

# Package ‘rgl’

May 8, 2012

**Version** 0.92.880

**Title** 3D visualization device system (OpenGL)

**Author** Daniel Adler <dadler@uni-goettingen.de>, Duncan Murdoch <murdoch@stats.uwo.ca>

**Maintainer** Duncan Murdoch <murdoch@stats.uwo.ca>

**Depends** R (>= 2.7.0),stats,grDevices

**Suggests** MASS

**Description** 3D visualization device (OpenGL)

**License** GPL

**URL** <http://rgl.neoscientists.org>

**SystemRequirements** OpenGL, GLU Library, zlib (optional), libpng (>=1.2.9, optional), FreeType (optional)

**BugReports** <https://r-forge.r-project.org/projects/rgl/>

**Repository** CRAN

**Date/Publication** 2012-05-08 14:22:16

## R topics documented:

rgl-package	2
abclines	4
addNormals	5
aspect3d	6
attributes	7
axes3d	8
bg	10
cylinder3d	11
ellipse3d	12
grid3d	14

light . . . . .	15
matrices . . . . .	16
mesh3d . . . . .	18
par3d . . . . .	20
par3dinterp . . . . .	24
persp3d . . . . .	25
planes . . . . .	28
play3d . . . . .	29
plot3d . . . . .	31
points3d . . . . .	33
r3d . . . . .	34
rgl.bbox . . . . .	36
rgl.bringtotop . . . . .	38
rgl.material . . . . .	38
rgl.pixels . . . . .	41
rgl.postscript . . . . .	42
rgl.primitive . . . . .	44
rgl.setMouseCallbacks . . . . .	45
rgl.snapshot . . . . .	46
rgl.surface . . . . .	47
rgl.Sweave . . . . .	49
rgl.user2window . . . . .	51
scene . . . . .	52
select3d . . . . .	54
shapelist3d . . . . .	55
spheres . . . . .	56
spin3d . . . . .	57
sprites . . . . .	58
subdivision3d . . . . .	59
surface3d . . . . .	60
texts . . . . .	61
viewpoint . . . . .	63
writeWebGL . . . . .	65
<b>Index</b>	<b>68</b>

---

 rgl-package

*3D visualization device system*


---

## Description

3D real-time rendering system.

## Usage

```
# Low level rgl.* interface
rgl.open()      # open new device
rgl.close()     # close current device
rgl.cur()       # returns active device ID
rgl.set(which, silent=FALSE) # set device as active
rgl.quit()      # shutdown rgl device system
rgl.init(initValue=0) # re-initialize rgl
```

## Arguments

which	device ID
silent	whether to suppress update of window titles
initValue	value for internal use only

## Details

RGL is a 3D real-time rendering device driver system for R. Multiple devices are managed at a time, where one has the current focus that receives instructions from the R command-line. The device design is oriented towards the R device metaphor. If you send scene management instructions, and there's no device open, it will be opened automatically. Opened devices automatically get the current device focus. The focus may be changed by using `rgl.set()`. `rgl.quit()` shuts down the rgl subsystem and all open devices, detaches the package including the shared library and additional system libraries.

The `rgl.open()` function attempts to open a new RGL window. If the `"rgl.antialias"` option is set, it will be used to select the requested antialiasing. (See [open3d](#) for more description of antialiasing and an alternative way to set the value.)

If `rgl.open()` fails (e.g. because X windows is not running, or its `DISPLAY` variable is not set properly), then you can retry the initialization by calling `rgl.init()`. Do not do this when windows have already been successfully opened: they will be orphaned, with no way to remove them other than closing R. In fact, it's probably a good idea not to do this at all: quitting R and restarting it is a better solution.

This package also includes a higher level interface which is described in the [r3d](#) help topic. That interface is designed to act more like classic 2D R graphics. We recommend that you avoid mixing `rgl.*` and `*3d` calls.

See the first example below to display the `ChangeLog`.

## See Also

[r3d](#), [rgl.clear](#), [rgl.pop](#), [rgl.viewpoint](#), [rgl.light](#), [rgl.bg](#), [rgl.bbox](#), [rgl.points](#), [rgl.lines](#), [rgl.triangles](#), [rgl.quads](#), [rgl.texts](#), [rgl.surface](#), [rgl.spheres](#), [rgl.sprites](#), [rgl.snapshot](#)

## Examples

```
file.show(system.file("NEWS", package="rgl"))
example(surface3d)
example(plot3d)
```

---

`abclines`*Lines intersecting the bounding box*

---

### Description

This adds mathematical lines to a scene. Their intersection with the current bounding box will be drawn.

### Usage

```
rgl.abclines(x, y = NULL, z = NULL, a, b = NULL, c = NULL, ...)  
abclines3d(x, y = NULL, z = NULL, a, b = NULL, c = NULL, ...)
```

### Arguments

<code>x, y, z</code>	Coordinates of points through which each line passes.
<code>a, b, c</code>	Coordinates of the direction vectors for the lines.
<code>...</code>	Material properties.

### Details

These functions draw the segment of a line that intersects the current bounding box of the scene using the parametrization  $(x, y, z) + (a, b, c) * s$  where  $s$  is a real number.

Any reasonable way of defining the coordinates `x, y, z` and `a, b, c` is acceptable. See the function [xyz.coords](#) for details.

### Value

A shape ID of the object is returned invisibly.

### See Also

[planes3d](#), [rgl.planes](#) for mathematical planes.

[segments3d](#) draws sections of lines that do not adapt to the bounding box.

### Examples

```
plot3d(rnorm(100), rnorm(100), rnorm(100))  
abclines3d(0,0,0, a=diag(3), col="gray")
```

---

addNormals	<i>Add normal vectors to objects so they render more smoothly.</i>
------------	--

---

### Description

This generic function adds normals at each of the vertices of a polyhedron by averaging the normals of each incident face. This has the effect of making the surface of the object appear smooth rather than faceted when rendered.

### Usage

```
addNormals(x, ...)
```

### Arguments

x	An object to which to add normals.
...	Additional parameters which will be passed to the methods. Currently unused.

### Details

Currently methods are supplied for "mesh3d" and "shapelist3d" classes.

### Value

A new object of the same class as x, with normals added.

### Author(s)

Duncan Murdoch

### Examples

```
open3d()
y <- subdivision3d(tetrahedron3d(col="red"), depth=3)
shade3d(y) # No normals
y <- addNormals(y)
shade3d(translate3d(y, x=1, y=0, z=0)) # With normals
```

---

`aspect3d`*Set the aspect ratios of the current plot*

---

**Description**

This function sets the apparent ratios of the x, y, and z axes of the current bounding box.

**Usage**

```
aspect3d(x, y = NULL, z = NULL)
```

**Arguments**

x	The ratio for the x axis, or all three ratios, or "iso"
y	The ratio for the y axis
z	The ratio for the z axis

**Details**

If the ratios are all 1, the bounding box will be displayed as a cube approximately filling the display. Values may be set larger or smaller as desired. Aspect "iso" signifies that the coordinates should all be displayed at the same scale, i.e. the bounding box should not be rescaled. (This corresponds to the default display before `aspect3d` has been called.) Partial matches to "iso" are allowed.

`aspect3d` works by modifying `par3d("scale")`.

**Value**

The previous value of the scale is returned invisibly.

**Author(s)**

Duncan Murdoch

**See Also**

[plot3d](#), [par3d](#)

**Examples**

```
x <- rnorm(100)
y <- rnorm(100)*2
z <- rnorm(100)*3

open3d()
plot3d(x, y, z)
aspect3d(1,1,0.5)
open3d()
plot3d(x, y, z)
aspect3d("iso")
```

---

attributes                      *Get information about shapes*

---

### Description

Retrieves information about the shapes in a scene.

### Usage

```
rgl.attrib.count(id, attrib)
rgl.attrib(id, attrib, first = 1,
           last = rgl.attrib.count(id, attrib))
```

### Arguments

<code>id</code>	A shape identifier, as returned by <code>rgl.ids</code> .
<code>attrib</code>	An attribute of a shape. Currently supported: one of <code>c("vertices", "normals", "colors", "texcoords")</code> or unique prefixes to one of those.
<code>first, last</code>	Specify these to retrieve only those rows of the result.

### Details

If the identifier is not found or is not a shape that has the given attribute, zero will be returned by `rgl.attrib.count`, and an empty matrix will be returned by `rgl.attrib`.

The first four `attrib` names correspond to the usual OpenGL properties; `"dim"` is used just for surfaces, defining the rows and columns in the rectangular grid; `"cex"` and `"adj"` apply only to text objects.

### Value

`rgl.attrib.count` returns the count of the requested attribute. `rgl.attrib` returns the values of the attribute. Attributes are mostly real-valued, with the following sizes:

**vertices** 3 values, x, y, z

**normals** 3 values, x, y, z

**centers** 3 values, x, y, z

**colors** 4 values, r, g, b, a

**texcoords** 2 values, s, t

**dim** 2 values, r, c

**cex** 1 value

**adj** 2 values, x, y

**radii** 1 value

The `"texts"` attribute is character-valued.

These are returned as matrices with the row count equal to the count for the attribute, and the columns as listed above.

**Author(s)**

Duncan Murdoch

**See Also**[rgl.ids](#)**Examples**

```
p <- plot3d(rnorm(100), rnorm(100), rnorm(100), type="s", col="red")
rgl.attrib.count(p["data"], "vertices")
rgl.attrib(p["data"], "vertices", last=10)
```

axes3d

*Draw boxes, axes and other text outside the data***Description**

These functions draw axes, boxes and text outside the range of the data. `axes3d`, `box3d` and `title3d` are the higher level functions; normally the others need not be called directly by users.

**Usage**

```
axes3d(edges = "bbox", labels = TRUE, tick = TRUE, nticks = 5, box=FALSE, ...)
box3d(...)
title3d(main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
        zlab = NULL, line = NA, ...)
axis3d(edge, at = NULL, labels = TRUE, tick = TRUE, line = 0,
        pos = NULL, nticks = 5, ...)
mtext3d(text, edge, line = 0, at = NULL, pos = NA, ...)
```

**Arguments**

<code>edges</code>	a code to describe which edge(s) of the box to use; see Details below
<code>labels</code>	whether to label the axes, or (for <code>axis3d</code> ) the labels to use
<code>tick</code>	whether to use tick marks
<code>nticks</code>	suggested number of ticks
<code>box</code>	draw the full box if "bbox" axes are used
<code>main</code>	the main title for the plot
<code>sub</code>	the subtitle for the plot
<code>xlab, ylab, zlab</code>	the axis labels for the plot
<code>line</code>	the "line" of the plot margin to draw the label on
<code>edge, pos</code>	the position at which to draw the axis or text
<code>text</code>	the text to draw
<code>at</code>	the value of a coordinate at which to draw the axis
<code>...</code>	additional parameters which are passed to <a href="#">bbox3d</a> or <a href="#">material3d</a>

## Details

The rectangular prism holding the 3D plot has 12 edges. They are identified using 3 character strings. The first character ('x', 'y', or 'z') selects the direction of the axis. The next two characters are each '-' or '+', selecting the lower or upper end of one of the other coordinates. If only one or two characters are given, the remaining characters default to '-'. For example `edge = 'x+'` draws an x-axis at the high level of y and the low level of z.

By default, `axes3d` uses the `bbox3d` function to draw the axes. The labels will move so that they do not obscure the data. Alternatively, a vector of arguments as described above may be used, in which case fixed axes are drawn using `axis3d`.

If `pos` is a numeric vector of length 3, `edge` determines the direction of the axis and the tick marks, and the values of the other two coordinates in `pos` determine the position. See the examples.

## Value

These functions are called for their side effects. They return the object IDs of objects added to the scene.

## Author(s)

Duncan Murdoch

## See Also

`axis`, `box`, `title`, `mtext`, `bbox3d`

## Examples

```
open3d()
points3d(rnorm(10),rnorm(10),rnorm(10))

# First add standard axes
axes3d()

# and one in the middle (the NA will be ignored, a number would
# do as well)
axis3d('x',pos=c(NA, 0, 0))

# add titles
title3d('main','sub','xlab','ylab','zlab')

rgl.bringtotop()

open3d()
points3d(rnorm(10),rnorm(10),rnorm(10))

# Use fixed axes
axes3d(c('x','y','z'))

# Put 4 x-axes on the plot
```

```
axes3d(c('x--','x--','x--','x--'))
axis3d('x',pos=c(NA, 0, 0))
title3d('main','sub','xlab','ylab','zlab')
```

---

 bg

*Set up Background*


---

### Description

Setup the background environment of the scene.

### Usage

```
bg3d(...)
rgl.bg( sphere = FALSE, fogtype = "none", color=c("black","white"),
        back="lines", ...)
```

### Arguments

fogtype	fog type: <b>"none"</b> no fog <b>"linear"</b> linear fog function <b>"exp"</b> exponential fog function <b>"exp2"</b> squared exponential fog function
sphere	logical, if true, an environmental sphere geometry is used for the background decoration.
color	Primary color is used for background clearing and as fog color. Secondary color is used for background sphere geometry. See <a href="#">rgl.material</a> for details.
back	Specifies the fill style of the sphere geometry. See <a href="#">rgl.material</a> for details.
...	Material properties. See <a href="#">rgl.material</a> for details.

### Details

If sphere is set to TRUE, an environmental sphere enclosing the whole scene is drawn.

### See Also

[rgl.material](#)

**Examples**

```

rgl.open()

# a simple white background

bg3d("white")

# the holo-globe (inspired by star trek):

rgl.bg(sphere=TRUE, color=c("black","green"), lit=FALSE, back="lines" )

# an environmental sphere with a nice texture.

rgl.bg(sphere=TRUE, texture=system.file("textures/sunsleep.png", package="rgl"),
       back="filled" )

```

---

cylinder3d

*Create cylindrical or "tube" plots.*


---

**Description**

This function converts a description of a space curve into a `"mesh3d"` object forming a cylindrical tube around the curve.

**Usage**

```

cylinder3d(center, radius = 1, twist = 0, e1 = NULL, e2 = NULL, e3 = NULL,
           sides = 8, closed = 0, debug = FALSE, keepVars = FALSE)

```

**Arguments**

center	An n by 3 matrix whose columns are the x, y and z coordinates of the space curve.
radius	The radius of the cross-section of the tube at each point in the center.
twist	The amount by which the polygon forming the tube is twisted at each point.
e1, e2, e3	The Frenet coordinates to use at each point on the space curve.
sides	The number of sides in the polygon cross section.
closed	Whether to treat the first and last points of the space curve as identical, and close the curve. If <code>closed &gt; 0</code> , it represents the number of points of overlap in the coordinates.
debug	If TRUE, plot the local Frenet coordinates at each point.
keepVars	If TRUE, return the local variables in attribute <code>"vars"</code> .

**Details**

The number of points in the space curve is determined by the vector lengths in center, after using `xyz.coords` to convert it to a list. The other arguments `radius`, `twist`, `e1`, `e2`, and `e3` are extended to the same length.

The three optional arguments `e1`, `e2`, and `e3` determine the local coordinate system used to create the vertices at each point in center. If missing, they are computed by simple numerical approximations. `e1` should be the tangent coordinate, giving the direction of the curve at the point. The cross-section of the polygon will be orthogonal to `e1`. `e2` defaults to an approximation to the normal or curvature vector; it is used as the image of the y axis of the polygon cross-section. `e3` defaults to an approximation to the binormal vector, to which the x axis of the polygon maps. The vectors are orthogonalized and normalized at each point.

**Value**

A "mesh3d" object holding the cylinder, possibly with attribute "vars" containing the local environment of the function.

**Author(s)**

Duncan Murdoch

**Examples**

```
# A trefoil knot
open3d()
theta <- seq(0, 2*pi, len=25)
knot <- cylinder3d(cbind(sin(theta)+2*sin(2*theta), 2*sin(3*theta), cos(theta)-2*cos(2*theta)),
                  e1=cbind(cos(theta)+4*cos(2*theta), 6*cos(3*theta), sin(theta)+4*sin(2*theta)),
                  radius=0.8, closed=TRUE)

shade3d(addNormals(subdivision3d(knot, depth=2)), col="green")
```

---

ellipse3d

*Make an ellipsoid*

---

**Description**

A generic function and several methods returning an ellipsoid or other outline of a confidence region for three parameters.

**Usage**

```
ellipse3d(x, ...)
## Default S3 method:
ellipse3d(x, scale = c(1, 1, 1), centre = c(0, 0, 0), level = 0.95,
          t = sqrt(qchisq(level, 3)), which = 1:3, subdivide = 3, smooth = TRUE, ...)
## S3 method for class 'lm'
```

```

ellipse3d(x, which = 1:3, level = 0.95, t = sqrt(3 * qf(level,
                                                    3, x$df.residual)), ...)

## S3 method for class 'glm'
ellipse3d(x, which = 1:3, level = 0.95, t, dispersion, ...)
## S3 method for class 'nls'
ellipse3d(x, which = 1:3, level = 0.95, t = sqrt(3 * qf(level,
                                                    3, s$df[2])), ...)

```

## Arguments

x	An object. In the default method the parameter x should be a square positive definite matrix at least 3x3 in size. It will be treated as the correlation or covariance of a multivariate normal distribution.
...	Additional parameters to pass to the default method or to <a href="#">qmesh3d</a> .
scale	If x is a correlation matrix, then the standard deviations of each parameter can be given in the scale parameter. This defaults to <code>c(1, 1, 1)</code> , so no rescaling will be done.
centre	The centre of the ellipse will be at this position.
level	The confidence level of a simultaneous confidence region. The default is 0.95, for a 95% region. This is used to control the size of the ellipsoid.
t	The size of the ellipse may also be controlled by specifying the value of a t-statistic on its boundary. This defaults to the appropriate value for the confidence region.
which	This parameter selects which variables from the object will be plotted. The default is the first 3.
subdivide	This controls the number of subdivisions (see <a href="#">subdivision3d</a> ) used in constructing the ellipsoid. Higher numbers give a smoother shape.
smooth	If TRUE, smooth interpolation of normals is used; if FALSE, a faceted ellipsoid will be displayed.
dispersion	The value of dispersion to use. If specified, it is treated as fixed, and chi-square limits for t are used. If missing, it is taken from <code>summary(x)</code> .

## Value

A [mesh3d](#) object representing the ellipsoid.

## Examples

```

# Plot a random sample and an ellipsoid of concentration corresponding to a 95%
# probability region for a
# trivariate normal distribution with mean 0, unit variances and
# correlation 0.8.
if (require(MASS)) {
  Sigma <- matrix(c(10,3,0,3,2,0,0,0,1), 3,3)
  Mean <- 1:3
  x <- mvrnorm(1000, Mean, Sigma)
}

```

```

open3d()

plot3d(x, box=FALSE)

plot3d( ellipse3d(Sigma, centre=Mean), col="green", alpha=0.5, add = TRUE)
}

# Plot the estimate and joint 90% confidence region for the displacement and cylinder
# count linear coefficients in the mtcars dataset

data(mtcars)
fit <- lm(mpg ~ disp + cyl , mtcars)

open3d()
plot3d(ellipse3d(fit, level = 0.90), col="blue", alpha=0.5, aspect=TRUE)

```

---

grid3d

*Add a grid to a 3D plot*


---

## Description

This function adds a reference grid to an RGL plot.

## Usage

```
grid3d(side, at = NULL, col = "gray", lwd = 1, lty = 1, n = 5)
```

## Arguments

side	Where to put the grid; see the Details section.
at	How to draw the grid; see the Details section.
col	The color of the grid lines.
lwd	The line width of the grid lines.
lty	The line type of the grid lines.
n	Suggested number of grid lines; see the Details section.

## Details

This function is similar to [grid](#) in classic graphics, except that it draws a 3D grid in the plot.

The grid is drawn in a plane perpendicular to the coordinate axes. The first letter of the `side` argument specifies the direction of the plane: "x", "y" or "z" (or uppercase versions) to specify the coordinate which is constant on the plane.

If `at = NULL` (the default), the grid is drawn at the limit of the box around the data. If the second letter of the `side` argument is "-" or is not present, it is the lower limit; if "+" then at the upper limit. The grid lines are drawn at values chosen by [pretty](#) with `n` suggested locations. The default locations should match those chosen by [axis3d](#) with `nticks = n`.

If `at` is a numeric vector, the grid lines are drawn at those values.

If `at` is a list, then the "x" component is used to specify the x location, the "y" component specifies the y location, and the "z" component specifies the z location. Missing components are handled using the default as for `at = NULL`.

Multiple grids may be drawn by specifying multiple values for `side` or for the component of `at` that specifies the grid location.

### Value

A vector or matrix of object ids is returned invisibly.

### Note

If the scene is resized, the grid will not be resized; use [abclines3d](#) to draw grid lines that will automatically resize.

### Author(s)

Ben Bolker and Duncan Murdoch

### See Also

[axis3d](#)

### Examples

```
x <- 1:10
y <- 1:10
z <- matrix(outer(x-5,y-5) + rnorm(100), 10, 10)
open3d()
persp3d(x, y, z, col="red", alpha=0.7, aspect=c(1,1,0.5))
grid3d(c("x", "y+", "z"))
```

---

light

*add light source*

---

### Description

add a light source to the scene.

### Usage

```
light3d(theta = 0, phi = 15, ...)
rgl.light(theta = 0, phi = 0, viewpoint.rel = TRUE, ambient = "#FFFFFF",
          diffuse = "#FFFFFF", specular = "#FFFFFF")
```



**Arguments**

<code>x, y, z, angle, matrix</code>	See details
<code>obj</code>	An object to be transformed
<code>...</code>	Additional parameters to be passed to methods

**Details**

OpenGL uses homogeneous coordinates to handle perspective and affine transformations. The homogeneous point  $(x, y, z, w)$  corresponds to the Euclidean point  $(x/w, y/w, z/w)$ . The matrices produced by the functions `scaleMatrix`, `translationMatrix`, and `rotationMatrix` are to be left-multiplied by a row vector of homogeneous coordinates; alternatively, the transpose of the result can be right-multiplied by a column vector. The generic functions `scale3d`, `translate3d` and `rotate3d` apply these transformations to the `obj` argument. The `transform3d` function is a synonym for `rotate3d(obj, matrix=matrix)`.

By default, it is assumed that `obj` is a row vector (or a matrix of row vectors) which will be multiplied on the right by the corresponding matrix, but users may write methods for these generics which operate differently. Methods are supplied for `mesh3d` objects.

To compose transformations, use matrix multiplication. The effect is to apply the matrix on the left first, followed by the one on the right.

`identityMatrix` returns an identity matrix.

`scaleMatrix` scales each coordinate by the given factor. In Euclidean coordinates,  $(u, v, w)$  is transformed to  $(x*u, y*v, z*w)$ .

`translationMatrix` translates each coordinate by the given translation, i.e.  $(u, v, w)$  is transformed to  $(u+x, v+y, w+z)$ .

`rotationMatrix` can be called in three ways. With arguments `angle, x, y, z` it represents a rotation of `angle` radians about the axis `x, y, z`. If `matrix` is a 3x3 rotation matrix, it will be converted into the corresponding matrix in 4x4 homogeneous coordinates. Finally, if a 4x4 matrix is given, it will be returned unchanged. (The latter behaviour is used to allow `transform3d` to act like a generic function, even though it is not.)

Use `asHomogeneous(x)` to convert the Euclidean vector `x` to homogeneous coordinates, and `asEuclidean(x)` for the reverse transformation.

**Value**

`identityMatrix`, `scaleMatrix`, `translationMatrix`, and `rotationMatrix` produce a 4x4 matrix representing the requested transformation in homogeneous coordinates.

`scale3d`, `translate3d` and `rotate3d` transform the object and produce a new object of the same class.

**Author(s)**

Duncan Murdoch

**See Also**

[par3d](#) for a description of how rgl uses matrices in rendering.

**Examples**

```
# A 90 degree rotation about the x axis:

rotationMatrix(pi/2, 1, 0, 0)

# Find what happens when you rotate (2,0,0) by 45 degrees about the y axis:

x <- asHomogeneous(c(2,0,0))
y <- x
asEuclidean(y)

# or more simply...

rotate3d(c(2,0,0), pi/4, 0, 1, 0)
```

---

 mesh3d

*3D Mesh objects*


---

**Description**

3D triangle and quadrangle mesh object creation and a collection of sample objects.

**Usage**

```
qmesh3d(vertices, indices, homogeneous = TRUE, material = NULL,
         normals = NULL, texcoords = NULL)
tmesh3d(vertices, indices, homogeneous = TRUE, material = NULL,
         normals = NULL, texcoords = NULL)

cube3d(trans = identityMatrix(), ...)
tetrahedron3d(trans = identityMatrix(), ...)
octahedron3d(trans = identityMatrix(), ...)
icosahedron3d(trans = identityMatrix(), ...)
dodecahedron3d(trans = identityMatrix(), ...)
cuboctahedron3d(trans = identityMatrix(), ...)

oh3d(trans = identityMatrix(), ...) # an 'o' object

dot3d(x, ...) # draw dots at the vertices of an object
## S3 method for class 'mesh3d'
dot3d(x, override = TRUE, ...)
wire3d(x, ...) # draw a wireframe object
## S3 method for class 'mesh3d'
```

```
wire3d(x, override = TRUE, ...)
  shade3d(x, ...) # draw a shaded object
  ## S3 method for class 'mesh3d'
  shade3d(x, override = TRUE, ...)
```

### Arguments

x	a mesh3d object (class qmesh3d or tmesh3d)
vertices	3- or 4-component vector of coordinates
indices	4-component vector of vertex indices
homogeneous	logical indicating if homogeneous (four component) coordinates are used.
material	material properties for later rendering
normals	normals at each vertex
texcoords	texture coordinates at each vertex
trans	transformation to apply to objects; see below for defaults
...	additional rendering parameters
override	should the parameters specified here override those stored in the object?

### Details

These functions create and work with mesh3d objects, which consist of a matrix of vertex coordinates together with a matrix of indices indicating which vertex is part of which face. Such objects may have triangular faces, planar quadrilateral faces, or both.

The sample objects optionally take a matrix transformation `trans` as an argument. This transformation is applied to all vertices of the default shape. The default is an identity transformation.

The "shape3d" class is a general class for shapes that can be plotted by `dot3d`, `wire3d` or `shade3d`.

The "mesh3d" class is a class of objects that form meshes: the vertices are in member `vb`, as a 3 or 4 by `n` matrix. Meshes with triangular faces will contain `it`, a  $3 * n$  matrix giving the indices of the vertices in each face. Quad meshes will have vertex indices in `ib`, a  $4 * n$  matrix.

### Value

`qmesh3d`, `cube3d`, `oh3d`, `tmesh3d`, `tetrahedron3d`, `octahedron3d`, `icosahedron3d` and `dodecahedron3d` return objects of class `c("mesh3d", "shape3d")`. The first three of these are quad meshes, the rest are triangle meshes.

`dot3d`, `wire3d`, and `shade3d` are called for their side effect of drawing an object into the scene; they return an object ID (or vector of IDs, for some classes) invisibly.

See [`rgl.primitive`](#) for a discussion of texture coordinates.

### See Also

[`r3d`](#), [`par3d`](#), [`shapelist3d`](#) for multiple shapes

**Examples**

```

# generate a quad mesh object

vertices <- c(
  -1.0, -1.0, 0, 1.0,
  1.0, -1.0, 0, 1.0,
  1.0, 1.0, 0, 1.0,
  -1.0, 1.0, 0, 1.0
)
indices <- c( 1, 2, 3, 4 )

open3d()
wire3d( qmesh3d(vertices,indices) )

# render 4 meshes vertically in the current view

open3d()
bg3d("gray")
l0 <- oh3d(tran = par3d("userMatrix"), color = "green" )
shade3d( translate3d( l0, -6, 0, 0 ) )
l1 <- subdivision3d( l0 )
shade3d( translate3d( l1, -2, 0, 0 ), color="red", override = FALSE )
l2 <- subdivision3d( l1 )
shade3d( translate3d( l2, 2, 0, 0 ), color="red", override = TRUE )
l3 <- subdivision3d( l2 )
shade3d( translate3d( l3, 6, 0, 0 ), color="red" )

# render all of the Platonic solids
open3d()
shade3d( translate3d( tetrahedron3d(col="red"), 0, 0, 0 ) )
shade3d( translate3d( cube3d(col="green"), 3, 0, 0 ) )
shade3d( translate3d( octahedron3d(col="blue"), 6, 0, 0 ) )
shade3d( translate3d( dodecahedron3d(col="cyan"), 9, 0, 0 ) )
shade3d( translate3d( icosahedron3d(col="magenta"), 12, 0, 0 ) )

```

---

par3d

*Set or Query RGL Parameters*


---

**Description**

par3d can be used to set or query graphical parameters in rgl. Parameters can be set by specifying them as arguments to par3d in tag = value form, or by passing them as a list of tagged values.

**Usage**

```
par3d(..., no.readonly = FALSE)
```

```
open3d(..., params=get("r3dDefaults", envir=.GlobalEnv))
```

## Arguments

...	arguments in tag = value form, or a list of tagged values. The tags must come from the graphical parameters described below.
no.readonly	logical; if TRUE and there are no other arguments, only those parameters which can be set by a subsequent par3d() call are returned.
params	a list of graphical parameters

## Details

Parameters are queried by giving one or more character vectors to par3d.

par3d() (no arguments) or par3d(no.readonly=TRUE) is used to get *all* the graphical parameters (as a named list).

**R.O.** indicates *read-only arguments*: These may only be used in queries, i.e., they do *not* set anything.

open3d opens a new rgl device, and sets the parameters as requested. The r3dDefaults list will be used as default values for parameters. As installed this sets the point of view to 'world coordinates' (i.e. x running from left to right, y from front to back, z from bottom to top), the mouseMode to (zAxis, zoom, fov), and the field of view to 30 degrees. Users may create their own variable of that name in the global environment and it will override the installed one. If there is a bg element in the list or the arguments, it should be a list of arguments to pass to the [bg3d](#) function to set the background.

The arguments to open3d may include material, a list of material properties as in [r3dDefaults](#), but note that high level functions such as [plot3d](#) normally use the r3dDefaults values in preference to this setting.

## Value

When parameters are set, their former values are returned in an invisible named list. Such a list can be passed as an argument to par3d to restore the parameter values. Use par3d(no.readonly = TRUE) for the full list of parameters that can be restored.

When just one parameter is queried, its value is returned directly. When two or more parameters are queried, the result is a list of values, with the list names giving the parameters.

Note the inconsistency: setting one parameter returns a list, but querying one parameter returns an object.

## Parameters

antialias **R.O.** in par3d, may be set in open3d. The (requested) number of hardware antialiasing planes to use (with multisample antialiasing). The OpenGL driver may not support the requested number, in which case par3d("antialias") will report what was actually set.

cex real. The default size for text.

family character. The default device independent family name; see [text3d](#).

font integer. The default font number (from 1 to 5; see [text3d](#)).

useFreeType logical. Should FreeType fonts be used?

fontname **R.O.**; the system-dependent name of the current font.

FOV real. The field of view, from 1 to 179 degrees. This controls the degree of parallax in the perspective view. Isometric perspective (which would correspond to FOV=0) is not currently possible, but one can approximate it by specifying `par3d(FOV=1)`.

ignoreExtent logical. Set to TRUE so that subsequently plotted objects will be ignored in calculating the bounding box of the scene.

modelMatrix ***R.O.***; a 4 by 4 matrix describing the position of the user data.

mouseMode character. A vector of 3 strings describing what the 3 mouse buttons do. Partial matching is used. Possible values for mouseMode are

"none" No action for this button.

"trackball" Mouse acts as a virtual trackball, rotating the scene.

"xAxis" Similar to "trackball", but restricted to X axis rotation.

"yAxis" Y axis rotation.

"zAxis" Z axis rotation.

"polar" Mouse rotates the scene by moving in polar coordinates.

"selecting" Mouse is used for selection. This is not normally set by the user, but is used internally by the `select3d` function.

"zoom" Mouse is used to zoom the display.

"fov" Mouse changes the field of view of the display.

"user" Used when a user handler is set by `rgl.setMouseCallbacks`.

projMatrix ***R.O.***; a 4 by 4 matrix describing the current projection of the scene.

scale real. A vector of 3 values indicating the amount by which to rescale each axis before display. Set by `aspect3d`.

skipRedraw whether to update the display. Set to TRUE to suspend updating while making multiple changes to the scene. See `demo(hist3d)` for an example.

userMatrix a 4 by 4 matrix describing user actions to display the scene.

viewport ***R.O.***; real. A vector giving the dimensions of the window in pixels.

zoom real. A positive value indicating the current magnification of the scene.

bbox ***R.O.***; real. A vector of six values indicating the current values of the bounding box of the scene (xmin, xmax, ymin, ymax, zmin, zmax)

windowRect integer. A vector of four values indicating the left, top, right and bottom of the displayed window (in pixels).

#### note

The "xAxis", "yAxis" and "zAxis" mouse modes rotate relative to the coordinate system of the data, regardless of the current orientation of the scene.

#### Rendering

The parameters returned by `par3d` are sufficient to determine where `rgl` would render a point on the screen. Given a column vector  $(x, y, z)$ , it performs the equivalent of the following operations:

1. It converts the point to homogeneous coordinates by appending  $w=1$ , giving the vector  $v = (x, y, z, 1)$ .

2. It calculates the  $M = \text{par3d}(\text{"modelMatrix"})$  as a product from right to left of the following matrices:
  - A matrix to translate the centre of the bounding box to the origin.
  - A matrix to rescale according to  $\text{par3d}(\text{"scale"})$ .
  - The  $\text{par3d}(\text{"userMatrix"})$  as set by the user.
  - A matrix which may be set by mouse movements.
  - A matrix to translate the origin to the centre of the viewing region.
3. It multiplies the point by  $M$  giving  $u = M \%*\% v$ . Using this location and information on the normals (which have been similarly transformed), it performs lighting calculations.
4. It obtains the projection matrix  $P = \text{par3d}(\text{"projMatrix"})$  and multiplies the point by it giving  $P \%*\% u = (x2, y2, z2, w2)$ .
5. It converts back to Euclidean coordinates by dividing the first 3 coordinates by  $w2$ .
6. The new value  $z2/w2$  represents the depth into the scene of the point. Depending on what has already been plotted, this depth might be obscured, in which case nothing more is plotted.
7. If the point is not culled due to depth, the  $x2$  and  $y2$  values are used to determine the point in the image. The  $\text{par3d}(\text{"viewport"})$  values are used to translate from the range  $(-1, 1)$  to pixel locations, and the point is plotted.
8. If hardware antialiasing is enabled, then the whole process is repeated multiple times (at least conceptually) with different locations in each pixel sampled to determine what is plotted there, and then the images are combined into what is displayed.

See [?matrices](#) for more information on homogeneous and Euclidean coordinates.

## References

OpenGL Architecture Review Board (1997). OpenGL Programming Guide. Addison-Wesley.

## See Also

[rgl.viewpoint](#) to set FOV and zoom.

## Examples

```
r3dDefaults
open3d()
shade3d(cube3d(color=rep(rainbow(6),rep(4,6))))
save <- par3d(userMatrix = rotationMatrix(90*pi/180, 1,0,0))
save
par3d("userMatrix")
par3d(save)
par3d("userMatrix")
```

---

par3dinterp                      *Interpolator for par3d parameters*

---

### Description

Returns a function which interpolates par3d parameter values, suitable for use in animations.

### Usage

```
par3dinterp(times = NULL, userMatrix, scale, zoom, FOV,
            method = c("spline", "linear"),
            extrapolate = c("oscillate", "cycle", "constant", "natural"))
```

### Arguments

times	Times at which values are recorded or a list; see below
userMatrix	Values of par3d("userMatrix")
scale	Values of par3d("scale")
zoom	Values of par3d("zoom")
FOV	Values of par3d("FOV")
method	Method of interpolation
extrapolate	How to extrapolate outside the time range

### Details

This function is intended to be used in constructing animations. It produces a function that returns a list suitable to pass to [par3d](#), to set the viewpoint at a given point in time.

All of the parameters are optional. Only those par3d parameters that are specified will be returned.

The input values other than `times` may each be specified as lists, giving the parameter value settings at a fixed time, or as matrices or arrays. If not lists, the following formats should be used: `userMatrix` can be a  $4 \times 4 \times n$  array, or a  $4 \times 4n$  matrix; `scale` should be an  $n \times 3$  matrix; `zoom` and `FOV` should be length  $n$  vectors.

An alternative form of input is to put all of the above arguments into a list (i.e. a list of lists, or a list of arrays/matrices/vectors), and pass it as the first argument. This is the most convenient way to use this function with the [tkrgl](#) function [par3dsave](#).

Interpolation is by cubic spline or linear interpolation in an appropriate coordinate-wise fashion. Extrapolation may oscillate (repeat the sequence forward, backward, forward, etc.), cycle (repeat it forward), be constant (no repetition outside the specified time range), or be natural (linear on an appropriate scale). In the case of cycling, the first and last specified values should be equal, or the last one will be dropped. Natural extrapolation is only supported with spline interpolation.

### Value

A function is returned. The function takes one argument, and returns a list of par3d settings interpolated to that time.

**Note**

Due to a bug in R (fixed in R 2.6.0), using `extrapolate` equal to "oscillate" will sometimes fail when only two points are given.

**Author(s)**

Duncan Murdoch

**See Also**

[play3d](#) to play the animation.

**Examples**

```
f <- par3dinterp( zoom = c(1,2,3,1) )
f(0)
f(1)
f(0.5)
## Not run:
play3d(f)

## End(Not run)
```

---

persp3d

*Surface plots*

---

**Description**

This function draws plots of surfaces over the x-y plane. `persp3d` is a generic function.

**Usage**

```
persp3d(x, ...)
```

## Default S3 method:

```
persp3d(x = seq(0, 1, len = nrow(z)), y = seq(0, 1, len = ncol(z)), z,
        xlim = range(x, na.rm = TRUE), ylim = range(y, na.rm = TRUE), zlim = range(z, na.rm = TRUE),
        xlab = NULL, ylab = NULL, zlab = NULL, add = FALSE, aspect = !add, ...)
```

**Arguments**

`x`, `y` locations of grid lines at which the values in `z` are measured. These may be given as vectors or matrices. If vectors, they must be in ascending order. Either one or both may be matrices. If `x` is a list, its components `x$x` and `x$y` are used for `x` and `y`, respectively.

`z` a matrix containing the values to be plotted. Note that `x` can be used instead of `z` for convenience.

xlim, ylim, zlim	x-, y- and z-limits. The plot is produced so that the rectangular volume defined by these limits is visible.
xlab, ylab, zlab	titles for the axes. N.B. These must be character strings; expressions are not accepted. Numbers will be coerced to character strings.
add	whether to add the points to an existing plot.
aspect	either a logical indicating whether to adjust the aspect ratio, or a new ratio
...	additional material parameters to be passed to <a href="#">surface3d</a> and <a href="#">decorate3d</a> .

### Details

This is similar to [persp](#) with user interaction. See [plot3d](#) for more general details.

One difference from [persp](#) is that colors are specified on each vertex, rather than on each facet of the surface. To emulate the [persp](#) color handling, you need to do the following. First, convert the color vector to an  $(nx-1)$  by  $(ny-1)$  matrix; then add an extra row before row 1, and an extra column after the last column, to convert it to  $nx$  by  $ny$ . (These extra colors will not be used). For example, `col <- rbind(0, cbind(matrix(col, nx-1, ny-1), 0))`. Finally, call `persp3d` with material property `smooth = FALSE`.

If the `x` or `y` argument is a matrix, then it must be of the same dimension as `z`, and the values in the matrix will be used for the corresponding coordinates. This is used to plot shapes such as spheres or cylinders where `z` is not a function of `x` and `y`. See the fourth and fifth examples below.

### Value

This function is called for the side effect of drawing the plot. A vector of shape IDs is returned.

### Author(s)

Duncan Murdoch

### See Also

[plot3d](#), [persp](#). The [curve3d](#) function in the `emdbook` package draws surface plots of functions.

### Examples

```
# (1) The Obligatory Mathematical surface.
#   Rotated sinc function.

x <- seq(-10, 10, length= 30)
y <- x
f <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
z[is.na(z)] <- 1
open3d()
bg3d("white")
material3d(col="black")
```

```

persp3d(x, y, z, aspect=c(1, 1, 0.5), col = "lightblue",
        xlab = "X", ylab = "Y", zlab = "Sinc( r )")

# (2) Add to existing persp plot:

xE <- c(-10,10); xy <- expand.grid(xE, xE)
points3d(xy[,1], xy[,2], 6, col = "red")
lines3d(x, y=10, z= 6 + sin(x), col = "green")

phi <- seq(0, 2*pi, len = 201)
r1 <- 7.725 # radius of 2nd maximum
xr <- r1 * cos(phi)
yr <- r1 * sin(phi)
lines3d(xr,yr, f(xr,yr), col = "pink", lwd = 2)

# (3) Visualizing a simple DEM model

z <- 2 * volcano      # Exaggerate the relief
x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)

open3d()
bg3d("slategray")
material3d(col="black")
persp3d(x, y, z, col = "green3", aspect="iso",
        axes = FALSE, box = FALSE)

# (4) A cylindrical plot

z <- matrix(seq(0, 1, len=50), 50, 50)
theta <- t(z)
r <- 1 + exp( -pmin( (z - theta)^2, (z - theta - 1)^2, (z - theta + 1)^2 )/0.01 )
x <- r*cos(theta*2*pi)
y <- r*sin(theta*2*pi)

open3d()
persp3d(x, y, z, col="red")

# (5) A globe

lat <- matrix(seq(90,-90, len=50)*pi/180, 50, 50, byrow=TRUE)
long <- matrix(seq(-180, 180, len=50)*pi/180, 50, 50)

r <- 6378.1 # radius of Earth in km
x <- r*cos(lat)*cos(long)
y <- r*cos(lat)*sin(long)
z <- r*sin(lat)

open3d()
persp3d(x, y, z, col="white",
        texture=system.file("textures/worldsmall.png",package="rgl"),
        specular="black", axes=FALSE, box=FALSE, xlab="", ylab="", zlab="",
        normal_x=x, normal_y=y, normal_z=z)

```

```

play3d(spin3d(axis=c(0,0,1), rpm=8), duration=5)

## Not run:
# This looks much better, but is slow because the texture is very big
persp3d(x, y, z, col="white",
        texture=system.file("textures/world.png",package="rgl"),
        specular="black", axes=FALSE, box=FALSE, xlab="", ylab="", zlab="",
        normal_x=x, normal_y=y, normal_z=z)

## End(Not run)

```

---

planes

*add planes*

---

### Description

This adds mathematical planes to a scene. Their intersection with the current bounding box will be drawn.

### Usage

```

planes3d(a, b = NULL, c = NULL, d = 0, ...)
rgl.planes(a, b = NULL, c = NULL, d = 0, ...)

```

### Arguments

a, b, c	Coordinates of the normal to the plane. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
d	Coordinates of the "offset". See the details.
...	Material properties. See <a href="#">rgl.material</a> for details.

### Details

This draws planes using the parametrization  $ax + by + cz + d = 0$ . Multiple planes may be specified by giving multiple values for any of a, b, c, d; the other values will be recycled as necessary.

### Value

A shape ID of the planes object is returned invisibly.

### See Also

[abclines3d](#), [rgl.abclines](#) for mathematical lines.

[triangles3d](#), [rgl.triangles](#) or the corresponding functions for quadrilaterals may be used to draw sections of planes that do not adapt to the bounding box.

## Examples

```
# Show regression plane with z as dependent variable

x <- rnorm(100)
y <- rnorm(100)
z <- 0.2*x - 0.3*y + rnorm(100, sd=0.3)
fit <- lm(z ~ x + y)
plot3d(x,y,z, type="s", col="red", size=1)

coefs <- coef(fit)
a <- coefs["x"]
b <- coefs["y"]
c <- -1
d <- coefs["(Intercept)"]
planes3d(a, b, c, d, alpha=0.5)
```

---

play3d

*Play animation of rgl scene*

---

## Description

play3d calls a function repeatedly, passing it the elapsed time in seconds, and using the result of the function to reset the viewpoint. movie3d does the same, but records each frame to a file to make a movie.

## Usage

```
play3d(f, duration = Inf, dev = rgl.cur(), ..., startTime = 0)
movie3d(f, duration, dev = rgl.cur(), ..., fps = 10,
        movie = "movie", frames = movie, dir = tempdir(),
        convert = TRUE, clean = TRUE, verbose=TRUE,
        top = TRUE, type = "gif", startTime = 0)
```

## Arguments

f	A function returning a list that may be passed to <a href="#">par3d</a>
duration	The duration of the animation
dev	Which rgl device to select
...	Additional parameters to pass to f.
startTime	Initial time at which to start the animation
fps	Number of frames per second
movie	The base of the output filename, not including .gif
frames	The base of the name for each frame



```

        duration=4 )
## Not run:
movie3d( spin3d(), duration=5 )

## End(Not run)

```

---

plot3d

*3D Scatterplot*


---

## Description

Draws a 3D scatterplot.

## Usage

```

plot3d(x, ...)
## Default S3 method:
plot3d(x, y, z,
       xlab, ylab, zlab, type = "p", col,
       size, lwd, radius,
       add = FALSE, aspect = !add, ...)
## S3 method for class 'mesh3d'
plot3d(x, xlab = "x", ylab = "y", zlab = "z", type = c("shade", "wire", "dots"),
       add = FALSE, ...)
decorate3d(xlim, ylim, zlim,
           xlab = "x", ylab = "y", zlab = "z",
           box = TRUE, axes = TRUE, main = NULL, sub = NULL,
           top = TRUE, aspect = FALSE, ...)

```

## Arguments

<code>x, y, z</code>	vectors of points to be plotted. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
<code>xlab, ylab, zlab</code>	labels for the coordinates.
<code>type</code>	For the default method, a single character indicating the type of item to plot. Supported types are: 'p' for points, 's' for spheres, 'l' for lines, 'h' for line segments from z=0, and 'n' for nothing. For the mesh3d method, one of 'shade', 'wire', or 'dots'. Partial matching is used.
<code>col</code>	the colour to be used for plotted items.
<code>size</code>	the size for plotted points.
<code>lwd</code>	the line width for plotted items.
<code>radius</code>	the radius of spheres: see Details below.
<code>add</code>	whether to add the points to an existing plot.

aspect	either a logical indicating whether to adjust the aspect ratio, or a new ratio.
...	additional parameters which will be passed to <a href="#">par3d</a> , <a href="#">material3d</a> or <a href="#">decorate3d</a> .
xlim, ylim, zlim	limits to use for the coordinates.
box, axes	whether to draw a box and axes.
main, sub	main title and subtitle.
top	whether to bring the window to the top when done.

### Details

`plot3d` is a partial 3D analogue of `plot.default`.

Note that since `rgl` does not currently support clipping, all points will be plotted, and `xlim`, `ylim`, and `zlim` will only be used to increase the respective ranges.

Missing values in the data are skipped, as in standard graphics.

If `aspect` is `TRUE`, aspect ratios of `c(1, 1, 1)` are passed to [aspect3d](#). If `FALSE`, no aspect adjustment is done. In other cases, the value is passed to [aspect3d](#).

With `type = "s"`, spheres are drawn centered at the specified locations. The radius may be controlled by `size` (specifying the size relative to the plot display, with the default `size=3` giving a radius about 1/20 of the plot region) or `radius` (specifying it on the data scale if an isometric aspect ratio is chosen, or on an average scale if not).

### Value

`plot3d` is called for the side effect of drawing the plot; a vector of object IDs is returned.

`decorate3d` adds the usual decorations to a plot: labels, axes, etc.

### Author(s)

Duncan Murdoch

### See Also

[plot.default](#), [open3d](#), [par3d](#).

### Examples

```
open3d()
x <- sort(rnorm(1000))
y <- rnorm(1000)
z <- rnorm(1000) + atan2(x,y)
plot3d(x, y, z, col=rainbow(1000))
```

---

points3d                      *add primitive set shape*

---

### Description

Adds a shape node to the current scene

### Usage

```
points3d(x, y = NULL, z = NULL, ...)  
lines3d(x, y = NULL, z = NULL, ...)  
segments3d(x, y = NULL, z = NULL, ...)  
triangles3d(x, y = NULL, z = NULL, ...)  
quads3d(x, y = NULL, z = NULL, ...)
```

### Arguments

x, y, z	coordinates. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
...	Material properties (see <a href="#">rgl.material</a> ), normals and texture coordinates (see <a href="#">rgl.primitive</a> ).

### Details

The functions `points3d`, `lines3d`, `segments3d`, `triangles3d` and `quads3d` add points, joined lines, line segments, filled triangles or quadrilaterals to the plots. They correspond to the OpenGL types `GL_POINTS`, `GL_LINE_STRIP`, `GL_LINES`, `GL_TRIANGLES` and `GL_QUADS` respectively.

Points are taken in pairs by `segments3d`, triplets as the vertices of the triangles, and quadruplets for the quadrilaterals. Colours are applied vertex by vertex; if different at each end of a line segment, or each vertex of a polygon, the colours are blended over the extent of the object. Quadrilaterals must be entirely in one plane and convex, or the results are undefined.

These functions call the lower level functions [rgl.points](#), [rgl.linestrips](#), and so on, and are provided for convenience.

The appearance of the new objects are defined by the material properties. See [rgl.material](#) for details.

The two principal differences between the `rgl.*` functions and the `*3d` functions are that the former set all unspecified material properties to defaults, whereas the latter use current values as defaults; the former make persistent changes to material properties with each call, whereas the latter make temporary changes only for the duration of the call.

### Value

Each function returns the integer object ID of the shape that was added to the scene. These can be passed to [rgl.pop](#) to remove the object from the scene.

**Author(s)**

Ming Chen and Duncan Murdoch

**Examples**

```
# Show 12 random vertices in various ways.

M <- matrix(rnorm(36), 3, 12, dimnames=list(c('x','y','z'),
                                             rep(LETTERS[1:4], 3)))

# Force 4-tuples to be convex in planes so that quads3d works.

for (i in c(1,5,9)) {
  quad <- as.data.frame(M[,i+0:3])
  coeffs <- runif(2,0,3)
  if (mean(coeffs) < 1) coeffs <- coeffs + 1 - mean(coeffs)
  quad$C <- with(quad, coeffs[1]*(B-A) + coeffs[2]*(D-A) + A)
  M[,i+0:3] <- as.matrix(quad)
}

open3d()

# Rows of M are x, y, z coords; transpose to plot

M <- t(M)
shift <- matrix(c(-3,3,0), 12, 3, byrow=TRUE)

points3d(M)
lines3d(M + shift)
segments3d(M + 2*shift)
triangles3d(M + 3*shift, col='red')
quads3d(M + 4*shift, col='green')
text3d(M + 5*shift, texts=1:12)

# Add labels

shift <- outer(0:5, shift[1,])
shift[,1] <- shift[,1] + 3
text3d(shift,
        texts = c('points3d','lines3d','segments3d',
                  'triangles3d', 'quads3d','text3d'),
        adj = 0)
rgl.bringtotop()
```

---

r3d

*Generic 3D interface*


---

**Description**

Generic 3D interface for 3D rendering and computational geometry.

## Details

R3d is a design for an interface for 3d rendering and computation without dependency on a specific rendering implementation. R3d includes a collection of 3D objects and geometry algorithms. All r3d interface functions are named \*3d. They represent generic functions that delegate to implementation functions.

The interface can be grouped into 8 categories: Scene Management, Primitive Shapes, High-level Shapes, Geometry Objects, Visualization, Interaction, Transformation, Subdivision.

The rendering interface gives an abstraction to the underlying rendering model. It can be grouped into four categories:

**Scene Management:** A 3D scene consists of shapes, lights and background environment.

**Primitive Shapes:** Generic primitive 3D graphics shapes such as points, lines, triangles, quadrangles and texts.

**High-level Shapes:** Generic high-level 3D graphics shapes such as spheres, sprites and terrain.

**Interaction:** Generic interface to select points in 3D space using the pointer device.

In this package we include an implementation of r3d using the underlying rgl.\* functions.

3D computation is supported through the use of object structures that live entirely in R.

**Geometry Objects:** Geometry and mesh objects allow to define high-level geometry for computational purpose such as triangle or quadrangle meshes (see [mesh3d](#)).

**Transformation:** Generic interface to transform 3d objects.

**Visualization:** Generic rendering of 3d objects such as dotted, wired or shaded.

**Computation:** Generic subdivision of 3d objects.

At present, there are two main practical differences between the r3d functions and the rgl.\* functions is that the r3d functions call [open3d](#) if there is no device open, and the rgl.\* functions call [rgl.open](#). By default [open3d](#) sets the initial orientation of the coordinate system in 'world coordinates', i.e. a right-handed coordinate system in which the x-axis increases from left to right, the y-axis increases with depth into the scene, and the z-axis increases from bottom to top of the screen. rgl.\* functions, on the other hand, use a right-handed coordinate system similar to that used in OpenGL. The x-axis matches that of r3d, but the y-axis increases from bottom to top, and the z-axis decreases with depth into the scene. Since the user can manipulate the scene, either system can be rotated into the other one.

The r3d functions also preserve the rgl.material setting across calls (except for texture elements, in the current implementation), whereas the rgl.\* functions leave it as set by the last call.

The example code below illustrates the two coordinate systems.

## See Also

[points3d](#) [lines3d](#) [segments3d](#) [triangles3d](#) [quads3d](#) [text3d](#) [spheres3d](#) [sprites3d](#) [terrain3d](#) [select3d](#) [dot3d](#) [wire3d](#) [shade3d](#) [transform3d](#) [rotate3d](#) [subdivision3d](#) [mesh3d](#) [cube3d](#) [rgl](#)

**Examples**

```

x <- c(0,1,0,0)
y <- c(0,0,1,0)
z <- c(0,0,0,1)
labels <- c("Origin", "X", "Y", "Z")
i <- c(1,2,1,3,1,4)

# rgl.* interface

rgl.open()
rgl.texts(x,y,z,labels)
rgl.texts(1,1,1,"rgl.* coordinates")
rgl.lines(x[i],y[i],z[i])

# *3d interface

open3d()
text3d(x,y,z,labels)
text3d(1,1,1,"*3d coordinates")
segments3d(x[i],y[i],z[i])

```

---

rgl.bbox

*Set up Bounding Box decoration*


---

**Description**

Set up the bounding box decoration.

**Usage**

```

rgl.bbox(
  xat=NULL, xlab=NULL, xunit=0, xlen=5,
  yat=NULL, ylab=NULL, yunit=0, ylen=5,
  zat=NULL, zlab=NULL, zunit=0, zlen=5,
  marklen=15.0, marklen.rel=TRUE, expand=1,
  draw_front=FALSE, ...)
bbox3d(xat=NULL, yat=NULL, zat=NULL,
       xunit="pretty", yunit="pretty", zunit="pretty",
       expand=1.03, nticks=5,
       draw_front=FALSE, ...)

```

**Arguments**

`xat,yat,zat` vector specifying the tickmark positions  
`xlab,ylab,zlab` character vector specifying the tickmark labeling  
`xunit,yunit,zunit` value specifying the tick mark base for uniform tick mark layout

xlen, ylen, zlen	value specifying the number of tickmarks
marklen	value specifying the length of the tickmarks
marklen.rel	logical, if TRUE tick mark length is calculated using $1/\text{marklen} * \text{axis length}$ , otherwise tick mark length is marklen in coordinate space
expand	value specifying how much to expand the bounding box around the data
nticks	suggested number of ticks to use on axes
draw_front	draw the front faces of the bounding box
...	Material properties (or other rgl.bbox parameters in the case of bbox3d). See <a href="#">rgl.material</a> for details.

### Details

Four different types of tick mark layouts are possible. If `at` is not NULL, the ticks are set up at custom positions. If `unit` is numeric but not zero, it defines the tick mark base. If it is "pretty" (the default in `bbox3d`), ticks are set at [pretty](#) locations. If `length` is not zero, it specifies the number of ticks that are automatically specified. The first colour specifies the bounding box, while the second one specifies the tick mark and font colour.

`bbox3d` defaults to [pretty](#) locations for the axis labels and a slightly larger box, whereas `rgl.bbox` covers the exact range.

[axes3d](#) offers more flexibility in the specification of the axes, but they are static, unlike those drawn by [rgl.bbox](#) and [bbox3d](#).

### Value

This function is called for the side effect of setting the bounding box decoration. A shape ID is returned to allow [rgl.pop](#) to delete it.

### See Also

[rgl.material](#), [axes3d](#)

### Examples

```
rgl.open()
rgl.points(rnorm(100), rnorm(100), rnorm(100))
rgl.bbox(color=c("#333377", "white"), emission="#333377",
         specular="#3333FF", shininess=5, alpha=0.8 )

open3d()
points3d(rnorm(100), rnorm(100), rnorm(100))
bbox3d(color=c("#333377", "black"), emission="#333377",
       specular="#3333FF", shininess=5, alpha=0.8)
```

---

`rgl.bringtotop`            *Assign focus to an RGL window*

---

**Description**

'`rgl.bringtotop`' brings the current RGL window to the front of the window stack (and gives it focus).

**Usage**

```
rgl.bringtotop(stay = FALSE)
```

**Arguments**

`stay`                    whether to make the window stay on top.

**Details**

If `stay` is TRUE, then the window will stay on top of normal windows.

**Note**

not completely implemented for X11 graphics (`stay` not implemented; window managers such as KDE may block this action (set "Focus stealing prevention level" to None in Control Center/Window Behavior/Advanced)). Not currently implemented under OS/X.

**Author(s)**

Ming Chen/Duncan Murdoch

**Examples**

```
rgl.open()
rgl.points(rnorm(1000), rnorm(1000), rnorm(1000), color=heat.colors(1000))
rgl.bringtotop(stay = TRUE)
```

---

`rgl.material`            *Generic Appearance setup*

---

**Description**

Set material properties for geometry appearance.

**Usage**

```

rgl.material(
  color      = c("white"),
  alpha     = c(1.0),
  lit       = TRUE,
  ambient   = "black",
  specular  = "white",
  emission  = "black",
  shininess = 50.0,
  smooth    = TRUE,
  texture   = NULL,
  textype   = "rgb",
  texmipmap = FALSE,
  texminfilter = "linear",
  texmagfilter = "linear",
  texenvmap = FALSE,
  front     = "fill",
  back      = "fill",
  size      = 3.0,
  lwd       = 1.0,
  fog       = TRUE,
  point_antialias = FALSE,
  line_antialias = FALSE,
  depth_mask = TRUE,
  depth_test = "less",
  ...
)
material3d(...)

```

**Arguments**

color	vector of R color characters. Represents the diffuse component in case of lighting calculation (lit = TRUE), otherwise it describes the solid color characteristics.
lit	logical, specifying if lighting calculation should take place on geometry
ambient, specular, emission, shininess	properties for lighting calculation. ambient, specular, emission are R color character string values; shininess represents a numerical.
alpha	vector of alpha values between 0.0 (fully transparent) .. 1.0 (opaque).
smooth	logical, specifying whether Gouraud shading (smooth) or flat shading should be used.
texture	path to a texture image file. Supported formats: png.
textype	specifies what is defined with the pixmap <b>"alpha"</b> alpha values <b>"luminance"</b> luminance <b>"luminance.alpha"</b> luminance and alpha

	" <b>rgb</b> " color
	" <b>rgba</b> " color and alpha texture
texmipmap	Logical, specifies if the texture should be mipmapped.
texmagfilter	specifies the magnification filtering type (sorted by ascending quality): " <b>nearest</b> " texel nearest to the center of the pixel " <b>linear</b> " weighted linear average of a 2x2 array of texels
texminfilter	specifies the minification filtering type (sorted by ascending quality): " <b>nearest</b> " texel nearest to the center of the pixel " <b>linear</b> " weighted linear average of a 2x2 array of texels " <b>nearest.mipmap.nearest</b> " low quality mipmapping " <b>nearest.mipmap.linear</b> " medium quality mipmapping " <b>linear.mipmap.nearest</b> " medium quality mipmapping " <b>linear.mipmap.linear</b> " high quality mipmapping
texenvmap	logical, specifies if auto-generated texture coordinates for environment-mapping should be performed on geometry.
front, back	Determines the polygon mode for the specified side: " <b>fill</b> " filled polygon " <b>line</b> " wireframed polygon " <b>points</b> " point polygon " <b>cull</b> " culled (hidden) polygon
size	numeric, specifying the size of points in pixels
lwd	numeric, specifying the line width in pixels
fog	logical, specifying if fog effect should be applied on the corresponding shape
point_antialias, line_antialias	logical, specifying if points and lines should be antialiased
depth_mask	logical, specifying whether the object's depth should be stored.
depth_test	Determines which depth test is used to see if this object is visible, depending on its apparent depth in the scene compared to the stored depth. Possible values are "never", "less" (the default), "equal", "lequal" (less than or equal), "greater", "notequal", "gequal" (greater than or equal), "always".
...	Any of the arguments above; see Details below.

### Details

Only one side at a time can be culled.

material3d is an alternate interface to the material properties, modelled after [par3d](#): rather than setting defaults for parameters that are not specified, they will be left unchanged. material3d may also be used to query the material properties; see the examples below.

The current implementation does not return parameters for textures.

If point\_antialias is TRUE, points will be drawn as circles; otherwise, they will be drawn as squares. Lines tend to appear heavier with line\_antialias==TRUE.

The material member of the [r3dDefaults](#) list may be used to set default values for material properties.

The ... parameter to rgl.material is ignored.

**Value**

`rgl.material()` is called for the side effect of setting the material properties. It returns a value invisibly which is not intended for use by the user.

Users should use `material3d()` to query material properties. It returns values similarly to `par3d` as follows: When setting properties, it returns the previous values in a named list. A named list is also returned when more than one value is queried. When a single value is queried it is returned directly.

**See Also**

[rgl.primitive](#), [rgl.bbox](#), [rgl.bg](#), [rgl.light](#)

**Examples**

```
save <- material3d("color")
material3d(color="red")
material3d("color")
material3d(color=save)

# this illustrates the effect of depth_test
x <- c(1:3); xmid <- mean(x)
y <- c(2,1,3); ymid <- mean(y)
z <- 1
open3d()
tests <- c("never", "less", "equal", "lequal", "greater",
           "notequal", "gequal", "always")
for (i in 1:8) {
  triangles3d(x,y,z+i, col=heat.colors(8)[i])
  texts3d(xmid,ymid,z+i, paste(i, tests[i], sep="."), depth_test=tests[i])
}
```

---

`rgl.pixels`

*Extract pixel information from window*

---

**Description**

This function extracts single components of the pixel information from the topmost window.

**Usage**

```
rgl.pixels(component = c("red", "green", "blue"), viewport=par3d("viewport"), top = TRUE)
```

**Arguments**

<code>component</code>	Which component(s)?
<code>viewport</code>	Lower left corner and size of desired region.
<code>top</code>	Whether to bring window to top before reading.

**Details**

The possible components are "red", "green", "blue", "alpha", "depth", and "luminance" (the sum of the three colours). All are scaled from 0 to 1.

Note that the luminance is kept below 1 by truncating the sum; this is the definition used for the GL\_LUMINANCE component in OpenGL.

**Value**

A vector, matrix or array containing the desired components. If one component is requested, a vector or matrix will be returned depending on the size of block requested (length 1 dimensions are dropped); if more, an array, whose last dimension is the list of components.

**Author(s)**

Duncan Murdoch

**See Also**

[rgl.snapshot](#) to write a copy to a file, `demo("stereo")` for functions that make use of this to draw a random dot stereogram and an anaglyph.

**Examples**

```
example(surface3d)
depth <- rgl.pixels(component="depth")
if (length(depth) && is.matrix(depth)) # Protect against empty or single pixel windows
  contour(depth)
```

---

rgl.postscript

*export screenshot*

---

**Description**

Saves the screenshot to a file in PostScript or other vector graphics format.

**Usage**

```
rgl.postscript( filename, fmt="eps", drawText=TRUE )
```

**Arguments**

filename	full path to filename.
fmt	export format, currently supported: ps, eps, tex, pdf, svg, pgf
drawText	logical, whether to draw text

**Details**

Animations can be created in a loop modifying the scene and saving a screenshot to a file. (See example below)

This function is a wrapper for the GL2PS library by Christophe Geuzaine, and has the same limitations as that library: not all OpenGL features are supported, and some are only supported in some formats. See the reference for full details.

**Author(s)**

Christophe Geuzaine / Albrecht Gebhardt

**References**

GL2PS: an OpenGL to PostScript printing library by Christophe Geuzaine, <http://www.geuz.org/gl2ps/>, version 1.3.2.

**See Also**

[rgl.viewpoint](#), [rgl.snapshot](#)

**Examples**

```
x <- y <- seq(-10,10,length=20)
z <- outer(x,y,function(x,y) x^2 + y^2)
persp3d(x,y,z, col='lightblue')

title3d("Using LaTeX text", col='red', line=3)
rgl.postscript("persp3da.ps", "ps", drawText=FALSE)
rgl.postscript("persp3da.pdf", "pdf", drawText=FALSE)
rgl.postscript("persp3da.tex", "tex")
rgl.pop()
title3d("Using ps/pdf text", col='red', line=3)
rgl.postscript("persp3db.ps", "ps")
rgl.postscript("persp3db.pdf", "pdf")
rgl.postscript("persp3db.tex", "tex", drawText=FALSE)

## Not run:

#
# create a series of frames for an animation
#

rgl.open()
shade3d(oh3d(), color="red")
rgl.viewpoint(0,20)

for (i in 1:45) {
  rgl.viewpoint(i,20)
  filename <- paste("pic",formatC(i,digits=1,flag="0"), ".eps", sep="")
  rgl.postscript(filename, fmt="eps")
}
```

```

}

## End(Not run)

```

---

rgl.primitive	<i>add primitive set shape</i>
---------------	--------------------------------

---

### Description

Adds a shape node to the current scene

### Usage

```

rgl.points(x, y = NULL, z = NULL, ... )
rgl.lines(x, y = NULL, z = NULL, ... )
rgl.linestrips(x, y = NULL, z = NULL, ...)
rgl.triangles(x, y = NULL, z = NULL, normals=NULL, texcoords=NULL, ... )
rgl.quads(x, y = NULL, z = NULL, normals=NULL, texcoords=NULL, ... )

```

### Arguments

<code>x, y, z</code>	coordinates. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
<code>normals</code>	Normals at each point.
<code>texcoords</code>	Texture coordinates at each point.
<code>...</code>	Material properties. See <a href="#">rgl.material</a> for details.

### Details

Adds a shape node to the scene. The appearance is defined by the material properties. See [rgl.material](#) for details.

For triangles and quads, the normals at each vertex may be specified using `normals`. These may be given in any way that would be acceptable as a single argument to [xyz.coords](#). These need not match the actual normals to the polygon: curved surfaces can be simulated by using other choices of normals.

Texture coordinates may also be specified. These may be given in any way that would be acceptable as a single argument to [xy.coords](#), and are interpreted in terms of the bitmap specified as the material texture, with (0,0) at the lower left, (1,1) at the upper right. The texture is used to modulate the colour of the polygon.

These are the lower level functions called by [points3d](#), [lines3d](#), etc. The two principal differences between the `rgl.*` functions and the `*3d` functions are that the former set all unspecified material properties to defaults, whereas the latter use current values as defaults; the former make persistent changes to material properties with each call, whereas the latter make temporary changes only for the duration of the call.

**Value**

Each primitive function returns the integer object ID of the shape that was added to the scene. These can be passed to [rgl.pop](#) to remove the object from the scene.

**See Also**

[rgl.material](#), [rgl.spheres](#), [rgl.texts](#), [rgl.surface](#), [rgl.sprites](#)

**Examples**

```
rgl.open()
rgl.points(rnorm(1000), rnorm(1000), rnorm(1000), color=heat.colors(1000))
```

---

`rgl.setMouseCallbacks` *User callbacks on mouse events*

---

**Description**

This function sets user callbacks on mouse events.

**Usage**

```
rgl.setMouseCallbacks(button, begin = NULL, update = NULL, end = NULL)
```

**Arguments**

button	Which button?
begin	Called when mouse down event occurs
update	Called when mouse moves
end	Called when mouse is released

**Details**

This function sets an event handler on mouse events that occur within the current rgl window. The `begin` and `update` events should be functions taking two arguments; these will be the mouse coordinates when the event occurs. The `end` event handler takes no arguments.

Alternatively, the handlers may be set to `NULL`, the default value, in which case no action will occur.

**Value**

This function is called for the side effect of setting the mouse event handlers.

**Author(s)**

Duncan Murdoch

**See Also**

[par3d](#) to set built-in handlers

**Examples**

```
## Not quite right --- this doesn't play well with rescaling

pan3d <- function(button) {
  start <- list()

  begin <- function(x, y) {
    start$userMatrix <<- par3d("userMatrix")
    start$viewport <<- par3d("viewport")
    start$scale <<- par3d("scale")
    start$projection <<- rgl.projection()
    start$pos <<- rgl.window2user( x/start$viewport[3], 1 - y/start$viewport[4], 0.5,
                                projection=start$projection)
  }

  update <- function(x, y) {
    xlat <- (rgl.window2user( x/start$viewport[3], 1 - y/start$viewport[4], 0.5,
                            projection = start$projection) - start$pos)*start$scale
    mouseMatrix <- translationMatrix(xlat[1], xlat[2], xlat[3])
    par3d(userMatrix = start$userMatrix %*% t(mouseMatrix) )
  }

  rgl.setMouseCallbacks(button, begin, update)
  cat("Callbacks set on button", button, "of rgl device",rgl.cur(),"\n")
}
pan3d(3)
```

---

rgl.snapshot

*export screenshot*

---

**Description**

Saves the screenshot as png file.

**Usage**

```
rgl.snapshot( filename, fmt="png", top=TRUE )
snapshot3d( ... )
```

**Arguments**

filename	full path to filename.
fmt	image export format, currently supported: png
top	whether to call <a href="#">rgl.bringtotop</a>
...	arguments to pass to <code>rgl.snapshot</code>

## Details

Animations can be created in a loop modifying the scene and saving each screenshot to a file. Various graphics programs (e.g. ImageMagick) can put these together into a single animation. (See [movie3d](#) or the example below.)

## Note

On some systems, the snapshot will include content from other windows if they cover the active rgl window. Setting `top=TRUE` (the default) will use [rgl.bringtotop](#) before the snapshot to avoid this. (See <http://www.opengl.org/resources/faq/technical/rasterization.htm#rast0070> for more details.)

## See Also

[movie3d](#), [rgl.viewpoint](#)

## Examples

```
## Not run:

#
# create animation
#

shade3d(oh3d(), color="red")
rgl.bringtotop()
rgl.viewpoint(0,20)

setwd(tempdir())
for (i in 1:45) {
  rgl.viewpoint(i,20)
  filename <- paste("pic",formatC(i,digits=1,flag="0"),".png",sep="")
  rgl.snapshot(filename)
}
## Now run ImageMagick command:
##   convert -delay 10 *.png -loop 0 pic.gif

## End(Not run)
```

---

rgl.surface

*add height-field surface shape*

---

## Description

Adds a surface to the current scene. The surface is defined by a matrix defining the height of each grid point and two vectors defining the grid.

**Usage**

```
rgl.surface(x, z, y, coords=1:3, ...,
           normal_x=NULL, normal_y=NULL, normal_z=NULL,
           texture_s=NULL, texture_t=NULL)
```

**Arguments**

x	values corresponding to rows of y, or matrix of x coordinates
y	matrix of height values
z	values corresponding to columns of y, or matrix of z coordinates
coords	See details
...	Material and texture properties. See <a href="#">rgl.material</a> for details.
normal_x, normal_y, normal_z	matrices of the same dimension as y giving the coordinates of normals at each grid point
texture_s, texture_t	matrices of the same dimension as z giving the coordinates within the current texture of each grid point

**Details**

Adds a surface mesh to the current scene. The surface is defined by the matrix of height values in y, with rows corresponding to the values in x and columns corresponding to the values in z.

The coords parameter can be used to change the geometric interpretation of x, y, and z. The first entry of coords indicates which coordinate (1=X, 2=Y, 3=Z) corresponds to the x parameter. Similarly the second entry corresponds to the y parameter, and the third entry to the z parameter. In this way surfaces may be defined over any coordinate plane.

If the normals are not supplied, they will be calculated automatically based on neighbouring points.

Texture coordinates run from 0 to 1 over each dimension of the texture bitmap. If texture coordinates are not supplied, they will be calculated to render the texture exactly once over the grid. Values greater than 1 can be used to repeat the texture over the surface.

rgl.surface always draws the surface with the 'front' upwards (i.e. towards higher y values). This can be used to render the top and bottom differently; see [rgl.material](#) and the example below.

If the x or z argument is a matrix, then it must be of the same dimension as y, and the values in the matrix will be used for the corresponding coordinates. This is used to plot shapes such as cylinders where y is not a function of x and z.

NA values in the height matrix are not drawn.

**Value**

The object ID of the displayed surface is returned invisibly.

**See Also**

[rgl.material](#), [surface3d](#), [terrain3d](#). See [persp3d](#) for a higher level interface.

## Examples

```
#
# volcano example taken from "persp"
#

data(volcano)

y <- 2 * volcano      # Exaggerate the relief

x <- 10 * (1:nrow(y)) # 10 meter spacing (S to N)
z <- 10 * (1:ncol(y)) # 10 meter spacing (E to W)

ylim <- range(y)
ylen <- ylim[2] - ylim[1] + 1

colorlut <- terrain.colors(ylen) # height color lookup table

col <- colorlut[ y-ylim[1]+1 ] # assign colors to heights for each point

rgl.open()
rgl.surface(x, z, y, color=col, back="lines")
```

---

 rgl.Sweave

*Integrating rgl with Sweave*


---

## Description

As of R 2.13.0, it is possible to include rgl graphics into a [Sweave](#) document. These functions support that integration.

## Usage

```
Sweave.snapshot()
rgl.Sweave(name, width, height, options, ...)
rgl.Sweave.off()
```

## Arguments

name, width, height, options, ...

These arguments are passed by [Sweave](#) to rgl.Sweave when it opens the device.

## Details

The rgl.Sweave function is not normally called by the user. The user specifies it as the graphics driver when opening the code chunk, e.g. by using

```
<<fig=TRUE, pdf=FALSE, grdevice=rgl.Sweave>>=
```

When the `rgl` device is closed at the end of the code chunk, `rgl.Sweave.off()` will be called automatically. It will save a snapshot of the last image (by default in `' .png'` format) for inclusion in the Sweave document and (by default) close the device. Alternatively, the `Sweave.snapshot()` function can be called to save the image before the end of the chunk. Only one snapshot will be taken per chunk.

Several chunk options are used by the `rgl.Sweave` device:

**stayopen** (default FALSE). If TRUE then the `rgl` device will *not* be closed at the end of the chunk, instead a call to `Sweave.snapshot()` will be used if it has not been called explicitly. Subsequent chunks can add to the scene.

**outputtype** (default png). The output may be specified as `outputtype=pdf` or `outputtype=eps` instead, in which case the `rgl.postscript` function will be used to write output in the specified format. Note that `rgl.postscript` has limitations and does not always render scenes correctly.

**delay** (default 0.1). After creating the display window, `Sys.sleep` will be called to delay this many seconds, to allow the display system to initialize. This is needed in X11 systems which open the display asynchronously. If the default time is too short, `rgl.Sweave` may falsely report that the window is too large to open.

### Value

These functions are called for their side effects.

### Note

We recommend turning off all other graphics drivers in a chunk that uses `grdevice=rgl.Sweave`. The `rgl` functions do not write to a standard graphics device.

### Note

It is possible to use this driver in R version 2.13.0, but there are problems because there is no automatic call to `rgl.Sweave.off()` at the end of a chunk, because the chunk is always run at least twice in that version of R, and because custom string options are not supported. An explicit call to `rgl.Sweave.off()` as the last line of the code chunk is recommended, though efforts are made to do the right thing if this is omitted. Some other notes that apply to R version 2.13.0:

- Calling `rgl.close()` within the chunk should work.
- If neither `rgl.Sweave.off()` nor `rgl.close()` is called and a new chunk with `grdevice=rgl.Sweave` is opened, a call to `rgl.Sweave.off()` will be inserted. However, the last chunk with `grdevice=rgl.Sweave` will not be automatically closed.
- Running `Sweave()` within R is likely to leave a number of open `rgl` device windows.
- It is not possible to specify the output type using the `outputtype` chunk option. Snapshots will always be saved in `' .png'` format.

Most of these problems are fixed in later versions of R, including 2.13.0-patched. Within the R 2.13.x series, it is recommended to use the `figs.only=TRUE`, `pdf=FALSE` options (and leave the other graphics drivers at their default FALSE settings), which cause the code in the chunk to be run only once.

**Note**

The rgl package relies on your graphics hardware to render OpenGL scenes, and the default '.png' output copies a bitmap from the hardware device. All such devices have limitations on the size of the bitmap, but they do not always signal these limitations in a way that rgl will detect. If you find that images are not being produced properly, try reducing the size using the resolution, width or height chunk options.

**Author(s)**

Duncan Murdoch

**See Also**

[RweaveLatex](#) for a description of alternate graphics drivers in Sweave, and standard options that can be used in code chunks.

---

rgl.user2window

*Convert between rgl user and window coordinates*

---

**Description**

This function converts from 3-dimensional user coordinates to 3-dimensional window coordinates.

**Usage**

```
rgl.user2window(x, y = NULL, z = NULL, projection = rgl.projection())  
rgl.window2user(x, y = NULL, z = 0, projection = rgl.projection())  
rgl.projection()
```

**Arguments**

x, y, z	Input coordinates. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
projection	The rgl projection to use

**Details**

These functions convert between user coordinates and window coordinates.

Window coordinates run from 0 to 1 in X, Y, and Z. X runs from 0 on the left to 1 on the right; Y runs from 0 at the bottom to 1 at the top; Z runs from 0 foremost to 1 in the background. rgl does not currently display vertices plotted outside of this range, but in normal circumstances will automatically resize the display to show them. In the example below this has been suppressed.

**Value**

The coordinate conversion functions produce a matrix with columns corresponding to the X, Y, and Z coordinates.

`rgl.projection()` returns a list containing the model matrix, projection matrix and viewport. See [par3d](#) for more details.

**Author(s)**

Ming Chen / Duncan Murdoch

**See Also**

[select3d](#)

**Examples**

```
open3d()
points3d(rnorm(100), rnorm(100), rnorm(100))
if (interactive() || !.Platform$OS=="unix") {
# Calculate a square in the middle of the display and plot it
square <- rgl.window2user(c(0.25, 0.25, 0.75, 0.75, 0.25),
                          c(0.25, 0.75, 0.75, 0.25, 0.25), 0.5)
par3d(ignoreExtent = TRUE)
lines3d(square)
par3d(ignoreExtent = FALSE)
}
```

---

scene

*scene management*

---

**Description**

Clear shapes, lights, bbox

**Usage**

```
clear3d( type = c("shapes", "bboxdeco", "material"), defaults )
rgl.clear( type = "shapes" )
pop3d( ... )
rgl.pop( type = "shapes", id = 0 )
rgl.ids( type = "shapes" )
```

**Arguments**

type	Select subtype(s): <b>"shapes"</b> shape stack <b>"lights"</b> light stack <b>"bboxdeco"</b> bounding box <b>"viewpoint"</b> viewpoint <b>"material"</b> material properties <b>"background"</b> scene background <b>"all"</b> all of the above
defaults	default values to use after clearing
id	vector of ID numbers of items to remove
...	generic arguments passed through to RGL-specific (or other) functions

**Details**

RGL holds two stacks. One is for shapes and the other is for lights. `clear3d` and `rgl.clear` clear the specified stack, or restore the defaults for the bounding box (not visible) or viewpoint. By default with `id=0` `rgl.pop` removes the top-most (last added) node on the shape stack. The `id` argument may be used to specify arbitrary item(s) to remove from the specified stack.

`rgl.clear` and `clear3d` may also be used to clear material properties back to their defaults.

`clear3d` has an optional `defaults` argument, which defaults to `r3dDefaults`. Only the materials component of this argument is currently used by `clear3d`.

`rgl.ids` returns a dataframe containing the IDs in the currently active rgl window, along with an indicator of their type.

For convenience, `type="shapes"` and `id=1` signifies the bounding box.

Note that clearing the light stack leaves the scene in darkness; it should normally be followed by a call to `rgl.light` or `light3d`.

**See Also**

[rgl](#), [rgl.bbox](#), [rgl.light](#), [open3d](#) to open a new window.

**Examples**

```
x <- rnorm(100)
y <- rnorm(100)
z <- rnorm(100)
p <- plot3d(x, y, z, type='s')
rgl.ids()
lines3d(x, y, z)
rgl.ids()
if (interactive()) {
  readline("Hit enter to change spheres")
  rgl.pop(id = p["data"])
  spheres3d(x, y, z, col="red", radius=1/5)
  box3d()
}
```

---

`select3d`*Select a rectangle in an RGL scene*

---

**Description**

This function allows the user to use the mouse to select a region in an RGL scene.

**Usage**

```
rgl.select3d(button = c("left", "middle", "right"))
select3d(...)
```

**Arguments**

<code>button</code>	Which button to use for selection.
<code>...</code>	Button argument to pass to <code>rgl.select3d</code>

**Details**

This function selects 3-dimensional regions by allowing the user to use a mouse to draw a rectangle showing the projection of the region onto the screen. It returns a function which tests points for inclusion in the selected region.

If the scene is later moved or rotated, the selected region will remain the same, no longer corresponding to a rectangle on the screen.

**Value**

Returns a function  $f(x, y, z)$  which tests whether each of the points  $(x, y, z)$  is in the selected region, returning a logical vector. This function accepts input in a wide variety of formats as it uses [xyz.coords](#) to interpret its parameters.

**Author(s)**

Ming Chen / Duncan Murdoch

**See Also**

[locator](#)

**Examples**

```
# Allow the user to select some points, and then redraw them
# in a different color

if (interactive()) {
  x <- rnorm(1000)
  y <- rnorm(1000)
```

```

z <- rnorm(1000)
open3d()
points3d(x,y,z)
f <- select3d()
keep <- f(x,y,z)
rgl.pop()
points3d(x[keep],y[keep],z[keep],color='red')
points3d(x[!keep],y[!keep],z[!keep])
}

```

---

shapelist3d

*Create and plot a list of shapes*


---

### Description

These functions create and plot a list of shapes.

### Usage

```
shapelist3d(shapes, x = 0, y = NULL, z = NULL, size = 1, matrix = NULL, override = TRUE,
            ..., plot = TRUE)
```

### Arguments

shapes	A single shape3d object, or a list of them.
x, y, z	Translation(s) to apply
size	Scaling(s) to apply
matrix	A single matrix transformation, or a list of them.
override	Whether the material properties should override the ones in the shapes.
...	Material properties to apply.
plot	Whether to plot the result.

### Details

shapelist3d is a quick way to create a complex object made up of simpler ones. Each of the arguments shapes through override may be a vector of values (a list in the case of shapes or matrix). All values will be recycled to produce a list of shapes as long as the longest of them.

The `xyz.coords` function will be used to process the x, y and z arguments, so a matrix may be used as x to specify all three. If a vector is used for x but y or z is missing, default values of 0 will be used.

The "shapelist3d" class is simply a list of "shape3d" objects.

Methods for `dot3d`, `wire3d`, `shade3d`, `translate3d`, `scale3d`, and `rotate3d` are defined for these objects.

**Value**

An object of class `c("shapelist3d", "shape3d")`.

**Author(s)**

Duncan Murdoch

**See Also**

[mesh3d](#)

**Examples**

```
shapelist3d(icosahedron3d(), x=rnorm(10), y=rnorm(10), z=rnorm(10), col=1:5, size=0.3)
```

---

spheres

*add sphere set shape*

---

**Description**

Adds a sphere set shape node to the scene

**Usage**

```
spheres3d(x, y = NULL, z = NULL, radius = 1, ...)
rgl.spheres(x, y = NULL, z = NULL, radius, ...)
```

**Arguments**

<code>x, y, z</code>	Numeric vector of point coordinates corresponding to the center of each sphere. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
<code>radius</code>	Vector or single value defining the sphere radius/radii
<code>...</code>	Material properties. See <a href="#">rgl.material</a> for details.

**Details**

If a non-isometric aspect ratio is chosen, these functions will still draw objects that appear to the viewer to be spheres. Use [ellipse3d](#) to draw shapes that are spherical in the data scale.

When the scale is not isometric, the radius is measured in an average scale. In this case the bounding box calculation is iterative, since rescaling the plot changes the shape of the spheres in user-coordinate, which changes the bounding box. Versions of `rgl` prior to 0.92.802 did not do this iterative adjustment.

If any coordinate or radius is NA, the sphere is not plotted.

**Value**

A shape ID of the spheres object is returned.

**See Also**

[rgl.material](#), [aspect3d](#) for setting non-isometric scales

**Examples**

```
open3d()
spheres3d(rnorm(10), rnorm(10), rnorm(10), radius=runif(10), color=rainbow(10))
```

---

spin3d

*Create a function to spin a scene at a fixed rate*

---

**Description**

This creates a function to use with [play3d](#) to spin an rgl scene at a fixed rate.

**Usage**

```
spin3d(axis = c(0, 0, 1), rpm = 5)
```

**Arguments**

axis	The desired axis of rotation
rpm	The rotation speed in rotations per minute

**Value**

A function with header `function(time)`. This function calculates and returns a list containing `userMatrix` updated by spinning it for `time` seconds at `rpm` revolutions per minute about the specified axis.

**Author(s)**

Duncan Murdoch

**See Also**

[play3d](#) to play the animation

**Examples**

```
open3d()
plot3d(oh3d(col="lightblue", alpha=0.5))
play3d(spin3d(axis=c(1,0,0), rpm=20), duration=3)
```

---

sprites                      *add sprite set shape*

---

### Description

Adds a sprite set shape node to the scene.

### Usage

```
sprites3d(x, y = NULL, z = NULL, radius = 1, ...)
particles3d(x, y = NULL, z = NULL, radius = 1, ...)
rgl.sprites(x, y = NULL, z = NULL, radius = 1, ...)
```

### Arguments

x, y, z	point coordinates. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
radius	vector or single value defining the sphere radius
...	material properties, texture mapping is supported

### Details

Sprites are rectangle planes that are directed towards the viewpoint. Their primary use is for fast (and faked) atmospherical effects, e.g. particles and clouds using alpha blended textures. Particles are Sprites using an alpha-blended particle texture giving the illusion of clouds and gasses.

If any coordinate is NA, the sprite is not plotted.

### Value

These functions are called for the side effect of displaying the sprites. The shape ID of the displayed object is returned.

### See Also

[rgl.material](#)

### Examples

```
open3d()
particles3d( rnorm(100), rnorm(100), rnorm(100), color=rainbow(100) )
# is the same as
sprites3d( rnorm(100), rnorm(100), rnorm(100), color=rainbow(100),
  lit=FALSE, alpha=.2,
  textype="alpha", texture=system.file("textures/particle.png", package="rgl") )
```

---

subdivision3d                      *generic subdivision surface method*

---

### Description

The Subdivision surface algorithm divide and refine (deform) a given mesh recursively to certain degree (depth). The mesh3d algorithm consists of two stages: divide and deform. The divide step generates for each triangle or quad four new triangles or quads, the deform step drags the points (refinement step).

### Usage

```
subdivision3d( x, ... )
## S3 method for class 'mesh3d'
subdivision3d( x, depth=1, normalize=FALSE, deform=TRUE, ... )
  divide.mesh3d(mesh, vb=mesh$vb, ib=mesh$ib, it=mesh$it )
  normalize.mesh3d(mesh)
  deform.mesh3d(mesh, vb=mesh$vb, ib=mesh$ib, it=mesh$it )
```

### Arguments

x	3d geometry mesh
mesh	3d geometry mesh
depth	recursion depth
normalize	normalize mesh3d coordinates after division if deform is TRUE
deform	deform mesh
it	indices for triangular faces
ib	indices for quad faces
vb	matrix of vertices: 4xn matrix (rows x,y,z,h) or equivalent vector, where h indicates scaling of each plotted quad
...	other arguments (unused)

### Details

Generic subdivision surface method. Currently there exists an algorithm that can be applied on mesh3d objects.

### See Also

[r3d mesh3d](#)

### Examples

```
open3d()
shade3d( subdivision3d( cube3d(), depth=3 ), color="red", alpha=0.5 )
```

---

 surface3d

*add height-field surface shape*


---

### Description

Adds a surface to the current scene. The surface is defined by a matrix defining the height of each grid point and two vectors defining the grid.

### Usage

```
surface3d(x, y, z, ..., normal_x=NULL, normal_y=NULL, normal_z=NULL)
terrain3d(x, y, z, ..., normal_x=NULL, normal_y=NULL, normal_z=NULL)
```

### Arguments

x	values corresponding to rows of z, or matrix of x coordinates
y	values corresponding to the columns of z, or matrix of y coordinates
z	matrix of heights
...	Material and texture properties. See <a href="#">rgl.material</a> for details.
normal_x, normal_y, normal_z	matrices of the same dimension as z giving the coordinates of normals at each grid point

### Details

Adds a surface mesh to the current scene. The surface is defined by the matrix of height values in z, with rows corresponding to the values in x and columns corresponding to the values in y. This is the same parametrization as used in [persp](#).

If the x or y argument is a matrix, then it must be of the same dimension as z, and the values in the matrix will be used for the corresponding coordinates. This is used to plot shapes such as cylinders where z is not a function of x and y.

If the normals are not supplied, they will be calculated automatically based on neighbouring points.

surface3d always draws the surface with the 'front' upwards (i.e. towards higher z values). This can be used to render the top and bottom differently; see [rgl.material](#) and the example below.

For more flexibility in defining the surface, use [rgl.surface](#).

surface3d and terrain3d are synonyms.

### See Also

[rgl.material](#), [rgl.surface](#). See [persp3d](#) for a higher level interface.

## Examples

```
#
# volcano example taken from "persp"
#

data(volcano)

z <- 2 * volcano      # Exaggerate the relief

x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)

zlim <- range(z)
zlen <- zlim[2] - zlim[1] + 1

colorlut <- terrain.colors(zlen) # height color lookup table

col <- colorlut[ z-zlim[1]+1 ] # assign colors to heights for each point

open3d()
surface3d(x, y, z, color=col, back="lines")
```

---

texts	<i>add text</i>
-------	-----------------

---

## Description

Adds text to the scene. The text is positioned in 3D space. Text is always oriented towards the camera.

## Usage

```
rgl.texts(x, y = NULL, z = NULL, text, adj = 0.5, justify, family = par3d("family"), font = par3d("font")
         cex = par3d("cex"), useFreeType=par3d("useFreeType"), ...)
text3d(x, y = NULL, z = NULL, texts, adj = 0.5, justify, ...)
texts3d(x, y = NULL, z = NULL, texts, adj = 0.5, justify, ...)
rglFonts(...)
```

## Arguments

x, y, z	point coordinates. Any reasonable way of defining the coordinates is acceptable. See the function <a href="#">xyz.coords</a> for details.
text	text character vector to draw
texts	text character vector to draw
adj	one value specifying the horizontal adjustment, or two, specifying horizontal and vertical adjustment respectively. .

justify	(deprecated, please use <code>adj</code> instead) character string specifying the horizontal adjustment; options are "left", "right", "center".
family	A device-independent font family name, or ""
font	A numeric font number from 1 to 5
cex	A numeric character expansion value
useFreeType	logical. Should FreeType be used to draw text? (See details below.)
...	In <code>rgl.texts</code> , material properties; see <a href="#"><code>rgl.material</code></a> for details. In <code>rglFonts</code> , device dependent font definitions for use with FreeType. In the other functions, additional parameters to pass to <code>rgl.texts</code> .

### Details

The `adj` parameter determines the position of the text relative to the specified coordinate. Use `adj = c(0,0)` to place the left bottom corner at  $(x, y, z)$ , `adj = c(0.5, 0.5)` to center the text there, and `adj = c(1,1)` to put the right top corner there. The optional second coordinate for vertical adjustment defaults to 0.5. Placement is done using the "advance" of the string and the "ascent" of the font relative to the baseline, when these metrics are known.

`text3d` and `texts3d` draw text using the [`r3d`](#) conventions. These are synonyms; the former is singular to be consistent with the classic 2-D graphics functions, and the latter is plural to be consistent with all the other graphics primitives. Take your choice!

If any coordinate or text is NA, that text is not plotted.

### Value

The text drawing functions return the object ID of the text object `i` invisibly.

`rglFonts` returns the current set of font definitions.

### Fonts

Fonts are specified using the `family`, `font`, `cex`, and `useFreeType` arguments. Defaults for the currently active device may be set using [`par3d`](#), or for future devices using [`r3dDefaults`](#).

The `family` specification is the same as for standard graphics, i.e. families `c("serif", "sans", "mono", "symbol")` are normally available, but users may add additional families. `font` numbers are restricted to the range 1 to 4 for standard, bold, italic and bold italic respectively; with font 5 recoded as family "symbol" font 1.

Using an unrecognized value for "family" will result in the system standard font as used in `rgl` up to version 0.76. That font is not resizable and font values are ignored.

If `useFreeType` is TRUE, then `rgl` will use the FreeType anti-aliased fonts for drawing. This is generally desirable, and it is the default if `rgl` was built to support FreeType.

FreeType fonts are specified using the `rglFonts` function. This function takes a vector of four filenames of TrueType font files which will be used for the four styles regular, bold, italic and bold italic. The vector is passed with a name to be used as the family name, e.g. `rglFonts(sans = c("/path/to/FreeSans.ttf"`  
In order to limit the file size, `rgl` ships with just 3 font files, for regular versions of the `serif`, `sans` and `mono` families. Additional free font files are available from the Amaya project at <http://dev.w3.org/cvsweb/Amaya/fonts/>. See the example below for how to specify a full set of fonts.

Full pathnames should normally be used to specify font files. If relative paths are used, they are interpreted differently by platform. Currently Windows fonts are looked for in the Windows fonts folder, while other platforms use the current working directory.

If FreeType fonts are not used, then bitmapped fonts will be used instead. On Windows these will be based on the fonts specified using the `windowsFonts` function, and are resizable. Other platforms will use the default bitmapped font which is not resizable. Currently MacOSX defaults to the bitmapped font, as our font library appears unable to read fonts properly on that platform.

### See Also

[r3d](#)

### Examples

```
open3d()
famnum <- rep(1:4, 8)
family <- c("serif", "sans", "mono", "symbol")[famnum]
font <- rep(rep(1:4, each=4), 2)
cex <- rep(1:2, each=16)
text3d(font, cex, famnum, text=paste(family, font),adj = 0.5,
        color="blue", family=family, font=font, cex=cex)

## Not run:
# These FreeType fonts are available from the Amaya project, and are not shipped
# with rgl. You would normally install them to the rgl/fonts directory
# and use fully qualified pathnames, e.g.
# system.file("fonts/FreeSerif.ttf", package= "rgl")

rglFonts(serif=c("FreeSerif.ttf", "FreeSerifBold.ttf", "FreeSerifItalic.ttf",
                "FreeSerifBoldItalic.ttf"),
         sans =c("FreeSans.ttf", "FreeSansBold.ttf", "FreeSansOblique.ttf",
                "FreeSansBoldOblique.ttf"),
         mono =c("FreeMono.ttf", "FreeMonoBold.ttf", "FreeMonoOblique.ttf",
                "FreeMonoBoldOblique.ttf"),
         symbol=c("ESSTIX10.TTF", "ESSTIX12.TTF", "ESSTIX9_.TTF",
                "ESSTIX11.TTF"))

## End(Not run)
```

---

viewpoint

*Set up viewpoint*

---

### Description

Set the viewpoint orientation.

### Usage

```
view3d( theta = 0, phi = 15, ...)
rgl.viewpoint( theta = 0, phi = 15, fov = 60, zoom = 1, scale = par3d("scale"),
              interactive = TRUE, userMatrix )
```

**Arguments**

theta,phi	polar coordinates
...	additional parameters to pass to <code>rgl.viewpoint</code>
fov	field-of-view angle in degrees
zoom	zoom factor
scale	real length 3 vector specifying the rescaling to apply to each axis
interactive	logical, specifying if interactive navigation is allowed
userMatrix	4x4 matrix specifying user point of view

**Details**

The viewpoint can be set in an orbit around the data model, using the polar coordinates `theta` and `phi`. Alternatively, it can be set in a completely general way using the 4x4 matrix `userMatrix`. If `userMatrix` is specified, `theta` and `phi` are ignored.

The pointing device of your graphics user-interface can also be used to set the viewpoint interactively. With the pointing device the buttons are by default set as follows:

**left** adjust viewpoint position

**middle** adjust field of view angle

**right or wheel** adjust zoom factor

If the `fov` angle is set to 0, a parallel or orthogonal projection is used. Small non-zero values (e.g. 0.01 or less, but not 0.0) are likely to lead to rendering errors due to OpenGL limitations.

**See Also**

[par3d](#)

**Examples**

```
## Not run:
# animated round trip tour for 10 seconds

rgl.open()
shade3d(oh3d(), color="red")

start <- proc.time()[3]
while ((i <- 36*(proc.time()[3]-start)) < 360) {
  rgl.viewpoint(i,i/4);
}

## End(Not run)
```

---

writeWebGL	<i>Write scene to HTML.</i>
------------	-----------------------------

---

### Description

Writes the current scene to a collection of files that contain WebGL code to reproduce it in a browser.

### Usage

```
writeWebGL(dir = "webGL", filename = file.path(dir, "index.html"),
           template = system.file(file.path("WebGL", "template.html"), package = "rgl"),
           prefix = "",
           snapshot = TRUE, font = "Arial", width, height)
```

### Arguments

<code>dir</code>	Where to write the files.
<code>filename</code>	The filename to use for the main file.
<code>template</code>	The template web page to which to write the Javascript for the scene. See Details below.
<code>prefix</code>	An optional prefix to use on global identifiers in the scene; use different prefixes for different scenes displayed on the same web page. If not blank, it should be a legal identifier in Javascript and HTML.
<code>snapshot</code>	Whether to include a snapshot of the scene, to be displayed in browsers that don't support WebGL.
<code>font</code>	The font to use for text.
<code>width, height</code>	The (optional) width and height in pixels of the image to display. If omitted, the <code>par3d("windowRect")</code> dimensions will be used.

### Details

This function writes out a web page containing Javascript that reconstructs the scene in WebGL.

Use the `template` argument to give the filename of a web page that is to contain the code for the new scene. It should contain a single line containing `paste("%", prefix, "WebGL%", e.g. %WebGL%` with the default empty prefix. That line will be replaced by the Javascript and other code necessary to reproduce the current scene.

To put more than one scene into a web page, use different values of `prefix` for each. The prefix will be used in identifiers in both Javascript and HTML, so it is safest to start with a letter and only use alphanumeric characters.

WebGL is a fairly new technology for displaying 3D scenes in browsers. Most current browsers support it to some extent, though it may not be enabled by default; see <http://get.webgl.org> for details. A major exception currently is Microsoft's Internet Explorer, though plugins are available.

Currently `writeWebGL` has a number of known limitations, some of which will be gradually eliminated as development progresses:

- The bounding box decorations are fixed; labels do not move as they do within R.
- User-defined mouse controls are not supported.
- No automatic movement (e.g. rotation via `spin3d`) is supported.
- Missing values are not handled properly.
- It only handles a single light source, and the geometry of specular reflections is simpler than in R.
- Polygons will only be rendered as filled; there is no support in WebGL for wireframe or point rendering.

**Value**

The filename is returned.

**Note**

The output includes a binary copy of the CanvasMatrix Javascript library. Its source contains the following copyright notice:

```
/*
 * Copyright (C) 2009 Apple Inc. All Rights Reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY APPLE INC. ‘‘AS IS’’ AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL APPLE INC. OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
 * OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

**Author(s)**

Duncan Murdoch.

## References

<http://www.webgl.org>

## Examples

```
plot3d(rnorm(100), rnorm(100), rnorm(100), type="s", col="red")
# This writes a copy into temporary directory 'webGL', and then displays it
browseURL(paste("file://", writeWebGL(dir=file.path(tempdir(), "webGL"),
width=500), sep=""))
```

# Index

- \*Topic **dplot**
  - ellipse3d, 12
  - par3dinterp, 24
  - play3d, 29
  - spin3d, 57
- \*Topic **dynamic**
  - abclines, 4
  - addNormals, 5
  - aspect3d, 6
  - axes3d, 8
  - bg, 10
  - cylinder3d, 11
  - grid3d, 14
  - light, 15
  - matrices, 16
  - mesh3d, 18
  - par3d, 20
  - persp3d, 25
  - planes, 28
  - plot3d, 31
  - points3d, 33
  - r3d, 34
  - rgl-package, 2
  - rgl.bbox, 36
  - rgl.bringtotop, 38
  - rgl.material, 38
  - rgl.pixels, 41
  - rgl.postscript, 42
  - rgl.primitive, 44
  - rgl.setMouseCallbacks, 45
  - rgl.snapshot, 46
  - rgl.surface, 47
  - rgl.user2window, 51
  - scene, 52
  - select3d, 54
  - shapelist3d, 55
  - spheres, 56
  - sprites, 58
  - subdivision3d, 59
  - surface3d, 60
  - texts, 61
  - viewpoint, 63
- \*Topic **graphics**
  - attributes, 7
  - writeWebGL, 65
- \*Topic **utilities**
  - rgl.Sweave, 49
- abclines, 4
- abclines3d, 15, 28
- abclines3d (abclines), 4
- addNormals, 5
- asEuclidean (matrices), 16
- asHomogeneous (matrices), 16
- aspect3d, 6, 22, 32, 57
- attributes, 7
- axes3d, 8, 37
- axis, 9
- axis3d, 14, 15
- axis3d (axes3d), 8
- bbox3d, 8, 9, 37
- bbox3d (rgl.bbox), 36
- bg, 10
- bg3d, 21
- bg3d (bg), 10
- box, 9
- box3d (axes3d), 8
- clear3d (scene), 52
- cube3d, 35
- cube3d (mesh3d), 18
- cubeoctahedron3d (mesh3d), 18
- curve3d, 26
- cylinder3d, 11
- decorate3d, 26
- decorate3d (plot3d), 31
- deform.mesh3d (subdivision3d), 59

- divide.mesh3d (subdivision3d), 59
- dodecahedron3d (mesh3d), 18
- dot3d, 35, 55
- dot3d (mesh3d), 18
- ellipse3d, 12, 56
- grid, 14
- grid3d, 14
- icosahedron3d (mesh3d), 18
- identityMatrix (matrices), 16
- light, 15
- light3d, 53
- light3d (light), 15
- lines3d, 35, 44
- lines3d (points3d), 33
- locator, 54
- material3d, 8, 32
- material3d (rgl.material), 38
- matrices, 16, 23
- mesh3d, 13, 17, 18, 35, 56, 59
- movie3d, 47
- movie3d (play3d), 29
- mtext3d (axes3d), 8
- normalize.mesh3d (subdivision3d), 59
- octahedron3d (mesh3d), 18
- oh3d (mesh3d), 18
- open3d, 3, 32, 35, 53
- open3d (par3d), 20
- par3d, 6, 18, 19, 20, 24, 29, 32, 40, 41, 46, 52, 62, 64
- par3dinterp, 24, 30
- par3dsave, 24
- particles3d (sprites), 58
- persp, 26, 60
- persp3d, 25, 48, 60
- planes, 28
- planes3d, 4
- planes3d (planes), 28
- play3d, 25, 29, 57
- plot.default, 32
- plot3d, 6, 21, 26, 31
- points3d, 33, 35, 44
- pop3d (scene), 52
- pretty, 14, 37
- qmesh3d, 13
- qmesh3d (mesh3d), 18
- quads3d, 35
- quads3d (points3d), 33
- r3d, 3, 19, 34, 59, 62, 63
- r3dDefaults, 21, 40, 53, 62
- r3dDefaults (par3d), 20
- rgl, 35, 53
- rgl (rgl-package), 2
- rgl-package, 2
- rgl.abclines, 28
- rgl.abclines (abclines), 4
- rgl.antialias (rgl-package), 2
- rgl.attrib (attributes), 7
- rgl.bbox, 3, 36, 37, 41, 53
- rgl.bg, 3, 41
- rgl.bg (bg), 10
- rgl.bringtotop, 30, 38, 46, 47
- rgl.clear, 3, 16
- rgl.clear (scene), 52
- rgl.close (rgl-package), 2
- rgl.cur (rgl-package), 2
- rgl.ids, 7, 8
- rgl.ids (scene), 52
- rgl.init (rgl-package), 2
- rgl.light, 3, 41, 53
- rgl.light (light), 15
- rgl.lines, 3
- rgl.lines (rgl.primitive), 44
- rgl.linestrips, 33
- rgl.linestrips (rgl.primitive), 44
- rgl.material, 10, 28, 33, 37, 38, 44, 45, 48, 56–58, 60, 62
- rgl.open, 35
- rgl.open (rgl-package), 2
- rgl.pixels, 41
- rgl.planes, 4
- rgl.planes (planes), 28
- rgl.points, 3, 33
- rgl.points (rgl.primitive), 44
- rgl.pop, 3, 16, 33, 37, 45
- rgl.pop (scene), 52
- rgl.postscript, 42, 50
- rgl.primitive, 19, 33, 41, 44
- rgl.projection (rgl.user2window), 51
- rgl.quads, 3

- rgl.quads (rgl.primitive), 44
- rgl.quit (rgl-package), 2
- rgl.select3d (select3d), 54
- rgl.set (rgl-package), 2
- rgl.setMouseCallbacks, 22, 45
- rgl.snapshot, 3, 30, 42, 43, 46
- rgl.spheres, 3, 45
- rgl.spheres (spheres), 56
- rgl.sprites, 3, 45
- rgl.sprites (sprites), 58
- rgl.surface, 3, 45, 47, 60
- rgl.Sweave, 49
- rgl.texts, 3, 45
- rgl.texts (texts), 61
- rgl.triangles, 3, 28
- rgl.triangles (rgl.primitive), 44
- rgl.user2window, 51
- rgl.viewpoint, 3, 23, 43, 47
- rgl.viewpoint (viewpoint), 63
- rgl.window2user (rgl.user2window), 51
- rglFonts (texts), 61
- rotate3d, 35, 55
- rotate3d (matrices), 16
- rotationMatrix (matrices), 16
- RweaveLatex, 51
- scale3d, 55
- scale3d (matrices), 16
- scaleMatrix (matrices), 16
- scene, 52
- segments3d, 4, 35
- segments3d (points3d), 33
- select3d, 22, 35, 52, 54
- shade3d, 35, 55
- shade3d (mesh3d), 18
- shape3d (mesh3d), 18
- shapelist3d, 19, 55
- snapshot3d (rgl.snapshot), 46
- spheres, 56
- spheres3d, 35
- spheres3d (spheres), 56
- spin3d, 30, 57, 66
- sprintf, 30
- sprites, 58
- sprites3d, 35
- sprites3d (sprites), 58
- subdivision3d, 13, 35, 59
- surface3d, 26, 48, 60
- Sweave, 49
- Sweave.snapshot (rgl.Sweave), 49
- Sys.sleep, 30, 50
- terrain3d, 35, 48
- terrain3d (surface3d), 60
- tetrahedron3d (mesh3d), 18
- text3d, 21, 35
- text3d (texts), 61
- texts, 61
- texts3d (texts), 61
- title3d (axes3d), 8
- tkrgl, 24
- tmesh3d (mesh3d), 18
- transform3d, 35
- transform3d (matrices), 16
- translate3d, 55
- translate3d (matrices), 16
- translationMatrix (matrices), 16
- triangles3d, 28, 35
- triangles3d (points3d), 33
- view3d (viewpoint), 63
- viewpoint, 63
- wire3d, 35, 55
- wire3d (mesh3d), 18
- writeWebGL, 65
- xy.coords, 44
- xyz.coords, 4, 12, 28, 31, 33, 44, 51, 54–56, 58, 61