Package 'cgalMeshes'

May 18, 2023

Type Package

Title R6 Based Utilities for 3D Meshes using 'CGAL'

Version 2.1.0

Maintainer Stéphane Laurent <laurent_step@outlook.fr>

Description Provides some utilities for 3D meshes: clipping of a mesh to the volume bounded by another mesh, decomposition into convex parts, distance between a mesh and a point, Hausdorff distance between two meshes, triangulation, geodesic distance, Boolean operations (intersection, union, difference), connected components, subdivision algorithms, random sampling on a mesh, volume, area, and centroid. Also provides two algorithms for surface reconstruction from a cloud of points. Meshes are represented by R6 classes. All algorithms are performed by the 'C++' library 'CGAL' (<https://www.cgal.org/>).

License GPL-3

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

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

Depends R (>= 2.10)

Imports data.table, grDevices, methods, onion, R6, Rcpp (>= 1.0.9), rgl, tools, utils

Suggests randomcoloR, rmarchingcubes, spray, viridisLite

LinkingTo BH, Rcpp, RcppCGAL, RcppColors, RcppEigen

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.2.3

SystemRequirements C++ 17, gmp, mpfr

NeedsCompilation yes

Author Stéphane Laurent [aut, cre]

Repository CRAN

Date/Publication 2023-05-18 10:00:09 UTC

R topics documented:

`AFSreconstruction` *Advancing front surface reconstruction*

Description

Reconstruction of a surface from a cloud of 3D points.

Usage

```
AFSreconstruction(points, jetSmoothing = NULL)
```

Arguments

`points`	numeric matrix which stores the points, one point per row
`jetSmoothing`	if not `NULL`, must be an integer higher than two, and then the points 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

Details

See Advancing Front Surface Reconstruction.

Value

A cgalMesh object.

Examples

```
library(cgalMeshes)
data(bunny, package = "onion")
mesh <- AFSreconstruction(bunny)
rglMesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512))
shade3d(rglMesh, color = "firebrick")

# jet smoothing example ####
library(cgalMeshes)
# no smoothing
mesh1 <- AFSreconstruction(SolidMobiusStrip)
mesh1$computeNormals()
rglMesh1 <- mesh1$getMesh()
# jet smoothing
mesh2 <- AFSreconstruction(SolidMobiusStrip, jetSmoothing = 30)
mesh2$computeNormals()
rglMesh2 <- mesh2$getMesh()
# plot
library(rgl)
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(20, -40, zoom = 0.85)
shade3d(rglMesh1, color = "gold")
next3d()
view3d(20, -40, zoom = 0.85)
shade3d(rglMesh2, color = "gold")
```

---

cgalMesh *R6 class to represent a CGAL mesh*

Description

R6 class to represent a CGAL mesh.

Methods

Public methods:

- cgalMesh$new()
- cgalMesh$print()
- cgalMesh$area()
- cgalMesh$assignFaceColors()
- cgalMesh$assignFaceScalars()

- cgalMesh$assignNormals()
- cgalMesh$assignVertexColors()
- cgalMesh$assignVertexScalars()
- cgalMesh$boundingBox()
- cgalMesh$boundsVolume()
- cgalMesh$CatmullClark()
- cgalMesh$centroid()
- cgalMesh$clip()
- cgalMesh$clipToPlane()
- cgalMesh$clipToIsoCuboid()
- cgalMesh$collectGarbage()
- cgalMesh$computeNormals()
- cgalMesh$connectedComponents()
- cgalMesh$convexParts()
- cgalMesh$copy()
- cgalMesh$distance()
- cgalMesh$DooSabin()
- cgalMesh$facesAroundVertex()
- cgalMesh$fair()
- cgalMesh$fillBoundaryHole()
- cgalMesh$fillFaceColors()
- cgalMesh$filterMesh()
- cgalMesh$fixManifoldness()
- cgalMesh$geoDists()
- cgalMesh$getBorders()
- cgalMesh$getEdges()
- cgalMesh$getFaces()
- cgalMesh$getFacesInfo()
- cgalMesh$getFaceColors()
- cgalMesh$getFaceScalars()
- cgalMesh$getVertexColors()
- cgalMesh$getVertexScalars()
- cgalMesh$getNormals()
- cgalMesh$getMesh()
- cgalMesh$getVertices()
- cgalMesh$HausdorffApproximate()
- cgalMesh$HausdorffEstimate()
- cgalMesh$intersection()
- cgalMesh$isClosed()
- cgalMesh$isotropicRemeshing()
- cgalMesh$isOutwardOriented()
- cgalMesh$isTriangle()

- cgalMesh$isValid()
- cgalMesh$isValidFaceGraph()
- cgalMesh$isValidHalfedgeGraph()
- cgalMesh$isValidPolygonMesh()
- cgalMesh$LoopSubdivision()
- cgalMesh$merge()
- cgalMesh$orientToBoundVolume()
- cgalMesh$removeSelfIntersections()
- cgalMesh$reverseOrientation()
- cgalMesh$sample()
- cgalMesh$selfIntersects()
- cgalMesh$sharpEdges()
- cgalMesh$Sqrt3Subdivision()
- cgalMesh$subtract()
- cgalMesh$triangulate()
- cgalMesh$union()
- cgalMesh$volume()
- cgalMesh$whereIs()
- cgalMesh$writeMeshFile()

Method `new()`**:** Creates a new `cgalMesh` object.

Usage:

```
cgalMesh$new(mesh, vertices, faces, normals = NULL, clean = FALSE)
```

Arguments:

`mesh` there are four possibilities for this argument: it can be missing, in which case the arguments `vertices` and `faces` must be given, or it can be the path to a mesh file (accepted formats: `off`, `obj`, `stl`, `ply`, `ts`, `vtp`), or it can be a **rgl** mesh (i.e. a `mesh3d` object), or it can be a list containing (at least) the fields `vertices` (numeric matrix with three columns) and `faces` (matrix of integers or list of vectors of integers), and optionally a field `normals` (numeric matrix with three columns); if this argument is a **rgl** mesh containing some colors, these colors will be assigned to the created `cgalMesh` object

`vertices` if `mesh` is missing, must be a numeric matrix with three columns

`faces` if `mesh` is missing, must be either a matrix of integers (each row gives the vertex indices of a face) or a list of vectors of integers (each one gives the vertex indices of a face)

`normals` if `mesh` is missing, must be `NULL` or a numeric matrix with three columns and as many rows as vertices

`clean` Boolean value indicating whether to clean the mesh (merge duplicated vertices and duplicated faces, remove isolated vertices); set to `FALSE` if you know your mesh is already clean

Returns: A `cgalMesh` object.

Examples:

```
library(cgalMeshes)
meshFile <- system.file(
  "extdata", "bigPolyhedron.off", package = "cgalMeshes"
)
mesh <- cgalMesh$new(meshFile)
rglmesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "tomato")
plotEdges(
  mesh$getVertices(), mesh$getEdges(), color = "darkred"
)

# this one has colors: ####
meshFile <- system.file(
  "extdata", "pentagrammicDipyramid.ply", package = "cgalMeshes"
)
mesh <- cgalMesh$new(meshFile)
rmesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.85)
shade3d(rmesh, meshColor = "faces")
```

Method `print()`**:** Print a `cgalMesh` object.

Usage:

```
cgalMesh$print(...)
```

Arguments:

`...` ignored

Returns: No value returned, just prints some information about the mesh.

Method `area()`**:** Compute the area of the mesh. The mesh must be triangle and must not self-intersect.

Usage:

```
cgalMesh$area()
```

Returns: A number, the mesh area.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$area()
```

Method `assignFaceColors()`**:** Assign colors (or any character strings) to the faces of the mesh.

Usage:

```
cgalMesh$assignFaceColors(colors)
```

Arguments:

`colors` a character vector whose length equals the number of faces, or a single character string to be assigned to each face

Returns: The current `cgalMesh` object, invisibly.

Method `assignFaceScalars()`**:** Assign scalars to the faces of the mesh.

Usage:

```
cgalMesh$assignFaceScalars(scalars)
```

Arguments:

`scalars` a numeric vector whose length equals the number of faces

Returns: The current `cgalMesh` object, invisibly.

Method `assignNormals()`**:** Assign per-vertex normals to the mesh.

Usage:

```
cgalMesh$assignNormals(normals)
```

Arguments:

`normals` a numeric matrix with three columns and as many rows as the number of vertices

Returns: The current `cgalMesh` object, invisibly.

Method `assignVertexColors()`**:** Assign colors (or any character strings) to the vertices of the mesh.

Usage:

```
cgalMesh$assignVertexColors(colors)
```

Arguments:

`colors` a character vector whose length equals the number of vertices

Returns: The current `cgalMesh` object, invisibly.

Method `assignVertexScalars()`**:** Assign scalars to the vertices of the mesh.

Usage:

```
cgalMesh$assignVertexScalars(scalars)
```

Arguments:

`scalars` a numeric vector whose length equals the number of vertices

Returns: The current `cgalMesh` object, invisibly.

Method `boundingBox()`**:** Bounding box of the mesh.

Usage:

```
cgalMesh$boundingBox()
```

Returns: A list containing the smallest corner point and the largest corner point of the bounding box, named `lcorner` and `ucorner` respectively. Use `isoCuboidMesh` to get a mesh of this bounding box.

Examples:

```
library(cgalMeshes)
library(rgl)
rmesh <- cyclideMesh(a = 97, c = 32, mu = 57)
mesh <- cgalMesh$new(rmesh)
bbox <- mesh$boundingBox()
bxmesh <- isoCuboidMesh(bbox[["lcorner"]], bbox[["ucorner"]])
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, -60)
shade3d(rmesh, color = "gold")
wire3d(bxmesh, color = "black")
```

Method `boundsVolume()`**:** Check whether the mesh bounds a volume. The mesh must be triangle.

Usage:

```
cgalMesh$boundsVolume()
```

Returns: A Boolean value, whether the mesh bounds a volume.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$boundsVolume() # TRUE
mesh$reverseOrientation()
mesh$boundsVolume() # TRUE
```

Method `CatmullClark()`**:** Performs the Catmull-Clark subdivision and deformation. The mesh must be triangle.

Usage:

```
cgalMesh$CatmullClark(iterations = 1)
```

Arguments:

`iterations` number of iterations

Returns: The modified reference mesh, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$CatmullClark(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "red")
wire3d(hopfMesh, color = "black")
next3d()
```

```
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "red")
wire3d(rmesh, color = "black")
```

Method `centroid()`**:** Centroid of the mesh. The mesh must be triangle.

Usage:

```
cgalMesh$centroid()
```

Returns: The Cartesian coordinates of the centroid of the mesh.

Examples:

```
library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(icosahedron3d())
mesh$centroid()
```

Method `clip()`**:** Clip mesh to the volume bounded by another mesh. The mesh must be triangle. Face properties (colors and scalars) are preserved. **WARNING**: the reference mesh is then replaced by its clipped version.

Usage:

```
cgalMesh$clip(clipper, clipVolume)
```

Arguments:

`clipper` a `cgalMesh` object; it must represent a closed triangle mesh which doesn't self-intersect

`clipVolume` Boolean, whether the clipping has to be done on the volume bounded by the reference mesh rather than on its surface (i.e. the reference mesh will be kept closed if it is closed); if `TRUE`, the mesh to be clipped must not self-intersect

Returns: The reference mesh is always replaced by the result of the clipping. If `clipVolume=TRUE`, this function returns two `cgalMesh` objects: the two parts of the clipped mesh contained in the reference mesh and the clipping mesh respectively. Otherwise, this function returns the modified reference mesh.

Examples:

```
# cube clipped to sphere ####
library(cgalMeshes)
library(rgl)
mesh    <- cgalMesh$new(cube3d())$triangulate()
clipper <- cgalMesh$new(sphereMesh(r= sqrt(2)))
mesh$assignFaceColors("blue")
clipper$assignFaceColors("red")
meshes <- mesh$clip(clipper, clipVolume = TRUE)
mesh1 <- meshes[[1]]
mesh2 <- meshes[[2]]
mesh2$computeNormals()
rglmesh1 <- mesh1$getMesh()
rglmesh2 <- mesh2$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(45, 45, zoom = 0.9)
shade3d(rglmesh1, meshColor = "faces")
```

```
shade3d(rglmesh2, meshColor = "faces")

# Togliatti surface clipped to a ball ####
library(rmarchingcubes)
library(rgl)
library(cgalMeshes)
# Togliatti surface equation: f(x,y,z) = 0
f <- function(x, y, z) {
  64*(x-1) *
   (x^4 - 4*x^3 - 10*x^2*y^2 - 4*x^2 + 16*x - 20*x*y^2 + 5*y^4 + 16 - 20*y^2) -
    5*sqrt(5-sqrt(5))*(2*z - sqrt(5-sqrt(5))) *
    (4*(x^2 + y^2 - z^2) + (1 + 3*sqrt(5)))^2
}
# grid
n <- 200L
x <- y <- seq(-5, 5, length.out = n)
z <- seq(-4, 4, length.out = n)
Grid <- expand.grid(X = x, Y = y, Z = z)
# calculate voxel
voxel <- array(with(Grid, 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)
rglMesh <- tmesh3d(
  vertices = t(contour_shape[["vertices"]]),
  indices  = t(contour_shape[["triangles"]]),
  normals  = contour_shape[["normals"]],
  homogeneous = FALSE
)
# make CGAL mesh
mesh <- cgalMesh$new(rglMesh)
# clip to sphere of radius 4.8
sphere <- sphereMesh(r = 4.8)
clipper <- cgalMesh$new(sphere)
mesh$clip(clipper, clipVolume = FALSE)
rglClippedMesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 900, 450))
mfrow3d(1L, 2L)
view3d(0, -70, zoom = 0.8)
shade3d(rglMesh, color = "firebrick")
next3d()
view3d(0, -70, zoom = 0.8)
shade3d(rglClippedMesh, color = "firebrick")
shade3d(sphere, color = "yellow", alpha = 0.15)
```

Method `clipToPlane()`**:** Clip the mesh to a plane. The mesh must be triangle. Face properties

(colors, scalars) are preserved.

Usage:

```
cgalMesh$clipToPlane(planePoint, planeNormal, clipVolume)
```

Arguments:

`planePoint` numeric vector of length three, a point belonging to the plane

`planeNormal` numeric vector of length three, a vector orthogonal to the plane

`clipVolume` Boolean, whether to clip on the volume

Returns: If `clipVolume=FALSE`, the modified reference mesh is invisibly returned. If `clipVolume=TRUE`, a list of two `cgalMesh` objects is returned: the first one is the part of the clipped mesh corresponding to the original mesh, the second one is the part of the clipped mesh corresponding to the plane.

Examples:

```
library(cgalMeshes)
library(rgl)
rmesh <- sphereMesh()
mesh <- cgalMesh$new(rmesh)
nfaces <- nrow(mesh$getFaces())
if(require("randomcoloR")) {
  colors <-
    randomColor(nfaces, hue = "random", luminosity = "dark")
} else {
  colors <- rainbow(nfaces)
}
mesh$assignFaceColors(colors)
meshes <- mesh$clipToPlane(
  planePoint  = c(0, 0, 0),
  planeNormal = c(0, 0, 1),
  clipVolume = TRUE
)
mesh1 <- meshes[[1]]
mesh2 <- meshes[[2]]
mesh1$computeNormals()
rClippedMesh1 <- mesh1$getMesh()
rClippedMesh2 <- mesh2$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(70, 0)
shade3d(rClippedMesh1, meshColor = "faces")
shade3d(rClippedMesh2, color = "orange")
```

Method `clipToIsoCuboid()`**:** Clip the mesh to an iso-oriented cuboid. The mesh must be triangle. Face properties (colors, scalars) are preserved.

Usage:

```
cgalMesh$clipToIsoCuboid(lcorner, ucorner, clipVolume)
```

Arguments:

`lcorner, ucorner` two diagonally opposite vertices of the iso-oriented cuboid, the smallest and the largest (see `isoCuboidMesh`)

`clipVolume` Boolean, whether to clip on the volume

Returns: If `clipVolume=FALSE`, the modified reference mesh is invisibly returned. If `clipVolume=TRUE`, a list of two `cgalMesh` objects is returned: the first one is the part of the clipped mesh corresponding to the original mesh, the second one is the part of the clipped mesh corresponding to the cuboid.

Examples:

```
library(cgalMeshes)
library(rgl)
rmesh <- HopfTorusMesh(nu = 200, nv = 200)
mesh <- cgalMesh$new(rmesh)
mesh$assignFaceColors("orangered")
lcorner <- c(-7, -7, -5)
ucorner <- c(7, 6, 5)
bxmesh <- isoCuboidMesh(lcorner, ucorner)
mesh$clipToIsoCuboid(
  lcorner, ucorner, clipVolume = FALSE
)
mesh$computeNormals()
rClippedMesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(-40, 0)
shade3d(rClippedMesh, meshColor = "faces")
shade3d(bxmesh, color = "cyan", alpha = 0.3)
```

Method `collectGarbage()`**:** tmp.

Usage:

```
cgalMesh$collectGarbage()
```

Returns: tmp.

Method `computeNormals()`**:** Compute per-vertex normals of the mesh.

Usage:

```
cgalMesh$computeNormals()
```

Returns: The current `cgalMesh` object, invisibly. To get the normals, use the `getNormals` method.

Method `connectedComponents()`**:** Decomposition into connected components. All face properties (colors, scalars) and vertex properties (colors, scalars, normals) are preserved.

Usage:

```
cgalMesh$connectedComponents(triangulate = FALSE)
```

Arguments:

`triangulate` Boolean, whether to triangulate the connected components.

Returns: A list of `cgalMesh` objects, one for each connected component.

Examples:

```
library(cgalMeshes)
library(rmarchingcubes)
# isosurface function (slice of a seven-dimensional toratope)
f <- function(x, y, z, a) {
  (sqrt(
   (sqrt((sqrt((x*sin(a))^2 + (z*cos(a))^2) - 5)^2 + (y*sin(a))^2) - 2.5)^2 +
      (x*cos(a))^2) - 1.25
  )^2 + (sqrt((sqrt((z*sin(a))^2 + (y*cos(a))^2) - 2.5)^2) - 1.25)^2
}
# make grid
n <- 200L
x <- seq(-10, 10, len = n)
y <- seq(-10, 10, len = n)
z <- seq(-10, 10, len = n)
Grid <- expand.grid(X = x, Y = y, Z = z)
# compute isosurface
voxel <- array(with(Grid, f(X, Y, Z, a = pi/2)), dim = c(n, n, n))
isosurface <- contour3d(voxel, level = 0.25, x = x, y = y, z = z)
# make CGAL mesh
mesh <- cgalMesh$new(
  vertices = isosurface[["vertices"]],
  faces = isosurface[["triangles"]],
  normals = isosurface[["normals"]]
)
# connected components
components <- mesh$connectedComponents()
ncc <- length(components)
# plot
library(rgl)
colors <- rainbow(ncc)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(30, 50)
for(i in 1L:ncc) {
  rglMesh <- components[[i]]$getMesh()
  shade3d(rglMesh, color = colors[i])
}
```

Method `convexParts()`**:** Decomposition into convex parts. The mesh must be triangle.

Usage:

```
cgalMesh$convexParts(triangulate = TRUE)
```

Arguments:

`triangulate` Boolean, whether to triangulate the convex parts

Returns: A list of `cgalMesh` objects, one for each convex part.

Examples:

```
library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(pentagrammicPrism)$triangulate()
```

```
cxparts <- mesh$convexParts()
ncxparts <- length(cxparts)
colors <- hcl.colors(ncxparts, palette = "plasma")
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(20, -20, zoom = 0.8)
for(i in 1L:ncxparts) {
  cxmesh <- cxparts[[i]]$getMesh()
  shade3d(cxmesh, color = colors[i])
}
```

Method `copy()`**:** Copy the mesh. This can change the order of the vertices.

Usage:

```
cgalMesh$copy()
```

Returns: A new `cgalMesh` object.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())
tmesh <- mesh$copy()$triangulate()
tmesh$isTriangle() # TRUE
mesh$isTriangle() # FALSE
```

Method `distance()`**:** Distance from one or more points to the mesh. The mesh must be triangle.

Usage:

```
cgalMesh$distance(points)
```

Arguments:

`points` either one point given as a numeric vector or several points given as a numeric matrix with three columns

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

Examples:

```
# cube example ####
library(cgalMeshes)
mesh <- cgalMesh$new(rgl::cube3d())$triangulate()
points <- rbind(
  c(0, 0, 0),
  c(1, 1, 1)
)
mesh$distance(points) # should be 1 and 0

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

Method `DooSabin()`**:** Performs the Doo-Sabin subdivision and deformation.

Usage:

```
cgalMesh$DooSabin(iterations = 1)
```

Arguments:

`iterations` number of iterations

Returns: The modified reference mesh, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$DooSabin(iterations = 2)
mesh$triangulate()
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "brown")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "brown")
wire3d(rmesh, color = "black")
```

Method `facesAroundVertex()`**:** Faces containing a given vertex.

Usage:

```
cgalMesh$facesAroundVertex(v)
```

Arguments:

`v` a vertex index

Returns: An integer vector, the indices of the faces containing `v`.

Method `fair()`**:** Fair a region of the mesh, i.e. make it smooth. The mesh must be triangle. This modifies the reference mesh. All face properties and vertex properties except the normals are preserved.

Usage:

```
cgalMesh$fair(indices)
```

Arguments:

`indices` the indices of the vertices in the region to be faired

Returns: The modified `cgalMesh` object.

Examples:

```
library(cgalMeshes)
rglHopf <- HopfTorusMesh(nu = 100, nv = 100)
hopf <- cgalMesh$new(rglHopf)
# squared norms of the vertices
normsq <- apply(hopf$getVertices(), 1L, crossprod)
# fair the region where the squared norm is > 19
indices <- which(normsq > 19)
hopf$fair(indices)
rglHopf_faired <- hopf$getMesh()
# plot
library(rgl)
open3d(windowRect = 50 + c(0, 0, 900, 450))
mfrow3d(1L, 2L)
view3d(0, 0, zoom = 0.8)
shade3d(rglHopf, color = "orangered")
next3d()
view3d(0, 0, zoom = 0.8)
shade3d(rglHopf_faired, color = "orangered")
```

Method `fillBoundaryHole()`**:** Fill a hole in the mesh. The face properties and the vertex properties are preserved.

Usage:

```
cgalMesh$fillBoundaryHole(border, fair = TRUE)
```

Arguments:

`border` index of the boundary cycle forming the hole to be filled; the boundary cycles can be identified with `$getBorders()`

`fair` Boolean, whether to fair (i.e. smooth) the filled hole

Returns: The filled hole as a `cgalMesh` object. The reference mesh is updated.

Examples:

```
library(cgalMeshes)
library(rgl)
# make a sphere
sphere <- sphereMesh()
mesh <- cgalMesh$new(sphere)
# make a hole in this sphere
mesh$clipToPlane(
  planePoint  = c(0.5, 0, 0),
  planeNormal = c(1, 0, 0),
  clipVolume  = FALSE
)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# fill the hole
hole <- mesh$fillBoundaryHole(1, fair = TRUE)
hole$computeNormals()
rhole <- hole$getMesh()
# plot
```

```
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(30, 30)
shade3d(rmesh, color = "red")
shade3d(rhole, color = "blue")
```

Method `fillFaceColors()`**:** Replace missing face colors with a color.

Usage:

```
cgalMesh$fillFaceColors(color)
```

Arguments:

`color` the color to replace the missing face colors

Returns: The reference mesh, invisibly.

Method `filterMesh()`**:** Split the mesh into two meshes according to a given set of selected faces. Face properties are preserved.

Usage:

```
cgalMesh$filterMesh(faces)
```

Arguments:

`faces` a vector of face indices

Returns: Two `cgalMesh` objects. The first one is the mesh consisting of the faces of the reference mesh given in the `faces` argument. The second one is the complementary mesh.

Examples:

```
library(rgl)
library(cgalMeshes)
rmesh <- HopfTorusMesh(nu = 80, nv = 60)
mesh <- cgalMesh$new(rmesh)
areas <- mesh$getFacesInfo()[, "area"]
bigFaces <- which(areas > 1)
meshes <- mesh$filterMesh(bigFaces)
rmesh1 <- meshes[[1]]$getMesh()
rmesh2 <- meshes[[2]]$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0)
shade3d(rmesh1, color = "red")
shade3d(rmesh2, color = "blue")
wire3d(rmesh)
```

Method `fixManifoldness()`**:** Duplicate non-manifold vertices.

Usage:

```
cgalMesh$fixManifoldness()
```

Returns: The possibly modified reference mesh, invisibly.

Method `geoDists()`**:** Estimated geodesic distances between vertices. The mesh must be triangle.

Usage:

```
cgalMesh$geoDists(index)
```

Arguments:

`index` index of the source vertex

Returns: The estimated geodesic distances from the source vertex to each vertex.

Examples:

```
# torus ####
library(cgalMeshes)
library(rgl)
rglmesh <- torusMesh(R = 3, r = 2, nu = 90, nv = 60)
mesh <- cgalMesh$new(rglmesh)
# estimated geodesic distances
geodists <- mesh$geoDists(1L)
# normalization to (0, 1)
geodists <- geodists / max(geodists)
# color each vertex according to its geodesic distance from the source
fcolor <- colorRamp(viridisLite::turbo(200L))
colors <- fcolor(geodists)
colors <- rgb(colors[, 1L], colors[, 2L], colors[, 3L], maxColorValue = 255)
rglmesh[["material"]] <- list("color" = colors)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
shade3d(rglmesh)
wire3d(rglmesh, color = "black")
if(!rgl.useNULL()) {
  play3d(spin3d(axis = c(1, 1, 1), rpm = 5), duration = 20)
}

# a trefoil knot (taken from `?rgl::cylinder3d`) ####
library(cgalMeshes)
library(rgl)
theta <- seq(0, 2*pi, length.out = 50L)
knot <- cylinder3d(
  center = cbind(
    sin(theta) + 2*sin(2*theta),
    2*sin(3*theta),
    cos(theta) - 2*cos(2*theta)),
  e1 = cbind(
    cos(theta) + 4*cos(2*theta),
    6*cos(3*theta),
    sin(theta) + 4*sin(2*theta)),
  radius = 0.8,
  closed = TRUE)
knot <- subdivision3d(knot, depth = 2)
mesh <- cgalMesh$new(knot)$triangulate()
rglmesh <- mesh$getMesh()
# estimated geodesic distances
geodists <- mesh$geoDists(1L)
```

```
# normalization to (0, 1)
geodists <- geodists / max(geodists)
# color each vertex according to its geodesic distance from the source
fcolor <- colorRamp(viridisLite::inferno(200L))
colors <- fcolor(geodists)
colors <- rgb(colors[, 1L], colors[, 2L], colors[, 3L], maxColorValue = 255)
rglmesh[["material"]] <- list("color" = colors)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
shade3d(rglmesh)
if(!rgl.useNULL()) {
  play3d(spin3d(axis = c(1, 1, 0), rpm = 5), duration = 20)
}
```

Method `getBorders()`**:** Get the borders of the mesh.

Usage:

```
cgalMesh$getBorders()
```

Returns: A list of matrices representing the boundary cycles. Each matrix has three columns: `"edge"`, an edge index, and `"v1"` and `"v2"`, the vertex indices of this edge.

Examples:

```
library(cgalMeshes)
library(rgl)
# isosurface f=0
f <- function(x, y, z) {
  sin_x <- sin(x)
  sin_y <- sin(y)
  sin_z <- sin(z)
  cos_x <- cos(x)
  cos_y <- cos(y)
  cos_z <- cos(z)
  d <- sqrt(
    (-sin_x * sin_y + cos_x * cos_z) ** 2
    + (-sin_y * sin_z + cos_y * cos_x) ** 2
    + (-sin_z * sin_x + cos_z * cos_y) ** 2
  )
  (
    cos(
      x - (-sin_x * sin_y + cos_x * cos_z) / d
    )
    * sin(
      y - (-sin_y * sin_z + cos_y * cos_x) / d
    )
    + cos(
      y - (-sin_y * sin_z + cos_y * cos_x) / d
    )
    * sin(
      z - (-sin_z * sin_x + cos_z * cos_y)/ d
```

```
    )
    + cos(
      z - (-sin_z * sin_x + cos_z * cos_y) / d
    )
    * sin(
      x - (-sin_x * sin_y + cos_x * cos_z) / d
    )
  ) * (
    (
      cos(
        x + (-sin_x * sin_y + cos_x * cos_z) / d
      )
      * sin(
        y + (-sin_y * sin_z + cos_y * cos_x) / d
      )
      + cos(
        y + (-sin_y * sin_z + cos_y * cos_x) / d
      )
      * sin(
        z + (-sin_z * sin_x + cos_z * cos_y) / d
      )
      + cos(
        z + (-sin_z * sin_x + cos_z * cos_y) / d
      )
      * sin(
        x + (-sin_x * sin_y + cos_x * cos_z) / d
      )
    )
  )
}
# construct the isosurface f=0
ngrid <- 200L
x <- y <- z <- seq(-8.1, 8.1, length.out = ngrid)
Grid <- expand.grid(X = x, Y = y, Z = z)
voxel <- array(
  with(Grid, f(X, Y, Z)), dim = c(ngrid, ngrid, ngrid)
)
library(rmarchingcubes)
contour_shape <- contour3d(
  griddata = voxel, level = 0,
  x = x, y = y, z = z
)
# make mesh
mesh <- cgalMesh$new(
  list(
    "vertices" = contour_shape[["vertices"]],
    "faces"    = contour_shape[["triangles"]]
  )
```

```
)
# clip the mesh to the ball of radius 8
spheremesh <- cgalMesh$new(sphereMesh(r = 8))
mesh$clip(spheremesh, clipVolume = FALSE)
# compute normals
mesh$computeNormals()
# we will plot the borders
borders <- mesh$getBorders()
# plot
rmesh <- mesh$getMesh()
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.7)
shade3d(rmesh, color = "darkred")
vertices <- mesh$getVertices()
for(border in borders){
  plotEdges(
    vertices, border[, c("v1", "v2")], color = "gold",
    lwd = 3, edgesAsTubes = FALSE, verticesAsSpheres = FALSE
  )
}
```

Method `getEdges()`**:** Get the edges of the mesh.

Usage:

```
cgalMesh$getEdges()
```

Returns: A dataframe with five columns; the first two ones give the vertex indices of each edge (one edge per row), the third one gives the lengths of each edge, the fourth one indicates whether the edges is a border edge, and the fifth one gives the dihedral angles in degrees between the two faces adjacent to each edge

Examples:

```
library(rgl)
mesh <- cgalMesh$new(dodecahedron3d())
head(mesh$getEdges())
```

Method `getFaces()`**:** Get the faces of the mesh.

Usage:

```
cgalMesh$getFaces()
```

Returns: The faces in a matrix if the mesh is triangle or quad, otherwise in a list.

Method `getFacesInfo()`**:** Get the centroids and the areas of the faces, for a triangle mesh only.

Usage:

```
cgalMesh$getFacesInfo()
```

Returns: A matrix with four columns: the first three ones provide the Cartesian coordinates of the centroids, the fourth one provides the areas.

Method `getFaceColors()`**:** Get the face colors (if there are).

Usage:

```
cgalMesh$getFaceColors()
```

Returns: The vector of colors (or any character vector) attached to the faces of the mesh, or NULL if nothing is assigned to the faces.

Method `getFaceScalars()`**:** Get the face scalars (if there are).

Usage:

```
cgalMesh$getFaceScalars()
```

Returns: The vector of scalars attached to the faces of the mesh, or NULL if nothing is assigned to the faces.

Method `getVertexColors()`**:** Get the vertex colors (if there are).

Usage:

```
cgalMesh$getVertexColors()
```

Returns: The vector of colors (or any character vector) attached to the vertices of the mesh, or NULL if nothing is assigned to the vertices.

Method `getVertexScalars()`**:** Get the vertex scalars (if there are).

Usage:

```
cgalMesh$getVertexScalars()
```

Returns: The vector of scalars attached to the vertices of the mesh, or NULL if nothing is assigned to the vertices.

Method `getNormals()`**:** Get the per-vertex normals (if there are).

Usage:

```
cgalMesh$getNormals()
```

Returns: The matrix of per-vertex normals if they have been given or computed (see computeNormals, or NULL otherwise.

Method `getMesh()`**:** Get the mesh.

Usage:

```
cgalMesh$getMesh(rgl = TRUE, ...)
```

Arguments:

rgl	Boolean, whether to return a **rgl** mesh if possible, i.e. if the mesh only has triangular or quadrilateral faces
...	arguments passed to mesh3d (if a **rgl** mesh is returned)

Returns: A **rgl** mesh or a list with two or three fields: vertices, faces, and normals if XXXXXXXXXXXXXXXXXXXXXXXXX the argument normals is set to TRUE

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$getMesh()
```

Method `getVertices()`**:** Get the vertices of the mesh.

Usage:

```
cgalMesh$getVertices()
```

Returns: The vertices in a matrix.

Method `HausdorffApproximate()`**:** Approximate Hausdorff distance between the reference mesh and another mesh.

Usage:

```
cgalMesh$HausdorffApproximate(mesh2, symmetric = TRUE)
```

Arguments:

`mesh2` a `cgalMesh` object

`symmetric` Boolean, whether to consider the symmetric Hausdorff distance.

Returns: A number. The algorithm uses some simulations and thus the result can vary.

Method `HausdorffEstimate()`**:** Estimate of Hausdorff distance between the reference mesh and another mesh.

Usage:

```
cgalMesh$HausdorffEstimate(mesh2, errorBound = 1e-04, symmetric = TRUE)
```

Arguments:

`mesh2` a `cgalMesh` object

`errorBound` a positive number, a bound on the error of the estimate

`symmetric` Boolean, whether to consider the symmetric Hausdorff distance.

Returns: A number.

Method `intersection()`**:** Intersection with another mesh.

Usage:

```
cgalMesh$intersection(mesh2)
```

Arguments:

`mesh2` a `cgalMesh` object

Returns: A `cgalMesh` object.

Examples:

```
library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
mesh1$triangulate()
mesh2$triangulate()
# intersection
imesh <- mesh1$intersection(mesh2)
rglimesh <- imesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(imesh$getEdges())
# plot
```

```
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglimesh, color = "red")
plotEdges(imesh$getVertices(), extEdges)
shade3d(rglmesh1, color = "yellow", alpha = 0.2)
shade3d(rglmesh2, color = "cyan", alpha = 0.2)
```

Method `isClosed()`**:** Check whether the mesh is closed.

Usage:

```
cgalMesh$isClosed()
```

Returns: A Boolean value, whether the mesh is closed.

Method `isotropicRemeshing()`**:** Isotropic remeshing.

Usage:

```
cgalMesh$isotropicRemeshing(targetEdgeLength, iterations = 1, relaxSteps = 1)
```

Arguments:

`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

Returns: The modified `cgalMesh` object, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(HopfTorusMesh(nu = 80, nv = 50))
mesh$isotropicRemeshing(targetEdgeLength = 0.7)
# squared norms of the vertices
normsq <- apply(mesh$getVertices(), 1L, crossprod)
# fair the region where the squared norm is > 19
mesh$fair(which(normsq > 19))
# plot
mesh$computeNormals()
rmesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0)
shade3d(rmesh, color = "maroon")
wire3d(rmesh)
```

Method `isOutwardOriented()`**:** Check whether the mesh is outward oriented. The mesh must be triangle.

Usage:

```
cgalMesh$isOutwardOriented()
```

Returns: A Boolean value, whether the mesh is outward oriented.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$isOutwardOriented() # TRUE
mesh$reverseOrientation()
mesh$isOutwardOriented() # FALSE
```

Method `isTriangle()`**:** Check whether the mesh is triangle.

Usage:

```
cgalMesh$isTriangle()
```

Returns: A Boolean value, whether the mesh is triangle.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())
mesh$isTriangle()
```

Method `isValid()`**:** Check whether the mesh is valid.

Usage:

```
cgalMesh$isValid()
```

Returns: A Boolean value, whether the mesh is valid.

Method `isValidFaceGraph()`**:** Check whether the mesh is valid.

Usage:

```
cgalMesh$isValidFaceGraph()
```

Returns: A Boolean value, whether the mesh is valid.

Method `isValidHalfedgeGraph()`**:** Check whether the mesh is valid.

Usage:

```
cgalMesh$isValidHalfedgeGraph()
```

Returns: A Boolean value, whether the mesh is valid.

Method `isValidPolygonMesh()`**:** Check whether the mesh is valid.

Usage:

```
cgalMesh$isValidPolygonMesh()
```

Returns: A Boolean value, whether the mesh is valid.

Method `LoopSubdivision()`**:** Performs the Loop subdivision and deformation.

Usage:

```
cgalMesh$LoopSubdivision(iterations = 1)
```

Arguments:

`iterations` number of iterations

Returns: The modified reference mesh, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$LoopSubdivision(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
```

```
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "gold")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "gold")
wire3d(rmesh, color = "black")
```

Method `merge()`**:** Merge the mesh and another mesh.

Usage:

```
cgalMesh$merge(mesh2)
```

Arguments:

`mesh2` a `cgalMesh` object

Returns: The updated reference mesh, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
mesh1 <- cgalMesh$new(sphereMesh())
mesh1$assignFaceColors("red")
mesh2 <- cgalMesh$new(sphereMesh(x = 3))
mesh2$assignFaceColors("blue")
mesh1$merge(mesh2)
rmesh <- mesh1$getMesh()
open3d(windowRect = c(50, 50, 562, 562))
shade3d(rmesh, meshColor = "faces")
```

Method `orientToBoundVolume()`**:** Reorient the connected components of the mesh in order that it bounds a volume. The mesh must be triangle.

Usage:

```
cgalMesh$orientToBoundVolume()
```

Returns: The modified `cgalMesh` object, invisibly. **WARNING**: even if you store the result in a new variable, the original mesh is modified.

Examples:

```
# two disjoint tetrahedra ####
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 <- cgalMesh$new(vertices = vertices, faces = faces)
mesh$boundsVolume() # FALSE
mesh$orientToBoundVolume()
mesh$boundsVolume() # TRUE
```

Method `removeSelfIntersections()`**:** Remove self-intersections (experimental). The mesh must be triangle.

Usage:

```
cgalMesh$removeSelfIntersections()
```

Returns: The modified `cgalMesh` object, invisibly.

Method `reverseOrientation()`**:** Reverse the orientation of the faces of the mesh.

Usage:

```
cgalMesh$reverseOrientation()
```

Returns: The modified `cgalMesh` object, invisibly. **WARNING**: even if you store the result in a new variable, the original mesh is modified.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$isOutwardOriented() # TRUE
mesh$reverseOrientation()
mesh$isOutwardOriented() # FALSE
```

Method `sample()`**:** Random sampling on the mesh. The mesh must be triangle.

Usage:

```
cgalMesh$sample(nsims)
```

Arguments:

`nsims` integer, the desired number of simulations

Returns: A `nsims` x `3` matrix containing the simulations.

Method `selfIntersects()`**:** Check whether the mesh self-intersects. The mesh must be triangle.

Usage:

```
cgalMesh$selfIntersects()
```

Returns: A Boolean value, whether the mesh self-intersects.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(dodecahedron3d())
mesh$selfIntersects()
```

Method `sharpEdges()`**:** Returns edges considered to be sharp according to the given angle bound.

Usage:

```
cgalMesh$sharpEdges(angleBound)
```

Arguments:

`angleBound` angle bound in degrees; an edge whose corresponding dihedral angle is smaller than this bound is considered as sharp

Returns: An integer matrix with three columns: `"edge"`, an edge index, and `"v1"` and `"v2"`, the vertex indices of this edge.

Examples:

```
library(cgalMeshes)
library(rgl)
# astroidal ellipsoid
f <- function(u, v) {
  rbind(
    cos(u)^3 * cos(v)^3,
    sin(u)^3 * cos(v)^3,
    sin(v)^3
  )
}
rmesh <- parametricMesh(
  f, urange = c(0, 2*pi), vrange = c(0, 2*pi),
  periodic = c(TRUE, TRUE), nu = 120, nv = 110
)
mesh <- cgalMesh$new(rmesh)
sharpEdges <- mesh$sharpEdges(30)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
shade3d(addNormals(rmesh), color = "chartreuse")
plotEdges(
  mesh$getVertices(), sharpEdges[, c("v1", "v2")],
  edgesAsTubes = FALSE, lwd = 5, verticesAsSpheres = FALSE
)
```

Method `Sqrt3Subdivision()`**:** Performs the 'Sqrt3' subdivision and deformation. The mesh must be triangle.

Usage:

```
cgalMesh$Sqrt3Subdivision(iterations = 1)
```

Arguments:

`iterations` number of iterations

Returns: The modified reference mesh, invisibly.

Examples:

```
library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$Sqrt3Subdivision(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "cyan")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "cyan")
wire3d(rmesh, color = "black")
```

Method `subtract()`**:** Subtract a mesh. Both meshes must be triangle. Face properties of the two meshes are copied to the new mesh. **WARNING**: this modifies the reference mesh and `mesh2`.

Usage:

```
cgalMesh$subtract(mesh2)
```

Arguments:

`mesh2` a `cgalMesh` object

Returns: A `cgalMesh` object, the difference between the reference mesh and `mesh2`. Both the reference mesh and `mesh2` are modified: they are corefined.

Examples:

```
library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
mesh1$triangulate()
mesh2$triangulate()
# assign colors
mesh1$assignFaceColors("red")
mesh2$assignFaceColors("navy")
# difference
mesh <- mesh1$subtract(mesh2)
```

```
rglmesh <- mesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(mesh$getEdges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, meshColor = "faces")
plotEdges(mesh$getVertices(), extEdges)
shade3d(rglmesh2, color = "cyan", alpha = 0.2)
```

Method `triangulate()`**:** Triangulate mesh.

Usage:

```
cgalMesh$triangulate()
```

Returns: The modified `cgalMesh` object, invisibly. **WARNING**: even if you store the result in a new variable, the original mesh is modified (see the example). You may want to triangulate a copy of the mesh; see the copy method.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())
mesh$isTriangle() # FALSE
# warning: triangulating the mesh modifies it
mesh$triangulate()
mesh$isTriangle() # TRUE
```

Method `union()`**:** Union with another mesh. Both meshes must be triangle. Face properties of the two united meshes are copied to the union mesh. **WARNING**: this modifies the reference mesh and `mesh2`.

Usage:

```
cgalMesh$union(mesh2)
```

Arguments:

`mesh2` a `cgalMesh` object

Returns: A `cgalMesh` object, the union of the reference mesh with `mesh2`. Both the reference mesh and `mesh2` are modified: they are corefined.

Examples:

```
library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
mesh1$triangulate()
mesh2$triangulate()
# assign a color to the faces; they will be retrieved in the union
mesh1$assignFaceColors("yellow")
```

```
mesh2$assignFaceColors("navy")
# union
umesh <- mesh1$union(mesh2)
rglumesh <- umesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(umesh$getEdges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglumesh, meshColor = "faces")
plotEdges(umesh$getVertices(), extEdges)
```

Method `volume()`**:** Compute the volume of the mesh. The mesh must be closed, triangle, and must not self-intersect.

Usage:

```
cgalMesh$volume()
```

Returns: A number, the mesh volume.

Examples:

```
library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$volume()
```

Method `whereIs()`**:** Locate points with respect to a closed triangle mesh.

Usage:

```
cgalMesh$whereIs(points)
```

Arguments:

`points` a numeric matrix with three columns

Returns: An integer vector taking values `-1` for outside, `1` for inside, and `0` if the point is on the boundary.

Examples:

```
library(cgalMeshes)
mesh <- cgalMesh$new(sphereMesh())
pt1 <- c(0, 0, 0) # inside
pt2 <- c(2, 0, 0) # outside
mesh$whereIs(rbind(pt1, pt2))
```

Method `writeMeshFile()`**:** Write mesh to a file.

Usage:

```
cgalMesh$writeMeshFile(filename, precision = 17, comments = "", binary = FALSE)
```

Arguments:

`filename` path to the file to be written, with extension `off` or `ply`

`precision` a positive integer, the desired number of decimal places

`comments` for `ply` extension only, a string to be included in the header of the PLY file

`binary` Boolean, for `ply` extension only, whether to write a binary `ply` file; the mesh properties (vertex colors, face colors, normals) are lost with this format

Returns: Nothing, just writes a file.

Examples

```
## ------------------------------------------------
## Method `cgalMesh$new`
## ------------------------------------------------

library(cgalMeshes)
meshFile <- system.file(
  "extdata", "bigPolyhedron.off", package = "cgalMeshes"
)
mesh <- cgalMesh$new(meshFile)
rglmesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "tomato")
plotEdges(
  mesh$getVertices(), mesh$getEdges(), color = "darkred"
)

# this one has colors: ####
meshFile <- system.file(
  "extdata", "pentagrammicDipyramid.ply", package = "cgalMeshes"
)
mesh <- cgalMesh$new(meshFile)
rmesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.85)
shade3d(rmesh, meshColor = "faces")

## ------------------------------------------------
## Method `cgalMesh$area`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$area()

## ------------------------------------------------
## Method `cgalMesh$boundingBox`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
rmesh <- cyclideMesh(a = 97, c = 32, mu = 57)
mesh <- cgalMesh$new(rmesh)
bbox <- mesh$boundingBox()
bxmesh <- isoCuboidMesh(bbox[["lcorner"]], bbox[["ucorner"]])
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, -60)
shade3d(rmesh, color = "gold")
wire3d(bxmesh, color = "black")
```

```
## ------------------------------------------------
## Method `cgalMesh$boundsVolume`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$boundsVolume() # TRUE
mesh$reverseOrientation()
mesh$boundsVolume() # TRUE

## ------------------------------------------------
## Method `cgalMesh$CatmullClark`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$CatmullClark(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "red")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "red")
wire3d(rmesh, color = "black")

## ------------------------------------------------
## Method `cgalMesh$centroid`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(icosahedron3d())
mesh$centroid()

## ------------------------------------------------
## Method `cgalMesh$clip`
## ------------------------------------------------

# cube clipped to sphere ####
library(cgalMeshes)
library(rgl)
mesh    <- cgalMesh$new(cube3d())$triangulate()
clipper <- cgalMesh$new(sphereMesh(r= sqrt(2)))
mesh$assignFaceColors("blue")
clipper$assignFaceColors("red")
meshes <- mesh$clip(clipper, clipVolume = TRUE)
```

```
mesh1 <- meshes[[1]]
mesh2 <- meshes[[2]]
mesh2$computeNormals()
rglmesh1 <- mesh1$getMesh()
rglmesh2 <- mesh2$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(45, 45, zoom = 0.9)
shade3d(rglmesh1, meshColor = "faces")
shade3d(rglmesh2, meshColor = "faces")

# Togliatti surface clipped to a ball ####
library(rmarchingcubes)
library(rgl)
library(cgalMeshes)
# Togliatti surface equation: f(x,y,z) = 0
f <- function(x, y, z) {
  64*(x-1) *
    (x^4 - 4*x^3 - 10*x^2*y^2 - 4*x^2 + 16*x - 20*x*y^2 + 5*y^4 + 16 - 20*y^2) -
    5*sqrt(5-sqrt(5))*(2*z - sqrt(5-sqrt(5))) *
    (4*(x^2 + y^2 - z^2) + (1 + 3*sqrt(5)))^2
}
# grid
n <- 200L
x <- y <- seq(-5, 5, length.out = n)
z <- seq(-4, 4, length.out = n)
Grid <- expand.grid(X = x, Y = y, Z = z)
# calculate voxel
voxel <- array(with(Grid, 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)
rglMesh <- tmesh3d(
  vertices = t(contour_shape[["vertices"]]),
  indices  = t(contour_shape[["triangles"]]),
  normals  = contour_shape[["normals"]],
  homogeneous = FALSE
)
# make CGAL mesh
mesh <- cgalMesh$new(rglMesh)
# clip to sphere of radius 4.8
sphere <- sphereMesh(r = 4.8)
clipper <- cgalMesh$new(sphere)
mesh$clip(clipper, clipVolume = FALSE)
rglClippedMesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 900, 450))
mfrow3d(1L, 2L)
view3d(0, -70, zoom = 0.8)
shade3d(rglMesh, color = "firebrick")
next3d()
view3d(0, -70, zoom = 0.8)
```

```
shade3d(rglClippedMesh, color = "firebrick")
shade3d(sphere, color = "yellow", alpha = 0.15)

## ------------------------------------------------
## Method `cgalMesh$clipToPlane`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
rmesh <- sphereMesh()
mesh <- cgalMesh$new(rmesh)
nfaces <- nrow(mesh$getFaces())
if(require("randomcoloR")) {
  colors <-
    randomColor(nfaces, hue = "random", luminosity = "dark")
} else {
  colors <- rainbow(nfaces)
}
mesh$assignFaceColors(colors)
meshes <- mesh$clipToPlane(
  planePoint  = c(0, 0, 0),
  planeNormal = c(0, 0, 1),
  clipVolume = TRUE
)
mesh1 <- meshes[[1]]
mesh2 <- meshes[[2]]
mesh1$computeNormals()
rClippedMesh1 <- mesh1$getMesh()
rClippedMesh2 <- mesh2$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(70, 0)
shade3d(rClippedMesh1, meshColor = "faces")
shade3d(rClippedMesh2, color = "orange")

## ------------------------------------------------
## Method `cgalMesh$clipToIsoCuboid`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
rmesh <- HopfTorusMesh(nu = 200, nv = 200)
mesh <- cgalMesh$new(rmesh)
mesh$assignFaceColors("orangered")
lcorner <- c(-7, -7, -5)
ucorner <- c(7, 6, 5)
bxmesh <- isoCuboidMesh(lcorner, ucorner)
mesh$clipToIsoCuboid(
  lcorner, ucorner, clipVolume = FALSE
)
mesh$computeNormals()
rClippedMesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(-40, 0)
```

```
shade3d(rClippedMesh, meshColor = "faces")
shade3d(bxmesh, color = "cyan", alpha = 0.3)

## ------------------------------------------------
## Method `cgalMesh$connectedComponents`
## ------------------------------------------------

library(cgalMeshes)
library(rmarchingcubes)
# isosurface function (slice of a seven-dimensional toratope)
f <- function(x, y, z, a) {
  (sqrt(
    (sqrt((sqrt((x*sin(a))^2 + (z*cos(a))^2) - 5)^2 + (y*sin(a))^2) - 2.5)^2 +
      (x*cos(a))^2) - 1.25
  )^2 + (sqrt((sqrt((z*sin(a))^2 + (y*cos(a))^2) - 2.5)^2) - 1.25)^2
}
# make grid
n <- 200L
x <- seq(-10, 10, len = n)
y <- seq(-10, 10, len = n)
z <- seq(-10, 10, len = n)
Grid <- expand.grid(X = x, Y = y, Z = z)
# compute isosurface
voxel <- array(with(Grid, f(X, Y, Z, a = pi/2)), dim = c(n, n, n))
isosurface <- contour3d(voxel, level = 0.25, x = x, y = y, z = z)
# make CGAL mesh
mesh <- cgalMesh$new(
  vertices = isosurface[["vertices"]],
  faces = isosurface[["triangles"]],
  normals = isosurface[["normals"]]
)
# connected components
components <- mesh$connectedComponents()
ncc <- length(components)
# plot
library(rgl)
colors <- rainbow(ncc)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(30, 50)
for(i in 1L:ncc) {
  rglMesh <- components[[i]]$getMesh()
  shade3d(rglMesh, color = colors[i])
}

## ------------------------------------------------
## Method `cgalMesh$convexParts`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(pentagrammicPrism)$triangulate()
cxparts <- mesh$convexParts()
ncxparts <- length(cxparts)
```

```
colors <- hcl.colors(ncxparts, palette = "plasma")
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(20, -20, zoom = 0.8)
for(i in 1L:ncxparts) {
  cxmesh <- cxparts[[i]]$getMesh()
  shade3d(cxmesh, color = colors[i])
}

## ------------------------------------------------
## Method `cgalMesh$copy`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())
tmesh <- mesh$copy()$triangulate()
tmesh$isTriangle() # TRUE
mesh$isTriangle() # FALSE

## ------------------------------------------------
## Method `cgalMesh$distance`
## ------------------------------------------------

# cube example ####
library(cgalMeshes)
mesh <- cgalMesh$new(rgl::cube3d())$triangulate()
points <- rbind(
  c(0, 0, 0),
  c(1, 1, 1)
)
mesh$distance(points) # should be 1 and 0

# cyclide example ####
library(cgalMeshes)
a <- 100; c <- 30; mu <- 80
mesh <- cgalMesh$new(cyclideMesh(a, c, mu, nu = 100L, nv = 100L))
O2 <- c(c, 0, 0)
# should be a - mu = 20 (see ?cyclideMesh):
mesh$distance(O2)

## ------------------------------------------------
## Method `cgalMesh$DooSabin`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$DooSabin(iterations = 2)
mesh$triangulate()
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
```

```
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "brown")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "brown")
wire3d(rmesh, color = "black")

## ------------------------------------------------
## Method `cgalMesh$fair`
## ------------------------------------------------

library(cgalMeshes)
rglHopf <- HopfTorusMesh(nu = 100, nv = 100)
hopf <- cgalMesh$new(rglHopf)
# squared norms of the vertices
normsq <- apply(hopf$getVertices(), 1L, crossprod)
# fair the region where the squared norm is > 19
indices <- which(normsq > 19)
hopf$fair(indices)
rglHopf_faired <- hopf$getMesh()
# plot
library(rgl)
open3d(windowRect = 50 + c(0, 0, 900, 450))
mfrow3d(1L, 2L)
view3d(0, 0, zoom = 0.8)
shade3d(rglHopf, color = "orangered")
next3d()
view3d(0, 0, zoom = 0.8)
shade3d(rglHopf_faired, color = "orangered")

## ------------------------------------------------
## Method `cgalMesh$fillBoundaryHole`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# make a sphere
sphere <- sphereMesh()
mesh <- cgalMesh$new(sphere)
# make a hole in this sphere
mesh$clipToPlane(
  planePoint  = c(0.5, 0, 0),
  planeNormal = c(1, 0, 0),
  clipVolume  = FALSE
)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# fill the hole
hole <- mesh$fillBoundaryHole(1, fair = TRUE)
hole$computeNormals()
rhole <- hole$getMesh()
```

```
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(30, 30)
shade3d(rmesh, color = "red")
shade3d(rhole, color = "blue")

## ------------------------------------------------
## Method `cgalMesh$filterMesh`
## ------------------------------------------------

library(rgl)
library(cgalMeshes)
rmesh <- HopfTorusMesh(nu = 80, nv = 60)
mesh <- cgalMesh$new(rmesh)
areas <- mesh$getFacesInfo()[, "area"]
bigFaces <- which(areas > 1)
meshes <- mesh$filterMesh(bigFaces)
rmesh1 <- meshes[[1]]$getMesh()
rmesh2 <- meshes[[2]]$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0)
shade3d(rmesh1, color = "red")
shade3d(rmesh2, color = "blue")
wire3d(rmesh)

## ------------------------------------------------
## Method `cgalMesh$geoDists`
## ------------------------------------------------

# torus ####
library(cgalMeshes)
library(rgl)
rglmesh <- torusMesh(R = 3, r = 2, nu = 90, nv = 60)
mesh <- cgalMesh$new(rglmesh)
# estimated geodesic distances
geodists <- mesh$geoDists(1L)
# normalization to (0, 1)
geodists <- geodists / max(geodists)
# color each vertex according to its geodesic distance from the source
fcolor <- colorRamp(viridisLite::turbo(200L))
colors <- fcolor(geodists)
colors <- rgb(colors[, 1L], colors[, 2L], colors[, 3L], maxColorValue = 255)
rglmesh[["material"]] <- list("color" = colors)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
shade3d(rglmesh)
wire3d(rglmesh, color = "black")
if(!rgl.useNULL()) {
  play3d(spin3d(axis = c(1, 1, 1), rpm = 5), duration = 20)
}

# a trefoil knot (taken from `?rgl::cylinder3d`) ####
library(cgalMeshes)
```

```
library(rgl)
theta <- seq(0, 2*pi, length.out = 50L)
knot <- cylinder3d(
  center = cbind(
    sin(theta) + 2*sin(2*theta),
    2*sin(3*theta),
    cos(theta) - 2*cos(2*theta)),
  e1 = cbind(
    cos(theta) + 4*cos(2*theta),
    6*cos(3*theta),
    sin(theta) + 4*sin(2*theta)),
  radius = 0.8,
  closed = TRUE)
knot <- subdivision3d(knot, depth = 2)
mesh <- cgalMesh$new(knot)$triangulate()
rglmesh <- mesh$getMesh()
# estimated geodesic distances
geodists <- mesh$geoDists(1L)
# normalization to (0, 1)
geodists <- geodists / max(geodists)
# color each vertex according to its geodesic distance from the source
fcolor <- colorRamp(viridisLite::inferno(200L))
colors <- fcolor(geodists)
colors <- rgb(colors[, 1L], colors[, 2L], colors[, 3L], maxColorValue = 255)
rglmesh[["material"]] <- list("color" = colors)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
shade3d(rglmesh)
if(!rgl.useNULL()) {
  play3d(spin3d(axis = c(1, 1, 0), rpm = 5), duration = 20)
}

## ------------------------------------------------
## Method `cgalMesh$getBorders`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# isosurface f=0
f <- function(x, y, z) {
  sin_x <- sin(x)
  sin_y <- sin(y)
  sin_z <- sin(z)
  cos_x <- cos(x)
  cos_y <- cos(y)
  cos_z <- cos(z)
  d <- sqrt(
    (-sin_x * sin_y + cos_x * cos_z) ** 2
    + (-sin_y * sin_z + cos_y * cos_x) ** 2
    + (-sin_z * sin_x + cos_z * cos_y) ** 2
  )
  (
    cos(
```

```
      x - (-sin_x * sin_y + cos_x * cos_z) / d
    )
    * sin(
      y - (-sin_y * sin_z + cos_y * cos_x) / d
    )
    + cos(
      y - (-sin_y * sin_z + cos_y * cos_x) / d
    )
    * sin(
      z - (-sin_z * sin_x + cos_z * cos_y)/ d
    )
    + cos(
      z - (-sin_z * sin_x + cos_z * cos_y) / d
    )
    * sin(
      x - (-sin_x * sin_y + cos_x * cos_z) / d
    )
  ) * (
    (
      cos(
        x + (-sin_x * sin_y + cos_x * cos_z) / d
      )
      * sin(
        y + (-sin_y * sin_z + cos_y * cos_x) / d
      )
      + cos(
        y + (-sin_y * sin_z + cos_y * cos_x) / d
      )
      * sin(
        z + (-sin_z * sin_x + cos_z * cos_y) / d
      )
      + cos(
        z + (-sin_z * sin_x + cos_z * cos_y) / d
      )
      * sin(
        x + (-sin_x * sin_y + cos_x * cos_z) / d
      )
    )
  )
}
# construct the isosurface f=0
ngrid <- 200L
x <- y <- z <- seq(-8.1, 8.1, length.out = ngrid)
Grid <- expand.grid(X = x, Y = y, Z = z)
voxel <- array(
  with(Grid, f(X, Y, Z)), dim = c(ngrid, ngrid, ngrid)
)
library(rmarchingcubes)
contour_shape <- contour3d(
  griddata = voxel, level = 0,
  x = x, y = y, z = z
)
# make mesh
```

```
mesh <- cgalMesh$new(
  list(
    "vertices" = contour_shape[["vertices"]],
    "faces"    = contour_shape[["triangles"]]
  )
)
# clip the mesh to the ball of radius 8
spheremesh <- cgalMesh$new(sphereMesh(r = 8))
mesh$clip(spheremesh, clipVolume = FALSE)
# compute normals
mesh$computeNormals()
# we will plot the borders
borders <- mesh$getBorders()
# plot
rmesh <- mesh$getMesh()
open3d(windowRect = c(50, 50, 562, 562), zoom = 0.7)
shade3d(rmesh, color = "darkred")
vertices <- mesh$getVertices()
for(border in borders){
  plotEdges(
    vertices, border[, c("v1", "v2")], color = "gold",
    lwd = 3, edgesAsTubes = FALSE, verticesAsSpheres = FALSE
  )
}

## ------------------------------------------------
## Method `cgalMesh$getEdges`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(dodecahedron3d())
head(mesh$getEdges())

## ------------------------------------------------
## Method `cgalMesh$getMesh`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$getMesh()

## ------------------------------------------------
## Method `cgalMesh$intersection`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
```

```
mesh1$triangulate()
mesh2$triangulate()
# intersection
imesh <- mesh1$intersection(mesh2)
rglimesh <- imesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(imesh$getEdges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglimesh, color = "red")
plotEdges(imesh$getVertices(), extEdges)
shade3d(rglmesh1, color = "yellow", alpha = 0.2)
shade3d(rglmesh2, color = "cyan", alpha = 0.2)

## ------------------------------------------------
## Method `cgalMesh$isotropicRemeshing`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(HopfTorusMesh(nu = 80, nv = 50))
mesh$isotropicRemeshing(targetEdgeLength = 0.7)
# squared norms of the vertices
normsq <- apply(mesh$getVertices(), 1L, crossprod)
# fair the region where the squared norm is > 19
mesh$fair(which(normsq > 19))
# plot
mesh$computeNormals()
rmesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0)
shade3d(rmesh, color = "maroon")
wire3d(rmesh)

## ------------------------------------------------
## Method `cgalMesh$isOutwardOriented`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$isOutwardOriented() # TRUE
mesh$reverseOrientation()
mesh$isOutwardOriented() # FALSE

## ------------------------------------------------
## Method `cgalMesh$isTriangle`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())
mesh$isTriangle()

## ------------------------------------------------
```

```
## Method `cgalMesh$LoopSubdivision`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$LoopSubdivision(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "gold")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "gold")
wire3d(rmesh, color = "black")

## ------------------------------------------------
## Method `cgalMesh$merge`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
mesh1 <- cgalMesh$new(sphereMesh())
mesh1$assignFaceColors("red")
mesh2 <- cgalMesh$new(sphereMesh(x = 3))
mesh2$assignFaceColors("blue")
mesh1$merge(mesh2)
rmesh <- mesh1$getMesh()
open3d(windowRect = c(50, 50, 562, 562))
shade3d(rmesh, meshColor = "faces")

## ------------------------------------------------
## Method `cgalMesh$orientToBoundVolume`
## ------------------------------------------------

# two disjoint tetrahedra ####
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 <- cgalMesh$new(vertices = vertices, faces = faces)
mesh$boundsVolume() # FALSE
mesh$orientToBoundVolume()
mesh$boundsVolume() # TRUE

## ------------------------------------------------
## Method `cgalMesh$reverseOrientation`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(tetrahedron3d())
mesh$isOutwardOriented() # TRUE
mesh$reverseOrientation()
mesh$isOutwardOriented() # FALSE

## ------------------------------------------------
## Method `cgalMesh$selfIntersects`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(dodecahedron3d())
mesh$selfIntersects()

## ------------------------------------------------
## Method `cgalMesh$sharpEdges`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# astroidal ellipsoid
f <- function(u, v) {
  rbind(
    cos(u)^3 * cos(v)^3,
    sin(u)^3 * cos(v)^3,
    sin(v)^3
  )
}
rmesh <- parametricMesh(
  f, urange = c(0, 2*pi), vrange = c(0, 2*pi),
  periodic = c(TRUE, TRUE), nu = 120, nv = 110
)
mesh <- cgalMesh$new(rmesh)
sharpEdges <- mesh$sharpEdges(30)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
```

```
shade3d(addNormals(rmesh), color = "chartreuse")
plotEdges(
  mesh$getVertices(), sharpEdges[, c("v1", "v2")],
  edgesAsTubes = FALSE, lwd = 5, verticesAsSpheres = FALSE
)

## ------------------------------------------------
## Method `cgalMesh$Sqrt3Subdivision`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
hopfMesh <- HopfTorusMesh(nu = 80, nv = 40)
mesh <- cgalMesh$new(hopfMesh)
mesh$Sqrt3Subdivision(iterations = 2)
mesh$computeNormals()
rmesh <- mesh$getMesh()
# plot
open3d(windowRect = 50 + c(0, 0, 800, 400))
mfrow3d(1, 2)
view3d(0, 0, zoom = 0.9)
shade3d(hopfMesh, color = "cyan")
wire3d(hopfMesh, color = "black")
next3d()
view3d(0, 0, zoom = 0.9)
shade3d(rmesh, color = "cyan")
wire3d(rmesh, color = "black")

## ------------------------------------------------
## Method `cgalMesh$subtract`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
mesh1$triangulate()
mesh2$triangulate()
# assign colors
mesh1$assignFaceColors("red")
mesh2$assignFaceColors("navy")
# difference
mesh <- mesh1$subtract(mesh2)
rglmesh <- mesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(mesh$getEdges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, meshColor = "faces")
```

```
plotEdges(mesh$getVertices(), extEdges)
shade3d(rglmesh2, color = "cyan", alpha = 0.2)

## ------------------------------------------------
## Method `cgalMesh$triangulate`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())
mesh$isTriangle() # FALSE
# warning: triangulating the mesh modifies it
mesh$triangulate()
mesh$isTriangle() # TRUE

## ------------------------------------------------
## Method `cgalMesh$union`
## ------------------------------------------------

library(cgalMeshes)
library(rgl)
# take two cubes
rglmesh1 <- cube3d()
rglmesh2 <- translate3d(cube3d(), 1, 1, 1)
mesh1 <- cgalMesh$new(rglmesh1)
mesh2 <- cgalMesh$new(rglmesh2)
# the two meshes must be triangle
mesh1$triangulate()
mesh2$triangulate()
# assign a color to the faces; they will be retrieved in the union
mesh1$assignFaceColors("yellow")
mesh2$assignFaceColors("navy")
# union
umesh <- mesh1$union(mesh2)
rglumesh <- umesh$getMesh()
# extract edges for plotting
extEdges <- exteriorEdges(umesh$getEdges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglumesh, meshColor = "faces")
plotEdges(umesh$getVertices(), extEdges)

## ------------------------------------------------
## Method `cgalMesh$volume`
## ------------------------------------------------

library(rgl)
mesh <- cgalMesh$new(cube3d())$triangulate()
mesh$volume()

## ------------------------------------------------
## Method `cgalMesh$whereIs`
## ------------------------------------------------
```

```
library(cgalMeshes)
mesh <- cgalMesh$new(sphereMesh())
pt1 <- c(0, 0, 0) # inside
pt2 <- c(2, 0, 0) # outside
mesh$whereIs(rbind(pt1, pt2))
```

`cyclideMesh` *Cyclide mesh*

Description

Triangle mesh of a Dupin cyclide.

Usage

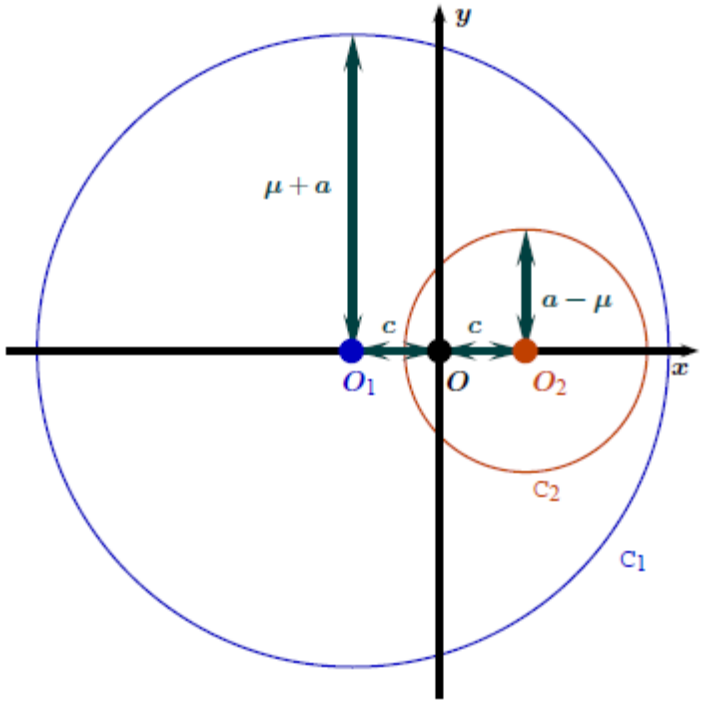
```
cyclideMesh(a, c, mu, nu = 90L, nv = 40L)
```

Arguments

`a, c, mu`	cyclide parameters, positive numbers such that `c < mu < a`
`nu, nv`	numbers of subdivisions, integers (at least 3)

Details

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



Value

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

Examples

```
library(cgalMeshes)
library(rgl)
mesh <- cyclideMesh(a = 97, c = 32, mu = 57)
sphere <- sphereMesh(x = 32, y = 0, z = 0, r = 40)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "chartreuse")
wire3d(mesh)
shade3d(sphere, color = "red")
wire3d(sphere)
```

---

exteriorEdges *Exterior edges of a mesh*

Description

Returns the edges of a mesh whose corresponding dihedral angles are not too flat.

Usage

```
exteriorEdges(edgesDF, angleThreshold = 1)
```

Arguments

`edgesDF`	the dataframe returned by the `edges` method of `cgalMesh`
`angleThreshold`	maximum deviation in degrees from the flat angle; for example if `angleThreshold=1`, then an edge is considered as exterior if its corresponding dihedral angle is lower than 179 or higher than 181

Value

An integer matrix giving the vertex indices of the exterior edges.

Note

Once you get the exterior edges, say in `extEdges`, then you can get the indices of the exterior vertices with `which(table(extEdges) != 2)`.

Examples

```
library(cgalMeshes)
library(rgl)
mesh <- cgalMesh$new(dodecahedron3d())
extEdges <- exteriorEdges(mesh$getEdges())
vertices <- mesh$getVertices()
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(dodecahedron3d(), color = "tomato")
plotEdges(vertices, extEdges)
```

---

gyroTriangle *Hyperbolic triangle*

Description

Mesh of a hyperbolic triangle.

Usage

```
gyroTriangle(A, B, C, s, iterations = 3)
```

Arguments

A, B, C	the vertices of the triangle
s	hyperbolic curvature, a positive number
iterations	number of iterations used to construct the mesh, at least 1

Value

A cgalMesh object.

Examples

```
library(cgalMeshes)
library(rgl)
mesh <- gyroTriangle(
  c(0, 0, 1), c(1, 0, 0), c(0, 1, 0), s = 0.7
)
mesh$computeNormals()
rmesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(20, 20)
shade3d(rmesh, color = "green")
wire3d(rmesh)

# hyperbolic icosahedron ####
library(cgalMeshes)
library(rgl)
```

```
icosahedron <- icosahedron3d()
vertices    <- icosahedron[["vb"]][-4L, ]
faces       <- icosahedron[["it"]]
colors <- hcl.colors(ncol(faces), palette = "Plasma")
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(15, -60, zoom = 0.7)
for(i in 1L:ncol(faces)) {
  triangle <- faces[, i]
  A <- vertices[, triangle[1L]]
  B <- vertices[, triangle[2L]]
  C <- vertices[, triangle[3L]]
  mesh <- gyroTriangle(A, B, C, s = 0.6)
  mesh$computeNormals()
  rmesh <- mesh$getMesh()
  shade3d(rmesh, color = colors[i])
  wire3d(rmesh)
}
```

`HopfTorusMesh` *Hopf torus mesh*

Description

Triangle mesh of a Hopf torus.

Usage

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

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)

Value

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

Examples

```
library(cgalMeshes)
library(rgl)
mesh <- HopfTorusMesh(nu = 90, nv = 90)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0, zoom = 0.75)
```

```
shade3d(mesh, color = "forestgreen")
wire3d(mesh)
mesh <- HopfTorusMesh(nu = 90, nv = 90, alpha = 1.5)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "yellowgreen")
wire3d(mesh)
```

---

isoCuboidMesh *Iso-oriented cuboid*

Description

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

Usage

```
isoCuboidMesh(lcorner, ucorner)
```

Arguments

lcorner	lower corner, a point whose coordinates must be lower than those of the upper corner
ucorner	upper corner, a point whose coordinates must be greater than those of the lower corner

Value

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

---

parametricMesh *Mesh of a parametric surface*

Description

Mesh of a parametric surface.

Usage

```
parametricMesh(f, urange, vrange, periodic = c(FALSE, FALSE), nu, nv)
```

Arguments

f	vectorized function of two variables returning a three-dimensional point
urange, vrange	the ranges of the two variables
periodic	two Boolean values, whether f is periodic in the first variable and in the second variable
nu, nv	numbers of subdivisions

Value

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

Examples

```
library(cgalMeshes)
library(rgl)
# Richmond surface
Richmond <- function(r, t){
  radius <- 0.5
  r <- r + 1/4
  exprho <- exp(r*(1 + 3*radius) - 2 - radius)
  u <- exprho * cospi(2*t)
  v <- exprho * sinpi(2*t)
  rbind(
    -v/(u*u+v*v) - u*u*v + v*v*v/3,
    3*u,
     u/(u*u+v*v) - u*v*v + u*u*u/3
  )
}
rmesh <- parametricMesh(
  Richmond, urange = c(0, 1), vrange = c(0, 1),
  periodic = c(FALSE, TRUE), nu = 100, nv = 100
)
rmesh <- addNormals(rmesh)
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(50, 10, zoom = 0.8)
shade3d(rmesh, color = "gold")
```

---

pentagrammicPrism *A 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

```
pentagrammicPrism
```

Format

A list (`vertices`, `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,
  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
spheresRadius	the radius of the spheres when verticesAsSpheres=TRUE
spheresColor	the color of the spheres when verticesAsSpheres=TRUE

Value

No value, just produces a 3D graphic.

Examples

```
library(cgalMeshes)
library(rgl)

mesh <- cgalMesh$new(pentagrammicPrism, clean = FALSE)
vertices <- mesh$getVertices()
```

```
edges <- mesh$getEdges()
extEdges <- exteriorEdges(edges)
tmesh <- mesh$triangulate()$getMesh()

open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(tmesh, color = "navy")
# we plot the exterior edges only
plotEdges(
  vertices, extEdges, color = "gold",
  tubesRadius = 0.02, spheresRadius = 0.02
)

# or only plot the edges whose corresponding dihedral angle is acute:
sharpEdges <- as.matrix(subset(edges, angle <= 91, select = c("i1", "i2")))
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(tmesh, color = "maroon")
plotEdges(
  vertices, sharpEdges, color = "darkred",
  tubesRadius = 0.02, spheresRadius = 0.02
)
```

`revolutionMesh` *Surface of revolution*

Description

Mesh of a surface of revolution. The axis of revolution is the z-axis.

Usage

```
revolutionMesh(x, y, n = 100)
```

Arguments

`x, y`	two numeric vectors of the same length defining the section to be revoluted
`n`	integer, the number of subdivisions used to construct the mesh

Value

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

Examples

```
library(cgalMeshes)
library(rgl)
t <- seq(0, 2*pi, length.out = 90)
x <- 4 + cos(t)/2
y <- sin(t)
rmesh <- revolutionMesh(x, y, n = 120)
rmesh <- addNormals(rmesh)
shade3d(rmesh, color = "red")
```

SolidMobiusStrip *A point cloud of a solid Möbius strip with noise*

Description

Some points forming a noisy solid Möbius strip.

Usage

```
SolidMobiusStrip
```

Format

A matrix with 10000 rows and 3 columns. The 10000 points have been sampled at random among more than 400000 points, obtained by the marching cubes algorithm, and some noise has been added to these 10000 points.

References

http://data.imaginary-exhibition.com/IMAGINARY-Moebiusband-Stephan-Klaus.pdf

sphereMesh *Sphere mesh*

Description

Triangle 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 (the mesh is obtained by iteratively subdividing the faces of an icosahedron)

Value

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

sphericalTriangle *Spherical triangle*

Description

Mesh of a spherical triangle.

Usage

```
sphericalTriangle(A, B, C, center = c(0, 0, 0), radius = 1, iterations = 4)
```

Arguments

A, B, C	the three vertices of the triangle, each given either as a pair of numbers: the polar angle (between 0 and pi) and the azimuthal angle (between 0 and 2pi), or as Cartesian coordinates
center	center of the sphere
radius	radius of the sphere, ignored if the three vertices are given as Cartesian coordinates
iterations	number of iterations used to construct the mesh of the sphere

Value

A cgalMesh object. The mesh has normals.

Examples

```
library(cgalMeshes)
library(rgl)
# orthant
A <- c(0, pi/2)
B <- c(pi/2, pi/2)
C <- c(0, 0)
mesh <- sphericalTriangle(A, B, C)
rmesh <- mesh$getMesh()
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(30, 30)
shade3d(rmesh, color = "red")
wire3d(rmesh)

# spherical icosahedron ####
library(cgalMeshes)
library(rgl)
icosahedron <- icosahedron3d()
vertices    <- icosahedron[["vb"]][-4L, ]
faces       <- icosahedron[["it"]]
colors <- rainbow(ncol(faces))
open3d(windowRect = 50 + c(0, 0, 512, 512))
```

```
for(i in 1L:ncol(faces)) {
  triangle <- faces[, i]
  A <- vertices[, triangle[1L]]
  B <- vertices[, triangle[2L]]
  C <- vertices[, triangle[3L]]
  mesh <- sphericalTriangle(A, B, C)
  rmesh <- mesh$getMesh()
  shade3d(rmesh, color = colors[i])
  wire3d(rmesh)
}
```

`SSSreconstruction` *Scale-space surface reconstruction*

Description

Reconstruction of a surface from a cloud of 3D points.

Usage

```
SSSreconstruction(
  points,
  scaleIterations = 1,
  neighbors = 12,
  samples = 300,
  separateShells = FALSE,
  forceManifold = TRUE,
  borderAngle = 45
)
```

Arguments

`points`	numeric matrix which stores the points, one point per row
`scaleIterations`	number of iterations used to increase the scale
`neighbors`	number of neighbors used to smooth the points cloud
`samples`	number of samples used to smooth the points cloud
`separateShells`	Boolean, whether to separate the shells
`forceManifold`	Boolean, whether to force a manifold output mesh
`borderAngle`	bound on the angle in degrees used to detect border edges

Details

See Scale-space Surface Reconstruction.

Value

A cgalMesh object.

Examples

```
library(cgalMeshes)
mesh <- SSSreconstruction(
  SolidMobiusStrip, scaleIterations = 4,
  forceManifold = TRUE, neighbors = 30
)
mesh$computeNormals()
rglMesh <- mesh$getMesh()
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(20, -40, zoom = 0.85)
shade3d(rglMesh, color = "tomato")
```

---

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 two fields: the field Vertices is a list of five matrices, each one giving the vertices of a tetrahedron; the field rglMeshes is the list of the five **rgl** meshes of the tetrahedra.

---

torusMesh *Torus mesh*

Description

Triangle mesh of a torus.

Usage

```
torusMesh(R, r, p1 = NULL, p2 = NULL, p3 = NULL, nu = 50, nv = 30)
```

Arguments

R, r	major and minor radii, positive numbers; R is ignored if p1, p2 and p3 are given
p1, p2, p3	three points or NULL; if not NULL, the function returns a mesh of the torus whose equator passes through these three points and with minor radius r; if NULL, the torus has equatorial plane z=0 and the z-axis as revolution axis
nu, nv	numbers of subdivisions, integers (at least 3)

Value

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

Examples

```
library(cgalMeshes)
library(rgl)
mesh <- torusMesh(R = 3, r = 1)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(0, 0, zoom = 0.75)
shade3d(mesh, color = "green")
wire3d(mesh)

# Villarceau circles ####
Villarceau <- function(beta, theta0, phi) {
  c(
    cos(theta0 + beta) * cos(phi),
    sin(theta0 + beta) * cos(phi),
    cos(beta) * sin(phi)
  ) / (1 - sin(beta) * sin(phi))
}
ncircles <- 30
if(require("randomcoloR")) {
  colors <-
    randomColor(ncircles, hue = "random", luminosity = "dark")
} else {
  colors <- rainbow(ncircles)
}
theta0_ <- seq(0, 2*pi, length.out = ncircles+1)[-1L]
phi <- 0.7
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.8)
for(i in seq_along(theta0_)) {
  theta0 <- theta0_[i]
  p1 <- Villarceau(0, theta0, phi)
  p2 <- Villarceau(2, theta0, phi)
  p3 <- Villarceau(4, theta0, phi)
  rmesh <- torusMesh(r = 0.05, p1 = p1, p2 = p2, p3 = p3)
  shade3d(rmesh, color = colors[i])
}
```

`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`.

Index