Package 'oblicubes'

March 10, 2025

Type Package

Title 3D Rendering Using Obliquely Projected Cubes and Cuboids

Version 1.0.0

Description Three-dimensional rendering for 'grid' and 'ggplot2' graphics using cubes and cuboids drawn with an oblique projection. As a special case also supports primary view orthographic projections. Can be viewed as an extension to the 'isocubes' package <https://github.com/coolbutuseless/isocubes>.

URL https://trevorldavis.com/R/oblicubes/

BugReports https://github.com/trevorld/oblicubes/issues

License MIT + file LICENSE

Imports grDevices, grid, utils

Suggests datasets, dplyr, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0), vdiffr

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

RoxygenNote 7.3.1

Config/testthat/edition 3

NeedsCompilation no

Author Trevor L. Davis [aut, cre] (<https://orcid.org/0000-0001-6341-4639>), Mike FC [aut] (Some code adapted from isocubes)

Maintainer Trevor L. Davis <trevor.l.davis@gmail.com>

Repository CRAN

Date/Publication 2025-03-10 08:30:02 UTC

Contents

`cheap_darken` *'light' effect helper functions*

Description

Helper functions to generate a "light" effect for `oblicubesGrob()`, `grid.oblicubes()`, `oblicuboidsGrob()`, and `grid.oblicuboids()`. `darken_face()` is the default `light` argument for `oblicubesGrob()`, `grid.oblicubes()`, `oblicuboidsGrob()`, and `grid.oblicuboids()`. `cheap_darken()` is the default darkening function used by `darken_face()`.

Usage

```
cheap_darken(col, amount)

darken_face(
  face,
  col,
  top = 0,
  west = 0.2,
  east = 0.2,
  south = 0.4,
  north = 0.4,
  darken_fn = cheap_darken
)
```

Arguments

`col`	Vector of colors to darken
`amount`	Fraction to darken by
`face`	Cube/cuboid face to color. One of "top", "west", "east", "south", or "north".
`top`	Amount to darken the "top" face.
`west`	Amount to darken the "west" face.
`east`	Amount to darken the "east" face.
`south`	Amount to darken the "south" face.
`north`	Amount to darken the "north" face.
`darken_fn`	Function to darken with. Should take two arguments: the first should be the colour and the second should be numeric amount to darken by. Default will be to use `cheap_darken()`. `colorspace::darken()` is a slower, "better" alternative.

Details

The `light` argument of `oblicubesGrob()`, `grid.oblicubes()`, `geom_oblicubes()`, `oblicuboidsGrob()`, `grid.oblicuboids()`, and `geom_oblicuboids()` needs a function that that takes two arguments: the first is `face` one of its five faces: "top", "west", "east", "south", or "north" and the second is `col` the cube/cuboid's fill color

Value

Vector of darkened colors.

Examples

```
demo_light <- function(light = darken_face, ...) {
  df <- data.frame(x=1, y=1, z=1)
  grid::grid.newpage()
  grid.oblicubes(df, ..., light=light, angle=45, lwd=4,
                 vp = grid::viewport(0.25, 0.25, 0.5, 0.5))
  grid.oblicubes(df, ..., light=light, angle=135, lwd=4,
                 vp = grid::viewport(0.75, 0.25, 0.5, 0.5))
  grid.oblicubes(df, ..., light=light, angle=-45, lwd=4,
                 vp = grid::viewport(0.25, 0.75, 0.5, 0.5))
  grid.oblicubes(df, ..., light=light, angle=-135, lwd=4,
                 vp = grid::viewport(0.75, 0.75, 0.5, 0.5))
}
demo_light()
demo_light(fill = "gold")
demo_light(light = function(face, col)
             darken_face(face, col, top = 0.3,
                         west = 0.6, east = 0.6,
                         south = 0.0, north = 0.0)
)
demo_light(light = function(face, col) {
             n <- length(col)
             switch(face,
               top = rep_len("grey90", n),
               west = rep_len("red", n),
               east = rep_len("green", n),
               south = rep_len("blue", n),
               north = rep_len("yellow", n))
           })
```

`geom_oblicubes` *Draw 2D/3D cubes with ggplot2*

Description

`geom_oblicubes()` creates a `ggplot2` geom that draws cubes.

Usage

```
geom_oblicubes(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  angle = 45,
  scale = 0.5,
  xoffset = 0,
  yoffset = 0,
  zoffset = 0,
  light = darken_face,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Aesthetics, used to set an aesthetic to a fixed value.

`angle` Oblique projection angle.

`scale` Oblique projection foreshortening factor. 0.5 corresponds to the "cabinet projection". 1.0 corresponds to the "cavalier projection". 0.0 corresponds to a "primary view orthographic projection".

`xoffset`, `yoffset`, `zoffset`
By default the x,y,z values are assumed to be the **center** of the cube. Use `xoffset`, `yoffset`, and/or `zoffset` to shift the x,y,z values a fixed amount.

`light` If `FALSE` don't perform a "light" effect. Otherwise a function that takes two arguments: the first `face` of the cube/cuboid face (one of "top", "west", "east", "south", "north"). the second `col` of the fill color. By default we use `darken_face()`.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

Details

`geom_oblicubes()` requires a fixed scale coordinate system with an aspect ratio of 1 as provided by `ggplot2::coord_fixed()`.

Value

A ggplot2 geom.

Aesthetics

`geom_oblicubes()` understands the following aesthetics (required aesthetics are in bold). See `oblicubesGrob()` for more details.

- `x`
- `y`
- `z`
- `fill`
- `colour`
- `linetype`
- `linewidth`

See Also

geom_oblicubes() is a wrapper around oblicubesGrob().

Examples

```
if (require("ggplot2")) {
  data("volcano", package = "datasets")
  df <- xyz_heightmap(volcano, scale = 0.3, min = 1)
  g <- ggplot(df, aes(x, y, z = z, fill = raw)) +
         geom_oblicubes(light = FALSE) +
         coord_fixed() +
         scale_fill_gradientn(name = "Height (m)",
                              colours=terrain.colors(256)) +
         labs(x = "East (10m)", y = "North (10m)",
              title = "Maungawhau (`datasets::volcano`)")
  plot(g)
}

if (require("ggplot2")) {
  # Using `scale_fill_identity()` if using `xyz_heightmap()`'s `fill` column
  df <- xyz_heightmap(volcano, scale = 0.3, min = 1,
                      col = grDevices::heat.colors)
  g <- ggplot(df, aes(x, y, z = z, fill = fill)) +
         geom_oblicubes() +
         coord_fixed() +
         scale_fill_identity()
  plot(g)
}

if (require("ggplot2") && require("dplyr")) {
  # Note you probably should not do 3D bar charts...
  df <- as.data.frame(datasets::Titanic) %>%
          filter(Age == "Child", Freq > 0) %>%
          group_by(Sex, Survived, Class) %>%
          summarize(Freq = seq.int(sum(Freq)), .groups = "drop")
  g <- ggplot(df, aes(x = Survived, y = Freq, fill = Survived)) +
      facet_grid(cols = vars(Class, Sex)) +
      coord_fixed() +
      geom_oblicubes(yoffset = -0.5, zoffset = -0.5, angle = -45, scale = 0.7) +
      scale_fill_manual(values = c("Yes" = "lightblue", "No" = "red")) +
      scale_y_continuous(expand = expansion(), name = "") +
      scale_x_discrete(name = "", breaks = NULL) +
      labs(title = "Children on the Titanic (by ticket class)")
  plot(g)
}
```

---

geom_oblicuboids *Draw 2D/3D cuboids with ggplot2*

Description

geom_oblicuboids() creates a ggplot2 geom that draws cuboids

Usage

```
geom_oblicuboids(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  angle = 45,
  scale = 0.5,
  xoffset = 0,
  yoffset = 0,
  zoffset = 0,
  light = darken_face,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Aesthetics, used to set an aesthetic to a fixed value.

`angle` Oblique projection angle.

`scale` Oblique projection foreshortening factor. 0.5 corresponds to the "cabinet projection". 1.0 corresponds to the "cavalier projection". 0.0 corresponds to a "primary view orthographic projection".

`xoffset`, `yoffset`, `zoffset`
By default the x,y values are assumed to be the **center** of the cuboid and the z value is assumed to be the **top** of the cuboid. Use `xoffset`, `yoffset`, and/or `zoffset` to shift the x,y,z values a fixed amount.

`light` If `FALSE` don't perform a "light" effect. Otherwise a function that takes two arguments: the first `face` of the cube/cuboid face (one of "top", "west", "east", "south", "north"). the second `col` of the fill color. By default we use `darken_face()`.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

Details

`geom_oblicuboids()` requires a fixed scale coordinate system with an aspect ratio of 1 as provided by `ggplot2::coord_fixed()`.

Value

A ggplot2 geom.

Aesthetics

`geom_oblicuboids()` understands the following aesthetics (required aesthetics are in bold). See `oblicuboidsGrob()` for more details.

- `x`
- `y`
- `z`

- fill
- colour
- linetype
- linewidth

See Also

geom_oblicuboids() is a wrapper around oblicuboidsGrob().

Examples

```
if (require("ggplot2")) {
  data("volcano", package = "datasets")
  df <- xyz_heightmap(volcano, scale = 0.3, min = 1)
  g <- ggplot(df, aes(x, y, z = z, fill = raw)) +
         geom_oblicuboids(light = FALSE) +
         coord_fixed() +
         scale_fill_gradientn(name = "Height (m)",
                              colours=terrain.colors(256)) +
         labs(x = "East (10m)", y = "North (10m)",
              title = "Maungawhau (`datasets::volcano`)")
  plot(g)
}
if (require("ggplot2")) {
  # Using `scale_fill_identity()` if using `xyz_heightmap()`'s `fill` column
  df <- xyz_heightmap(volcano, scale = 0.3, min = 1,
                      col = grDevices::heat.colors)
  g <- ggplot(df, aes(x, y, z = z, fill = fill)) +
         geom_oblicuboids() +
         coord_fixed() +
         scale_fill_identity()
  plot(g)
}
if (require("ggplot2") && require("dplyr")) {
  # Note you probably should not do 3D bar charts...
  df <- as.data.frame(datasets::Titanic) %>%
          filter(Age == "Child", Freq > 0) %>%
          group_by(Sex, Survived, Class) %>%
          summarize(Freq = seq.int(sum(Freq)), .groups = "drop")
  g <- ggplot(df, aes(x = Survived, y = Freq, fill = Survived)) +
      facet_grid(cols = vars(Class, Sex)) +
      coord_fixed() +
      geom_oblicuboids(yoffset = -0.5, scale = 0.7, angle = -45) +
      scale_fill_manual(values = c("Yes" = "lightblue", "No" = "red")) +
      scale_y_continuous(expand = expansion(), name = "") +
      scale_x_discrete(name = "", breaks = NULL) +
      labs(title = "Children on the Titanic (by ticket class)")
  plot(g)
}
```

oblicubesGrob *Render 2D/3D cubes via an oblique projection*

Description

`oblicubesGrob()` / `grid.oblicubes()` renders cubes using a 3D oblique projection. `oblicubesGrob()` returns a grid grob object while `grid.oblicubes()` also draws the grob to the graphic device. As a special case may also render a 2D primary view orthographic projection.

Usage

```
oblicubesGrob(
  x,
  y = NULL,
  z = NULL,
  ...,
  fill = NULL,
  light = darken_face,
  scale = 0.5,
  angle = 45,
  xo = NULL,
  yo = NULL,
  width = NULL,
  default.units = "snpc",
  name = NULL,
  gp = gpar(),
  vp = NULL
)

grid.oblicubes(
  x,
  y = NULL,
  z = NULL,
  ...,
  fill = NULL,
  light = darken_face,
  scale = 0.5,
  angle = 45,
  xo = NULL,
  yo = NULL,
  width = NULL,
  default.units = "snpc",
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

x	Integer vector of x coordinates (if necessary will be rounded to integers). May be a data.frame of x,y,z coordinates (and maybe fill color).
y	Integer vector of y coordinates (if necessary will be rounded to integers). If NULL and x is a data frame with a y column then we use that instead.
z	Integer vector of z coordinates (if necessary will be rounded to integers). If NULL and x is a data frame with a z column then we use that instead.
...	Passed to grid::gpar(). Will override any values set in gp.
fill	Fill color(s) for the cubes. If NULL and x is a data frame with a fill or col column then we use that column; if no such column but gp has a fill value we use that; otherwise we fall back to "grey90".
light	If FALSE don't perform a "light" effect. Otherwise a function that takes two arguments: the first face of the cube/cuboid face (one of "top", "west", "east", "south", "north"). the second col of the fill color. By default we use darken_face().
scale	Oblique projection foreshortening factor. 0.5 corresponds to the "cabinet projection". 1.0 corresponds to the "cavalier projection". 0.0 corresponds to a "primary view orthographic projection".
angle	Oblique projection angle.
xo, yo	The origin of the oblique projection coordinate system in grid units. The default is to try to guess a "good" value.
width	Width of the cube's (non-foreshortened) sides. The default will be to try to guess a "good" value.
default.units	Default units for the xo, yo, and width arguments.
name	A character identifier (for grid).
gp	A 'grid' gpar object. See grid::gpar(). Will be merged with the values in ... and the value of fill.
vp	A 'grid' viewport object. See grid::viewport().

Value

A grid grob. As a side effect grid.oblicubes() also draws to the active graphics device.

Examples

```
if (require("grid")) {
  # we support arbitrary oblique projection angles
  mat <- matrix(c(1, 2, 1, 2, 3, 2, 1, 2, 1), nrow = 3, ncol = 3, byrow = TRUE)
  coords <- xyz_heightmap(mat, col = c("red", "yellow", "green"))

  angles <- c(135, 90, 45, 180, 45, 0, -135, -90, -45)
  scales <- c(0.5, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.5)
  vp_x <- rep(1:3/3 - 1/6, 3)
  vp_y <- rep(3:1/3 - 1/6, each = 3)
  grid.newpage()
  for (i in 1:9) {
```

```
      pushViewport(viewport(x=vp_x[i], y=vp_y[i], width=1/3, height=1/3))
      grid.rect(gp = gpar(lty = "dashed"))
      grid.oblicubes(coords, width = 0.15, xo = 0.25, yo = 0.15,
                     angle = angles[i], scale = scales[i],
                     gp = gpar(lwd=4))
      if(i != 5)
          grid.text(paste("angle =", angles[i]), y=0.92, gp = gpar(cex = 1.2))
      else
          grid.text(paste("scale = 0"), y=0.92, gp = gpar(cex = 1.2))
      popViewport()
  }
}
# volcano example
mat <- datasets::volcano
mat <- 0.3 * (mat - min(mat)) + 1.0
coords <- xyz_heightmap(mat, col = grDevices::terrain.colors)
grid::grid.newpage()
grid.oblicubes(coords)
```

`oblicuboidsGrob` *Render 2D/3D cuboids via an oblique projection*

Description

`oblicuboidsGrob()` / `grid.oblicuboids()` renders cuboids using a 3D oblique projection. `oblicuboidsGrob()` returns a grid grob object while `grid.oblicuboids()` also draws the grob to the graphic device.
As a special case may also render a 2D primary view orthographic projection.

Usage

```
oblicuboidsGrob(
  x,
  y = NULL,
  z = NULL,
  ...,
  fill = NULL,
  light = darken_face,
  scale = 0.5,
  angle = 45,
  xo = NULL,
  yo = NULL,
  width = NULL,
  default.units = "snpc",
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

```
grid.oblicuboids(
  x,
  y = NULL,
  z = NULL,
  ...,
  fill = NULL,
  scale = 0.5,
  angle = 45,
  xo = NULL,
  yo = NULL,
  width = NULL,
  default.units = "snpc",
  name = NULL,
  gp = gpar(),
  vp = NULL
)
```

Arguments

x	Integer vector of x coordinates (if necessary will be rounded to integers). May be a `data.frame` of x,y,z coordinates (and maybe fill color). This will be the x-value at the *center* of the cuboid.
y	Integer vector of y coordinates (if necessary will be rounded to integers). If `NULL` and `x` is a data frame with a `y` column then we use that instead. This will be the x-value at the *center* of the cuboid.
z	Integer vector of z coordinates (if necessary will be rounded to integers). If `NULL` and `x` is a data frame with a `z` column then we use that instead. This will be the z-value at the *top* of the cuboid.
...	Passed to `grid::gpar()`. Will override any values set in gp.
fill	Fill color(s) for the cuboids. If `NULL` and `x` is a data frame with a `fill` or `col` column then we use that column; if no such column but gp has a `fill` value we use that; otherwise we fall back to "grey90".
light	If `FALSE` don't perform a "light" effect. Otherwise a function that takes two arguments: the first `face` of the cube/cuboid face (one of "top", "west", "east", "south", "north"). the second `col` of the fill color. By default we use `darken_face()`.
scale	Oblique projection foreshortening factor. 0.5 corresponds to the "cabinet projection". 1.0 corresponds to the "cavalier projection". 0.0 corresponds to a "primary view orthographic projection".
angle	Oblique projection angle.
xo, yo	The origin of the oblique projection coordinate system in grid units. The default is to try to guess a "good" value.
width	Width of the cuboids's (non-foreshortened) side. The default will be to try to guess a "good" value.
default.units	Default units for the `xo`, `yo`, and `width` arguments.
name	A character identifier (for grid).

gp	A 'grid' gpar object. See grid::gpar(). Will be merged with the values in ... and the value of fill.
vp	A 'grid' viewport object. See grid::viewport().

Value

A grid grob. As a side effect grid.oblicubes() also draws to the active graphics device.

Examples

```
if (require("grid")) {
  # we support arbitrary oblique projection angles
  mat <- matrix(c(1, 2, 1, 2, 3, 2, 1, 2, 1), nrow = 3, ncol = 3, byrow = TRUE)
  coords <- xyz_heightmap(mat, col = c("red", "yellow", "green"),
                          solid = FALSE)
  angles <- c(135, 90, 45, 180, 45, 0, -135, -90, -45)
  scales <- c(0.5, 0.5, 0.5, 0.5, 0.0, 0.5, 0.5, 0.5, 0.5)
  vp_x <- rep(1:3/3 - 1/6, 3)
  vp_y <- rep(3:1/3 - 1/6, each = 3)
  grid.newpage()
  for (i in 1:9) {
      pushViewport(viewport(x=vp_x[i], y=vp_y[i], width=1/3, height=1/3))
      grid.rect(gp = gpar(lty = "dashed"))
      grid.oblicuboids(coords, width = 0.15, xo = 0.25, yo = 0.15,
                       angle = angles[i], scale = scales[i],
                       gp = gpar(lwd=4))
      if(i != 5)
          grid.text(paste("angle =", angles[i]), y=0.92, gp = gpar(cex = 1.2))
      else
          grid.text(paste("scale = 0"), y=0.92, gp = gpar(cex = 1.2))
      popViewport()
  }
}
# volcano example
mat <- datasets::volcano
mat <- 0.3 * (mat - min(mat)) + 1.0
coords <- xyz_heightmap(mat, col = grDevices::terrain.colors,
                        solid = FALSE)
grid::grid.newpage()
grid.oblicuboids(coords)
```

---

xyz_heightmap *Calculate x,y,z coordinates from a height matrix*

Description

Calculate x,y,z coordinates from a height matrix

Usage

```
xyz_heightmap(
  mat,
  col = NULL,
  scale = 1,
  min = NULL,
  flipx = FALSE,
  flipy = TRUE,
  ground = "xy",
  solid = TRUE,
  verbose = FALSE
)
```

Arguments

mat	integer matrix. The matrix will be interpreted as cubes (or cuboids) flat on the page, with the value in the matrix interpreted as the height above the page.
col	matrix, vector, or (palette) function of colours. If a matrix it must be the same dimensions as the mat argument; each cube/cuboid corresponding to that x,y value will have that color. If a vector then if the max of z values is less than equal to the number of colors we will use the z integers as indices else we will use base::cut() to assign z values to colors. If a function we will call it with the argument max(z) to create a a vector of colors and then use the z values as indices. If col is not NULL then a fill column will be included in the final returned coordinates.
scale	scale factor for values in matrix. Default = 1
min	Minimum target z value. If NULL ignore else we "translate" the z-values so the minimum z-value is equal to this value.
flipx, flipy	Should the matrix be flipped in the horizontal/vertical directions (respectively)? Note: flipy defaults to TRUE as matrices are indexed from the top-down, but the coordinate space is increasing from the bottom up. Flipping the matrix vertically is usually what you want.
ground	Orientation of the ground plane. Default: "xy". Possible values "xy", "xz", "zy"
solid	Should the heightmap be made 'solid' i.e. without holes? This can be an expensive operation in terms of both memory and CPU, but should be OK for simple examples. Set to FALSE if things take too long or you will be rendering cuboids. This operation works by extruding cubes down from the top of the height map to the floor to ensure gaps do not appear when the slope is too great.
verbose	Be verbose? default: FALSE

Value

A data frame of x, y, z, raw, and possibly fill columns. The "raw" column is the (original) "z" column before any scale, min, and ground transformations have been performed (it may be repeated "down" if solid = TRUE). The "raw" column can be useful as the fill value in ggplot2 plots especially when adding a legend.

Examples

```
if (require("grDevices") && require("grid")) {
  mat <- datasets::volcano
  mat <- 0.3 * (mat - min(mat)) + 1.0

  grid.newpage()
  grid.rect(gp=gpar(col=NA, fill="grey5"))
  width <- convertWidth(unit(0.007, "snpc"), "cm")

  # Top view
  pushViewport(viewport(width = 0.7, height = 0.7, x = 0.65, y = 0.65))
  coords <- xyz_heightmap(mat, col = terrain.colors, solid = FALSE)
  grid.oblicubes(coords, scale = 0, width = width, gp = gpar(col=NA))
  popViewport()

  # South view
  pushViewport(viewport(width = 0.7, height = 0.3, x = 0.65, y = 0.15))
  coords <- xyz_heightmap(mat, col = terrain.colors, ground = "xz")
  grid.oblicubes(coords, scale = 0, width = width, gp = gpar(col=NA))
  popViewport()

  # West view
  pushViewport(viewport(width = 0.3, height = 0.7, x = 0.15, y = 0.65))
  coords <- xyz_heightmap(mat, col = terrain.colors, ground = "zy")
  grid.oblicubes(coords, scale = 0, width = width, gp = gpar(col=NA))
  popViewport()
}
if (require("grDevices") && require("ggplot2")) {
  data("volcano", package = "datasets")
  df <- xyz_heightmap(volcano, scale = 0.3, min = 1, solid = FALSE)
  g <- ggplot(df, aes(x, y, z = z, fill = raw)) +
         geom_oblicuboids(light = FALSE) +
         coord_fixed() +
         scale_fill_gradientn(name = "Height (m)", colours=terrain.colors(256)) +
       labs(x = "East (10m)", y = "North (10m)", title = "Maungawhau (`datasets::volcano`)")
  plot(g)
}
```

Index