Package 'doolkit'

May 14, 2022

Title Exploration of Dental Surface Topography

Version 1.42.1

Description Tools for exploring the topography of 3d triangle meshes. The functions were developed with dental surfaces in mind, but could be applied to any triangle mesh of class 'mesh3d'. More specifically, 'doolkit' allows to isolate the border of a mesh, or a subpart of the mesh using the polygon networks method; crop a mesh; compute basic descriptors (elevation, orientation, footprint area); compute slope, angularity and relief index (Ungar and Williamson (2000) <https://palaeo-electronica.org/2000_1/gorilla/issue1_00.htm>; Boyer (2008) <doi:10.1016/j.jhevol.2008.08.002>), inclination and occlusal relief index or gamma (Guy et al. (2013) <doi:10.1371/journal.pone.0066142>), OPC (Evans et al. (2007) <doi:10.1038/nature05433>), OPCR (Wilson et al. (2012) <doi:10.1038/nature10880>), DNE (Bunn et al. (2011) <doi:10.1002/ajpa.21489>; Pampush et al. (2016) <doi:10.1007/s10914-016-9326-0>), form factor (Horton (1932) <doi:10.1029/TR013i001p00350>), basin elongation (Schum (1956) <doi:10.1130/0016-7606(1956)67[597:EODSAS]2.0.CO;2>), lemniscate ratio (Chorley et al; (1957) <doi:10.2475/ajs.255.2.138>), enamel-dentine distance (Guy et al. (2015) <doi:10.1371/journal.pone.0138802>; Thiery et al. (2017) <doi:10.3389/fphys.2017.00524>), absolute crown strength (Schwartz et al. (2020) <doi:10.1098/rsbl.2019.0671>), relief rate (Thiery et al. (2019) <doi:10.1002/ajpa.23916>) and area-relative curvature; draw cumulative profiles of a topographic variable; and map a variable over a 3d triangle mesh.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

RoxygenNote 7.1.2

Imports ggplot2, igraph, Morpho, rgl, Rvcg, sp, MASS, tis, methods, concaveman, usethis

NeedsCompilation no

Author Ghislain Thiery [aut, cre],
Franck Guy [aut],
Vincent Lazzari [aut]

Maintainer Ghislain Thiery <ghislain.thiery@ntymail.com>

Repository CRAN

Date/Publication 2022-05-14 06:10:09 UTC

R topics documented:

`angularity` *angularity*

Description

Compute the angularity (delta slope).

Usage

```
angularity(mesh, ratio = FALSE)
```

Arguments

mesh	object of class mesh3d
ratio	logical, if false standard angularity will be computed (default), if true a relative angularity ratio will be computed (see below)

Value

If ratio = FALSE, a numeric vector of angularity values i.e. delta slope of each polygon compared to their adjacent polygons, for all the polygons of the mesh. If ratio = TRUE, a numeric vector of angularity ratio values. Ratio is computed by dividing polygon slope by 90, replacing vertex elevation by the average ratio of faces adjacent to the vertex, then dividing the slope of polygons from this new mesh by 90. Although it is a non-standard variable, it was implemented because it better discriminates sharp edges than basic angularity. Warning: both options can take up a significant amount of time for large meshes.

References

Ungar and Williamson (2000)

Examples

```
delta_slope <- angularity(dkmodel$complex, ratio = FALSE)
summary(delta_slope)
#angularity ratio:
delta_slope_ratio <- angularity(dkmodel$complex, ratio = TRUE)
summary(delta_slope_ratio)

#render on a map:
dkmap(dkmodel$complex, delta_slope, col = "slope",
legend.lab = "Angularity (°)")
#angularity ratio:
dkmap(dkmodel$complex, delta_slope_ratio, col = "slope",
legend.lab = "Angularity ratio")
```

---

arc *Average-Relative Curvature (ARC)*

Description

Compute a scale-free estimate of mean curvature.

Usage

```
arc(mesh, range = c(0.01, 0.99))
```

Arguments

mesh	object of class mesh3d
range	a numeric vector of the form c('minrange', 'maxrange') indicating the set limits for extreme values, beyond which arc values will be truncated. If 'minrange' and 'maxrange' are comprised between 0 and 1, they are used as arc percentages. If 'minrange' and 'maxrange' are not comprised between 0 and 1, they are used as raw arc values

Details

Area-relative curvature (ARC) is obtained by dividing the mean curvature of each triangle by the mean curvature of an hemisphere, the surface area of which is the same as the surface area of the total mesh object. As a result, ARC is a scale-free estimate of surface curvature. It can be used to estimate the sharpness of a mesh.

Value

A numeric vector of area-relative curvature values for all the polygons of the mesh.

Examples

```
curvature <- arc(dkmodel$complex)
summary(curvature)

#There is a default truncature of extreme values below 1% or above 99%.
#Without truncature:
curvature <- arc(dkmodel$complex, range = c(0, 1))
summary(curvature)

#mean positive ARC:
parc <- mean(curvature[curvature >= 0])
#mean negative ARC:
narc <- mean(curvature[curvature < 0])

#render on a map:
dkmap(dkmodel$complex, curvature, col = "arc",
min.range = -20, max.range = 20)
#absolute truncature of the values above 20 or below -20:
dkmap(dkmodel$complex, curvature, col = "arc", min.range = -20, max.range = 20)
```

---

area2d *2D surface area*

Description

Compute the area of a 2d projection on the (xy) plane.

Usage

```
area2d(mesh, method = "concave")
```

Arguments

mesh	object of class mesh3d
method	the method used to compute the hull of the 2d projection, either 'convex' (see chull), or 'concave' (see concaveman). The default method is 'concave'.

Value

A single 2D surface area value, corresponding to the footprint of the mesh.

See Also

rfi

Examples

```
#The following objects should have the exact same footprints:
area2d(dkmodel$basin)
area2d(dkmodel$complex)
area2d(dkmodel$cusp)
area2d(dkmodel$flat)

#Graphical rendering of convex hull
x <- dkmodel$cusp
FootprintVerts <- t(x$vb[1:2, ])
Hull <- grDevices::chull(x = FootprintVerts[, 1], y = FootprintVerts[, 2])
plot(x$vb[1, ], x$vb[2, ], xlab = "x", ylab = "y")
points(x$vb[1, Hull], x$vb[2, Hull], col = "orange1")

#Graphical rendering of concave hull
x <- dkmodel$cusp
FootprintVerts <- t(x$vb[1:2, ])
FootprintVerts[, 1] <- FootprintVerts[, 1] - min(FootprintVerts[, 1])
FootprintVerts[, 2] <- FootprintVerts[, 2] - min(FootprintVerts[, 2])
Hull <- concaveman::concaveman(points = FootprintVerts)
plot(x$vb[1, ] - min(x$vb[1, ]), x$vb[2, ] - min(x$vb[2, ]), xlab = "x", ylab = "y")
points(Hull, col = "green1")
```

---

dkborder *dkborder*

Description

Selects the border of a 3d triangle mesh.

Usage

```
dkborder(mesh)
```

Arguments

mesh	object of class mesh3d

Value

A vector of indices corresponding to the triangles with at least one vertex on the border of the mesh.

Examples

```
border <- dkborder(dkmodel$cusp)

# Map the border in orange:
is_border <- rep(1, Rvcg::nfaces(dkmodel$cusp))
is_border[border] <- 2
dkmap(dkmodel$cusp, is_border, col = c("white", "#E69F00"), col.levels = 2, legend = FALSE,
scalebar = FALSE)

# Compare with vcgBorder from the R package Rvcg, in blue:
vcgborder <- which(Rvcg::vcgBorder(dkmodel$cusp)$borderit == TRUE)
is_border[vcgborder] <- 3
dkmap(dkmodel$cusp, is_border, col = c("white", "#E69F00", "#56B4E9"), col.levels = 3,
legend = FALSE, scalebar = FALSE)
#As you can see, it all depends on what you want to select!
```

---

dkcrop *crop a mesh*

Description

Crop a 3d triangle mesh.

Usage

```
dkcrop(mesh, y)
```

Arguments

mesh	object of class mesh3d
y	numeric vector indicating which polygons should be cropped; or an object of class `polygon.network`

Value

A new object of class mesh3d for which all polygons out of y have been removed.

Examples

```
#Crop above a certain threshold:
mythreshold <- quantile(elev(dkmodel$basin), 0.5)
mypolynetwork <- poly.network(dkmodel$basin, elev(dkmodel$basin),
lwr.limit = mythreshold)
mynewmesh <- dkcrop(dkmodel$basin, mypolynetwork)
dkmap(mynewmesh, elev(mynewmesh))

#Crop the sharpest dental elements:
sharpmesh <- dkcrop(dkmodel$basin, poly.network(dkmodel$basin,
Rvcg::vcgCurve(dkmodel$basin)$meanitmax,
lwr.limit = quantile(Rvcg::vcgCurve(dkmodel$basin)$meanitmax, 0.8),
min.size = 50))
dkmap(sharpmesh, arc(sharpmesh), col = "arc", col.levels = 20,
min.range = -20, max.range = 20)
#Map of the sharpest elements' elevation, slope and orientation;
dkmap(sharpmesh, elev(sharpmesh), col = "elev")
dkmap(sharpmesh, slope(sharpmesh), col = "slope", col.levels = 9,
min.range = 0, max.range = 90)
dkmap(sharpmesh, orient(sharpmesh), col = "orient", col.levels = 8,
min.range = 0, max.range = 360)
```

---

dkmap *3d topographic map*

Description

Map topographic variables on a 3d triangle mesh.

Usage

```
dkmap(
  mesh,
  y,
  alpha = 1,
  alpha.above = TRUE,
  alpha.faces = NULL,
  alpha.thresh = NULL,
  bg = "white",
  col = "slope",
  col.levels = 100,
  col.main = "black",
  col.lab = "black",
  col.sub = "black",
  col.axis = "black",
  max.range = NULL,
  min.range = NULL,
```

```
  lit = TRUE,
  cex = 2,
  cex.axis = 2,
  cex.main = 4,
  cex.sub = 3,
  cex.lab = 2,
  family = "sans",
  font.axis = 1,
  font.lab = 2,
  font.main = 3,
  font.sub = 2,
  main = "",
  sub = "",
  legend = TRUE,
  legend.lab = "y",
  legend.type = "stack",
  windowRect = NULL,
  orient = "current",
  bbox = FALSE,
  origin = TRUE,
  scalebar = FALSE,
  smooth = TRUE
)
```

Arguments

mesh	object of class mesh3d
y	a vector of values for each polygon of the mesh, usually a topographic variable
alpha	an integer between 0 and 1 corresponding to alpha value for the selected polygons (see alpha.above, alpha.faces and alpha.thresh)
alpha.above	logical, if TRUE polygons affected by alpha should have a y value above alpha.thresh, if FALSE their y value should be below alpha.thresh
alpha.faces	a numeric vector of indices indicating which faces affected by alpha
alpha.thresh	a numeric value indicating a threshold for alpha
bg	the color to be used for the background. Defaults to "white".
col	a vector of colors for texturing the polygons according to y
col.levels	the number of color levels
col.main	the color to be used for legend main titles. Defaults to "black".
col.lab	the color to be used for the legend labels. Defaults to "black".
col.sub	the color to be used for plot sub-titles. Defaults to "black".
col.axis	the color to be used for legend axis annotation. Defaults to "black".
max.range	optional; the maximal range of the scale
min.range	optional; the minimal range of the scale
lit	logical, specifying if lighting calculation should take place on geometry

cex	a numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. This starts as 1 when a device is opened, and is reset when the layout is changed, e.g. by setting mfrow.
cex.axis	the magnification to be used for legend axis annotation relative to the current setting of cex.
cex.main	the magnification to be used for main titles relative to the current setting of cex.
cex.sub	the magnification to be used for sub-titles relative to the current setting of cex.
cex.lab	the magnification to be used for legend labels relative to the current setting of cex.
family	the name of a font family for drawing text. The maximum allowed length is 200 bytes. This name gets mapped by each graphics device to a device-specific font description. The default value is "" which means that the default device fonts will be used (and what those are should be listed on the help page for the device). Standard values are "serif", "sans" and "mono", and the Hershey font families are also available.
font.axis	the font to be used for axis annotation.
font.lab	the font to be used for the legend axis
font.main	the font to be used for plot main titles.
font.sub	the font to be used for plot sub-titles.
main	the main title (on top) using font, size (character expansion) and color par(c("font.main", "cex.main", "col.main"))
sub	sub-title (at bottom) using font, size and color par(c("font.sub", "cex.sub", "col.sub"))
legend	a logical indicating whether a legend should be displayed.
legend.lab	a label for the legend axis.
legend.type	a character string specifying the type of legend to be used; default is "stack", which corresponds to a stacked vertical legend; "pie" generates a pie-shaped legend and "log" generates a stacked vertical legend, but does a log transformation of the data (base: e=exp(1)). The "log" is mostly useful for DNE maps.
windowRect	the dimensions of the rgl window (default is the current size or, if size is below 1000*800, c(20, 20, 1020, 820))
orient	the orientation of the view. For more details, see dksetview
bbox	a logical, if TRUE a bounding box will be displayed around the surface object
origin	logical, whether to set the z of the mesh's lowermost point to zero
scalebar	A logical indicating whether a scalebar should be displayed
smooth	A logical indicating whether the color of polygons should blend with neighbor polygons for a smoother rendering

Value

An rgl window displaying the topography of a variable over a 3d mesh.

See Also

rgl

Examples

```
#Map of orientation:
orient <- orient(dkmodel$complex)
dkmap(dkmodel$complex, orient, col.levels = 8, col = "orient",
legend.lab = "Orientation (degrees)",legend.type = "pie", min.range = 0,
max.range = 360, orient = "occlusal")

#Map of area-relative curvature:
arc <- arc(dkmodel$complex)
dkmap(dkmodel$complex, arc, col = "arc", legend.lab = "ARC",
min.range = -20, max.range = 20, col.levels = 15, orient = "occlusal")

#Map of Dirichlet normal energy:
dne <- dne(dkmodel$complex)
dkmap(dkmodel$complex, dne, col = "dne", legend.lab = "DNE",
legend.type = "log", orient = "occlusal")

#Map of 3d-Area of polygons (for surface checking):
dkmap(dkmodel$complex, Rvcg::vcgArea(dkmodel$complex, perface = TRUE)$pertriangle,
legend.lab = "3d Area (mm\U00B2)", orient = "occlusal")
```

`dkmodel` *dkmodel*

Description

A list containing theoretical model surfaces corresponding to a flat surface, a single cusp, a shallow basin and a complex surface.

Usage

```
dkmodel
```

Format

An object of class `list` of length 4.

Source

https://github.com/nialsiG/A-comparison-of-relief-estimates-used-in-3d-dental-topography

Examples

```
Flat_surface <- dkmodel$flat
Single_cusp <- dkmodel$cusp
Shallow_basin <- dkmodel$basin
Complex_surface <- dkmodel$complex
```

dkorigin *dkorigin*

Description

Sets the lowermost point of the mesh to 0 on the Z-axis

Usage

```
dkorigin(mesh)
```

Arguments

mesh	object of class mesh3d

Value

An object of class mesh3d.

Examples

```
#Map of elevation before using dkorigin:
dkmap(dkpongo$OES, doolkit::elev(dkpongo$OES), col = "elev", legend.lab = "Elevation (mm)")

#Map of elevation after dkorigin:
leveled <- dkorigin(dkpongo$OES)
dkmap(leveled, doolkit::elev(leveled), col = "elev", legend.lab = "Elevation (mm)")
```

dkpongo *dkpongo*

Description

A dataset containing the OES and the EDJ surfaces of a *Pongo pygmaeus* upper second molar (SMF-1117)

Usage

```
dkpongo
```

Format

An object of class `list` of length 2.

Source

https://www.morphosource.org/Detail/MediaDetail/Show/media_id/42357

Examples

```
Pongo_OES <- dkpongo$OES
Pongo_EDJ <- dkpongo$EDJ
```

`dkprofile` *cumulative profile, its slope and the area under its curve*

Description

A function for drawing the cumulative profile of a variable, computing the area under the curve and the slope of the profile at the arithmetic mean of the variable.

Usage

```
dkprofile(
  x,
  type = "cartesian",
  xlab = paste("cumulated frequency (%)"),
  ylab = "",
  main = "",
  col = "red",
  alpha = 1,
  size = 1,
  linetype = "solid"
)
```

Arguments

`x`	a numeric vector
`type`	a character string indicating the type of coordinates to use ("cartesian", "polar" etc.). Currently only "cartesian" is supported.
`xlab`	title of the x axis
`ylab`	title of the y axis
`main`	main title of the plot
`col`	the color of data points
`alpha`	numeric indicating the alpha value of data points
`size`	the size of data points
`linetype`	the type of line to be traced (see ggplot2)

Value

A list containing (1) the area under the curve of the profile, (2) the profile to be drawn, and (3) the slope of the profile at the mean of the variable.

References

doi:10.3389/fphys.2017.00524Thiery et al. (2017)

Examples

```
#Elevation (hypsometric) profile (see Thiery et al., 2017):
dkprofile(elev(dkpongo$OES), main = "Elevation profile - Pongo pygmaeus",
ylab = "Elevation (%)", col = "#0072B2", linetype = "solid")

#Enamel-dentine distance (pachymetric) profile:
dkprofile(oedist(dkpongo$OES, dkpongo$EDJ),
main = "Elevation profile - Pongo pygmaeus", ylab = "Distance (%)",
col = "#F0E442", linetype = "dashed")

#Curvature (kurtometric) profile:
dkprofile(Rvcg::vcgCurve(dkpongo$OES)$meanitmax,
main = "Curvature profile - Pongo pygmaeus", ylab = "Curvature (%)",
col = "#D55E00", linetype = "dotted")
```

---

dksetview *preset orientations*

Description

A function to orient 3d topographical maps using preset values.

Usage

```
dksetview(orient = "occlusal")
```

Arguments

orient	a character string indicating the targeted orientation (default is occlusal)

Value

sets the orientation of the 'rgl' window.

See Also

dkmap

Examples

```
dkmap(dkmodel$cusp, inclin(dkmodel$cusp), col = "inclin", min.range = 0, max.range = 180)
dksetview()
#possible orientations are "distal", "left", "occlusal", "mesial" and "right"
```

dne *Dirichlet normal energy*

Description

Compute the Dirichlet normal energy.

Usage

```
dne(mesh, range = 0.999, total = FALSE)
```

Arguments

mesh	object of class mesh3d
range	an integer between 0 and 1 indicating the percentage of values to consider for the computation. Following Pampush et al. (2016) default is set to 0.999.
total	logical, should the result of the function be the total DNE.

Value

If total = FALSE, a numeric vector of dne values for all the polygons of the mesh. If total = TRUE, a single DNE value, calculated as the sum of the products of polygon normal energies * polygon areas.

References

doi:10.1002/ajpa.21489Bunn et al. (2011)

doi:10.1007/s1091401693260Pampush et al. (2016)

Examples

```
dne <- dne(dkmodel$complex)
summary(dkmodel$complex)

#total DNE value corresponds to the sum of products Dne * Area:
round(sum(dne*Rvcg::vcgArea(dkmodel$complex, perface = TRUE)$pertriangle), 3)
#can be directly computed using \code{dne}:
dne(dkmodel$complex, total = TRUE)

#render on a map:
dkmap(dkmodel$complex, dne, legend.type = "log", col = "dne")
```

elev *elevation*

Description

Compute the elevation (z component of triangle barycenter).

Usage

```
elev(mesh, origin = TRUE)
```

Arguments

mesh	object of class mesh3d
origin	logical, if TRUE the z of the mesh is adjusted so that the lowest z = 0 (see dkorigin)

Value

A numeric vector of elevation values for all the polygons of the mesh.

See Also

inclin

rfi

slope

Examples

```
elev <- elev(dkmodel$cusp)
summary(elev)

#render on a map:
dkmap(dkmodel$cusp, elev)
```

hypso *hypso*

Description

Compute the maximum height, length, width and corresponding hypsodonty index (ratio of the maximum height over the maximum length)

Usage

```
hypso(mesh, origin = TRUE)
```

Arguments

mesh	object of class mesh3d
origin	logical, whether to set the z of the mesh's lowermost point to zero.

Value

A list of values for hypsodonty index, height, length and width of the mesh. The hypsodonty index is not expressed relative to 100 but to 1. Note: the tooth surface is expected to be oriented such as the X-axis is the bucco-lingual axis, the Y-axis is the mesio-distal axis, and the occlusal plane is parallel to the (XY) plane and faces upward.

Examples

```
hypso(dkmodel$cusp)
```

---

inclin *inclin*

Description

Compute inclination i.e. the angle between triangles and the vertical plane in degrees, comprised between 0 and 180.

Usage

```
inclin(mesh)
```

Arguments

mesh	object of class mesh3d

Value

A numeric vector of inclination values for all the polygons of the mesh.

References

doi:10.1371/journal.pone.0066142Guy et al. (2013)

See Also

slope

Examples

```
inclin <- inclin(dkmodel$cusp)
summary(inclin)

#render on a map:
dkmap(dkmodel$cusp, inclin, col = "inclin",
min.range = 0, max.range = 180, legend = TRUE)
```

`oedist` *Distance from outer enamel surface to enamel dentine junction*

Description

Compute the distance from enamel vertices to dentine mesh.

Usage

```
oedist(oes, edj, ray = FALSE)
```

Arguments

`oes`	object of class mesh3d; should be the outer enamel surface
`edj`	object of class mesh3d; should be the enamel-dentine junction
`ray`	logical, if TRUE the search is along vertex normals (default is FALSE)

Value

A numeric vector of vertex-to-mesh distance values for all the polygons of the x mesh.

References

doi:10.1371/journal.pone.0066142Guy et al. (2013)

doi:10.1371/journal.pone.0138802Guy et al. (2015)

doi:10.3389/fphys.2017.00524Thiery et al. (2017)

doi:10.1098/rsbl.2019.0671Schwartz et al. (2020)

See Also

`meshDist`

Examples

```
edd <- oedist(dkmodel$cusp, dkmodel$flat)
summary(edd)
AETgeom <- mean(edd)
#Geometric relative enamel thickness, obtained by dividing AETgeom by the
#square root of EDJ area
#Note: it is different from classic RET which requires the volume of the
#dentine inside the enamel cap (see Thiery et al., 2017)
AETgeom/sqrt(Rvcg::vcgArea(dkmodel$flat))
#Absolute crown strength:
edj_radius <- max(dist(cbind(dkmodel$flat$vb[1,], dkmodel$flat$vb[2,])))/2
sqrt(mean(edd) * edj_radius)

#render on a map:
oedist <- doolkit::oedist(dkmodel$cusp, dkmodel$flat)
dkmap(dkmodel$cusp, oedist)
#distance map can also be rendered on EDJ surface:
eodist <- oedist(dkmodel$flat, dkmodel$cusp)
dkmap(dkmodel$flat, eodist)
```

---

opc *orientation patch count*

Description

Count the number of orientation patches using `poly.network`.

Usage

```
opc(mesh, bins = 8, min.size = 3, rotation = 0)
```

Arguments

`mesh`	object of class mesh3d
`bins`	the number of orientation bins to be defined (default set to 8)
`min.size`	the minimal amount of polygons defining a "patch" (default set to 3)
`rotation`	if applicable, the number of degrees to which bins are to be rotated. By default the bins start from an angle of pi/2 and rotates clockwise.

Value

A data.frame displaying the number of patches and their size (number of triangles) for each orientation bin. Note: if you want the surface area of each patch, see`poly.network`

References

doi:10.1038/nature05433Evans et al. (2007)

See Also

orient

opcr

Examples

```
#8 bins (default):
opc <- opc(dkmodel$complex)
#8 bins starting from mesial, as in Evans et al. 2007:
opc <- opc(dkmodel$complex, rotation = -(360/16))
#4 bins (mesial, buccal, distal and lingual):
opc <- opc(dkmodel$complex, bins = 4, rotation = -(360/8))
```

---

opcr *orientation patch count rotated*

Description

Compute the orientation patch count rotated of a triangle mesh.

Usage

```
opcr(mesh, bins = 8, min.size = 3)
```

Arguments

mesh	object of class mesh3d
bins	the number of orientation bins to be defined (default set to 8)
min.size	the minimal amount of polygons defining a "patch" (default set to 3)

Value

A data.frame displaying the number of patches and their size (number of triangles) for each orientation bin.

References

doi:10.1038/nature10880Wilson et al. (2012)

See Also

opc

orient

Examples

```
#8bins (default):
opcr <- opcr(dkmodel$complex)
#less bins:
opcr <- opcr(dkmodel$complex, bins = 4)
#larger patches:
opcr <- opcr(dkmodel$complex, min.size = 50)
```

`orient` *orientation of polygons*

Description

Returns the occlusal orientation (exposure in GIS)

Usage

```
orient(mesh)
```

Arguments

`mesh`	object of class mesh3d

Value

A numeric vector of occlusal orientation values in degrees for all the polygons of the mesh. Let the orientation from above be depicted as a trigonomical circle, then for a tooth positioned as in Guy et al. (2015) an orientation of 0 (mesial) would be located at an angle of pi/2, and an orientation of 90° (buccal) would be located at an angle of 2*pi.

See Also

`opc`

`opcr`

Examples

```
orient <- orient(dkmodel$complex)
summary(orient)
```

poly.network *Identify polygon networks*

Description

From a selected variable y, identifies patches of adjacent polygons that share a given range of y values. These patches are called 'polygon networks'.

Usage

```
poly.network(
  mesh,
  y,
  lwr.limit = stats::quantile(y, 0.75),
  upr.limit = stats::quantile(y, 1),
  min.size = 3
)
```

Arguments

mesh	object of class mesh3d
y	a vector of values to be used to select polygons
lwr.limit	the lower range of values to be selected from y
upr.limit	the upper range of values to be selected from y
min.size	the minimum amount of polygons defining a cluster. Default is set to 3.

Value

An object of class "polygon.network" composed of the face index and the membership of each triangle answering the set conditions. The function makes patches of contiguous triangles, and each patch is indexed with a unique number corresponding to its membership.

Examples

```
#Isolate cusps using elevation:
mythreshold <- quantile(elev(dkmodel$cusp), 0.65)
cusps <- poly.network(dkmodel$cusp, elev(dkmodel$cusp), lwr.limit = mythreshold,
min.size = 100)
myvector <- rep(0, Rvcg::nfaces(dkmodel$cusp))
myvector[cusps@faces] <- cusps@membership[]
myvector <- as.factor(myvector)
ncusps <- length(levels(myvector)) - 1
levels(myvector) <- c(0:ncusps + 1)
dkmap(dkmodel$cusp, as.numeric(myvector), col = cbPalette <- c("#000000", "#E69F00",
"#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7"),
col.levels = ncusps + 1, legend.lab = "Elevation (mm)")
```

```
#Any other variables could be used to define the clusters
#Mean curvature:
crests <- poly.network(dkmodel$complex, Rvcg::vcgCurve(dkmodel$complex)$meanitmax,
lwr.limit = quantile(Rvcg::vcgCurve(dkmodel$complex)$meanitmax, 0.8), min.size = 10)
doolkit::dkmap(mesh = dkmodel$complex, y = doolkit::arc(dkmodel$complex,
range = c(-20, 20)), col = "arc", col.levels = 256, min.range = -20,
max.range = 20, orient = "occlusal", legend.lab = "ARC",
alpha.thresh = quantile(doolkit::arc(dkmodel$complex), 0.8), alpha = 0.3,
alpha.above = FALSE)

valleys <- poly.network(dkmodel$complex, Rvcg::vcgCurve(dkmodel$complex)$meanitmax,
upr.limit = quantile(Rvcg::vcgCurve(dkmodel$complex)$meanitmax, 0.2), min.size = 10)
doolkit::dkmap(mesh = dkmodel$complex, y = doolkit::arc(dkmodel$complex,
range = c(-20, 20)), col = "arc", col.levels = 256, min.range = -20,
max.range = 20, orient = "occlusal", legend.lab = "ARC",
 alpha.thresh = quantile(doolkit::arc(dkmodel$complex), 0.2), alpha = 0.3,
 alpha.above = TRUE)

#Orientation and surface of patches:
patch_orient <- data.frame(bin = NULL, patch = NULL, size = NULL, surface = NULL)
for (i in 1:8) {
  Cluster <- poly.network(dkmodel$complex, orient(dkmodel$complex),
  lwr.limit = 45 * (i - 1), upr.limit = 45 * i)
  Patches <- levels(as.factor(Cluster@membership))
  Bins <- rep(paste(45 * (i - 1), "-", 45 * i), length(Patches))
  Areas <- rep(0, length(Patches))
  for (j in 1:length(Patches)) {
    test <- Cluster@faces[Cluster@membership == Patches[j]]
    Areas[j] <- round(sum(Rvcg::vcgArea(dkmodel$complex,
    perface = TRUE)$pertriangle[test]), 3)
  }
  patch_orient <- data.frame(rbind(patch_orient,
  cbind.data.frame(Bins, Patches, Areas)))
}
#Since patches made of 3 or less polygons are discarded,
#sum of patch areas < total surface area:
sum(patch_orient$Areas)
Rvcg::vcgArea(dkmodel$complex)
```

`polygon.network-class` *S4 class for polygon networks*

Description

Polygon networks are subgraphs made of polygons (i) sharing topographic features and (ii) in contact with the rest of the subgraph by at least 1 polygon edge. Objects of S4 class `polygon.network` are typically made using the function `poly.network`

Slots

`membership` a vector of numeric values specifying, for each triangle, the index number of the patch to which the triangle is assigned

`faces` a vector of numeric values indicating the mesh triangle indexes

---

`rfi` *relief index*

Description

Compute the relief index of a 3d triangle mesh.

Usage

```
rfi(mesh, method = "Ungar", hull = "concave")
```

Arguments

`mesh`	object of class mesh3d
`method`	a character string indicating which method is to be used for the computation of relief index
`hull`	the method used to compute the hull of the 2d projection, either 'convex' or 'concave'. The default method is 'convex'. See `area2d`

Value

A single relief index value.

References

doi:10.1016/j.jhevol.2008.08.002Boyer (2008) doi:10.1371/journal.pone.0066142Guy et al. (2013) Ungar and Williamson (2000)

See Also

`area2d`

Examples

```
rfi <- rfi(dkmodel$cusp, method = "Ungar")
gamma <- rfi(dkmodel$cusp, method = "Guy")
```

`rrate` *relief rate*

Description

Compute the relief rate from a sub-sample of a 3d triangle mesh. For instance, the relief rate could be computed from the portion of a molar above the lowermost point of its central basin, compared to the whole tooth.

Usage

```
rrate(uncropped, cropped, origin = TRUE)
```

Arguments

`uncropped`	object of class mesh3d. Should entirely contain the 'cropped' argument.
`cropped`	object of class mesh3d. Should be part of the 'uncropped' argument.
`origin`	logical, if TRUE both cropped and uncropped mesh are translated along the z-axis so that the lowest z of the uncropped mesh = 0; see `dkorigin`

Value

A single relief rate value.

References

doi:10.1002/ajpa.23916Thiery et al. (2019)

Examples

```
medelev <- median(elev(dkmodel$cusp))
basins <- dkcrop(dkmodel$cusp, which(elev(dkmodel$cusp) < medelev))
cusps <- dkcrop(dkmodel$cusp, which(elev(dkmodel$cusp) > medelev))

rrate(dkmodel$cusp, basins)
rrate(dkmodel$cusp, cusps)
```

shape.index *shape.index*

Description

Compute various shape indices.

Usage

```
shape.index(mesh, origin = TRUE)
```

Arguments

mesh	object of class mesh3d
origin	logical, if TRUE the z of the mesh is adjusted so that the lowest z = 0; see dkorigin

Details

A handful of indices have been developed to characterize the shape of natural landscapes, including drainage basins. While some indices are very scale-sensitive (e.g., Gravelius' compactness coefficient), others are dimensionless. Horton (1932) introduced a form factor computed as the quotient of the basin's surface area over the square of the maximum basin length. Schumm (1956) developed a basin elongation index computed as the quotient of twice the square root of surface area over the product of basin length and the squareroot of pi. Lastly, Chorley et al. (1957) developed a lemniscate ratio which corresponds to the ratio between the surface of a lemniscate of same length over the basin area,and computed as (pi*(Length^2))/(4*Area).

Value

A list of indices:

- Form factor (Horton, 1932)
- Basin elongation (Schum, 1956)
- Lemniscate ratio 'K' (Chorley et al., 1957)

References

doi:10.1029/TR013i001p00350Horton (1932)

doi:10.1130/00167606(1956)67[597:EODSAS]2.0.CO;2Schumm (1956)

doi:10.2475/ajs.255.2.138Chorley et al. (1957)

Examples

```
ShapInd <- shape.index(dkmodel$basin)
ShapInd$FormFactor
ShapInd$Elongation
ShapInd$K
```

slope *slope*

Description

Compute slope i.e. the angle between triangles and the horizontal plane in degrees, comprised between 0 and 90.

Usage

```
slope(mesh)
```

Arguments

mesh	object of class mesh3d

Value

A numeric vector of slope values for all the polygons of the mesh.

References

Ungar and Williamson (2000)

See Also

inclin

Examples

```
slope <- slope(dkmodel$cusp)
summary(slope)

#render on a map:
dkmap(dkmodel$cusp, slope, col.levels = 9, col = "slope",
min.range = 0, max.range = 90, legend = TRUE)
```

Index