Package ‘misc3d’

September 6, 2020

Title  Miscellaneous 3D Plots
Version  0.9-0
Author  Dai Feng and Luke Tierney
Maintainer  Luke Tierney <luke-tierney@uiowa.edu>
Suggests  rgl, tkrplot, MASS
Imports  grDevices, graphics, stats, tcltk
Description  A collection of miscellaneous 3d plots, including isosurfaces.
License  GPL
NeedsCompilation  no
Repository  CRAN
Date/Publication  2020-09-06 04:50:06 UTC

R topics documented:

computeContour3d  ...................................................... 2
contour3d  .............................................................. 3
drawScene  .............................................................. 6
exportScene  ............................................................ 9
image3d  ................................................................. 10
kde3d  ................................................................. 11
lighting  ................................................................. 12
linesTetrahedra  ....................................................... 13
parametric3d  .......................................................... 14
pointsTetrahedra  ...................................................... 17
slices3d  ................................................................. 18
surfaceTriangles  ...................................................... 19
teapot  ................................................................. 20
triangles  .............................................................. 21

Index  ................................................................. 23
computeContour3d

Compute Isosurface, a Three Dimension Contour

Description

Computes a 3D contours or isosurface by the marching cubes algorithm.

Usage

```r
computeContour3d(vol, maxvol = max(vol), level,
    x = 1:dim(vol)[1],
    y = 1:dim(vol)[2],
    z = 1:dim(vol)[3], mask)
```

Arguments

- `vol` a three dimensional array.
- `maxvol` maximum of the `vol` array.
- `level` The level at which to construct the contour surface.
- `x`, `y`, `z` locations of grid planes at which values in `vol` are measured.
- `mask` a function of 3 arguments returning a logical array, a three dimensional logical array, or `NULL`. If not `NULL`, only cells for which `mask` is true at all eight vertices are used in forming the contour.

Details

Uses the marching-cubes algorithm, with adjustments for dealing with face and internal ambiguities, to compute an isosurface. See references for the details. The function `contour3d` provides a higher-level interface.

Value

A matrix of three columns representing the triangles making up the contour surface. Each row represents a vertex and groups of three rows represent a triangle.

References


See Also

`contour3d`
Examples

```r
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))
con <- computeContour3d(v, max(v), 1)
drawScene(makeTriangles(con))
```

drawScene

Draw an Isosurface, a Three Dimension Contour Plot

Description

Computes and renders 3D contours or isosurfaces computed by the marching cubes algorithm.

Usage

```r
contour3d(f, level, x, y, z, mask = NULL, color = "white", color2 = NA,
          alpha = 1, fill = TRUE, col.mesh = if (fill) NA else color,
          material = "default", smooth = 0,
          add = FALSE, draw = TRUE, engine = "rgl", separate=FALSE, ...)
```

Arguments

- `f` a function of 3 arguments or a three dimensional array.
- `level` The level or levels at which to construct contour surfaces.
- `x, y, z` locations of grid planes at which values in `f` are measured or `f` is to be evaluated. Can be omitted if `f` is an array.
- `mask` a function of 3 arguments returning a logical array, a three dimensional logical array, or NULL. If not NULL, only cells for which `mask` is true at all eight vertices are used in forming the contour. Can also be a list of functions the same length as `level`.
- `color` color to use for the contour surface. Recycled to the length of 'levels'. Can also be a function, or list of functions, of three arguments. These are called for each level with three arguments, the coordinates of the midpoints of the triangles making up the surface. They should return a vector of colors to use for the triangles.
- `color2` opposite face color. Recycled to the length of 'levels'.
- `alpha` alpha channel level, a number between 0 and 1. Recycled to the length of 'levels'.
- `fill` logical; if TRUE, drawing should use filled surfaces; otherwise a wire frame should be drawn. Recycled to the length of 'levels'.
- `col.mesh` color to use for the wire frame. Recycled to the length of 'levels'.
- `smooth` integer or logical specifying Phong shading level for "standard" and "grid" engines or whether or not to use shading for the "rgl" engine. Recycled to the length of 'levels'.
material specification; currently only used by "standard" and "grid" engines. Currently possible values are the character strings "dull", "shiny", "metal", and "default". Recycled to the length of 'levels'.

add logical; if TRUE, add to current rgl graph.
draw logical; if TRUE, draw the results; otherwise, return contour triangles.
engine character; currently "rgl", "standard", "grid" or "none"; for "none" the computed triangles are returned.
separate logical and one for each level; if it is TRUE, and either the engine is "none" or draw is not true, the triangles from the corresponding level are separated into disconnected chunks, namely that triangles from different chunks have no vertex in common. The default is FALSE for each level.

... additional rendering arguments, e.g. material and texture properties for the "rgl" engine. See documentation for drawScene and drawScene.rgl

Details

Uses the marching-cubes algorithm, with adjustments for dealing with face and internal ambiguities, to draw isosurfaces. See references for the details.

Value

For the "rgl" engine the returned value is NULL. For the "standard" and "grid" engines the returned value is the viewing transformation as returned by persp. For the engine "none", or when draw is not true, the returned value is a structure representing the triangles making up the contour, or a list of such structures for multiple contours.

Note

The "rgl" engine now uses the standard rgl coordinates instead of negating y and swapping y and z. If you need to reproduce the previous behavior you can use options(old.misc3d.orientation=TRUE).

Transparency only works properly in the "rgl" engine. For standard or grid graphics on pdf or quartz devices using alpha levels less than 1 does work but the triangle borders show as a less transparent mesh.

References


Daniel Adler, Oleg Nenadic and Walter Zucchini (2003) RGL: A R-library for 3D visualization with OpenGL


See Also

triangles3d, material3d, surface3d.
Examples

#Example 1: Draw a ball
f <- function(x, y, z) x^2 + y^2 + z^2
x <- seq(-2, 2, len=20)
contour3d(f, x, x, x)
contour3d(f, x, x, x, engine = "standard")

# ball with one corner removed.
contour3d(f, x, x, x, mask = function(x,y,z) x > 0 | y > 0 | z > 0)
contour3d(f, x, x, x, mask = function(x,y,z) x > 0 | y > 0 | z > 0,
          engine="standard", screen = list(x = 290, y = -20),
          color = "red", color2 = "white")

# ball with computed colors
w <- function(x,y,z) {
  v <- sin(x) + cos(2 * y) * sin(5 * z)
  r <- range(v)
  n <- 100
  i <- pmax(pmin(ceiling(n * (v - r[1]) / (r[2] - r[1])), n), 1)
  terrain.colors(n)[i]
}
contour3d(f, x, x, x, color = w)

#Example 2: Nested contours of mixture of three tri-variate normal densities
nmix3 <- function(x, y, z, m, s) {
  0.4 * dnorm(x, m, s) * dnorm(y, m, s) * dnorm(z, m, s) +
  0.3 * dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
  0.3 * dnorm(x, m, s) * dnorm(y, -1.5 * m, s) * dnorm(z, m, s)
}
f <- function(x,y,z) nmix3(x,y,z,.5,.5)
g <- function(n = 40, k = 5, alo = 0.1, ahi = 0.5, cmap = heat.colors) {
  th <- seq(0.05, 0.2, len = k)
  col <- rev(cmap(length(th)))
  al <- seq(alo, ahi, len = length(th))
  x <- seq(-2, 2, len=n)
  contour3d(f, th, x, x, x, color=col, alpha=al)
  rgl::bg3d(col="white")
}
g(40,5)
gs <- function(n = 40, k = 5, cmap = heat.colors, ...) {
  th <- seq(0.05, 0.2, len = k)
  col <- rev(cmap(length(th)))
  x <- seq(-2, 2, len=n)
  m <- function(x,y,z) x > .25 | y < -.3
  contour3d(f, th, x, x, x, color=col, mask = m, engine = "standard",
            scale = FALSE, ...)
  rgl::bg3d(col="white")
}
gs(40, 5, screen=list(z = 130, x = -80), color2 = "lightgray", cmap=rainbow)

## Not run:
#Example 3: Nested contours for FMRI data.
library(AnalyzeFMRI)
a <- f.read.analyze.volume(system.file("example.img", package="AnalyzeFMRI"))
a <- a[,,1]
contour3d(a, 1:64, 1:64, 1.5*(1:21), lev=c(3000, 8000, 10000),
        alpha = c(0.2, 0.5, 1), color = c("white", "red", "green"))

# alternative masking out a corner
m <- array(TRUE, dim(a))
m[1:30,1:30,1:10] <- FALSE
contour3d(a, 1:64, 1:64, 1.5*(1:21), lev=c(3000, 8000, 10000),
          mask = m, color = c("white", "red", "green"))
contour3d(a, 1:64, 1:64, 1.5*(1:21), lev=c(3000, 8000, 10000),
          color = c("white", "red", "green"),
          color2 = c("gray", "red", "green"),
          mask = m, engine="standard",
          scale = FALSE, screen=list(z = 60, x = -120))

## End(Not run)

# Example 4: Separate the triangles from the contours of
# mixture of three tri-variate normal densities
nmix3 <- function(x, y, z, m, s) {
    0.3*dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
    0.3*dnorm(x, -2*m, s) * dnorm(y, -2*m, s) * dnorm(z, -2*m, s) +
    0.4*dnorm(x, -3*m, s) * dnorm(y, -3 * m, s) * dnorm(z, -3*m, s) }
f <- function(x,y,z) nmix3(x,y,z,0.5,.1)
n <- 20
x <- y <- z <- seq(-2, 2, len=n)
contour3dObj <- contour3d(f, 0.35, x, y, z, draw=FALSE, separate=TRUE)
for(i in 1:length(contour3dObj))
    contour3dObj[[i]]$color <- rainbow(length(contour3dObj))[i]
drawScene.rgl(contour3dObj)

drawScene

Rendering of Triangular Mesh Surface Data

Description

Draw scenes consisting of one or more surfaces described by triangular mesh data structures.

Usage

drawScene(scene, light = c(0, 0, 1),
          screen = list(z = 40, x = -60), scale = TRUE, R.mat = diag(4),
          perspective = FALSE, distance = if (perspective) 0.2 else 0,
          fill = TRUE, xlim = NULL, ylim = NULL, zlim = NULL,
          aspect = c(1, 1), col.mesh = if (fill) NA else "black",
          polynum = 100, lighting = phongLighting, add = FALSE,
          engine = "standard", col.bg = "transparent", depth = 0,
          newpage = TRUE)
drawScene.rgl(scene, add = FALSE, ...)
**drawScene**

**Arguments**

- **scene**: a triangle mesh object of class `Triangles3D` or a list of such objects representing the scene to be rendered.
- **light**: numeric vector of length 3 or 4. The first three elements represent the direction to the light in viewer coordinates; the viewer is at \((0,0,1 / \text{distance})\) looking down along the positive z-axis. The fourth element, if present, represents light intensity; the default is 1.
- **screen**: as for `panel.3dwire`, a list giving sequence of rotations to be applied to the scene before being rendered. The initial position starts with the viewing point along the positive z-axis, and the x and y axes in the usual position. Each component of the list should be named one of "x", "y" or "z"; repetitions are allowed. The values indicate the amount of rotation about that axis in degrees.
- **scale**: logical. Before viewing the x, y and z coordinates of the scene defining the surface are transformed to the interval [-0.5,0.5]. If scale is true the x, y and z coordinates are transformed separately. Otherwise, the coordinates are scaled so that aspect ratios are retained. Ignored if `draw = TRUE`.
- **R.mat**: initial rotation matrix in homogeneous coordinates, to be applied to the data before screen rotates the view further.
- **perspective**: logical, whether to render a perspective view. Setting this to `FALSE` is equivalent to setting `distance` to 0.
- **distance**: numeric, between 0 and 1, controls amount of perspective. The distance of the viewing point from the origin (in the transformed coordinate system) is \(1 / \text{distance}\). This is described in a little more detail in the documentation for `cloud`.
- **fill**: logical; if `TRUE`, drawing should use filled surfaces or wire frames as indicated by the object properties. Otherwise all objects in the scene should be rendered as wire frames.
- **xlim,ylim,zlim**: x-, y- and z-limits. The scene is rendered so that the rectangular volume defined by these limits is visible.
- **aspect**: vector of length 2. Gives the relative aspects of the y-size/x-size and z-size/x-size of the enclosing cube.
- **col.mesh**: color to use for the wire frame if `frames` is true.
- **polynum**: integer. Number of triangles to pass in batches to grid primitives for the "grid" engine. The default should be adequate.
- **lighting**: a lighting function. Current options are `phongLighting` and `perspLighting`.
- **add**: logical; if `TRUE`, add to current graph.
- **engine**: character; currently "standard" or "grid".
- **col.bg**: background color to use in color depth cueing.
- **depth**: numeric, between 0 and 1. Controls the amount of color blending to `col.bg` for objects farther from the viewer. `depth` equal to zero means no depth cueing.
- **newpage**: logical; if `TRUE`, and `add` is true, then the "grid" engine will call "grid.newpage"; otherwise the current page is used.
- **...**: rgl material and texture properties; see documentation for `rgl.material`
drawScene renders a scene consisting of one or more triangle mesh objects using standard or grid graphics. Object-specific rendering features such as smoothing and material are controlled by setting in the objects. Arguments to drawScene control global factors such as viewer and light position.

drawScene.rgl renders the scene in an rgl window.

If add=TRUE in standard or grid graphics then coordinates are not further scaled after the transformations implied by R.mat, and distance are applied. For the grid engine drawing occurs in the current viewport.

Value

drawScene.rgl returns NULL. The return value of drawScene is the viewing transformation as returned by persp.

Note

The "rgl" engine now uses the standard rgl coordinates instead of negating y and swapping y and z. If you need to reproduce the previous behavior you can use options(old.misc3d.orientation=TRUE).

Transparency only works properly in the "rgl" engine. For standard or grid graphics on devices that support transparency using alpha levels less than 1 does work but the triangle borders show as a less transparent mesh.

See Also

rgl.material

Examples

vtri <- local({
  z <- 2 * volcano
  x <- 10 * (1:nrow(z))
  y <- 10 * (1:ncol(z))
  surfaceTriangles(x, y, z, color="green3")
})
drawScene(vtri, scale = FALSE)
drawScene(vtri, screen=list(x=40, y=-40, z=-135), scale = FALSE)
drawScene(vtri, screen=list(x=40, y=-40, z=-135), scale = FALSE, perspective = TRUE)
drawScene(vtri, screen=list(x=40, y=-40, z=-135), scale = FALSE, perspective = TRUE, depth = 0.4)
exportScene

Writing Out Triangular Mesh Scenes

Description

Writing out scenes consisting of one or more surfaces represented by triangular mesh data structures to textual files.

Usage

exportScene(scene, filename, format=c("OFF", "IDTF", "ASY"))

Arguments

scene a triangle mesh object of class Triangles3D or a list of such objects representing the scene to be exported.
filename the name of the exported textual file.
format the format of the exported textual file. It must be one of "OFF", "IDTF", or "ASY" and can be abbreviated. The default is "OFF".

Details

exportScene writes out scenes to textual files, which can be used for other purposes, for example the generation of U3d and PRC files for interactive 3D visualization in a PDF.

Value

Textual files representing triangular mesh scenes.

Examples

nmix3 <- function(x, y, z, m, s) {
  0.4 * dnorm(x, m, s) * dnorm(y, m, s) * dnorm(z, m, s) +
  0.3 * dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
  0.3 * dnorm(x, m, s) * dnorm(y, -1.5 * m, s) * dnorm(z, m, s)
}

f <- function(x,y,z) nmix3(x,y,z,.5,.5)

gs1 <- function(n = 40, k = 5, cmap = heat.colors, ...) {
  th <- seq(0.05, 0.2, len = k)
  col <- rev(cmap(length(th)))
  x <- seq(-2, 2, len=n)
  m <- function(x,y,z) x > .25 | y < -.3
  contour3d(f,th,x,x,x,color=col, mask = m, engine = "none",
  scale = FALSE, ...)
}
```r
cnts <- gs1(40, 5, screen=list(z = 130, x = -80),
    color2 = "lightgray", cmap=rainbow)
filename <- file.path(tempdir(), "nmix")
exportScene(cnts, filename, "OFF")
```

---

**image3d**  

---

**Draw Points on a 3D Grid**

---

### Description

Plots points on a three dimensional grid representing values in a three dimensional array. Assumes high values are inside and uses alpha blending to make outside points more transparent.

### Usage

```r
image3d(v, x = 1:dim(v)[1], y = 1:dim(v)[2], z = 1:dim(v)[3],
    vlim = quantile(v, c(.9, 1), na.rm=TRUE),
    col = heat.colors(256), alpha.power = 2,
    alpha = ((1:length(col))/ length(col))^(alpha.power,
    breaks, sprites = TRUE, jitter = FALSE,
    radius = min(diff(x), diff(y), diff(z)),
    add = FALSE,...)
```

### Arguments

- **v**: three dimensional data array.
- **x, y, z**: locations of grid planes at which values in `v` are measured.
- **vlim**: minimum and maximum `v` values for which points are to be drawn.
- **col**: vector of colors for the points as generated by `heat.colors` or similar functions.
- **alpha.power**: used to calculate the alpha values. The larger the power, the smaller the alpha, the more transparent the point. Only used if `alpha` is not supplied.
- **alpha**: vector of alpha values between 0 and 1. The length of the vector should be equal to the length of `col`.
- **breaks**: breakpoints for the colors; must give one more breakpoint than colors.
- **sprites**: logical; if `TRUE`, use `sprites3d` to draw the points.
- **radius**: radius used in `sprites3d`.
- **jitter**: logical; if `TRUE`, add a small amount of noise to the point locations.
- **add**: logical; if `TRUE`, add to current `rgl` graph.
- **...**: material and texture properties. See `rgl.material` for details.

### References

Daniel Adler, Oleg Nenadic and Walter Zucchini (2003) RGL: A R-library for 3D visualization with OpenGL.
kde3d

See Also

image, sprites3d, points3d, jitter.

Examples

```r
# view density of mixture of tri-variate normals
nmix3 <- function(x, y, z, m, s) {
  0.4 * dnorm(x, m, s) * dnorm(y, m, s) * dnorm(z, m, s) +
  0.3 * dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
  0.3 * dnorm(x, m, s) * dnorm(y, -1.5 * m, s) * dnorm(z, m, s)
}
f <- function(x,y,z) nmix3(x,y,z,.5,.5)
x<-seq(-2,2,len=50)
g <- expand.grid(x = x, y = x, z = x)
v <- array(f(g$x, g$y, g$z), c(length(x), length(x), length(x)))
image3d(v)
image3d(v, jitter = TRUE)
```

kde3d

**Compute a Three Dimension Kernel Density Estimate**

Description

Evaluates a three dimensional kernel density estimate using a Gaussian kernel with diagonal covariance matrix on a regular grid.

Usage

```r
kde3d(x, y, z, h, n = 20, lims = c(range(x), range(y), range(z)))
```

Arguments

- `x, y, z`: x, y, and z coordinates of the data.
- `h`: vector of three bandwidths for the density estimate; recycled if length is less than three; default is based on the normal reference bandwidth (see `bandwidth.nrd`).
- `n`: numbers of grid points to use for each dimension; recycled if length is less than three.
- `lims`: lower and upper limits on the region for which the density estimate is to be computed, provides as a vector of length 6, corresponding to low and high values of x, y, and z; recycled if only two values are supplied.

Value

A list of four components, x, y, z, and d. x, y, and z are the coordinates of the grid points at which the density estimate has been evaluated, and d is a three dimensional array of the estimated density values.
Based on the function kde2d in package MASS.

See Also

kde2d.

Examples

with(quakes, {
  d <- kde3d(long, lat, -depth, n = 40)
  contour3d(d$d, exp(-12), d$x/22, d$y/28, d$z/640,
            color = "green", color2 = "gray", scale=FALSE,
            engine = "standard")
})

lighting

Lighting Functions

Description

Functions to compute colors modified for lighting effects.

Usage

phongLighting(normals, view, light, color, color2, alpha, material = "default")
perspLighting(normals, view, light, color, color2, alpha, material = "default")

Arguments

- **normals**: numeric matrix with three columns representing surface normal vectors.
- **view**: numeric vector of length 3 representing the direction to the viewer.
- **light**: numeric vector of length 3 or 4. The first three elements represent the direction to the light. The fourth element, if present, represents light intensity; the default is 1.
- **color**: colors to use for faces in the direction of the normal vectors.
- **color2**: opposite face color.
- **alpha**: alpha channel level, a number between 0 and 1.
- **material**: material specification. Currently possible values are the character strings "dull", "shiny", "metal", and "default".

Details

phongLighting uses the Phong lighting model to compute colors modified for view direction, light direction, and material properties. perspLighting implements approximately the same lighting model as the persp function.
linesTetrahedra

Value

Vector of color specifications.

Description

Creates a scene consisting of lines made up of small tetrahedra centered at points along them.

Usage

linesTetrahedra(x, y, z, delta=c(min(x[2]-x[1])/10,
                                min(y[2]-y[1])/10,
                                min(z[2]-z[1])/10),
               lwd = 0.01, color = "black", ...)

Arguments

x, y, z  numeric vectors of length two or matrices with two columns representing coordinates of starting and ending points of line(s).
delta  numeric; increase in each dimension used to locate points along the lines; recycled to length 3.
lwd  numeric; used for the size of the tetrahedron in each dimension; recycled to length 3.
color  color to use for the tetrahedra.
...  additional arguments to be passed on to makeTriangles.

Details

The function uses the Bresenham’s line algorithm to locate points along lines and then creates a triangle mesh scene representing tetrahedra centered at those points.

Value

Returns a triangle mesh scene representing the lines.

See Also

lines3d.
Examples

```r
p <- pointsTetrahedra(x=c(100,100, 257, 257),
                      y=c(100,100, 257, 257),
                      z=c(100,257, 257, 100), size=1)

l <- linesTetrahedra(x=matrix(c(100,257,
                               100,257), nrow=2, byrow=TRUE),
                      y=matrix(c(100,257,
                               100,257), nrow=2, byrow=TRUE),
                      z=matrix(c(100,257,
                               257,100), nrow=2, byrow=TRUE),
                      lwd=0.4,
                      col="red")

drawScene.rgl(list(p, l))
```

---

**parametric3d**

*Draw a 3D Parametric Plot*

**Description**

Plot a two-parameter surface in three dimensions.

**Usage**

```r
parametric3d(fx, fy, fz, u, v, umin, umax, vmin, vmax, n = 100,
             color = "white", color2 = NA, alpha = 1,
             fill = TRUE, col.mesh = if (fill) NA else color,
             smooth = 0, material = "default",
             add = FALSE, draw = TRUE, engine = "rgl", ...)
```

**Arguments**

- `fx, fy, fz`: vectorized functions of `u` and `v` to compute the `x`, `y`, and `z` coordinates.
- `u`: numeric vector of `u` values.
- `v`: numeric vector of `v` values.
- `umin`: numeric; the minimum value of `u`. Ignored if `u` is supplied.
- `umax`: numeric; the maximum value of `u`. Ignored if `u` is supplied.
- `vmin`: numeric; the minimum value of `v`. Ignored if `v` is supplied.
- `vmax`: numeric; the maximum value of `v`. Ignored if `v` is supplied.
- `n`: the number of equally spaced `u` and `v` values to use. Ignored if `u` and `v` are supplied.
- `color`: color to use for the surface. Can also be a function of three arguments. This is called with three arguments, the coordinates of the midpoints of the triangles making up the surface. The function should return a vector of colors to use for the triangles.
- `color2`: opposite face color.
alpha
  alpha channel level, a number between 0 and 1.
fill
  logical; if TRUE, drawing should use filled surfaces; otherwise a wire frame should be drawn.
col.mesh
  color to use for the wire frame.
smooth
  integer or logical specifying Phong shading level for "standard" and "grid" engines or whether or not to use shading for the "rgl" engine.
material
  material specification; currently only used by "standard" and "grid" engines. Currently possible values are the character strings "dull", "shiny", "metal", and "default".
add
  logical; if TRUE, add to current graph.
draw
  logical; if draw is TRUE, draw the results; otherwise, return triangle mesh structure.
engine
  character; currently "rgl", "standard", "grid" or "none"; for "none" the computed triangles are returned.
...  
  additional rendering arguments, e.g. material and texture properties for the "rgl" engine. See documentation for drawScene and drawScene.rgl

Details

Analogous to Mathematica's Param3D. Evaluates the functions fx, fy, and fz specifying the coordinates of the surface at a grid of values for the parameters u and v.

Value

For the "rgl" engine the returned value is NULL. For the "standard" and "grid" engines the returned value is the viewing transformation as returned by persp. For the engine "none", or when draw is not true, the returned value is a structure representing the triangles making up the surface.

Note

The "rgl" engine now uses the standard rgl coordinates instead of negating y and swapping y and z. If you need to reproduce the previous behavior you can use options(old.misc3d.orientation=TRUE).

Transparency only works properly in the "rgl" engine. For standard or grid graphics on pdf or quartz devices using alpha levels less than 1 does work but the triangle borders show as a less transparent mesh.

References

Daniel Adler, Oleg Nenadic and Walter Zucchini (2003) RGL: A R-library for 3D visualization with OpenGL

See Also

surface3d, material3d, scatterplot3d.
Examples

#Example 1: Ratio-of-Uniform sampling region of bivariate normal
parametric3d(fx = function(u, v) u * exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   fy = function(u, v) v * exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   fz = function(u, v) exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   umin = -20, umax = 20, vmin = -20, vmax = 20, n = 100)
parametric3d(fx = function(u, v) u * exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   fy = function(u, v) v * exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   fz = function(u, v) exp(-0.5 * (u^2 + v^2 - 2 * 0.75 * u * v)/sqrt(1-.75^2))^(1/3),
   u = qcauchy((1:100)/101), v = qcauchy((1:100)/101))
parametric3d(fx = function(u,v) u*(dt(u,2) * dt(v,2))^(1/3),
   fy = function(u,v) v*(dt(u,2) * dt(v,2))^(1/3),
   fz = function(u,v) (dt(u,2) * dt(v,2))^(1/3),
   umin = -20, umax = 20, vmin = -20, vmax = 20, n = 100, color = "green")
parametric3d(fx = function(u,v) u*(dt(u,2) * dt(v,2))^(1/3),
   fy = function(u,v) v*(dt(u,2) * dt(v,2))^(1/3),
   fz = function(u,v) (dt(u,2) * dt(v,2))^(1/3),
   u = qcauchy((1:100)/101), v = qcauchy((1:100)/101),
   color = "green")
parametric3d(fx = function(u,v) u*(dt(u,2) * dt(v,2))^(1/3),
   fy = function(u,v) v*(dt(u,2) * dt(v,2))^(1/3),
   fz = function(u,v) (dt(u,2) * dt(v,2))^(1/3),
   u = qcauchy((1:100)/101), v = qcauchy((1:100)/101),
   color = "green", engine = "standard", scale = FALSE)

#Example 2: Ratio-of-Uniform sampling region of Bivariate t
parametric3d(fx = function(u,v) u*(dt(u,2) * dt(v,2))^(1/3),
   fy = function(u,v) v*(dt(u,2) * dt(v,2))^(1/3),
   fz = function(u,v) (dt(u,2) * dt(v,2))^(1/3),
   u = qcauchy((1:100)/101), v = qcauchy((1:100)/101),
   color = "green")
parametric3d(fx = function(u,v) u*(dt(u,2) * dt(v,2))^(1/3),
   fy = function(u,v) v*(dt(u,2) * dt(v,2))^(1/3),
   fz = function(u,v) (dt(u,2) * dt(v,2))^(1/3),
   u = qcauchy((1:100)/101), v = qcauchy((1:100)/101),
   color = "green", engine = "standard", scale = FALSE)

#Example 3: Surface of revolution
parametric3d(fx = function(u,v) u,
   fy = function(u,v) sin(v)*(u^3+2*x^2-2*u+2)/5,
   fz = function(u,v) cos(v)*(u^3+2*x^2-2*u+2)/5,
   umin = -2.3, umax = 1.3, vmin = 0, vmax = 2*pi)
parametric3d(fx = function(u,v) u,
   fy = function(u,v) sin(v)*(u^3+2*x^2-2*u+2)/5,
   fz = function(u,v) cos(v)*(u^3+2*x^2-2*u+2)/5,
pointsTetrahedra

umin = -2.3, umax = 1.3, vmin = 0, vmax = 2*pi,
engine = "standard", scale = FALSE,
color = "red", color2 = "blue", material = "shiny")

pointsTetrahedra

Create a Set of Tetrahedra Centered at Data Points

Description

Creates a scene consisting of small tetrahedra centered at specified data points in three dimensions.

Usage

pointsTetrahedra(x, y, z, size = 0.01, color = "black", ...)

Arguments

x, y, z numeric vectors representing point coordinates.
size numeric; multiple of data range to use for the size of the tetrahedron in each
dimension; recycled to length 3.
color color to use for the tetrahedra.
... additional arguments to be passed on to makeTriangles.

Details

This function is useful, for example, for incorporating raw data along with a density estimate surface in a scene rendered using standard or grid graphics. For rgl rendering points3d is an alternative.

Value

Returns a triangle mesh scene representing the tetrahedra.

See Also

points3d.

Examples

with(quakes, {
  d <- kde3d(long, lat, -depth, n = 40)
  v <- contour3d(d$d, exp(-12), d$x/22, d$y/28, d$z/640,
                 color="green", color2="gray", draw=FALSE)
  p <- pointsTetrahedra(long/22, lat/28, -depth/640,
                        size = 0.005)
  drawScene(list(v, p))
})
slices3dInteractive Image Slices of 3D or 4D Volume Data

Description

Uses `tkrplot` to create an interactive slice view of three or four dimensional volume data.

Usage

```r
slices3d(vol1, vol2=NULL, rlim1, rlim2, col1, col2, main,
         scale = 0.8, alpha=1, cross = TRUE,
         layout=c("counterclockwise", "clockwise"))
```

Arguments

- `vol1`: a three or four dimensional real array. If two images are overlaid, then this is the one at bottom.
- `vol2`: a three or four dimensional real array. If two images are overlaid, then this is the one on top. The default value is `NULL`, when only `vol1` is drawn.
- `rlim1`: the minimum and maximum `vol1` values for which colors should be plotted, defaulting to the range of the values of `vol1`.
- `rlim2`: the minimum and maximum `vol2` values for which colors should be plotted, defaulting to the range of the values of `vol2`, if two images are overlaid.
- `col1`: a list of colors for `vol1`.
- `col2`: a list of colors for `vol2`.
- `main`: a character vector; main title for the plot.
- `scale`: real value for scaling embedded plot size.
- `alpha`: real value for transparency level, if two images are overlaid. The default value is 1.
- `cross`: logical; if TRUE, show cross hairs of current slices.
- `layout`: a character string specifying the layout. It must be either "counterclockwise" or "clockwise", and may be abbreviated. The default is "counterclockwise". Images corresponding to the x-y planes are always displayed in the third quadrant. If layout is counterclockwise, then the first quadrant shows images from the y-z planes and the second quadrant the x-z planes. Otherwise, the images in the first and the second quadrant are switched. The fourth quadrant is left for the slider used to select the value of the fourth index (if any) of input array(s).

Details

Shows slices of 3D array along the axes as produced by `image`, along with sliders for controlling which slices are shown. For 4D data an additional slider selects the value of the fourth index. Two images can be overlaid. This is useful for viewing medical imaging data (e.g. PET scans and fMRI data).
Examples

# Example 1: View of a mixture of three tri-variate normal densities
nmix3 <- function(x, y, z, m, s) {
    0.4 * dnorm(x, m, s) * dnorm(y, m, s) * dnorm(z, m, s) +
    0.3 * dnorm(x, -m, s) * dnorm(y, -m, s) * dnorm(z, -m, s) +
    0.3 * dnorm(x, m, s) * dnorm(y, 1.5 * m, s) * dnorm(z, m, s)
}
x <- seq(-2, 2, len=40)
g <- expand.grid(x = x, y = x, z = x)
v <- array(nmix3(g$x, g$y, g$z, .5, .5), c(40, 40, 40))
slices3d(vol1=v, main="View of a mixture of three tri-variate normals", col1=heat.colors(256))

## Not run:
# Example 2: Put a z-map from fMRI data on top of a structure
# image. The threshold value of the z-map is 2.
library(AnalyzeFMRI)
temp <- f.read.analyze.volume("standard.img")
z <- f.read.analyze.volume("z-map.img")
slices3d(vol1=temp, vol2=z[, , 1], rlim2=c(2, Inf), col2=heat.colors(20),
         main="Regions above threshold values.")

## End(Not run)

---

surfaceTriangles

Create a Triangle Mesh Representing a Surface

Description

Creates a triangle mesh object representing a surface over a rectangular grid.

Usage

surfaceTriangles(x, y, f, color = "red", color2 = NA, alpha = 1,
                 fill = TRUE, col.mesh = if (fill) NA else color,
                 smooth = 0, material = "default")

Arguments

x, y: numeric vectors.

f: numeric matrix of dimension length(x) by length(y) or vectorized function
   of two arguments.

color: color to use for the surface. Can also be a function of three arguments. This
       is called with three arguments, the coordinates of the midpoints of the triangles
       making up the surface. The function should return a vector of colors to use for
       the triangles.

color2: opposite face color.

alpha: alpha channel level, a number between 0 and 1.
fill
logical; if TRUE, drawing should use filled surfaces; otherwise a wire frame should be drawn.

col.mesh
color to use for the wire frame.

smooth
integer or logical specifying Phong shading level for "standard" and "grid" engines or whether or not to use shading for the "rgl" engine.

material
material specification; currently only used by "standard" and "grid" engines. Currently possible values are the character strings "dull", "shiny", "metal", and "default".

Value
Returns a triangle mesh object representing the surface.

See Also
persp, rgl.surface, surface3d.

Examples

```r
drawScene(surfaceTriangles(seq(-1,1,len=30), seq(-1,1,len=30),
                           function(x, y) (x^2 + y^2), color2 = "green"))
drawScene.rgl(surfaceTriangles(seq(-1,1,len=30), seq(-1,1,len=30),
                               function(x, y) (x^2 + y^2), color2 = "green"))
```

---

teapot

Description
The Utah teapot is a classic computer graphics example. This data set contains a representation in terms of triangles.

Usage
data(teapot)

Format
A list with components vertices and edges. vertices is a 3 by 1976 numeric matrix of the coordinates of the vertices. edges is a 3 by 3751 integer matrix of the indices of the triangles.

Source
Converted from the netCDF file that was at one time made available by Dave Forrest at http://www.maplepark.com/~drf5n/extras/teapot.nc.
**triangles**

*Triangle Mesh Functions*

**Description**

Functions to create and modify triangle mesh objects representing 3D surfaces.

**Usage**

```r
makeTriangles(v1, v2, v3, color = "red", color2 = NA, alpha = 1,
    fill = TRUE, col.mesh = if (fill) NA else color,
    smooth = 0, material = "default")
updateTriangles(triangles, color, color2, alpha, fill, col.mesh,
    material, smooth)
translateTriangles(triangles, x = 0, y = 0, z = 0)
scaleTriangles(triangles, x = 1, y = x, z = x)
transformTriangles(triangles, R)
```

**Arguments**

- `v1, v2, v3` specification of triangle coordinates. If all three are provided then they should be matrices with three columns representing coordinates of the first, second, and third vertices of the triangles. If only `v1` and `v2` are provided then `v1` should be a numeric matrix with three rows specifying coordinates of vertices, and `v2` should be an integer matrix with three rows specifying the indexes of the vertices in the triangles. If only `v1` is provided then it should be a matrix with three columns and number of rows divisible by three specifying the vertices of the triangles in groups of three.

- `triangles` triangle mesh object.

- `x, y, z` numeric of length one. Amounts by which to translate or scale corresponding coordinates.

- `color` color to use for the surface. Can also be a function of three arguments. This is called with three arguments, the coordinates of the midpoints of the triangles making up the surface. The function should return a vector of colors to use for the triangles.

- `color2` opposite face color.

- `alpha` alpha channel level, a number between 0 and 1.

- `fill` logical; if TRUE, drawing should use filled surfaces; otherwise a wire frame should be drawn.

- `col.mesh` color to use for the wire frame.

- `smooth` integer or logical specifying Phong shading level for "standard" and "grid" engines or whether or not to use shading for the "rgl" engine.

- `material` material specification; currently only used by "standard" and "grid" engines. Currently possible values are the character strings "dull", "shiny", "metal", and "default".

- `R` 4 by 4 homogeneous coordinate transformation matrix to apply.
Details

makeTriangles creates a triangle mesh object. updateTriangles modifies fields of such an object. Both may perform some consistency checks.

translateTriangles and scaleTriangles translate or scale the vertex locations of triangle mesh objects by specified amounts.

transformTriangles applies a transformation specified by a 4 by 4 homogeneous transformation matrix.

Value

A triangle mesh object of class Triangles3D.
Index

* datasets
teapot, 20
* dplot
  kde3d, 11
  kde2d, 12
  kde3d, 11
  kde2d, 12
* hplot
  computeContour3d, 2
  contour3d, 3
  drawScene, 6
  exportScene, 9
  image3d, 10
  lighting, 12
  linesTetrahedra, 13
  parametric3d, 14
  pointsTetrahedra, 17
  slices3d, 18
  surfaceTriangles, 19
  triangles, 21
  bandwidth.nrd, 11
cloud, 7
computeContour3d, 2
contour3d, 2, 3
drawScene, 4, 6, 15
drawScene.rgl, 4, 15
exportScene, 9
image, 11
image3d, 10
jitter, 11
kde2d, 12
kde3d, 11
lighting, 12
lines3d, 13
linesTetrahedra, 13
makeTriangles (triangles), 21
material3d, 4, 15
panel.3dwire, 7
parametric3d, 14
persp, 20
perspLighting (lighting), 12
phongLighting (lighting), 12
points3d, 11, 17
pointsTetrahedra, 17
rgl.material, 7, 8
rgl.surface, 20
scaleTriangles (triangles), 21
scatterplot3d, 15
slices3d, 18
sprites3d, 11
surface3d, 4, 15, 20
surfaceTriangles, 19
teapot, 20
transformTriangles (triangles), 21
translateTriangles (triangles), 21
triangles, 21
triangles3d, 4
updateTriangles (triangles), 21