Package ‘Rvcg’

September 8, 2021

Type Package

Title Manipulations of Triangular Meshes Based on the ‘VCGLIB’ API

Version 0.20.2

Date 2021-09-06

Description Operations on triangular meshes based on ‘VCGLIB’. This package integrates nicely with the R-package ‘rgl’ to render the meshes processed by ‘Rvcg’. The Visualization and Computer Graphics Library (VCG for short) is an open source portable C++ templated library for manipulation, processing and displaying with OpenGL of triangle and tetrahedral meshes. The library, composed by more than 100k lines of code, is released under the GPL license, and it is the base of most of the software tools of the Visual Computing Lab of the Italian National Research Council Institute ISTI <http://vcg.isti.cnr.it>, like ‘metro’ and ‘MeshLab’. The ‘VCGLIB’ source is pulled from trunk <https://github.com/cnr-isti-vclab/vcglib> and patched to work with options determined by the configure script as well as to work with the header files included by ‘RcppEigen’.

Depends R (>= 3.1.0)

Imports Rcpp, grDevices, stats, utils

Suggests Morpho, rgl

LinkingTo Rcpp, RcppEigen, RcppArmadillo

License GPL (>= 2) | file LICENSE

BugReports https://github.com/zarquon42b/Rvcg/issues

Copyright see files COPYRIGHTS for detailed information

LazyLoad yes

Biarch yes

URL https://github.com/zarquon42b/Rvcg,

https://github.com/cnr-isti-vclab/vcglib

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation yes
R topics documented:

Author  Stefan Schlager [aut, cre, cph],
        Girinon Francois [ctb],
        Tim Schaefer [ctb]

Maintainer  Stefan Schlager <zarquon42@gmail.com>

Repository  CRAN

Date/Publication  2021-09-08 13:40:05 UTC

R topics documented:

Rvcg-package ........................................... 3
ccheckFaceOrientation .................................. 4
dummyhead .................................................. 4
humface ..................................................... 5
meshInfo ..................................................... 5
meshIntegrity .............................................. 5
nfaces ....................................................... 6
nverts ....................................................... 6
setRays ..................................................... 7
vcgArea ..................................................... 7
vcgBallPivoting .......................................... 8
vcgBary ..................................................... 9
vcgBorder .................................................. 9
vcgClean ................................................... 10
vcgClose ................................................... 11
vcgCloseKd ................................................ 13
vcgCloseOnKdtreeFromBarycenters ..................... 15
vcgCreateKdtree .......................................... 16
vcgCreateKdtreeFromBarycenters ....................... 17
vcgCurve ................................................... 18
vcgDijkstra ............................................... 19
vcgFaceNormals .......................................... 20
vcgGeodesicPath ......................................... 20
vcgGeodist ................................................ 21
vcgGetEdge ............................................... 22
vcgImport .................................................. 23
vcgIsolated ............................................... 24
vcgIsosurface ........................................... 25
vcgIsotropicRemeshing .................................. 26
vcgKdtree ................................................. 27
vcgKmeans ............................................... 28
vcgMeshres ............................................... 29
vcgMetro ................................................... 30
vcgNonBorderEdge ........................................ 32
vcgObjWrite .............................................. 33
vcgOffWrite .............................................. 33
vcgPlyRead ............................................... 34
vcgPlyWrite .............................................. 35
Description

Provides meshing functionality from vcglib (meshlab) for R. E.g. mesh smoothing, mesh decimation, closest point search.

Details

Package: Rvcg
Type: Package
Version: 0.20.2
Date: 2021-09-06
License: GPL
LazyLoad: yes

Author(s)

Stefan Schlager
Maintainer: Stefan Schlager <zarquon42@gmail.com>

References

To be announced
checkFaceOrientation  
*check the orientation of a mesh*

**Description**  
check the orientation of a mesh assuming that expansion along normals increases centroid size  

**Usage**  
checkFaceOrientation(x, offset = NULL)  

**Arguments**  
- **x**: mesh of class mesh3d  
- **offset**: numeric; amount to offset the mesh along the vertex normals. If NULL a reasonable value will be estimated.  

**Details**  
assuming that a correctly (i.e outward) oriented mesh increases its centroid size when ’growing’ outwards, this function tests whether this is the case.  

**Value**  
returns TRUE if mesh is oriented correctly and FALSE otherwise  

**Examples**  
```r  
data(dummyhead)  
## now we invert faces inwards  
checkFaceOrientation(dummyhead.mesh)  
```  
```r  
if (requireNamespace("Morpho", quietly = TRUE)) {  
dummyinward <- Morpho::invertFaces(dummyhead.mesh)  
checkFaceOrientation(dummyinward)  
}  
```  

**dummyhead**  
*dummyhead - dummy head and landmarks*  

**Description**  
A triangular mesh representing a dummyhead - called by data(dummyhead)  

**Format**  
dummyhead.mesh: triangular mesh representing a dummyhead.  
dummyhead.lm: landmarks on mesh ’dummyhead’
### humface

**Example mesh and landmarks**

#### Description

A triangular mesh representing a human face - called by data(humface)

#### Format

- **humface**: triangular mesh representing a human face.
- **humfaceClean**: triangular mesh representing a human face but without errors or isolated pieces.
- **humface.lm**: landmarks on mesh 'humface' - called by data(humface)

### meshInfo

**print number of vertices and triangular faces of a mesh**

#### Description

print number of vertices and triangular faces of a mesh

#### Usage

`meshInfo(x)`

#### Arguments

- **x**: triangular mesh

### meshintegrity

**check if an object of class mesh3d contains valid data**

#### Description

checks for existance and validity of vertices, faces and vertex normals of an object of class "mesh3d"

#### Usage

`meshintegrity(mesh, facecheck = FALSE, normcheck = FALSE)`

#### Arguments

- **mesh**: object of class mesh3d
- **facecheck**: logical: check the existence of valid triangular faces
- **normcheck**: logical: check the existence of valid normals
Value

if mesh data are valid, the mesh is returned, otherwise it stops with an error message.

<table>
<thead>
<tr>
<th>nfaces</th>
<th>get number of vertices from a mesh</th>
</tr>
</thead>
</table>

Description

get number of vertices from a mesh

Usage

nfaces(x)

Arguments

x     triangular mesh

Value

integer: number of triangular faces

<table>
<thead>
<tr>
<th>nverts</th>
<th>get number of vertices from a mesh</th>
</tr>
</thead>
</table>

Description

get number of vertices from a mesh

Usage

nverts(x)

Arguments

x     triangular mesh

Value

integer: number of vertices
**setRays**

*helper function to create an object to be processed by vcgRaySearch*

**Description**

create a search structure from a matrix of coordinates and one of directional vectors to be processed by vcgRaySearch

**Usage**

```r
setRays(coords, dirs)
```

**Arguments**

- `coords`: k x 3 matrix (or a vector of length 3) containing the starting points of the rays
- `dirs`: k x 3 matrix (or a vector of length 3) containing the directions of the rays. The i-th row of `dirs` corresponds to the coordinate stored in the i-th row of `coords`

**Value**

an object of class "mesh3d" (without faces) and the vertices representing the starting points of the rays and the normals storing the directions.

---

**vcgArea**

*compute surface area of a triangular mesh*

**Description**

compute surface area of a triangular mesh

**Usage**

```r
vcgArea(mesh, perface = FALSE)
```

**Arguments**

- `mesh`: triangular mesh of class mesh3d
- `perface`: logical: if TRUE, a list containing the overall area, as well as the individual per-face area are reported.

**Value**

surface area of mesh

**Examples**

```r
data(humface)
vcgArea(humface)
```
Ball pivoting surface reconstruction

Usage

vcgBallPivoting(x, radius = 0, clustering = 0.2, angle = pi/2, deleteFaces = FALSE)

Arguments

x k x 3 matrix or object of class mesh3d
radius The radius of the ball pivoting (rolling) over the set of points. Gaps that are larger than the ball radius will not be filled; similarly the small pits that are smaller than the ball radius will be filled. 0 = autoguess.
clustering Clustering radius (fraction of ball radius). To avoid the creation of too small triangles, if a vertex is found too close to a previous one, it is clustered/merged with it.
age Angle threshold (radians). If we encounter a crease angle that is too large we should stop the ball rolling.
deleteFaces in case x is a mesh and deleteFaces=TRUE, existing faces will be deleted beforehand.

Value

triangular face of class mesh3d

Examples

if (requireNamespace("Morpho", quietly = TRUE)) {
  require(Morpho)
data(nose)
nosereko <- vcgBallPivoting(shortnose.lm)
}
vcgBary

get barycenters of all faces of a triangular mesh

Description
get barycenters of all faces of a triangular mesh

Usage
vcgBary(mesh)

Arguments
mesh triangular mesh of class "mesh3d"

Value
n x 3 matrix containing 3D-coordinates of the barycenters (where n is the number of faces in mesh.

Examples
data(humface)
bary <- vcgBary(humface)
## Not run:
require(rgl)
points3d(bary,col=2)
wire3d(humface)
## End(Not run)

vcgBorder

find all border vertices and faces of a triangular mesh

Description
Detect faces and vertices at the borders of a mesh and mark them.

Usage
vcgBorder(mesh)

Arguments
mesh triangular mesh of class "mesh3d"
vcgClean

Clean triangular surface meshes

Description

Apply several cleaning algorithms to surface meshes

Usage

vcgClean(mesh, sel = 0, tol = 0, silent = FALSE, iterate = FALSE)

Arguments

mesh triangular mesh of class 'mesh3d'
sel integer vector selecting cleaning type (see "details").
tol numeric value determining Vertex Displacement Ratio used for splitting non-manifold vertices.
silent logical, if TRUE no console output is issued.
iterate logical: if TRUE, vcgClean is repeatedly run until nothing more is to be cleaned (see details).
Details

the vector sel determines which operations are performed in which order. E.g. removing degenerate
tables may generate unreferenced vertices, thus the ordering of cleaning operations is important,
multiple calls are possible (sel=c(1,3,1) will remove unreferenced vertices twice). available options
are:

• 0 = only duplicated vertices and faces are removed
• 1 = remove unreferenced vertices
• 2 = Remove non-manifold Faces
• 3 = Remove degenerate faces
• 4 = Remove non-manifold vertices
• 5 = Split non-manifold vertices by threshold
• 6 = merge close vertices (radius=tol)
• 7 = coherently orient faces
CAVEAT: sel=6 will not work keep vertex colors

Value

cleaned mesh with an additional entry
remvert vector of length = number of vertices before cleaning. Entries = 1 indicate that
this vertex was removed; 0 otherwise.

Examples

data(humface)
cleanface <- humface
##add duplicated faces
cleanface$it <- cbind(cleanface$it, cleanface$it[,1:100])
## add duplicated vertices
cleanface$vb <- cbind(cleanface$vb,cleanface$vb[,1:100])
## ad unreferenced vertices
cleanface$vb <- cbind(cleanface$vb,rbind(matrix(rnorm(18),3,6),1))
cleanface <- vcgClean(cleanface, sel=1)

vcgClost

Project coordinates onto a target triangular surface mesh.

Description

For a set of 3D-coordinates/triangular mesh, the closest matches on a target surface are determined
and normals at as well as distances to that point are calculated.
Usage

vcgClost(
  x,
  mesh,
  sign = TRUE,
  barycentric = FALSE,
  smoothNormals = FALSE,
  borderchk = FALSE,
  tol = 0,
  facenormals = FALSE,
  ...
)

Arguments

  x       k x 3 matrix containing 3D-coordinates or object of class "mesh3d".
  mesh    triangular surface mesh stored as object of class "mesh3d".
  sign    logical: if TRUE, signed distances are returned.
  barycentric    logical: if TRUE, barycentric coordinates of the hit points are returned.
  smoothNormals    logical: if TRUE, laplacian smoothed normals are used.
  borderchk    logical: request checking if the hit face is at the border of the mesh.
  tol      maximum distance to search. If distance is beyond that, the original point will be kept and the distance set to NaN. If tol = 0, tol is set to 2*diagonal of the bounding box of mesh.
  facenormals    logical: if TRUE only the facenormal of the face the closest point has hit is returned, the weighted average of the surrounding vertex normals otherwise.
  ...          additional parameters, currently unused.

Value

returns an object of class "mesh3d" with:

  vb       4 x n matrix containing n vertices as homolougous coordinates.
  normals  4 x n matrix containing vertex normals.
  quality  numeric vector containing distances to target.
  it       3 x m integer matrix containing vertex indices forming triangular faces. Only available, when x is a mesh.
  border   integer vector of length n: if borderchk = TRUE, for each closest point the value will be 1 if the hit face is at the border of the target mesh and 0 otherwise.
  barycoords  3 x m Matrix containing barycentric coordinates of closest points; only available if barycentric=TRUE.
  faceptr  vector of face indeces on which the closest points are located
Note

If large part of the reference mesh are far away from the target surface, calculation can become very slow. In that case, the function `vcgClostKD` will be significantly faster.

Author(s)

Stefan Schlager

References


See Also

`vcgPlyRead`

Examples

data(humface)
clost <- vcgClost(humface.lm, humface)

---

### Description

For a set of 3D-coordinates/triangular mesh, the closest matches on a target surface are determined (by using KD-tree search) and normals at as well as distances to that point are calculated.

### Usage

```r
cvgClostKD(x, mesh, sign = TRUE, barycentric = FALSE, smoothNormals = FALSE, borderchk = FALSE, k = 50, n = 16, maxDepth = 64, angdev = NULL,
```
weightnorm = FALSE, 
facenormals = FALSE, 
threads = 1, 
...

Arguments

x                k x 3 matrix containing 3D-coordinates or object of class "mesh3d".
mesh            triangular surface mesh stored as object of class "mesh3d".
sign            logical: if TRUE, signed distances are returned.
barycentric      logical: if TRUE, barycentric coordinates of the hit points are returned.
smoothNormals   logical: if TRUE, laplacian smoothed normals are used.
borderchk       logical: request checking if the hit face is at the border of the mesh.
k                integer: check the kdtree for the k closest faces (using faces’ barycenters.
nofPoints       integer: number of points per cell in the kd-tree (don’t change unless you know what you are doing!)
maxDepth        integer: depth of the kd-tree (don’t change unless you know what you are doing!)
angdev          maximum deviation between reference and target normals. If the none of the k closest triangles match this criterion, the closest point on the closest triangle is returned but the corresponding distance in $quality is set to 1e5.
weightnorm      logical if angdev is set, this requests the normal of the closest points to be estimated by weighting the surrounding vertex normals. Otherwise, simply the hit face’s normal is used (faster but slightly less accurate)
facenormals     logical: if TRUE only the facenormal of the face the closest point has hit is returned, the weighted average of the surrounding vertex normals otherwise.
threads         integer: threads to use in closest point search.
...
additional parameters, currently unused.

Value

returns an object of class "mesh3d" with:

vb               4 x n matrix containing n vertices as homolougous coordinates.
normals         4 x n matrix containing vertex normals.
quality          numeric vector containing distances to target.
it               3 x m integer matrix containing vertex indices forming triangular faces.Only available, when x is a mesh.
border         integer vector of length n: if borderchk = TRUE, for each closest point the value will be 1 if the hit face is at the border of the target mesh and 0 otherwise.
barycoords      3 x m Matrix containing barycentric coordinates of closest points; only available if barycentric=TRUE.
Note
Other than vcgClosest this does not search a grid, but first uses a KD-tree search to find the k closest barycenters for each point and then searches these faces for the closest match.

Author(s)
Stefan Schlager

References

See Also
vcgPlyRead

---

vcgClosestOn KDtreeFromBarycenters

*search a KD-tree from Barycenters for multiple closest point searches on a mesh*

---

**Description**

search a KD-tree from Barycenters for multiple closest point searches on a mesh

**Usage**

```
vcgClosestOn KDtreeFromBarycenters(
  x,
  query,
  k = 50,
  sign = TRUE,
  barycentric = FALSE,
  borderchk = FALSE,
  angdev = NULL,
  weightnorm = FALSE,
  facenormals = FALSE,
  threads = 1
)
```

**Arguments**

- `x` object of class "vcgKDtreeWithBarycenters"
- `query` matrix or triangular mesh containing coordinates
- `k` integer: check the kdtree for the k closest faces (using faces’ barycenters).
vcgCreateKDtree

create a KD-tree

Description

create a KD-tree

Usage

vcgCreateKDtree(mesh, nofPointsPerCell = 16, maxDepth = 64)
vcgCreateKDtreeFromBarycenters

Arguments

- mesh: matrix or triangular mesh containing coordinates
- nofPointsPerCell: number of points per kd-cell
- maxDepth: maximum tree depth

Value

returns an object of class vcgKDtree containing external pointers to the tree and the target points

See Also

vcgSearchKDtree

Examples

data(humface)
mytree <- vcgCreateKDtree(humface)

vcgCreateKDtreeFromBarycenters

create a KD-tree from Barycenters for multiple closest point searches on a mesh

Description

create a KD-tree from Barycenters for multiple closest point searches on a mesh

Usage

vcgCreateKDtreeFromBarycenters(mesh, nofPointsPerCell = 16, maxDepth = 64)

Arguments

- mesh: matrix or triangular mesh containing coordinates
- nofPointsPerCell: number of points per kd-cell
- maxDepth: maximum tree depth

Value

returns an object of class vcgKDtreeWithBarycenters containing external pointers to the tree, the barycenters and the target mesh

See Also

vcgClostOnKDtreeFromBarycenters, vcgSearchKDtree, vcgCreateKDtree
vcgCurve

*calculate curvature of a triangular mesh*

**Description**

calculate curvature of faces/vertices of a triangular mesh using various methods.

**Usage**

```
vcgCurve(mesh)
```

**Arguments**

- `mesh` triangular mesh (object of class 'mesh3d')

**Value**

- `gaussv` per vertex gaussian curvature
- `meanv` per vertex mean curvature
- `RMSv` per vertex RMS curvature
- `gausitmax` per face maximum gaussian curvature of adjacent vertices
- `borderi` per face information if it is on the mesh's border (0=FALSE, 1=TRUE)
- `borderv` per vertex information if it is on the mesh’s border (0=FALSE, 1=TRUE)
- `meanitmax` per face maximum mean curvature of adjacent vertices
- `K1` Principal Curvature 1
- `K2` Principal Curvature 2

**Examples**

```
data(humface)
curv <- vcgCurve(humface)
##visualise per vertex mean curvature
## Not run:
require(Morpho)
meshDist(humface,distvec=curv$meanv,from=-0.2,to=0.2,tol=0.01)
## End(Not run)
```
vcgDijkstra

Compute pseudo-geodesic distances on a triangular mesh

Description

Compute pseudo-geodesic distances on a triangular mesh

Usage

vcgDijkstra(x, vertpointer, maxdist = NULL)

Arguments

x    triangular mesh of class mesh3d
vertpointer    integer: references indices of vertices on the mesh, typically only a single query vertex.
maxdist    positive scalar double, the maximal distance to travel along the mesh when computing distances. Leave at NULL to traverse the full mesh. This can be used to speed up the computation if you are only interested in geodesic distances to neighbors within a limited distance around the query vertices.

Value

returns a vector of shortest distances for each of the vertices to one of the vertices referenced in vertpointer. If maxdist is in use (not NULL), the distance values for vertices outside the requested maxdist are not computed and appear as 0.

Note

Make sure to have a clean manifold mesh. Note that this computes the length of the pseudo-geodesic path (following the edges) between the two vertices.

Examples

## Compute geodesic distance between all mesh vertices and the first vertex of a mesh
data(humface)
geo <- vcgDijkstra(humface,1)
if (interactive()) {
  require(Morpho);require(rgl)
  meshDist(humface,distvec = geo)
spheres3d(vert2points(humface)[1,],col=2)
}
vcgFaceNormals  
*Compute normalized face normals for a mesh.*

**Description**

Compute normalized face normals for a mesh.

**Usage**

```r
cvgFaceNormals(mesh)
```

**Arguments**

- `mesh`: triangular mesh of class `mesh3d`, from `rgl`

**Value**

3xn numeric matrix of face normals for the mesh, where n is the number of faces.

**Examples**

```r
data(humface);
hf_facenormals <- vcgFaceNormals(humface);
```

---

vcgGeodesicPath  
*Compute geodesic path and path length between vertices on a mesh*

**Description**

Compute geodesic path and path length between vertices on a mesh.

**Usage**

```r
cvgGeodesicPath(x, source, targets, maxdist = 1e+06)
```

**Arguments**

- `x`: triangular mesh of class `mesh3d` from the `rgl` package.
- `source`: scalar positive integer, the source vertex index.
- `targets`: positive integer vector, the target vertex indices.
- `maxdist`: numeric, the maximal distance to travel along the mesh edges during geodesic distance computation.
vcgGeodist

Value

named list with two entries as follows. ‘paths’: list of integer vectors, representing the paths. ‘geodist’: double vector, the geodesic distances from the source vertex to all vertices in the graph.

Note

Currently no reachability checks are performed, so you have to be sure that the mesh is connected, or at least that the source and target vertices are reachable from one another.

Examples

data(humface)
p = vcgGeodesicPath(humface, 50, c(500, 5000))
p$paths[[1]]; # The path 50..500
p$geodist[500]; # Its path length.

vcgGeodist

Compute pseudo-geodesic distance between two points on a mesh

Description

Compute pseudo-geodesic distance between two points on a mesh

Usage

vcgGeodist(x, pt1, pt2)

Arguments

x triangular mesh of class mesh3d
pt1 3D coordinate on mesh or index of vertex
pt2 3D coordinate on mesh or index of vertex

Value

returns the geodesic distance between pt1 and pt2.

Note

Make sure to have a clean manifold mesh. Note that this computes the length of the pseudo-geodesic path (following the edges) between the two vertices closest to these points.

Examples

data(humface)
pt1 <- humface.lm[1,]
pt2 <- humface.lm[5,]
vcgGeodist(humface, pt1, pt2)
v cgGetEdge  

Get all edges of a triangular mesh

Description

Extract all edges from a mesh and retrieve adjacent faces and vertices

Usage

vcgGetEdge(mesh, unique = TRUE)

Arguments

mesh  
triangular mesh of class 'mesh3d'

unique  
logical: if TRUE each edge is only reported once, if FALSE, all occurrences are reported.

Value

returns a dataframe containing:

vert1  
integer indicating the position of the first vertex belonging to this edge

vert2  
integer indicating the position of the second vertex belonging to this edge

facept  
integer pointing to the (or a, if unique = TRUE) face adjacent to the edge

border  
integer indicating if the edge is at the border of the mesh. 0 = no border, 1 = border

Examples

require(rgl)
data(humface)
edges <- vcgGetEdge(humface)
## Not run:
## show first edge
lines3d(t(humface$vb[1:3,])[c(edges$vert1[1],edges$vert2[2]),],col=2,lwd=3)
shade3d(humface, col=3)
## now find the edge - hint: it is at the neck.

## End(Not run)
vcgImport

Import common mesh file formats.

Description

Import common mesh file formats and store the results in an object of class "mesh3d" - momentarily only triangular meshes are supported.

Usage

vcgImport(
  file,
  updateNormals = TRUE,
  readcolor = FALSE,
  clean = TRUE,
  silent = FALSE
)

Arguments

- **file**: character: file to be read.
- **updateNormals**: logical: if TRUE and the imported file contais faces, vertex normals will be (re)calculated. Otherwise, normals will be a matrix containing zeros.
- **readcolor**: if TRUE, vertex colors and texture (face and vertex) coordinates will be processed - if available, otherwise all vertices will be colored white.
- **clean**: if TRUE, duplicated and unreferenced vertices as well as duplicate faces are removed (be careful when importing point clouds).
- **silent**: logical, if TRUE no console output is issued.

Value

Object of class "mesh3d"

with:

- **vb**: 4 x n matrix containing n vertices as homolougous coordinates
- **it**: 3 x m matrix containing vertex indices forming triangular faces
- **normals**: 4 x n matrix containing vertex normals (homologous coordinates)

in case the imported files contains face or vertex quality, these will be stored as vectors named $quality (for vertex quality) and $facequality

if the imported file contains vertex colors and readcolor = TRUE, these will be saved in $material<Color according to "mesh3d" specifications.

Note

currently only meshes with either color or texture can be processed. If both are present, the function will mark the mesh as non-readable.
Author(s)
Stefan Schlager

See Also
vcgSmooth

Examples

data(humface)
vcgPlyWrite(humface)
readit <- vcgImport("humface.ply")

vcgIsolated

Remove isolated pieces from a surface mesh or split into connected components

Description
Remove isolated pieces from a surface mesh, selected by a minimum amount of faces or of a diameter below a given threshold. Also the option only to keep the largest piece can be selected or to split a mesh into connected components.

Usage

vcgIsolated(
  mesh,
  facenum = NULL,
  diameter = NULL,
  split = FALSE,
  keep = 0,
  silent = FALSE
)

Arguments

mesh triangular mesh of class "mesh3d".
facenum integer: all connected pieces with less components are removed. If not specified or 0 and diameter is NULL, then only the component with the most faces is kept.
diameter numeric: all connected pieces smaller diameter are removed. diameter = 0 removes all component but the largest ones. This option overrides the option facenum.
split logical: if TRUE, a list with all connected components (optionally matching requirements facenum/diameter) of the mesh will be returned.
keep integer: if split=T, keep specifies the number of largest chunks (number of faces) to keep.
silent logical, if TRUE no console output is issued.
Value

returns the reduced mesh.

Author(s)

Stefan Schlager

See Also

vcgPlyRead

Examples

```r
## Not run:
data(humface)
cleanface <- vcgIsolated(humface)
## End(Not run)
```

---

vcgIsosurface Create Isosurface from 3D-array

Description

Create Isosurface from 3D-array using Marching Cubes algorithm

Usage

```r
vcgIsosurface(
  vol,
  threshold,
  from = NULL,
  to = NULL,
  spacing = NULL,
  origin = NULL,
  direction = NULL,
  IJK2RAS = diag(c(-1, -1, 1, 1)),
  as.int = FALSE
)
```

Arguments

- **vol**: an integer valued 3D-array
- **threshold**: threshold for creating the surface
- **from**: numeric: the lower threshold of a range (overrides `threshold`)
- **to**: numeric: the upper threshold of a range (overrides `threshold`
vcgIsotropicRemeshing

Isotropically remesh a triangular surface mesh

Description

Isotropically remesh a triangular surface mesh

Usage

vcgIsotropicRemeshing(
  x,
  TargetLen = 1,
  FeatureAngleDeg = 10,
  MaxSurfDist = 1,
  iterations = 3,
  Adaptive = FALSE,
  split = TRUE,
  collapse = TRUE,
  swap = TRUE,

spacing numeric 3D-vector: specifies the voxel dimensons in x,y,z direction.
origin numeric 3D-vector: origin of the original data set, will transpose the mesh onto that origin.
direction a 3x3 direction matrix
IJK2RAS 4x4 IJK2RAS transformation matrix
as.int logical: if TRUE, the array will be stored as integer (might decrease RAM usage)

Value

returns a triangular mesh of class "mesh3d"

Examples

#this is the example from the package "misc3d"
x <- seq(-2,2,len=50)
g <- expand.grid(x = x, y = x, z = x)
v <- array(g$x^4 + g$y^4 + g$z^4, rep(length(x),3))
storage.mode(v) <- "integer"
## Not run:
mesh <- vcgIsosurface(v,threshold=10)
require(rgl)
wire3d(mesh)
##now smooth it a little bit
wire3d(vcgSmooth(mesh,"HC",iteration=3),col=3)
## End(Not run)
smooth = TRUE,
project = TRUE,
surfDistCheck = TRUE
)

Arguments

x mesh of class mesh3d
TargetLen numeric: edge length of the target surface
FeatureAngleDeg define Crease angle (in degree).
MaxSurfDist Max. surface distance
iterations ToDo
Adaptive enable adaptive remeshing
split enable refine step
collapse enable collapse step
swap enable dge swap
smooth enable smoothing
project enable reprojection step
surfDistCheck check distance to surface

Value
returns the remeshed surface mesh

Examples

## Not run:
data(humface)
resampledMesh <- vcgIsotropicRemeshing(humface,TargetLen=2.5)
## End(Not run)

vcgKDtree perform kdtree search for 3D-coordinates.

Description
perform kdtree search for 3D-coordinates.

Usage
vcgKDtree(target, query, k, nofPoints = 16, maxDepth = 64, threads = 1)
Arguments

target  n x 3 matrix with 3D coordinates or mesh of class "mesh3d". These coordinates are to be searched.
query    m x 3 matrix with 3D coordinates or mesh of class "mesh3d". We search the closest coordinates in target for each of these.
k         number of neighbours to find
nofPoints integer: number of points per cell in the kd-tree (don’t change unless you know what you are doing!)
maxDepth integer: depth of the kd-tree (don’t change unless you know what you are doing!)
threads   integer: threads to use in closest point search.

Value

a list with

   index    integer matrices with indeces of closest points
   distances corresponding distances

vcgKmeans  fast Kmean clustering for 1D, 2D and 3D data

Description

fast Kmean clustering for 1D, 2D and 3D data

Usage

vcgKmeans(x, k = 10, iter.max = 10, getClosest = FALSE, threads = 0)

Arguments

x        matrix containing coordinates or mesh3d
k         number of clusters
iter.max maximum number of iterations
getClosest logical: if TRUE the indices of the points closest to the k-centers are sought.
threads   integer: number of threads to use

Value

returns a list containing

centers   cluster center
class     vector with cluster association for each coordinate

If getClosest=TRUE

selected  vector with indices of points closest to the centers
vcgMeshres

See Also
vcgSample

Examples

```r
require(Rvcg);require(rgl)
data(humface)
set.seed(42)
clust <- vcgKmeans(humface,k=1000,threads=1)
```

data(humface)
mres <- vcgMeshres(humface)
#histogram of edgelength distribution
hist(mres$edgelength)
#visualise average edgelength
points( mres$res, 1000, pch=20, col=2, cex=2)

vcgMeshres calculates the average edge length of a triangular mesh

Description
calculates the average edge length of a triangular mesh, iterating over all faces.

Usage
cvgMeshres(mesh)

Arguments

mesh triangular mesh stored as object of class "mesh3d"

Value

res average edge length (a.k.a. mesh resolution)
edgelength vector containing lengths for each edge

Author(s)

Stefan Schlager

Examples
data(humface)
mres <- vcgMeshres(humface)
#histogram of edgelength distribution
hist(mres$edgelength)
#visualise average edgelength
points( mres$res, 1000, pch=20, col=2, cex=2)
vcgMetro

evaluate the difference between two triangular meshes.

Description

Implementation of the command line tool "metro" to evaluate the difference between two triangular meshes.

Usage

vcgMetro(
  mesh1, mesh2, nSamples = 0, nSamplesArea = 0, vertSamp = TRUE, edgeSamp = TRUE, faceSamp = TRUE, unrefVert = FALSE, samplingType = c("SS", "MC", "SD"), searchStruct = c("SGRID", "AABB", "OCTREE", "HGRID"), from = 0, to = 0, colormeshes = FALSE, silent = FALSE)

Arguments

mesh1 triangular mesh (object of class 'mesh3d').
mesh2 triangular mesh (object of class 'mesh3d').
nSamples set the required number of samples if 0, this will be set to approx. 10x the face number.
nSamplesArea set the required number of samples per area unit, override nSamples.
vertSamp logical: if FALSE, disable vertex sampling.
edgeSamp logical: if FALSE, disable edge sampling.
faceSamp logical: if FALSE, disable face sampling.
unrefVert logical: if FALSE, ignore unreferred vertices.
samplingType set the face sampling mode. options are: SS (similar triangles sampling), SD (subdivision sampling), MC (montecarlo sampling).
searchStruct set search structures to use. options are: SGRID (static Uniform Grid), OCTREE, AABB (AxisAligned Bounding Box Tree), HGRID (Hashed Uniform Grid).
from numeric: minimum value for color mapping.
vcgMetro

Parameters

to numeric: maximum value for color mapping.
colormeshes if TRUE, meshes with vertices colored according to distance are returned
silent logical: if TRUE, output to console is suppressed.

Value

ForwardSampling, BackwardSampling
lists containing information about forward (mesh1 to mesh2) and backward (mesh2 to mesh1) sampling with the following entries

- maxdist maximal Hausdorff distance
- meandist mean Hausdorff distance
- RMSdist RMS of the Hausdorff distances
- area mesh area (of mesh1 in ForwardSampling and mesh2 in BackwardSampling)
- RMSdist RMS of the Hausdorff distances
- nvbsamples number of vertices sampled
- nsamples number of samples
distances1, distances2 vectors containing vertex distances from mesh1 to mesh2 and mesh2 to mesh1.
forward_hist, backward_hist Matrices tracking the sampling results

if colormeshes == TRUE

mesh1, mesh2 meshes with color coded distances and an additional entry called quality containing the sampled per-vertex distances

Note

this is a straightforward implementation of the command line tool metro http://vcg.isti.cnr.it/vcglib/metro.html

References


Examples

if (requireNamespace("Morpho", quietly = TRUE)) {
  require(Morpho)
data(humface)
data(dummyhead)
## align humface to dummyhead.mesh
humfalign <- rotmesh.onto(humface,humface.lm,dummyhead.lm)
samp <- vcgMetro(humfalign$mesh,dummyhead.mesh,faceSamp=FALSE,edgeSamp=FALSE)
## create heatmap using Morpho's meshDist function
## Not run:
## create custom heatmaps based on distances
mD <- meshDist(humfalign$mesh,distvec=samp$distances1)

## End(Not run)

vcgNonBorderEdge

### Get all non-border edges

#### Description
Get all non-border edges and both faces adjacent to them.

#### Usage
vcgNonBorderEdge(mesh, silent = FALSE)

#### Arguments
- **mesh**: triangular mesh of class ‘mesh3d’
- **silent**: logical: suppress output of information about number of border edges

#### Value
returns a dataframe containing:
- **vert1**: integer indicating the position of the first vertex belonging to this edge
- **vert2**: integer indicating the position of the second vertex belonging to this edge
- **border**: integer indicating if the edge is at the border of the mesh. 0 = no border, 1 = border
- **face1**: integer pointing to the first face adjacent to the edge
- **face2**: integer pointing to the first face adjacent to the edge

#### See Also
vcgGetEdge

#### Examples

data(humface)
edges <-vcgNonBorderEdge(humface)
## show first edge (not at the border)
## Not run:
require(Morpho)
require(rgl)
lines3d(t(humface$vb[1:3,])[,c(edges$vert1[1],edges$vert2[2],),],col=2,lwd=3)

## plot barycenters of adjacent faces
bary <- barycenter(humface)
points3d(bary[c(edges$face1[1],edges$face2[1],)])
shade3d(humface, col=3)

## now find the edge - hint: it is at the neck.

## End(Not run)

---

vcgObjWrite  
Export meshes to OBJ-files

**Description**

Export meshes to OBJ-files

**Usage**

```r
vcgObjWrite(mesh, filename = dataname, writeNormals = TRUE)
```

**Arguments**

- **mesh**: triangular mesh of class 'mesh3d' or a numeric matrix with 3-columns
- **filename**: character: filename (file extension '.obj' will be added automatically.)
- **writeNormals**: write existing normals to file

**Examples**

```r
data(humface)
vcgObjWrite(humface, filename = "humface")
unlink("humface.obj")
```

---

vcgOffWrite  
Export meshes to OFF-files

**Description**

Export meshes to OFF-files

**Usage**

```r
vcgOffWrite(mesh, filename = dataname)
```
vcgPlyRead

**Arguments**

- **mesh**: triangular mesh of class 'mesh3d' or a numeric matrix with 3-columns
- **filename**: character: filename (file extension '.off' will be added automatically.

**Examples**

data(humface)
vcgOffWrite(humface, filename = "humface")
unlink("humface.off")

**Description**

Reads Polygon File Format (PLY) files and stores the results in an object of class "mesh3d" - momentarily only triangular meshes are supported.

**Usage**

vcgPlyRead(file, updateNormals = TRUE, clean = TRUE)

**Arguments**

- **file**: character: file to be read.
- **updateNormals**: logical: if TRUE and the imported file contains faces, vertex normals will be (re)calculated.
- **clean**: logical: if TRUE, duplicated and unreferenced vertices will be removed.

**Value**

Object of class "mesh3d"

with:

- **vb**: 3 x n matrix containing n vertices as homologous coordinates
- **normals**: 3 x n matrix containing vertex normals
- **it**: 3 x m integer matrix containing vertex indices forming triangular faces
- **material$color**: Per vertex colors if specified in the imported file

**Note**

from version 0.8 on this is only a wrapper for vcgImport (to avoid API breaking).

**Author(s)**

Stefan Schlager
vcgPlyWrite

See Also
vcgSmooth.

vcgPlyWrite  Export meshes to PLY-files

Description
Export meshes to PLY-files (binary or ascii)

Usage
vcgPlyWrite(mesh, filename, binary = TRUE, ...)

## S3 method for class 'mesh3d'
vcgPlyWrite(
  mesh,
  filename = dataname,
  binary = TRUE,
  addNormals = FALSE,
  writeCol = TRUE,
  writeNormals = TRUE,
  ...
)

## S3 method for class 'matrix'
vcgPlyWrite(mesh, filename = dataname, binary = TRUE, addNormals = FALSE, ...)

Arguments
mesh    triangular mesh of class 'mesh3d' or a numeric matrix with 3-columns
filename character: filename (file extension '.ply' will be added automatically, if missing.
binary   logical: write binary file
...      additional arguments, currently not used.
addNormals logical: compute per-vertex normals and add to file
writeCol   logical: export existing per-vertex color stored in mesh$material$color
writeNormals write existing normals to file

Examples
data(humface)
vcgPlyWrite(humface, filename = "humface")
## remove it
unlink("humface.ply")
vcgQEdecim

Performs Quadric Edge Decimation on triangular meshes.

Description

Decimates a mesh by adapting the faces of a mesh either to a target face number, a percentage or an approximate mesh resolution (a.k.a. mean edge length)

Usage

vcgQEdecim(
  mesh,
  tarface = NULL,
  percent = NULL,
  edgeLength = NULL,
  topo = FALSE,
  quality = TRUE,
  bound = FALSE,
  optiplace = FALSE,
  scaleindi = TRUE,
  normcheck = FALSE,
  qweightFactor = 100,
  qthresh = 0.3,
  boundweight = 1,
  normalthr = pi/2,
  silent = FALSE
)

Arguments

- **mesh** Triangular mesh of class "mesh3d"
- **tarface** Integer: set number of target faces.
- **percent** Numeric: between 0 and 1. Set amount of reduction relative to existing face number. Overrides tarface argument.
- **edgeLength** Numeric: tries to decimate according to a target mean edge length. Under the assumption of regular triangles, the edges are half as long by dividing the triangle into 4 regular smaller triangles.
- **topo** logical: if TRUE, mesh topology is preserved.
- **quality** logical: if TRUE, vertex quality is considered.
- **bound** logical: if TRUE, mesh boundary is preserved.
- **optiplace** logical: if TRUE, mesh boundary is preserved (may lead to unwanted distortions in some cases).
- **scaleindi** logical: if TRUE, decimation is scale independent.
- **normcheck** logical: if TRUE, normal directions are considered.
vcgRaySearch

Check if a mesh is intersected by a set of rays

Details

This is basically an adaption of the cli tridecimator from vcglib

Value

Returns a reduced mesh of class mesh3d.

Author(s)

Stefan Schlager

See Also

vcgSmooth

Examples

data(humface)
## reduce faces to 50%
decimface <- vcgQEdecim(humface, percent=0.5)
## view
## Not run:
require(rgl)
shade3d(decimface, col=3)

## some light smoothing
decimface <- vcgSmooth(decimface, iteration = 1)

## End(Not run)
vcgRaySearch

Usage
vcgRaySearch(x, mesh, mintol = 0, maxtol = 1e+15, mindist = FALSE, threads = 1)

Arguments

x
  a triangular mesh of class 'mesh3d' or a list containing vertices and vertex nor-
mals (fitting the naming conventions of 'mesh3d'). In the second case x must
contain x$v = 3 x n matrix containing 3D-coordinates and x$normals = 3 x n
matrix containing normals associated with x$v.

mesh
  triangular mesh to be intersected.

mintol
  minimum distance to target mesh

maxtol
  maximum distance to search along ray

mindist
  search both ways (ray and -ray) and select closest point.

threads
  number of threads used during search.

Details
vcgRaySearch projects a mesh (or set of 3D-coordinates) along a set of given rays (stored as nor-
mals) onto a target and return the hit points as well as information if the target mesh was hit at all.
If nothing is hit along the ray(within the given thresholds), the ordinary closest point’s value will be
returned and the corresponding entry in quality will be zero.

Value
list with following items:

vb
  4 x n matrix containing intersection points

normals
  4 x n matrix containing homogenous coordinates of normals at intersection
  points

quality
  integer vector containing a value for each vertex of x: 1 indicates that a ray has
  intersected ‘mesh’, while 0 means not

distance
  numeric vector: distances to intersection

Examples
data(humface)
  #get normals of landmarks
lms <- vcgClost(humface.lm, humface)
  # offset landmarks along their normals for a negative amount of -5mm
lms$v[1:3,] <- lms$v[1:3,]+lms$normals[1:3,]*-5
interact <- vcgRaySearch(lms, humface)
  ## Not run:
require(Morpho)
require(rgl)
spheres3d(vert2points(lms),radius=0.5,col=3)
plotNormals(lms,long=5)
spheres3d(vert2points(interact),col=2) #plot intersections
wire3d(humface,col="white")##
vcgSample

Subsamples points on a mesh surface

Description

Subsamples surface of a triangular mesh and returns a set of points located on that mesh.

Usage

```r
vcgSample(
  mesh,
  SampleNum = 100,
  type = c("km", "pd", "mc"),
  MCsamp = 20,
  geodes = TRUE,
  strict = FALSE,
  iter.max = 100,
  threads = 0
)
```

Arguments

- **mesh**: triangular mesh of class `mesh3d`
- **SampleNum**: integer: number of sampled points (see details below)
- **type**: character: select sampling type ("mc"=MonteCarlo Sampling, "pd"=PoissonDisk Sampling,"km"=kmeans clustering)
- **MCsamp**: integer: MonteCarlo sample iterations used in PoissonDisk sampling.
- **geodes**: logical: maximise geodesic distance between sample points (only for Poisson Disk sampling)
- **strict**: logical: if type="pd" and the amount of coordinates exceeds SampleNum, the resulting coordinates will be subsampled again by kmean clustering to reach the requested number.
- **iter.max**: integer: maximum iterations to use in k-means clustering.
- **threads**: integer number of threads to use for k-means clustering

Details

Poisson disk subsampling will not generate the exact amount of coordinates specified in SampleNum, depending on MCsamp the result will contain more or less coordinates.
vcgSearchKDtree

Value

sampled points

Examples

data(humface)
ss <- vcgSample(humface, SampleNum = 500, type="km", threads=1)
## Not run:
require(rgl)
points3d(ss)
## End(Not run)

vcgSearchKDtree  

search an existing KD-tree

Description

search an existing KD-tree

Usage

vcgSearchKDtree(kdtree, query, k, threads = 0)

Arguments

kdtree  
object of class vcgKDtree

query  
array or triangular mesh containing coordinates

k  
number of k-closest neighbours to query

threads  
integer: number of threads to use

Value

a list with

index  
integer matrices with indeces of closest points

distances  
corresponding distances

See Also

vcgCreateKDtree
### Examples

```r
## Not run:
data(humface);data(dummyhead)
mytree <- vcgCreateKDtree(humface)
## get indices and distances for 10 closest points.
closest <- vcgSearchKDtree(mytree,dummyhead.mesh,k=10,threads=1)
## End(Not run)
```

---

**vcgSmooth**

*Smooths a triangular mesh*

---

**Description**

Applies different smoothing algorithms on a triangular mesh.

**Usage**

```r
vcgSmooth(
  mesh,
  type = c("taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight",
           "surfPreserveLaplace"),
  iteration = 10,
  lambda = 0.5,
  mu = -0.53,
  delta = 0.1
)
```

**Arguments**

- `mesh` triangular mesh stored as object of class "mesh3d".
- `type` character: select smoothing algorithm. Available are "taubin", "laplace", "HClaplace", "fujiLaplace", "angWeight" (and any sensible abbreviations).
- `iteration` integer: number of iterations to run.
- `lambda` numeric: parameter for Taubin smooth (see reference below).
- `mu` numeric: parameter for Taubin smooth (see reference below).
- `delta` numeric: parameter for Scale dependent laplacian smoothing (see reference below) and maximum allowed angle (in radians) for deviation between normals Laplacian (surface preserving).

**Details**

The algorithms available are Taubin smoothing, Laplacian smoothing and an improved version of Laplacian smoothing ("HClaplace"). Also available are Scale dependent laplacian smoothing ("fujiLaplace") and Laplacian angle weighted smoothing ("angWeight")
Value

returns an object of class "mesh3d" with:

- `vb`: 4xn matrix containing n vertices as homologous coordinates.
- `normals`: 4xn matrix containing vertex normals.
- `quality`: vector: containing distances to target.
- `it`: 4xm matrix containing vertex indices forming triangular faces.

Note

The additional parameters for taubin smooth are hardcoded to the default values of meshlab, as they appear to be the least distorting.

Author(s)

Stefan Schlager

References


See Also

vcgPlyRead, vcgClean

Examples

data(humface)
smoothface <- vcgSmooth(humface)
## view
## Not run:
require(rgl)
shade3d(smoothface, col=3)
## End(Not run)
vcgSphere  

create platonic objects as triangular meshes

Description

create platonic objects as triangular meshes

Usage

vcgSphere(subdivision = 3, normals = TRUE)

vcgSphericalCap(angleRad = pi/2, subdivision = 3, normals = TRUE)

vcgTetrahedron(normals = TRUE)

vcgDodecahedron(normals = TRUE)

vcgOctahedron(normals = TRUE)

vcgIcosahedron(normals = TRUE)

vcgHexahedron(normals = TRUE)

vcgSquare(normals = TRUE)

vcgBox(mesh = vcgSphere(), normals = TRUE)

vcgCone(r1, r2, h, normals = TRUE)

Arguments

subdivision subdivision level for sphere (the larger the denser the mesh will be)
normals if TRUE vertex normals are calculated
angleRad angle of the spherical cap
mesh mesh to take the bounding box from
r1 radius1 of the cone
r2 radius2 of the cone
h height of the cone
vcgStlWrite  
Export meshes to STL-files

Description

Export meshes to STL-files (binary or ascii)

Usage

vcgStlWrite(mesh, filename = dataname, binary = FALSE)

Arguments

mesh  
triangular mesh of class 'mesh3d' or a numeric matrix with 3-columns
filename  
character: filename (file extension `.stl' will be added automatically.
binary  
logical: write binary file

Examples

data(humface)
vcgStlWrite(humface, filename = "humface")
unlink("humface.stl")

vcgSubdivide  
subdivide the triangles of a mesh

Description

subdivide the triangles of a mesh

Usage

vcgSubdivide(
  x,
  threshold = NULL,
  type = c("Butterfly", "Loop"),
  looptype = c("loop", "regularity", "continuity"),
  iterations = 3,
  silent = FALSE
)
Arguments

- **x**: triangular mesh of class "mesh3d"
- **threshold**: minimum edge length to subdivide
- **type**: character: algorithm used. Options are Butterfly and Loop (see notes)
- **looptype**: character: method for type = loop options are "loop","regularity","continuity" (see notes)
- **iterations**: integer: number of iterations
- **silent**: logical: suppress output.

Value

returns subdivided mesh

Note

The different algorithms are (from meshlab description):

- **Butterfly Subdivision**: Apply Butterfly Subdivision Surface algorithm. It is an interpolated method, defined on arbitrary triangular meshes. The scheme is known to be C1 but not C2 on regular meshes

- **Loop Subdivision**: Apply Loop’s Subdivision Surface algorithm. It is an approximant subdivision method and it works for every triangle and has rules for extraordinary vertices. Options are "loop" a simple subdivision, "regularity" to enhance the mesh’s regularity and "continuity" to enhance the mesh’s continuity.

Examples

```
data(humface)
subdivide <- vcgSubdivide(humface,type="Loop",looptype="regularity")
```

v<em>c</em>gUniformRemesh  
Resample a mesh uniformly

Description

Resample a mesh uniformly

Usage

```
v<em>c</em>gUniformRemesh(  
  x,  
  voxelSize = NULL,  
  offset = 0,  
  discretize = FALSE,  
  multiSample = FALSE,
)```
vcgUpdateNormals

absDist = FALSE,
mergeClosest = FALSE,
silent = FALSE
)

Arguments

x triangular mesh
voxelSize voxel size for space discretization
offset Offset of the created surface (i.e. distance of the created surface from the original one).
discretize If TRUE, the position of the intersected edge of the marching cube grid is not computed by linear interpolation, but it is placed in fixed middle position. As a consequence the resampled object will look severely aliased by a stairstep appearance.
multiSample If TRUE, the distance field is more accurately compute by multisampling the volume (7 sample for each voxel). Much slower but less artifacts.
absDist If TRUE, an unsigned distance field is computed. In this case you have to choose a not zero Offset and a double surface is built around the original surface, inside and outside.
mergeClosest logical: merge close vertices
silent logical: suppress messages

Value

resampled mesh

Examples

## Not run:
data(humface)
humResample <- vcgUniformRemesh(humface, voxelSize = 1, multiSample = TRUE)
require(rgl)
shade3d(humResample, col = 3)
## End(Not run)

vcgUpdateNormals updates vertex normals of a triangular meshes or point clouds

Description

update vertex normals of a triangular meshes or point clouds

Usage

vcgUpdateNormals(mesh, type = 0, pointcloud = c(10, 0), silent = FALSE)
vcgVertexNeighbors

Compute mesh adjacency list representation or the vertex neighborhoods of specific mesh vertices.

Arguments

mesh	triangular mesh of class 'mesh3d' or a n x 3 matrix containing 3D-coordinates.
type	select the method to compute per-vertex normals: 0 = area weighted average of surrounding face normals; 1 = angle weighted vertex normals.
pointcloud	integer vector of length 2: containing optional parameters for normal calculation of point clouds. The first entry specifies the number of neighbouring points to consider. The second entry specifies the amount of smoothing iterations to be performed.
silent	logical, if TRUE no console output is issued.

Value

mesh with updated/created normals, or in case mesh is a matrix, a list of class "mesh3d" with

vb	4 x n matrix containing coordinates (as homologous coordinates
normals	4 x n matrix containing normals (as homologous coordinates

Examples

data(humface)
humface$normals <- NULL # remove normals
humface <- vcgUpdateNormals(humface)
## Not run:
pointcloud <- t(humface$vb[1:3,]) #get vertex coordinates
pointcloud <- vcgUpdateNormals(pointcloud)

require(Morpho)
plotNormals(pointcloud)#plot normals

## End(Not run)

vcgVertexNeighbors(x, vi = NULL, numstep = 1L, include_self = FALSE)
Arguments

- **x**: tmesh3d instance from the rgl package
- **vi**: optional, vector of positive vertex indices for which to compute the neighborhoods. All vertices are used if left at the default value NULL.
- **numstep**: positive integer, the number of times to extend the neighborhood from the source vertices (the k for computing the k-ring neighborhood). Setting this to high values significantly increases the computational cost.
- **include_self**: logical, whether the returned neighborhood for a vertex i should include i itself.

Value

list of positive integer vectors, the neighborhoods.

Examples

data(humface)
adjacency_list <- vcgVertexNeighbors(humface)
v500_5ring = vcgVertexNeighbors(humface, vi=c(500), numstep = 5)

vcgVFadj  

**Description**

find all faces belonging to each vertex in a mesh

**Usage**

vcgVFadj(mesh)

Arguments

- **mesh**: triangular mesh of class "mesh3d"

Value

list containing one vector per vertex containing the indices of the adjacent faces
Compute volume for manifold meshes

Description
Compute volume for manifold meshes

Usage
vcgVolume(x)

Arguments
x triangular mesh of class mesh3d

Value
returns volume

Note
Please note, that this function only works reliably on watertight, coherently oriented meshes that constitute a manifold. In case your mesh has some issues regarding non-manifoldness or there are isolated pieces flying around, you can use vcgIsolated and vcgClean to remove those.

Examples
mysphere <- vcgSphere()
vcgVolume(mysphere)
## Not run:
## here is an example where the mesh has some non-manifold vertices

mysphere <- vcgSphere(normals=FALSE)
## add a degenerate face
mysphere$it <- cbind(mysphere$it,c(1,2,1))
try(vcgVolume(mysphere))

## fix the error using vcgClean():
vcgVolume(vcgClean(mysphere,sel=0:6,iterate=TRUE))

## End(Not run)
vcgWrlWrite  

Export meshes to WRL-files

Description
Export meshes to WRL-files

Usage
vcgWrlWrite(mesh, filename = dataname, writeCol = TRUE, writeNormals = TRUE)

Arguments
mesh  triangular mesh of class 'mesh3d' or a numeric matrix with 3-columns
filename character: filename (file extension '.wrl' will be added automatically.
writeCol logical: export existing per-vertex color stored in mesh$material$color
writeNormals write existing normals to file

Examples
data(humface)
vcgWrlWrite(humface, filename = "humface")
unlink("humface.wrl")
Index

* datasets
  dummyhead, 4
  humface, 5
* package
  Rvcg-package, 3
  checkFaceOrientation, 4
  dummyhead, 4
  humface, 5
  humfaceClean (humface), 5
  meshInfo, 5
  meshintegrity, 5
  nfaces, 6
  nverts, 6
  Rvcg (Rvcg-package), 3
  Rvcg-package, 3
  setRays, 7
  vcgArea, 7
  vcgBallPivoting, 8
  vcgBary, 9
  vcgBorder, 9
  vcgBox (vcgSphere), 43
  vcgClean, 10, 42
  vcgClost, 11
  vcgClostKD, 13
  vcgClostOnKDtreetFromBarycenters, 15, 17
  vcgCone (vcgSphere), 43
  vcgCreateKDtree, 16, 16, 17, 40
  vcgCreateKDtreeFromBarycenters, 16, 17
  vcgCurve, 18
  vcgDijkstra, 19
  vcgDodecahedron (vcgSphere), 43
  vcgFaceNormals, 20
  vcgGeodesicPath, 20
  vcgGeodist, 21
  vcgGetEdge, 22, 32
  vcgHexahedron (vcgSphere), 43
  vcgIcosahedron (vcgSphere), 43
  vcgImport, 23
  vcgIsolated, 24
  vcgIsosurface, 25
  vcgIsotropicRemeshing, 26
  vcgKDtree, 27
  vcgKmeans, 28
  vcgMeshes, 29
  vcgMetro, 30
  vcgNonBorderEdge, 32
  vcgObjWrite, 33
  vcgOctahedron (vcgSphere), 43
  vcgOffWrite, 33
  vcgPlyRead, 10, 13, 15, 25, 34, 42
  vcgPlyWrite, 35
  vcgQEdicim, 36
  vcgRaySearch, 37
  vcgSample, 29, 39
  vcgSearchKDtree, 16, 17, 40
  vcgSmooth, 24, 35, 37, 41
  vcgSphere, 43
  vcgSphericalCap (vcgSphere), 43
  vcgSquare (vcgSphere), 43
  vcgStlWrite, 44
  vcgSubdivide, 44
  vcgTetrahedron (vcgSphere), 43
  vcgUniformRemesh, 45
  vcgUpdateNormals, 46
  vcgVerteexNormals, 47
  vcgVfadji, 48
  vcgVolume, 49
  vcgWrlWrite, 50