExact"
)
# now we can plot the truncated icosahedron
tmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
shade3d(tmesh, color = "gold")
plotEdges(mesh[["vertices"]], mesh[["edges0"]], color = "navy")
```

---

qsqrt *Rational approximation of square roots*

Description

Returns a rational approximation of the square root of an integer.

Usage

```
qsqrt(x, n)

qsqrt2(n)

qsqrt3(n)

qsqrtPhi(n)

## S3 method for class 'qsqrt'
print(x, ...)
```

Arguments

x	the positive integer whose square root is desired
n	a positive integer, the higher the better approximation
...	ignored

Value

The qsqrt function returns a **gmp** rational number (class bigq) approximating the square root of x. The qsqrt2, qsqrt3, and qsqrtPhi functions return a **gmp** rational number approximating the square root of 2, 3, and phi (the golden number) respectively. Their value converge more fastly than the value obtained with qsqrt.

Examples

```
library(MeshesOperations)
qsqrt(2, 7)
qsqrt2(7)
qsqrt3(22)
qsqrtPhi(17)
```

`readMeshFile` *Read a mesh file*

Description

Read mesh vertices and faces from a file.

Usage

```
readMeshFile(filepath)
```

Arguments

`filepath`	path to the mesh file; supported formats are `stl`, `ply`, `obj` and `off`

Value

A list with two fields: `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.

Examples

```
library(MeshesOperations)
library(rgl)
vf <- readMeshFile(
  system.file("extdata", "beethoven.ply", package = "MeshesOperations")
)
mesh <- Mesh(
  vf[["vertices"]], vf[["faces"]], normals = TRUE, clean = TRUE
)
rglmesh <- toRGL(mesh)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.8)
shade3d(rglmesh, color = "palevioletred")
```

sampleOnMesh *Sampling on a mesh*

Description

Uniformly samples points on a mesh.

Usage

```
sampleOnMesh(n, mesh)
```

Arguments

n	number of simulations, a positive integer
mesh	either a list containing (at least) two fields `vertices` (numeric matrix with three columns) and `faces` (integer matrix or list of integer vectors), otherwise a **rgl** mesh (i.e. a `mesh3d` object)

Value

The simulated points on a matrix with three columns.

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- torusMesh(R = 4, r = 2)
sims <- sampleOnMesh(200, mesh)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "yellow")
points3d(sims, size = 5)
```

smoothShape *Smoothing of the shape of a mesh*

Description

Smooths the overall shape of the mesh by using the mean curvature flow.

Usage

```
smoothShape(
  vertices,
  faces,
  mesh = NULL,
  time,
  iterations = 1,
  normals = FALSE
)
```

Arguments

vertices	a numeric matrix with three columns
faces	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
mesh	if not NULL, this argument takes precedence over vertices and faces, and must be either a list containing two fields vertices and faces as described above, otherwise a **rgl** mesh (i.e. a mesh3d object)
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
iterations	number of iterations, a positive integer
normals	Boolean, whether to compute the vertex normals of the output mesh

Value

A triangle mesh represented as the output of the Mesh function.

Examples

```
library(MeshesOperations)
library(rgl)

# parabola ####
x <- seq(-1, 1, length.out = 30)
parabola <- cylinder3d(cbind(x, x^2, 0), radius = 0.2, closed = -2)
vertices <- t(parabola$vb[-4L, ])
faces <- c(
  split(t(parabola$it), 1L:ncol(parabola$it)),
  split(t(parabola$ib), 1L:ncol(parabola$ib))
)
sparabola <- smoothShape(
  vertices, faces, time = 0.0005, iterations = 10
)
sparabola <- toRGL(sparabola)
open3d(windowRect = c(50, 50, 950, 500))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(parabola, color = "orange")
```

```
wire3d(parabola)
next3d()
view3d(0, 0)
shade3d(sparabola, color = "green")
wire3d(sparabola)

# Stanford bunny (light version)
vf <- readMeshFile(
  system.file("extdata", "bunny.off", package = "MeshesOperations")
)
mesh <- Mesh(
  vf[["vertices"]], vf[["faces"]], normals = TRUE
)
rglmesh <- toRGL(mesh)
smesh <- smoothShape(
  mesh = mesh,
  time = 0.00001, iterations = 1, normals = TRUE
)
srglmesh <- toRGL(smesh)
open3d(windowRect = c(50, 50, 900, 500))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.8)
shade3d(rglmesh, color = "purple")
next3d()
view3d(0, 0, zoom = 0.8)
shade3d(srglmesh, color = "violetred")
```

`sphereMesh` *Sphere mesh*

Description

Mesh of a sphere.

Usage

```
sphereMesh(x = 0, y = 0, z = 0, r = 1, iterations = 3L)
```

Arguments

`x, y, z`	coordinates of the center
`r`	radius
`iterations`	number of iterations

Value

A **rgl** mesh (class `mesh3d`).

tetrahedraCompound *Compound of five tetrahedra*

Description

Five tetrahedra in a pretty configuration. Each tetrahedron is centered at the origin.

Usage

```
tetrahedraCompound
```

Format

A list with three fields: the field `meshes` is a list of five elements, each one representing a tetrahedron by a list with two elements, the vertices and the faces; the field `rglmeshes` is the list of the five corresponding **rgl** meshes; the field `gmpmeshes` is the same as `meshes` except that the vertices are **gmp** rational numbers.

toRGL *Conversion to 'rgl' mesh*

Description

Converts a CGAL mesh (e.g. an output of the `Mesh` function) to a **rgl** mesh.

Usage

```
toRGL(mesh, ...)
```

Arguments

`mesh`	a CGAL mesh, that is to say a list of class `"cgalMesh"` (e.g. an output of the `Mesh` function); in order to be convertible to a **rgl** mesh, its faces must have at most four sides
`...`	arguments passed to `mesh3d`

Value

A **rgl** mesh, that is to say a list of class `"mesh3d"`.

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- Mesh(
  truncatedIcosahedron[["vertices"]], truncatedIcosahedron[["faces"]],
  triangulate = TRUE, numbersType = "lazyExact"
)
rglmesh <- toRGL(mesh, segments = t(mesh[["edges"]]))
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.9)
shade3d(rglmesh, color = "darkred")
```

`torusMesh` *Torus mesh*

Description

Triangle mesh of a torus.

Usage

```
torusMesh(R, r, nu = 50, nv = 30, rgl = TRUE)
```

Arguments

`R, r`	major and minor radii, positive numbers
`nu, nv`	numbers of subdivisions, integers (at least 3)
`rgl`	Boolean, whether to return a **rgl** mesh

Value

A triangle **rgl** mesh (class `mesh3d`) if `rgl=TRUE`, otherwise a `cgalMesh` list (vertices, faces, and normals).

Examples

```
library(MeshesOperations)
library(rgl)
mesh <- torusMesh(R = 3, r = 1)
open3d(windowRect = c(50, 50, 562, 562))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "green")
wire3d(mesh)
```

truncatedIcosahedron *A 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

```
truncatedIcosahedron
```

Format

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

writeMeshFile *Export mesh to a file*

Description

Export a mesh to a file.

Usage

```
writeMeshFile(mesh, filename, precision = 17L, binary = FALSE)
```

Arguments

`mesh`	a mesh given either as a list containing (at least) the fields `vertices` and `faces`, otherwise a **rgl** mesh (i.e. a `mesh3d` object)
`filename`	name of the file to be written, with extension `stl`, `ply`, `obj` or `off`
`precision`	positive integer, number of decimal digits for the vertices
`binary`	Boolean, whether to write a binary file or an ASCII file

Value

No value, just generates the file.

Index