Package 'cgalMeshes'

November 9, 2022

Type Package

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

Version 1.0.0

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, triangulation, geodesic distance, Boolean operations (intersection, union, difference), connected components, volume, area, and centroid. Also provides an algorithm 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, methods, R6, Rcpp (>= 1.0.9), rgl

Suggests onion, rmarchingcubes, viridisLite

LinkingTo BH, Rcpp, RcppCGAL, RcppEigen

Encoding UTF-8

LazyData TRUE

RoxygenNote 7.2.1

SystemRequirements C++ 17, gmp, mpfr

NeedsCompilation yes

Author Stéphane Laurent [aut, cre]

Repository CRAN

Date/Publication 2022-11-09 11:50:12 UTC

R topics documented:

`AFSreconstruction` *Advancing front surface reconstruction*

Description

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

Usage

```
AFSreconstruction(points)
```

Arguments

`points`	numeric matrix which stores the points, one point per row

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)
shade3d(rglMesh, color = "firebrick")
```

`cgalMesh` *R6 class to represent a CGAL mesh*

Description

R6 class to represent a CGAL mesh.

Methods

Public methods:

- `cgalMesh$new()`
- `cgalMesh$print()`
- `cgalMesh$area()`
- `cgalMesh$boundsVolume()`
- `cgalMesh$centroid()`
- `cgalMesh$clip()`
- `cgalMesh$connectedComponents()`
- `cgalMesh$convexParts()`
- `cgalMesh$copy()`
- `cgalMesh$distance()`
- `cgalMesh$edges()`
- `cgalMesh$fair()`
- `cgalMesh$geoDists()`
- `cgalMesh$getMesh()`
- `cgalMesh$intersection()`
- `cgalMesh$isClosed()`
- `cgalMesh$isOutwardOriented()`
- `cgalMesh$isTriangle()`
- `cgalMesh$isValid()`
- `cgalMesh$orientToBoundVolume()`
- `cgalMesh$removeSelfIntersections()`
- `cgalMesh$reverseOrientation()`
- `cgalMesh$selfIntersects()`
- `cgalMesh$subtract()`
- `cgalMesh$triangulate()`
- `cgalMesh$union()`
- `cgalMesh$vertices()`
- `cgalMesh$volume()`
- `cgalMesh$writeMeshFile()`

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

Usage:

```
cgalMesh$new(mesh, vertices, faces, clean = TRUE)
```

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)

`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)

`clean` Boolean, no effect if the mesh is given by a file, otherwise it indicates 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(normals = FALSE)
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "tomato")
plotEdges(
  mesh$vertices(), mesh$edges(), color = "darkred"
)
```

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 `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 `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. **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 modified `cgalObject`.

Examples:

```
# cube clipped to sphere ####
library(cgalMeshes)
library(rgl)
mesh    <- cgalMesh$new(cube3d())$triangulate()
clipper <- cgalMesh$new(sphereMesh(r= sqrt(2)))
mesh$clip(clipper, clipVolume = TRUE)
rglmesh <- mesh$getMesh(normals = FALSE)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(45, 45, zoom = 0.9)
shade3d(rglmesh, col = "darkorange")
```

```
# 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 `connectedComponents()`**:** Decomposition into connected components.

Usage:

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

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$new(
  vertices = isosurface[["vertices"]], faces = isosurface[["triangles"]]
)
# connected components
components <- mesh$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(normals = FALSE)
  shade3d(cxmesh, color = colors[i])
}
```

Method `copy()`**:** Copy the mesh.

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

Usage:

```
cgalMesh$edges()
```

Returns: A dataframe with four 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, and the fourth one gives the dihedral angles in degrees between the two faces adjacent to each edge

Examples:

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

Method `fair()`**:** Fair a region of the mesh, i.e. make it smooth. The mesh must be triangle. This modifies the reference mesh.

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$new(rglHopf)
# squared norms of the vertices
normsq <- apply(hopf$vertices(), 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 `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 `getMesh()`**:** Get the mesh.

Usage:

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

Arguments:

`normals` Boolean, whether to return the per-vertex normals

`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 the argument `normals` is set to `TRUE`

Examples:

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

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(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(imesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglimesh, color = "red")
plotEdges(imesh$vertices(), 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 `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 `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 `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 `subtract()`**:** Subtract another mesh. Both meshes must be triangle.

Usage:

```
cgalMesh$subtract(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()
# difference
mesh <- mesh1$subtract(mesh2)
rglmesh <- mesh$getMesh(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(mesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "red")
plotEdges(mesh$vertices(), 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.

Usage:

```
cgalMesh$union(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()
# union
umesh <- mesh1$union(mesh2)
rglumesh <- umesh$getMesh(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(umesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglumesh, color = "red")
plotEdges(umesh$vertices(), extEdges)
```

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

Usage:

```
cgalMesh$vertices()
```

Returns: The vertices in a matrix.

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

Usage:

```
cgalMesh$writeMeshFile(filename, precision = 17, 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

`binary` Boolean, whether to write a binary PLY file if `filename` has the `ply` extension

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(normals = FALSE)
library(rgl)
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "tomato")
plotEdges(
  mesh$vertices(), mesh$edges(), color = "darkred"
)

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

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

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

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

## ------------------------------------------------
## 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$new(sphereMesh(r= sqrt(2)))
mesh$clip(clipper, clipVolume = TRUE)
rglmesh <- mesh$getMesh(normals = FALSE)
open3d(windowRect = 50 + c(0, 0, 512, 512))
view3d(45, 45, zoom = 0.9)
shade3d(rglmesh, col = "darkorange")

# 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$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"]]
)
# 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(normals = FALSE)
  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$edges`
## ------------------------------------------------

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

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

library(cgalMeshes)
rglHopf <- HopfTorusMesh(nu = 100, nv = 100)
hopf <- cgalMesh$new(rglHopf)
# squared norms of the vertices
normsq <- apply(hopf$vertices(), 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$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$getMesh`
## ------------------------------------------------

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

```
## ------------------------------------------------
## 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(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(imesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglimesh, color = "red")
plotEdges(imesh$vertices(), extEdges)
shade3d(rglmesh1, color = "yellow", alpha = 0.2)
shade3d(rglmesh2, color = "cyan", alpha = 0.2)

## ------------------------------------------------
## 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$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$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()
```

```
# difference
mesh <- mesh1$subtract(mesh2)
rglmesh <- mesh$getMesh(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(mesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglmesh, color = "red")
plotEdges(mesh$vertices(), 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()
# union
umesh <- mesh1$union(mesh2)
rglumesh <- umesh$getMesh(normals = FALSE)
# extract edges for plotting
extEdges <- exteriorEdges(umesh$edges())
# plot
open3d(windowRect = 50 + c(0, 0, 512, 512), zoom = 0.9)
shade3d(rglumesh, color = "red")
plotEdges(umesh$vertices(), extEdges)

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

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

`cyclideMesh`	*Cyclide mesh*

Description

Triangle mesh of a Dupin cyclide.

Usage

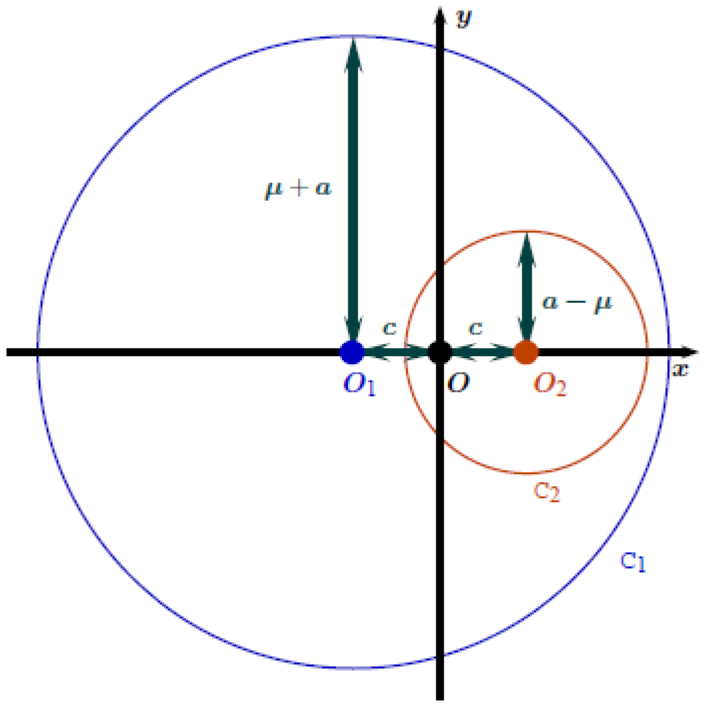
```
cyclideMesh(a, c, mu, nu = 90L, nv = 40L)
```

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

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)
```

`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,
  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, just produces a 3D graphic.

Examples

```
library(cgalMeshes)
library(rgl)

mesh <- cgalMesh$new(pentagrammicPrism, clean = FALSE)
vertices <- mesh$vertices()
edges <- mesh$edges()
extEdges <- exteriorEdges(edges)
tmesh <- mesh$triangulate()$getMesh(normals = FALSE)

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
)
```

`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`).

`torusMesh` *Torus mesh*

Description

Triangle mesh of a torus.

Usage

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

Arguments

`R, r`	major and minor radii, positive numbers
`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)
```

`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