Package ‘Rvcg’
February 19, 2015

Type Package
Title Manipulations of Triangular Meshes Based on the VCGLIB API
Version 0.10.1
Date 2014-12-17
Author Stefan Schlager; the authors of VCGLIB for the included version of the code
Maintainer Stefan Schlager <zarquon42@gmail.com>
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 (svn://svn.code.sf.net/p/vcg/code/trunk/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 (>= 3.0.0)
Imports Rcpp
Suggests Morpho, rgl
LinkingTo Rcpp, RcppEigen
License GPL (>= 2) | file LICENSE
Copyright see files COPYRIGHTS for detailed information
LazyLoad yes
Biarch yes
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-12-22 10:13:18
### R topics documented:

- `Rvcg-package` ............................................. 2
- `dummyhead` ............................................. 3
- `humface` .................................................. 3
- `meshintegrity` ................................. 4
- `setRays` ............................................... 4
- `vcgBary` ............................................... 5
- `vcgBorder` ............................................. 5
- `vcgClean` ............................................... 6
- `vcgClost` ............................................... 7
- `vcgClostKD` .......................................... 9
- `vcgCurve` ............................................. 10
- `vcgGetEdge` ......................................... 11
- `vcgImport` ............................................ 12
- `vcgIsolated` ....................................... 13
- `vcgIsosurface` .............................. 14
- `vcgKDtree` .......................................... 15
- `vcgMeshres` .......................................... 15
- `vcgNonBorderEdge` ......................... 16
- `vcgPlyRead` ......................................... 17
- `vcgPlyWrite` ....................................... 18
- `vcgQEdecim` ....................................... 19
- `vcgRaySearch` ................................... 20
- `vcgSample` ........................................... 21
- `vcgSmooth` .......................................... 22
- `vcgStlWrite` ....................................... 24
- `vcgUniformRemesh` ......................... 24
- `vcgUpdateNormals` ......................... 25
- `vcgVFadj` ............................................. 26

### Index

<table>
<thead>
<tr>
<th>Rvcg-package</th>
<th>Interface between R and vcglib libraries for mesh operations</th>
</tr>
</thead>
</table>

### Description

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

### Details

- **Package:** Rvcg
- **Type:** Package
- **Version:** 0.10.1
- **Date:** 2014-12-17
- **License:** GPL
Author(s)

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

References

To be announced

---

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.
humface.lm: landmarks on mesh 'humface' - called by data(humface)
**meshintegrity**  
*check if an object of class mesh3d contains valid data*

**Description**

checks for existence 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.

---

**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**

`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.
v cgBary

get barycenters of all faces of a triangular mesh

Description
get barycenters of all faces of a triangular mesh

Usage
v cgBary(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 <- v cgBary(humface)
## Not run:
require(rgl)
points3d(bary,col=2)
wire3d(humface)
## End(Not run)

v cgBorder

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
v cgBorder(mesh)

Arguments
mesh triangular mesh of class "mesh3d"
Value

bordervb  logical: vector containing boolean value for each vertex, if it is a border vertex.
bordervit  logical: vector containing boolean value for each face, if it is a border vertex.

Author(s)

Stefan Schlager

See Also

vcgPlyRead

Examples

data(humface)
borders <- vcgBorder(humface)
## view border vertices
## Not run:
require(rgl)
points3d(t(humface$vb)[1:3,][which(borders$bordervb == 1),],col=2)
wire3d(humface)
require(rgl)

## End(Not run)

---

vcgClean  

Clean triangular surface meshes

Description

Apply several cleaning algorithms to surface meshes

Usage

vcgClean(mesh, sel = 0, tol = 0, silent = 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.
vcgClost

Details

the vector sel determines which operations are performed in which order. E.g. removing degenerate faces 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

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, ...)


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 1e12.
- **...**: 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

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

vcgClostKD(x, mesh, sign = TRUE, barycentric = FALSE, smoothNormals = FALSE, borderchk = FALSE, k = 50,nofPoints = 16, maxDepth = 64, angdev = NULL, weightnorm = 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.
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).
...
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 vcgclost 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

---

**vcgCurve**
*calculate curvature of a triangular mesh*

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

**Usage**
vcgCurve(mesh)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh</td>
<td>triangular mesh (object of class ‘mesh3d’)</td>
</tr>
</tbody>
</table>

**Value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gaussvb</td>
<td>per vertex gaussian curvature</td>
</tr>
<tr>
<td>meanvb</td>
<td>per vertex mean curvature</td>
</tr>
<tr>
<td>RMSvb</td>
<td>per vertex RMS curvature</td>
</tr>
<tr>
<td>gaussitmax</td>
<td>per face maximum gaussian curvature of adjacent vertices</td>
</tr>
<tr>
<td>borderit</td>
<td>per face information if it is on the mesh’s border (0=FALSE, 1=TRUE)</td>
</tr>
<tr>
<td>bordervb</td>
<td>per vertex information if it is on the mesh’s border (0=FALSE, 1=TRUE)</td>
</tr>
<tr>
<td>meanitmax</td>
<td>per face maximum mean curvature of adjacent vertices</td>
</tr>
</tbody>
</table>

---
vcgGetEdge

Examples

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

## End(Not run)
```

vcgGetEdge

*Get all edges of a triangular mesh*

Description

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

Usage

```r
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

```r
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 contains 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 homologous 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
vcgIsolated

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, 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 of the mesh will be returned.
silent logical, if TRUE no console output is issued.

Value

returns the reduced mesh.

Author(s)

Stefan Schlager

See Also

vcgPlyRead
vcgIsosurface

Create Isosurface from 3D-array

Description

Create Isosurface from 3D-array using Marching Cubes algorithm

Usage

vcgIsosurface(vol, threshold, spacing = NULL, origin = NULL)

Arguments

vol an integer valued 3D-array
threshold threshold for creating the surface
spacing numeric 3D-vector: specifies the voxel dimensions in x,y,z direction.
origin numeric 3D-vector: origin of the original data set, will transpose the mesh onto that origin.

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)
**vcgKDtree**

*perform kdtree search for 3D-coordinates.*

**Description**

perform kdtree search for 3D-coordinates.

**Usage**

```r
vcgKDtree(target, query, k)
```

**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

**Value**

- `index`: integer matrices with indices of closest points
- `distances`: corresponding distances

---

**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**

```r
vcgMeshRes(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)

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 =
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
vcgPlyRead

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=1)
## now find the edge - hint: it is at the neck.

## End(Not run)

vcgPlyRead

Import ascii or binary PLY files.

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 unreference vertices will be removed.

Value

Object of class "mesh3d"
with:

vb 3 x n matrix containing n vertices as homolouguous 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

**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, ...)

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

**Arguments**

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

**Examples**

data(humface)
vcgPlyWrite(humface, filename = "humface")
vCGQEdcem

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

vCGQEdcem(mesh, tarface = NULL, percent = NULL, edgeLength = NULL, topo = FALSE, quality = TRUE, bound = FALSE, optiplace = TRUE, scaleindi = TRUE, normcheck = FALSE, safeheap = FALSE, 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.
- **scaleindi**: logical: if TRUE, decimation is scale independent.
- **normcheck**: logical: if TRUE, normal directions are considered.
- **safeheap**: logical: if TRUE, safeheap update option enabled.
- **qthresh**: numeric: Quality threshold for decimation process.
- **boundweight**: numeric: Weight assigned to mesh boundaries.
- **normalthr**: numeric: threshold for normal check in radians.
- **silent**: logical, if TRUE no console output is issued.

Details

This is basically an adaption of the cli tridecimator from vcglib

Value

Returns a reduced mesh of class mesh3d.
vchgRaySearch

check if a mesh is intersected by a set of rays

Description
check if a mesh is intersected by a set of rays (stored as normals)

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

Arguments

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

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.
vcgSample

Details

vcgRaySearch projects a mesh (or set of 3D-coordinates) along a set of given rays (stored as normals) 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.1m, humface)
# offset landmarks along their normals for a negative amount of -5mm
lms$vb[1:3,] <- lms$vb[1:3,]+lms$normals[1:3,]*-5
intersect <- vcgRaySearch(lms, humface)
## Not run:
require(Morpho)
require(rgl)
spheres3d(vert2points(lms),radius=0.5,col=3)
plotNormals(lms,long=5)
spheres3d(vert2points(intersect),col=2) # plot intersections
wire3d(humface,col="white")#

## End(Not run)

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

vcgSample(mesh, SampleNum = 100, type = c("km", "pd", "mc"), MCsamp = 20, geodes = TRUE, strict = FALSE)
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"=kmean 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.

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.

Value

- sampled points

Examples

```r
data(humface)
ss <- vcgSample(humface, SampleNum = 500, type="pd")
## Not run:
require(rgl)
points3d(ss)
## End(Not run)
```

vcgSmooth | Smoothes 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)
```
vcgSmooth

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 homolougous 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

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

## End(Not run)
```

---

**vcgStlWrite**

*Export meshes to STL-files*

**Description**

Export meshes to STL-files (binary or ascii)

**Usage**

```r
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 `.ply` will be added automatically.
- `binary` : logical: write binary file

**Examples**

```r
data(humface)
vcgStlWrite(humface, filename = "humface")
```

---

**vcgUniformRemesh**

*Resample a mesh uniformly*

**Description**

Resample a mesh uniformly

**Usage**

```r
vcgUniformRemesh(x, voxelSize = NULL, offset = 0, discretize = FALSE,
                   multiSample = FALSE, 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.
- **mergeClose**: logical: merge close vertices
- **silent**: logical: suppress messages

Value

resampled mesh

Examples

```r
## 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

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

vcgVFadj

find all faces belonging to each vertex in a mesh

Description

find all faces belonging to each vertex in a mesh and report their indices

Usage

vcgVFadj(mesh)

Arguments

mesh: triangular mesh of class "mesh3d"

Value

list containing one vector per vertex containing the indices of the adjacent faces
Index

*Topic `\textasciitildekwd1`
  vcgBorder, 5
  vcgClosest, 7
  vcgImport, 12
  vcgIsolated, 13
  vcgMeshres, 15
  vcgPlyRead, 17
  vcgQEdecom, 19
  vcgSmooth, 22

*Topic `\textasciitildekwd2`
  vcgBorder, 5
  vcgClosest, 7
  vcgImport, 12
  vcgIsolated, 13
  vcgMeshres, 15
  vcgPlyRead, 17
  vcgQEdecom, 19
  vcgSmooth, 22

*Topic `datasets`
  dummyhead, 3
  humface, 3

*Topic `package`
  Rvcg-package, 2

dummyhead, 3

humface, 3

meshintegrity, 4

Rvcg (Rvcg-package), 2
Rvcg-package, 2

setRays, 4

vcgBary, 5
vcgBorder, 5
vcgClean, 6, 23
vcgClosest, 7
vcgClosestKD, 9
vcgCurve, 10

vcgGetEdge, 11, 16
vcgImport, 12
vcgIsolated, 13
vcgIsosurface, 14
vcgKdtree, 15
vcgMeshes, 15
vcgNonBorderEdge, 16
vcgPlyRead, 6, 8, 10, 13, 17, 23
vcgPlyWrite, 18
vcgQEdecom, 19
vcgRaySearch, 20
vcgSample, 21
vcgSmooth, 13, 18, 20, 22
vcgStlWrite, 24
vcgUniformRemesh, 24
vcgUpdateNormals, 25
vcgVFadj, 26

27