Package ‘piecepackr’

June 17, 2022

Encoding UTF-8
Type Package
Title Board Game Graphics
Version 1.11.1
Description Functions to make board game graphics with the ‘ggplot2’, ‘grid’, ‘rayrender’, ‘rayvertex’, and ‘rgl’ packages. Specializes in game diagrams, animations, and “Print & Play” layouts for the ‘piecepack’ <https://www.ludism.org/ppwiki> but can make graphics for other board game systems. Includes configurations for several public domain game systems such as checkers, (double-18) dominoes, go, ‘piecepack’, playing cards, etc.
License MIT + file LICENSE
URL https://trevorldavis.com/piecepackr/ (blog),
    https://trevorldavis.com/R/piecepackr/ (pkgdown),
    https://groups.google.com/forum/#!forum/piecepackr (forum)
BugReports https://github.com/piecepackr/piecepackr/issues
LazyData true
LazyLoad yes
Imports grid, gridGeometry, grImport2, grDevices, purrr, jpeg, png,
    R6, rlang, stringr, tibble, tools, utils
Suggests animation (>= 2.7), ggplot2, gifski, gridpattern, magick,
    rayrender (>= 0.5.8), rayvertex (>= 0.3.3), readobj (>= 0.4.0),
    rgl (>= 0.106.8), scales (>= 0.5.0), systemfonts, testthat,
    tweenr, vdiffr, XML (>= 3.99-0.9)
SystemRequirements ghostscript
RoxygenNote 7.1.2
Config/testthat/edition 3
NeedsCompilation no
Author Trevor L Davis [aut, cre],
    Linux Foundation [dtc] (Uses some data from the “SPDX License List”
    <https://github.com/spdx/license-list-XML>),
    Delapouite <https://delapouite.com/> [ill] (Meeple shape extracted from
R topics documented:

``Meeple icon'' <https://game-icons.net/1x1/delapouite/meeple.html>
``CC BY 3.0'' <https://creativecommons.org/licenses/by/3.0/>
Creative Commons [ill] (`save_print_and_play()` uses `license badges`
from Creative Commons to describe the generated print-and-play
file's license)

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

Depends R (>= 2.10)

Repository CRAN

Date/Publication 2022-06-16 22:40:02 UTC

R topics documented:

piecepackr-package ................................................. 3
aabb_piece ........................................................... 3
AA_to_R ............................................................... 5
animate_piece ......................................................... 7
basicPieceGrobs ...................................................... 9
font_utils ............................................................ 10
game_systems ......................................................... 11
geom_piece ........................................................... 14
grid.cropmark ......................................................... 16
grid.piece ............................................................ 18
op_transform .......................................................... 22
piece ................................................................. 23
piece3d ............................................................... 25
piecepackr-defunct .................................................. 27
piece_mesh ........................................................... 28
pmap_piece ........................................................... 29
pp_cfg ............................................................... 31
pp_shape ............................................................. 34
pp_utils ............................................................. 37
render_piece ........................................................ 38
save_ellipsoid_obj .................................................. 40
save_piece_images ................................................... 42
save_piece_obj ....................................................... 42
save_print_and_play ............................................... 44
scale_x_piece ........................................................ 45
spdx_license_list ................................................... 47

Index 49
Description

Functions to make board game graphics. Specializes in game diagrams, animations, and "Print & Play" layouts for the 'piecepack' https://www.ludism.org/ppwiki but can make graphics for other board game systems. Includes configurations for several public domain game systems.

Package options

The following piecepackr function arguments may be set globally via base::options():

- **piecepack.at.inform** If FALSE turns off messages when affine transformation support not detected in active graphics device.
- **piecepackr.cfg** Sets a new default for the cfg argument
- **piecepackr.default.units** Sets a new default for the default.units argument
- **piecepackr.envir** Sets a new default for the envir argument
- **piecepackr.op_angle** Sets a new default for the op_angle argument
- **piecepackr.op_scale** Sets a new default for the op_scale argument
- **piecepackr.trans** Sets a new default for the trans argument

See Also

Useful links:

- blog: https://trevorldavis.com/piecepackr/
- pkgdown: https://trevorldavis.com/R/piecepackr/
- forum: https://groups.google.com/forum/#!forum/piecepackr
- Report bugs: https://github.com/piecepackr/piecepackr/issues

aabb_piece

Calculate axis-aligned bounding box for set of game pieces

Description

Calculate axis-aligned bounding box (AABB) for set of game pieces with and without an “oblique projection”.
Usage

aabb_piece(
  df,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  envir = getOption("piecepackr.envir"),
  op_scale = getOption("piecepackr.op_scale", 0),
  op_angle = getOption("piecepackr.op_angle", 45),
  ...
)

Arguments

df    A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.

cfg    Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().

envir    Environment (or named list) containing configuration list(s).

op_scale    How much to scale the depth of the piece in the oblique projection (viewed from the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5 is the most common scale used in the “cabinet” projection, and 1.0 is the scale used in the “cavalier” projection.

op_angle    What is the angle of the oblique projection? Has no effect if op_scale is 0.

...    Ignored

Details

The “oblique projection” of a set of \((x, y, z)\) points onto the xy-plane is \((x + \lambda \times z \times \cos(\alpha), y + \lambda \times z \times \sin(\alpha))\) where \(\lambda\) is the scale factor and \(\alpha\) is the angle.

Value

A named list of ranges with five named elements \(x\), \(y\), and \(z\) for the axis-aligned bounding cube in xyz-space plus \(x_{op}\) and \(y_{op}\) for the axis-aligned bounding box of the “oblique projection” onto the xy plane.

Examples

df_tiles <- data.frame(piece_side="tile_back",
  x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1),
  suit=NA, angle=NA, z=NA, stringsAsFactors=FALSE)
df_coins <- data.frame(piece_side="coin_back",
  x=rep(4:1, 4), y=rep(4:1, each=4),
  suit=1:16%%2+rep(c(1,3), each=8),
  angle=rep(c(180,0), each=8), z=1/4+1/16, stringsAsFactors=FALSE)
df <- rbind(df_tiles, df_coins)
aabb_piece(df, op_scale = 0)
aabb_piece(df, op_scale = 1, op_angle = 45)
aabb_piece(df, op_scale = 1, op_angle = -90)
## AA_to_R

### Helper functions for making geometric calculations.

#### Description

to_x, to_y, to_r, to_t convert between polar coordinates (in degrees) and Cartesian coordinates. to_degrees and to_radians converts between degrees and radians. AA_to_R and R_to_AA convert back and forth between (post-multiplied) rotation matrix and axis-angle representations of 3D rotations. \( R_x, R_y, \) and \( R_z \) build (post-multiplied) rotation matrices for simple rotations around the x, y, and z axes.

#### Usage

```r
AA_to_R(angle = 0, axis_x = 0, axis_y = 0, axis_z = NA, ...)  
R_to_AA(R = diag(3))  
R_x(angle = 0)  
R_y(angle = 0)  
R_z(angle = 0)  
to_radians(t)  
to_degrees(t)  
to_x(t, r)  
to_y(t, r)  
to_r(x, y)  
to_t(x, y)
```

#### Arguments

- **angle**: Angle in degrees (counter-clockwise)
- **axis_x**: First coordinate of the axis unit vector.
- **axis_y**: Second coordinate of the axis unit vector.
- **axis_z**: Third coordinate of the axis unit vector (usually inferred).
- **...**: Ignored
- **R**: 3D rotation matrix (post-multiplied)
- **t**: Angle in degrees (counter-clockwise)
- **r**: Radial distance
x Cartesian x coordinate
y Cartesian y coordinate

Details

ppCfg uses polar coordinates to determine where the "primary" and "directional" symbols are located on a game piece. They are also useful for drawing certain shapes and for making game diagrams on hex boards.

piecepackr and grid functions use angles in degrees but the base trigonometry functions usually use radians.

piecepackr’s 3D graphics functions save_piece_obj, piece, and piece3d use the axis-angle representation for 3D rotations. The axis-angle representation involves specifying a unit vector indicating the direction of an axis of rotation and an angle describing the (counter-clockwise) rotation around that axis. Because it is a unit vector one only needs to specify the first two elements, axis.x and axis.y, and we are able to infer the 3rd element axis.z. The default of axis = 0, axis.y = 0, and implied axis.z = 1 corresponds to a rotation around the z-axis which is reverse-compatible with the originally 2D angle interpretation in grid.piece. In order to figure out the appropriate axis-angle representation parameters R_to_AA, R.x, R.y, and R.z allow one to first come up with an appropriate (post-multiplied) 3D rotation matrix by chaining simple rotations and then convert them to the corresponding axis-angle representation. Pieces are rotated as if their center was at the origin.

See Also


Examples

to_x(90, 1)
to_y(180, 0.5)
to_t(0, -1)
to_r(0.5, 0)
all.equal(pi, to_radians(to_degrees(pi)))
# default axis-angle axis is equivalent to a rotation about the z-axis
all.equal(AA_to_R(angle=60), R_z(angle=60))
# axis-angle representation of 90 rotation about the x-axis
R_to_AA(R_x(90))
# find Axis-Angle representation of first rotating about x-axis 180 degrees
# and then rotating about z-axis 45 degrees
R_to_AA(R_x(180) %*% R_z(45))
animate_piece

Animate board game pieces

Description

animate_piece() animates board game pieces.

Usage

animate_piece(
  dfs,
  file = "animation.gif",
  annotate = TRUE,
  ...,
  .f = piecepackr::grid.piece,
  cfg = getOption("piecepackr.cfg", NULL),
  envir = getOption("piecepackr.envir", game_systems("sans")),
  n_transitions = 0L,
  n_pauses = 1L,
  fps = n_transitions + n_pauses,
  width = NULL,
  height = NULL,
  ppi = NULL,
  new_device = TRUE,
  annotation_scale = NULL
)

Arguments

dfs A list of data frames of game data to plot.
file Filename to save animation unless NULL in which case it uses the current graphics device.
annotate If TRUE or "algebraic" annotate the plot with "algebraic" coordinates, if FALSE or "none" don’t annotate, if "cartesian" annotate the plot with "cartesian" coordinates.
... Arguments to pmap_piece
.f Low level graphics function to use e.g. grid.piece(), piece3d(), piece(), or piece_mesh().
cfg A piecepackr configuration list
envir Environment (or named list) of piecepackr configuration lists
n_transitions Integer, if over zero (the default) how many transition frames to add between moves.
n_pauses Integer, how many paused frames per completed move.
fps Double, frames per second.
animate_piece

**width** Width of animation (in inches). Inferred by default.

**height** Height of animation (in inches). Inferred by default.

**ppi** Resolution of animation in pixels per inch. By default set so image max 600 pixels wide or tall.

**new_device** If file is NULL should we open up a new graphics device?

**annotation_scale** Multiplicative factor that scales (stretches) any annotation coordinates. By default uses attr(df, "scale_factor") %||% 1.

**Value**

Nothing, as a side effect creates an animation.

**Examples**

```r
# Basic tic-tac-toe animation
dfs <- list()
d.frame <- function(piece_side = "bit_back", ..., rank = 1L) {
  data.frame(piece_side = piece_side, ..., rank = rank,
  cfg = "checkers1", stringsAsFactors = FALSE)
}
df <- d.frame("board_back", suit = 2L, rank = 3L, x = 2, y = 2, id = "1")
dfs[[1L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 2, y = 2, id = "2"))
dfs[[2L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 1, y = 2, id = "3"))
dfs[[3L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 3, y = 1, id = "4"))
dfs[[4L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 1, y = 3, id = "5"))
dfs[[5L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 1, y = 1, id = "6"))
dfs[[6L]] <- df
df <- rbind(df, d.frame(suit = 2L, x = 3, y = 3, id = "7"))
dfs[[7L]] <- df
df <- rbind(df, d.frame(suit = 1L, x = 2, y = 1, id = "8"))
dfs[[8L]] <- df

## Press enter to walk through moves in a "game" in new graphics device
if (interactive()) {
  animate_piece(dfs, file = NULL)
}

## Save GIF of game with animation transitions
if ((require("animation") || require("gifski")) && require("tweenr")) {
  animate_piece(dfs, file = "tic-tac-toe.gif", n_transitions = 5L,
  n_pauses = 2L, fps = 9)
  unlink("tic-tac-toe.gif")
}
```
basicPieceGrobs

---

**Piece Grob Functions**

**Description**

`basicPieceGrob`, `pyramidTopGrob`, and `previewLayoutGrob` are the default “grob” functions that `grid.piece` uses to create grid graphical grob objects. `picturePieceGrobFn` is a function that returns a “grob” function that imports graphics from files found in its directory argument.

**Usage**

```r
basicPieceGrob(piece_side, suit, rank, cfg = pp_cfg())
picturePieceGrobFn(directory, filename_fn = find_pp_file)
pyramidTopGrob(piece_side, suit, rank, cfg = pp_cfg())
previewLayoutGrob(piece_side, suit, rank, cfg = pp_cfg())
```

**Arguments**

- `piece_side`: A string with piece and side separated by a underscore e.g. "coin_face"
- `suit`: Number of suit (starting from 1).
- `rank`: Number of rank (starting from 1)
- `cfg`: Piecepack configuration list or `pp_cfg` object.
- `directory`: Directory that `picturePieceGrobFn` will look in for piece graphics.
- `filename_fn`: Function that takes arguments `directory`, `piece_side`, `suit`, `rank`, and optionally `cfg` and returns the (full path) filename of the image that the function returned by `picturePieceGrobFn` should import.

**Examples**

```r
if (require("grid") && all(capabilities(c("cairo", "png")))) {
  op <- options()
on.exit(options(op))
options(piecepackr.at.inform = FALSE)
cfg <- pp_cfg(list(invert_colors=TRUE))

  pushViewport(viewport(width=unit(2, "in"), height=unit(2, "in")))
grid.draw(basicPieceGrob("tile_face", suit=1, rank=3))
popViewport()

  grid.newpage()
pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
grid.draw(basicPieceGrob("coin_back", suit=2, rank=0, cfg=cfg))
popViewport()
}
font_utils

Font utility functions

Description

get_embedded_font() returns which font is actually embedded by cairo_pdf() for a given character. has_font() tries to determine if a given font is available on the OS.

Usage

get_embedded_font(font, char)

has_font(font)

Arguments

font A character vector of font(s).

char A character vector of character(s) to be embedded by grid::grid.text()

Details

get_embedded_font() depends on pdffonts being on the system path (on many OSes found in a poppler-utils package).
game_systems

Value

get_embedded_font() returns character vector of fonts that were actually embedded by cairo_pdf(). NA's means no embedded font detected: this either means that no font was found or that a color emoji font was found and instead of a font an image was embedded.

Examples

```r
if ((Sys.which("pdffonts") != ")") && capabilities("cairo"){
  chars <- c("a", ♦")
  fonts <- c("sans", "Sans Noto", "Noto Sans", "Noto Sans Symbols2")
  get_embedded_font(fonts, chars)

  has_font("Dejavu Sans")
}
```

game_systems  Standard game systems

Description

game_systems returns a list of pp_cfg objects representing several game systems and pieces. to_subpack and to_hexpack will attempt to generate matching (piecepack stackpack) subpack and (piecepack) hexpack pp_cfg R6 objects respectively given a piecepack configuration.

Usage

game_systems(style = NULL, round = FALSE, pawn = "token")

to_hexpack(cfg = getOption("piecepackr.cfg", pp_cfg()))

to_subpack(cfg = getOption("piecepackr.cfg", pp_cfg()))

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>If NULL (the default) uses suit glyphs from the default &quot;sans&quot; font. If &quot;dejavu&quot; it will use suit glyphs from the &quot;DejaVu Sans&quot; font (must be installed on the system).</td>
</tr>
<tr>
<td>round</td>
<td>If TRUE the “shape” of “tiles” and “cards” will be “roundrect” instead of “rect” (the default).</td>
</tr>
<tr>
<td>pawn</td>
<td>If &quot;token&quot; (default) the piecepack pawn will be a two-sided token in a “halma” outline, if &quot;peg-doll&quot; the piecepack pawn will be a “peg doll” style pawn, and if &quot;joystick&quot; the piecepack pawn will be a “joystick” style pawn. Note for the latter two pawn styles only pawn_top will work with grid.piece.</td>
</tr>
<tr>
<td>cfg</td>
<td>List of configuration options</td>
</tr>
</tbody>
</table>
game_systems

Details

Contains the following game systems:

- **alquerque**  Boards and pieces in six color schemes for Alquerque
- **checkers1, checkers2**  Checkers and checkered boards in six color schemes. Checkers are represented by a piecepackr “bit”. The “board” “face” is a checkered board and the “back” is a lined board. Color is controlled by suit and number of rows/columns by rank. checkers1 has one inch squares and checkers2 has two inch squares.
- **chess1, chess2**  Chess pieces and checkered boards in six color schemes. Chess pieces are represented by a “bit” (face). The “board” “face” is a checkered board and the “back” is a lined board. Color is controlled by suit and number of rows/columns by rank. chess1 has one inch squares and chess2 has two inch squares.
- **dice**  Traditional six-sided pipped dice in six color schemes (color controlled by their suit).
- **dice_fudge**  “Fudge” dice in six color schemes (color controlled by their suit). “Fudge” dice have three ranks “+”, “ ”, and “−” repeated twice.
- **dominoes, dominoes_black, dominoes_blue, dominoes_green, dominoes_red, dominoes_white, dominoes_yellow**  Traditional pipped dominoes in six color schemes (dominoes and dominoes_white are the same). In each color scheme the number of pips on the “top” of the domino is controlled by their “rank” and on the “bottom” by their “suit”. Supports up to double-18 sets.
- **go**  Go stones and lined boards in six color schemes. Go stones are represented by a “bit” and the board is a “board”. Color is controlled by suit and number of rows/columns by rank Currently the "stones" look like "checkers" which is okay for 2D diagrams but perhaps unsatisfactory for 3D diagrams.
- **meeples**  Standard 16mm x 16mm x 10mm “meeples” in six colors represented by a “bit”.
- **morris**  Various morris aka mills aka merels games in six colors. Color is controlled by suit and “size” of morris board is controlled by rank e.g. “Six men’s morris” corresponds to a rank of 6 and “Nine men’s morris” corresponds to a rank of 9. Game pieces are represented by stones.
- **piecepack, dual_piecepacks_expansion, playing_cards_expansion, hexpack, subpack, piecepack_inverted**  The piecepack is a public domain game system invented by James "Kyle" Droscha. See https://www.ludism.org/ppwiki for more info about the piecepack and its accessories/expansions.
- **piecepack**  A standard piecepack. The configuration also contains the following piecepack accessories:
  - **piecepack dice cards**  An accessory proposed by John Braley. See https://www.ludism.org/ppwiki/PiecepackDiceCards.
  - **piecepack matchsticks**  A public domain accessory developed by Dan Burkey. See https://www.ludism.org/ppwiki/PiecepackMatchsticks.
  - **piecepack pyramids**  A public domain accessory developed by Tim Schutz. See https://www.ludism.org/ppwiki/PiecepackPyramids.
- **piecepack_inverted**  The standard piecepack with its color scheme inverted. Intended to aid in highlighting special pieces in diagrams.
- **dual_piecepacks_expansion**  A companion piecepack with a special suit scheme. See https://trevorldavis.com/piecepackr/dual-piecepacks-pnp.html.
playing_cards_expansion  A piecepack with the standard dQuoteFrench playing card suits. See https://www.ludism.org/ppwiki/PlayingCardsExpansion.

dhexpack  A hexagonal extrapolation of the piecepack designed by Nathan Morse and Daniel Wilcox. See https://boardgamegeek.com/boardgameexpansion/35424/hexpack.

subpack  A mini piecepack. Designed to be used with the piecepack to make piecepack “stackpack” diagrams. See https://www.ludism.org/ppwiki/StackPack.

playing_cards, playing_cards_colored, playing_cards_tarot  Poker-sized card components for various playing card decks:

playing_cards  A traditional deck of playing cards with 4 suits and 13 ranks (A, 2-10, J, Q, K) plus a 14th “Joker” rank.

playing_cards_colored  Like playing_cards but with five colored suits: red hearts, black spades, green clubs, blue diamonds, and yellow stars.

playing_cards_tarot  A (French Bourgeois) deck of tarot playing cards: first four suits are hearts, spades, clubs, and diamonds with 14 ranks (ace through jack, knight, queen, king) plus a 15th “Joker” rank and a fifth “suit” of 22 trump cards (1-21 plus an “excuse”).

reversi  Boards and pieces for Reversi. "board_face" provides lined boards with colored backgrounds. "board_back" provides checkered boards. "bit_face" / "bit_back" provides circular game tokens with differently colored sides: red paired with green, black paired with white, and blue paired with yellow.

See Also

pp_cfg  for information about the pp_cfg objects returned by game_systems.

Examples

cfgs <- game_systems()
names(cfgs)

if (require("grid") ) {
  op <- options()
on.exit(options(op))
options(piecepackr.at.inform = FALSE)

  # standard dice
  grid.newpage()
  grid.piece("die_face", x=1:6, default.units="in", rank=1:6, suit=1:6,
             op_scale=0.5, cfg=cfgs$dice)

  # dominoes
  grid.newpage()
  colors <- c("black", "red", "green", "blue", "yellow", "white")
  cfg <- paste0("dominoes_", rep(colors, 2))
  grid.piece("tile_face", x=rep(4:1, 3), y=rep(2*3:1, each=4), suit=1:12, rank=1:12+1,
             cfg=cfg, default.units="in", envir=cfgs, op_scale=0.5)

  # various piecepack expansions
  grid.newpage()
  df_tiles <- data.frame(piece_side="tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1),
```r
df_coins <- data.frame(piece_side="coin_back", x=rep(4:1, 4), y=rep(4:1, each=4), suit=c(1,4,1,4,1,4,1,2,3,2,3,3,2,3,2), angle=rep(c(180,0), each=8), z=1/4+1/16, stringsAsFactors=FALSE)
df <- rbind(df_tiles, df_coins)
pmap_piece(df, cfg = cfgs$playing_cards_expansion, op_scale=0.5, default.units="in")
grid.newpage()
pmap_piece(df, cfg = cfgs$dual_piecepacks_expansion, op_scale=0.5, default.units="in")
```

---

**geom_piece**

*Draw board game pieces with ggplot2*

**Description**

`geom_piece()` creates a `ggplot2` geom. `aes_piece()` takes a data frame and generates an appropriate `ggplot2::aes()` mapping.

**Usage**

```r
geom_piece(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  envir = getOption("piecepackr.envir", piecepackr::game_systems()),
  op_scale = getOption("piecepackr.op_scale", 0),
  op_angle = getOption("piecepackr.op_angle", 45),
  inherit.aes = TRUE
)
```

**Arguments**

- **mapping**
  - Set of aesthetic mappings created by `aes()` or `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, as a string.

**position**  
Position adjustment, either as a string, or the result of a call to a position adjustment function.

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

**envir**  
Environment (or named list) containing configuration list(s).

**op_scale**  
How much to scale the depth of the piece in the oblique projection (viewed from the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5 is the most common scale used in the “cabinet” projection, and 1.0 is the scale used in the “cavalier” projection.

**op_angle**  
What is the angle of the oblique projection? Has no effect if op_scale is 0.

**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().

**df**  
A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.

**Details**

tgeom_piece() requires a fixed scale coordinate system with an aspect ratio of 1 as provided by ggplot2::coord_fixed(). geom_piece() also requires that cfg is a character vector (and not a pp_cfg() object). In particular if using op_transform() one should set its argument cfg_class = “character” if intending for use with geom_piece().

**Aesthetics**

tgeom_piece() understands the following aesthetics (required aesthetics are in bold). See pieceGrob() for more details.

- x
- y
- z
- piece_side
- rank
- suit
- cfg
- width
- height
- depth
- angle
- scale
- type
See Also

`geom_piece()` is a wrapper around `pieceGrob()`. `scale_x_piece()` and `scale_y_piece()` are wrappers around `ggplot2::scale_x_continuous()` and `ggplot2::scale_y_continuous()` with better defaults for board game diagrams.

Examples

```r
if (require("ggplot2") && require("tibble")) {
  envir <- game_systems("sans")
  df_board <- tibble(piece_side = "board_face", suit = 3, rank = 8,
                     x = 4.5, y = 4.5)
  df_w <- tibble(piece_side = "bit_face", suit = 6, rank = 1,
                  x = rep(1:8, 2), y = rep(1:2, each=8))
  df_b <- tibble(piece_side = "bit_face", suit = 1, rank = 1,
                  x = rep(1:8, 2), y = rep(7:8, each=8))
  df <- rbind(df_board, df_w, df_b)
  # 2D example
  # 'cfg' must be a character vector for 'geom_piece'
  ggplot(df, aes_piece(df)) +
    geom_piece(cfg = "checkers1", envir = envir) +
    coord_fixed() +
    scale_x_piece() +
    scale_y_piece() +
    theme_minimal(28) +
    theme(panel.grid = element_blank())

  # 3D "oblique" projection example
  # 'cfg_class' must be "character" when using with 'geom_piece'
  df3d <- op_transform(df, cfg = "checkers1", envir = envir,
                       op_angle = 45, cfg_class = "character")
  ggplot(df3d, aes_piece(df3d)) +
    geom_piece(cfg = "checkers1", envir = envir,
               op_angle = 45, op_scale = 0.5) +
    coord_fixed() +
    theme_void()
}
```

---

grid.cropmark  

**Crop Mark Grob**

**Description**

`grid.cropmark()` draws “crop marks” to the active graphics device. `cropmarkGrob()` is its grid grob counterpart. Intended for use in adding crop marks around game pieces in print-and-play layouts.
grid.cropmark

Usage

cropmarkGrob(
  ..., 
  piece_side = "tile_back", 
  suit = NA, 
  rank = NA, 
  cfg =getOption("piecepackr.cfg", pp_cfg()), 
  x = unit(0.5, "npc"), 
  y = unit(0.5, "npc"), 
  angle = 0, 
  width = NA, 
  height = NA, 
  scale = 1, 
  default.units = "npc", 
  envir =getOption("piecepackr.envir"), 
  name = NULL, 
  gp = NULL, 
  vp = NULL, 
  bleed = unit(0.125, "in"), 
  cm_select = "12345678", 
  cm_width = unit(0.25, "mm"), 
  cm_length = unit(0.125, "in")
)

grid.cropmark(..., draw = TRUE)

Arguments

... cropmarkGrob() ignores; grid.cropmark() passes to cropmarkGrob().

piece_side A string with piece and side separated by a underscore e.g. "coin_face"

suit Number of suit (starting from 1).

rank Number of rank (starting from 1)

cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a
      character vector referring to names in envir or a character vector referring to 
      object names that can be retrieved by base::dynGet().

x Where to place piece on x axis of viewport

y Where to place piece on y axis of viewport

angle Angle (on xy plane) to draw piece at

width Width of piece

height Height of piece

scale Multiplicative scaling factor to apply to width, height, and depth.

default.units A string indicating the default units to use if ’x’, ’y’, ’width’, and/or ’height’ are
               only given as numeric vectors.

envir Environment (or named list) containing configuration list(s).
**name**
A character identifier (for grid)

**gp**
An object of class “gpar”.

**vp**
A grid viewport object (or NULL).

**bleed**
Bleed zone size to assume:
- If bleed is a `grid::unit()` simply use it
- If bleed is numeric then convert via `grid::unit(bleed, default.units)`
- If bleed is TRUE assume 1/8 inch bleed zone size
- If bleed is FALSE assume 0 inch bleed zone size

**cm_select**
A string of integers from "1" to "8" indicating which crop marks to draw. "1" represents the top right crop mark then we proceeding clockwise to "8" which represents the top left crop mark. Default "12345678" draws all eight crop marks.

**cm_width**
Width of crop mark.

**cm_length**
Length of crop mark.

**draw**
A logical value indicating whether graphics output should be produced.

---

### Examples

```r
if (require("grid")) {
  cfg <- pp_cfg(list(mat_color = "pink", mat_width=0.05, border_color=NA))
  grid.newpage()
  df <- data.frame(piece_side = "tile_face", suit = 2, rank = 2,
                   x = 2, y = 2, angle = 0,
                   stringsAsFactors = FALSE)
  pmap_piece(df, grid.cropmark, cfg = cfg, default.units = "in")
  pmap_piece(df, grid.piece, cfg = cfg, default.units = "in", bleed=TRUE)

  grid.newpage()
  df <- data.frame(piece_side = "coin_back", suit = 2, rank = 2,
                   x = 2, y = 2, angle = 0,
                   stringsAsFactors = FALSE)
  pmap_piece(df, grid.cropmark, cfg = cfg, default.units = "in", bleed=TRUE)
  pmap_piece(df, grid.piece, cfg = cfg, default.units = "in", bleed=TRUE)
}
```

---

**grid.piece**

*Draw board game pieces with grid*

---

### Description

`grid.piece()` draws board game pieces onto the graphics device. `pieceGrob()` is its grid “grob” counterpart.
Usage

```r
pieceGrob(
    piece_side = "tile_back",
    suit = NA,
    rank = NA,
    cfg = getOption("piecepackr.cfg", pp_cfg()),
    x = unit(0.5, "npc"),
    y = unit(0.5, "npc"),
    z = NA,
    angle = 0,
    use_pictureGrob = FALSE,
    width = NA,
    height = NA,
    depth = NA,
    op_scale = getOption("piecepackr.op_scale", 0),
    op_angle = getOption("piecepackr.op_angle", 45),
    default.units = getOption("piecepackr.default.units", "npc"),
    envir = getOption("piecepackr.envir"),
    name = NULL,
    gp = NULL,
    vp = NULL,
    ...
    scale = 1,
    alpha = 1,
    type = "normal",
    bleed = FALSE
)
```

```r
grid.piece(
    piece_side = "tile_back",
    suit = NA,
    rank = NA,
    cfg = getOption("piecepackr.cfg", pp_cfg()),
    x = unit(0.5, "npc"),
    y = unit(0.5, "npc"),
    z = NA,
    angle = 0,
    use_pictureGrob = FALSE,
    width = NA,
    height = NA,
    depth = NA,
    op_scale = getOption("piecepackr.op_scale", 0),
    op_angle = getOption("piecepackr.op_angle", 45),
    default.units = getOption("piecepackr.default.units", "npc"),
    envir = getOption("piecepackr.envir"),
    name = NULL,
    gp = NULL,
    draw = TRUE,
) 
```
vp = NULL,
..., 
scale = 1, 
alpha = 1, 
type = "normal", 
bleed = FALSE 
)

Arguments

piece_side A string with piece and side separated by a underscore e.g. "coin_face"
suit Number of suit (starting from 1).
rank Number of rank (starting from 1)
cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base:::dynGet().
x Where to place piece on x axis of viewport
y Where to place piece on y axis of viewport
z z-coordinate of the piece. Has no effect if op_scale is 0.
angle Angle (on xy plane) to draw piece at
use_pictureGrob Deprecated argument. If TRUE sets type argument to "picture".
width Width of piece
height Height of piece
depth Depth (thickness) of piece. Has no effect if op_scale is 0.

op_scale How much to scale the depth of the piece in the oblique projection (viewed from the top of the board). 0 (the default) leads to an “orthographic” projection, 0.5 is the most common scale used in the “cabinet” projection, and 1.0 is the scale used in the “cavalier” projection.

op_angle What is the angle of the oblique projection? Has no effect if op_scale is 0.
default.units A string indicating the default units to use if ‘x’, ‘y’, ‘width’, and/or ‘height’ are only given as numeric vectors.

envir Environment (or named list) containing configuration list(s).
name A character identifier (for grid)
gp An object of class “gpar”.
vp A grid viewport object (or NULL).
... Ignored.
scale Multiplicative scaling factor to apply to width, height, and depth.
alpha Alpha channel for transparency.
Type of grid grob to use. Either "normal" (default), "picture", "raster", or "transformation". "picture" exports to (temporary) svg and re-imports as a grImport2::pictureGrob. "raster" exports to (temporary) png and re-imports as a grid::rasterGrob. "transformation" uses the affine transformation feature only supported in R 4.2+ within select graphic devices. The latter three can be useful if drawing pieces really big or small and don’t want to mess with re-configuring font sizes and linewdths.

If FALSE do not add a “bleed” zone around the piece, otherwise add a “bleed” zone around the piece:

- If bleed is TRUE we will add 1/8 inch bleeds
- If bleed is a grid::unit() we will use it as bleed size
- If bleed is numeric we will convert to grid::unit() via grid::unit(bleed, default.units)

A non-FALSE bleed is incompatible with op_scale > 0 (drawing in an “oblique projection”).

A logical value indicating whether graphics output should be produced.

A grid grob object. If draw is TRUE then as a side effect grid.piece() will also draw it to the graphics device.

See Also

pmap_piece() which applies pieceGrob() over rows of a data frame.

Examples

if (require("grid")) {
  op <- options()
  on.exit(options(op))
  options(piecepackr.at.inform = FALSE)

draw_pp_diagram <- function(cfg=pp_cfg(), op_scale=0) {
  g.p <- function(...) {
    grid.piece(..., op_scale=op_scale, cfg=cfg, default.units="in")
  }
  g.p("tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1))
  g.p("tile_back", x=0.5+3, y=0.5+1, z=1/4+1/8)
  g.p("tile_back", x=0.5+3, y=0.5+1, z=2/4+1/8)
  g.p("die_face", suit=3, rank=5, x=1, y=1, z=1/4+1/4)
  g.p("pawn_face", x=1, y=4, z=1/4+1/2, angle=90)
  g.p("coin_back", x=3, y=4, z=1/4+1/16, angle=180)
  g.p("coin_back", suit=4, x=3, y=4, z=1/4+1/8+1/16, angle=180)
  g.p("coin_back", suit=2, x=3, y=1, z=3/4+1/8, angle=90)
}

# default piecepack, orthogonal projection
draw_pp_diagram(cfg=pp_cfg())
# custom configuration, orthogonal projection
grid.newpage()
dark_colorscheme <- list(suit_color="darkred,black,darkgreen,darkblue,black",
                      invert_colors.suited=TRUE, border_color="black", border_lex=2)
traditional_ranks <- list(use_suit_as_ace=TRUE, rank_text="a,2,3,4,5")
cfg <- c(dark_colorscheme, traditional_ranks)
draw_pp_diagram(cfg=pp_cfg(cfg))

# custom configuration, oblique projection
grid.newpage()
cfg3d <- list(width.pawn=0.75, height.pawn=0.75, depth.pawn=1,
               dm_text.pawn="", shape.pawn="convex6", invert_colors.pawn=TRUE,
               edge_color.coin="tan", edge_color.tile="tan")
cfg <- pp_cfg(c(cfg, cfg3d))
draw_pp_diagram(cfg=pp_cfg(cfg), op_scale=0.5)

---

op_transform

Oblique projection helper function

Description

Guesses z coordinates and sorting order to more easily make 3D graphics with pmap_piece.

Usage

op_transform(
  df,
  ...,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  envir = getOption("piecepackr.envir"),
  op_angle = getOption("piecepackr.op_angle", 45),
  pt_thickness = 0.01,
  as_top = character(0),
  cfg_class = "list"
)

Arguments

df A data frame with coordinates and dimensions in inches
...
Ignored
cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector of pp_cfg objects
envir Environment (or named list) containing configuration list(s).
op_angle Intended oblique projection angle (used for re-sorting)
\textbf{pt\_thickness} \hspace{1cm} Thickness of pyramid tip i.e. value to add to the \(z\)-value of a pyramid top if it is a (weakly) smaller ranked pyramid (top) placed on top of a larger ranked pyramid (top).

\textbf{as\_top} \hspace{1cm} Character vector of components whose “side” should be converted to “top” e.g. c("pawn\_face").

\textbf{cfg\_class} \hspace{1cm} Either "list" (default) or "character". Desired class of the cfg column in the returned tibble. "list" is more efficient for use with \texttt{pmap\_piece()} but \texttt{geom\_piece()} needs "character".

\textbf{Details}

The heuristics used to generate guesses for \(z\) coordinates and sorting order aren’t guaranteed to work in every case. In some cases you may get better sorting results by changing the \texttt{op\_angle} or the dimensions of pieces.

\textbf{Value}

A tibble with extra columns added and re-sorted rows

\textbf{See Also}


\textbf{Examples}

\begin{verbatim}
df <- tibble::tibble(piece_side="tile\_back",
x=c(2,2,2,4,6,6,4,2,5),
y=c(4,4,4,4,4,2,2,2,3))
pmap\_piece(df, op\_angle=135, trans=op\_transform,
op\_scale=0.5, default.units="in")
\end{verbatim}

---

\textbf{piece  \hspace{1cm} Create rayrender board game piece objects}

\textbf{Description}

piece creates 3d board game piece objects for use with the rayrender package.

\textbf{Usage}

\begin{verbatim}
piece(
piece\_side = "tile\_back",
suit = NA,
rank = NA,
cfg = getOption("piecepackr.cfg", pp\_cfg()),
x = 0,
\end{verbatim}
y = 0, 
z = NA, 
angle = 0, 
axis_x = 0, 
axis_y = 0, 
width = NA, 
height = NA, 
depth = NA, 
envir = getOption("piecepackr.envir"), 
..., 
scale = 1, 
res = 72 
)

Arguments

- **piece_side**: A string with piece and side separated by a underscore e.g. "coin_face"
- **suit**: Number of suit (starting from 1).
- **rank**: Number of rank (starting from 1)
- **cfg**: Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- **x**: Where to place piece on x axis of viewport
- **y**: Where to place piece on y axis of viewport
- **z**: z-coordinate of the piece. Has no effect if op_scale is 0.
- **angle**: Angle (on xy plane) to draw piece at
- **axis_x**: First coordinate of the axis unit vector.
- **axis_y**: Second coordinate of the axis unit vector.
- **width**: Width of piece
- **height**: Height of piece
- **depth**: Depth (thickness) of piece. Has no effect if op_scale is 0.
- **envir**: Environment (or named list) containing configuration list(s).
- **...**: Ignored.
- **scale**: Multiplicative scaling factor to apply to width, height, and depth.
- **res**: Resolution of the faces.

Value

A rayrender object.

See Also

See [https://www.rayrender.net](https://www.rayrender.net) for more information about the rayrender package. See `geometry_utils` for a discussion of the 3D rotation parameterization.
Examples

```r
if (require("rayrender") && all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$piecepack
  render_scene(piece("tile_face", suit = 3, rank = 3, cfg = cfg))
  render_scene(piece("coin_back", suit = 4, rank = 2, cfg = cfg))
  render_scene(piece("pawn_face", suit = 2, cfg = cfg))
}
```

**Description**

`piece3d` draws board games pieces using the `rgl` package.

**Usage**

```r
piece3d(
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  x = 0,
  y = 0,
  z = NA,
  angle = 0,
  axis_x = 0,
  axis_y = 0,
  width = NA,
  height = NA,
  depth = NA,
  envir = getOption("piecepackr.envir"),
  ...,
  scale = 1,
  res = 72,
  alpha = 1,
  lit = FALSE,
  shininess = 50,
  textype = NA
)
```

**Arguments**

- `piece_side`: A string with piece and side separated by a underscore e.g. "coin_face"
- `suit`: Number of suit (starting from 1).
rank
Number of rank (starting from 1)

cfg
Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a
character vector referring to names in envir or a character vector referring to
object names that can be retrieved by base:::dynGet()

x
Where to place piece on x axis of viewport

y
Where to place piece on y axis of viewport

z
z-coordinate of the piece. Has no effect if op_scale is 0.

angle
Angle (on xy plane) to draw piece at

axis_x
First coordinate of the axis unit vector.

axis_y
Second coordinate of the axis unit vector.

width
Width of piece

height
Height of piece

depth
Depth (thickness) of piece. Has no effect if op_scale is 0.

envir
Environment (or named list) containing configuration list(s).

... Ignored.

scale
Multiplicative scaling factor to apply to width, height, and depth.

res
Resolution of the faces.

alpha
Alpha channel for transparency.

lit
logical, specifying if rgl lighting calculation should take place.

shininess
Properties for rgl lighting calculation.

textype
Use "rgba" when sure texture will have alpha transparency. Use "rgb" when
sure texture will not have alpha transparency (in particular rgl's WebGL export
will likely work better). If NA we will read the texture and figure out a reasonable
value.

Value
A numeric vector of rgl object IDs.

See Also
See rgl-package for more information about the rgl package. See rgl::material3d() for more
info about setting rgl material properties. See geometry_utils for a discussion of the 3D rotation
parameterization.

Examples
if (require("rgl") & all(capabilities(c("cairo", "png")))) {
  open3d()
  cfg <- game_systems("sans3d")$piecepack
  piece3d("tile_back", suit = 3, rank = 3, cfg = cfg, x = 0, y = 0, z = 0)
  piece3d("coin_back", suit = 4, rank = 2, cfg = cfg, x = 0.5, y = 0.5, z = 0.25)
  piece3d("pawn_top", suit = 1, cfg = cfg, x = -0.5, y = 0.5, z = 0.6)
  piece3d("die_face", suit = 3, cfg = cfg, x = -0.5, y = -0.5, z = 0.375)
  piece3d("pyramid_top", suit = 2, rank = 3, cfg = cfg, x = 1.5, y = 0.0, z = 0.)
}
**Defunct functions**

These functions are Defunct and have been removed from piecepackr.

**Usage**

- `halmaGrob(...)`
- `kiteGrob(...)`
- `pyramidGrob(...)`
- `convexGrobFn(...)`
- `concaveGrobFn(...)`
- `gridlinesGrob(...)`
- `matGrob(...)`
- `checkersGrob(...)`
- `hexlinesGrob(...)`
- `get_shape_grob_fn(...)`

**Arguments**

... Ignored

**Details**

1. For `get_shape_grob_fn` use `pp_shape()$shape` instead.
2. For `gridlinesGrob()` use `pp_shape()$gridlines()` instead.
3. For `matGrob()` use `pp_shape()$mat()` instead.
4. For `checkersGrob()` use `pp_shape()$checkers()` instead.
5. For `hexlinesGrob()` use `pp_shape()$hexlines()` instead.
6. For `halmaGrob()` use `pp_shape("halma")$shape()` instead.
7. For `kiteGrob()` use `pp_shape("kite")$shape()` instead.
8. For `pyramidGrob()` use `pp_shape("pyramid")$shape()` instead.
9. For `convexGrobFn(n, t)` use `pp_shape(paste0("convex", n), t)$shape` instead.
10. For `concaveGrobFn(n, t, r)` use `pp_shape(paste0("concave", n), t, r)$shape` instead.
piece_mesh

Create rayvertex board game piece objects

Description

piece_mesh() creates 3d board game piece objects for use with the rayvertex package.

Usage

piece_mesh(
  piece_side = "tile_back",
  suit = NA,
  rank = NA,
  cfg = pp_cfg(),
  x = 0,
  y = 0,
  z = NA,
  angle = 0,
  axis_x = 0,
  axis_y = 0,
  width = NA,
  height = NA,
  depth = NA,
  envir = NULL,
  ...
  scale = 1,
  res = 72
)

Arguments

- **piece_side**: A string with piece and side separated by a underscore e.g. "coin_face"
- **suit**: Number of suit (starting from 1).
- **rank**: Number of rank (starting from 1)
- **cfg**: Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- **x**: Where to place piece on x axis of viewport
- **y**: Where to place piece on y axis of viewport
- **z**: z-coordinate of the piece. Has no effect if op_scale is 0.
- **angle**: Angle (on xy plane) to draw piece at
- **axis_x**: First coordinate of the axis unit vector.
- **axis_y**: Second coordinate of the axis unit vector.
- **width**: Width of piece
pmap_piece

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>height</td>
<td>Height of piece</td>
</tr>
<tr>
<td>depth</td>
<td>Depth (thickness) of piece. Has no effect if op_scale is 0.</td>
</tr>
<tr>
<td>envir</td>
<td>Environment (or named list) containing configuration list(s).</td>
</tr>
<tr>
<td>...</td>
<td>Ignored.</td>
</tr>
<tr>
<td>scale</td>
<td>Multiplicative scaling factor to apply to width, height, and depth.</td>
</tr>
<tr>
<td>res</td>
<td>Resolution of the faces.</td>
</tr>
</tbody>
</table>

Value

A rayvertex object.

See Also

See [https://www.rayvertex.com](https://www.rayvertex.com) for more information about the rayvertex package. See [geometry_utils](https://www.rayvertex.com) for a discussion of the 3D rotation parameterization.

Examples

```r
if (require("rayvertex") && all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$piecepack
  rs <- function(shape) {
    rasterize_scene(shape, light_info = directional_light(c(0, 0, 1)))
  }
  rs(piece_mesh("tile_face", suit = 3, rank = 3, cfg = cfg))
  rs(piece_mesh("coin_back", suit = 4, rank = 2, cfg = cfg))
  rs(piece_mesh("pawn_face", suit = 1, cfg = cfg))
}
```

---

**pmap_piece**

Create graphics using data frame input

**Description**

`pmap_piece()` operates on the rows of a data frame applying `.f` to each row (usually `grid.piece`).

**Usage**

```r
pmap_piece(
  .1,
  .f = pieceGrob,
  ...,
  cfg = getOption("piecepackr.cfg"),
  envir = getOption("piecepackr.envir"),
  trans = getOption("piecepackr.trans"),
  draw = TRUE,
)```
name = NULL,
gp = NULL,
vp = NULL
)

Arguments

.l A list of vectors, such as a data frame. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

.f Function to be applied to .l after adjustments to cfg and envir and the application of trans. Usually grid.piece(), pieceGrob(), piece3d(), or piece().

... Extra arguments to pass to .f.

cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().

envir Environment (or named list) containing configuration list(s).

trans Function to modify .l before drawing. Default (NULL) is to not modify .l. op_transform can help with using an oblique projection (i.e. op_scale over 0).

draw A logical value indicating whether graphics output should be produced.

name A character identifier (for grid)

gp An object of class “gpar”.

vp A grid viewport object (or NULL).

Details

pmap_piece() differs from purrr::pmap() in a few ways:

1. If cfg and/or envir are missing attempts to set reasonable defaults.
2. If not NULL will first apply function trans to .l.
3. If the output of .f is a grid grob object then pmap_piece will return a gTree object with specified name, gp, and vp values and if draw is true draw it.
4. If .l lacks a name column or if name column is non-unique attempts to generate a reasonable new default name column and use that to name the return gTree children or list values.

See Also

render_piece() is a higher-level function that wraps this function.

Examples

if (require("grid")) {
  dark_colorscheme <- list(suit_color="darkred,black,darkgreen,darkblue,black",
                         invert_colors.suited=TRUE, border_color="black", border_lex=2)
  traditional_ranks <- list(use_suit_as_ace=TRUE, rank_text="",a,2,3,4,5")
  cfg3d <- list(width.pawn=0.75, height.pawn=0.75, depth.pawn=1,"
```r
# Configuring the appearance of pieces

dm_text.pawn="", shape.pawn="convex6", invert_colors.pawn=TRUE,
edge_color.coin="tan", edge_color.tile="tan"

cfg <- pp_cfg(c(dark_colorscheme, traditional_ranks, cfg3d))

grid.newpage()

df_tiles <- data.frame(piece_side="tile_back", x=0.5+c(3,1,3,1), y=0.5+c(3,3,1,1),
suit=NA, angle=NA, z=NA, stringsAsFactors=FALSE)

df_coins <- data.frame(piece_side="coin_back", x=rep(4:1, 4), y=rep(4:1, each=4),
suit=1:16%%2+rep(c(1,3), each=8),
angle=rep(c(180,0), each=8), z=1/4+1/16, stringsAsFactors=FALSE)

df <- rbind(df_tiles, df_coins)

# Mapping the pieces
pmap_piece(df, cfg=cfg, op_scale=0.5, default.units="in")
```

---

**Description**

`pp_cfg()` and `as_pp_cfg()` create piecepack configuration list R6 objects. `is_pp_cfg()` returns `TRUE` if object is a piecepack configuration list R6 object. `as.list()` will convert it into a list.

**Usage**

```r
pp_cfg(cfg = list())

is_pp_cfg(cfg)

as_pp_cfg(cfg = list())
```

**Arguments**

- `cfg` List of configuration options

**Details**

- Customize the appearance of pieces drawn by `grid.piece()`.
- Speed up the drawing of graphics through use of caching.
- Allow the setting and querying of information about the board game components that maybe of use to developers:
  - Number of suits
  - Number of ranks
  - Suit colors
  - Which types of components are included and/or properly supported
  - What would be a good color to use when adding annotations on top of these components.
  - Title, Description, Copyright, License, and Credit metadata
pp_cfg R6 Class Method Arguments

- **piece_side**: A string with piece and side separated by an underscore e.g. "coin_face".
- **suit**: Number of suit (starting from 1).
- **rank**: Number of rank (starting from 1).
- **type**: Which type of grob to return, either "normal", "picture", "raster", or "transformation".
- **scale**: "scale" factor
- **alpha**: "alpha" value

pp_cfg R6 Class Methods

- **get_grob()**: Returns a grid “grob” for drawing the piece.
- **get_piece_opt()**: Returns a list with info useful for drawing the piece.
- **get_suit_color()**: Returns the suit colors.
- **get_width(), get_height(), get_depth()**: Dimensions (of the bounding cube) of the piece in inches

pp_cfg R6 Class Fields and Active Bindings

- **annotation_color**: Suggestion of a good color to annotate with
- **cache**: Cache object which stores intermediate graphical calculations. Default is a memory-cache that does not prune. This can be replaced by another cache that implements the cache API used by the cachem package
- **cache_grob**: Whether we should cache (2D) grobs
- **cache_grob_with_bleed_fn**: Whether we should cache the grob with bleed functions
- **cache_piece_opt**: Whether we should cache piece opt information
- **cache_op_fn**: Whether we should cache the oblique projection functions
- **cache_obj_fn**: Whether we should cache any 3D rendering functions
- **copyright**: Design copyright information
- **credit**: Design credits
- **description**: Design description
- **fontfamily**: Main font family
- **has_bits**: Whether we should assume this supports "bit" pieces
- **has_boards**: Whether we should assume this supports "board" pieces
- **has_cards**: Whether we should assume this supports "card" pieces
- **has_coins**: Whether we should assume this supports "coin" pieces
- **has_dice**: Whether we should assume this supports "die" pieces
- **has_matchsticks**: Whether we should assume this supports "matchstick" pieces
- **has_pawns**: Whether we should assume this supports "pawn" pieces
- **has_piecepack**: Binding which simultaneously checks/sets has_coins, has_tiles, has_pawns, has_dice
has_pyramids  Whether we should assume this supports "pyramid" pieces
has_saucers  Whether we should assume this supports "saucer" pieces
has_tiles    Whether we should assume this supports "tile" pieces
spdx_id     SPDX Identifier for graphical design license. See https://spdx.org/licenses/ for full list.
title       Design title

**Deprecated pp_cfg R6 Class attributes**

cache_shadow Use cache_op_fn instead
get_shadow_fn get_op_grob() returns complete oblique projection grob
i_unsuit    Instead add 1L to n_suits

**Defunct pp_cfg R6 Class attributes which have been removed**

get_pictureGrob() Use get_grob(..., type = "picture") instead

**See Also**

game_systems() for functions that return configuration list objects for several game systems. https://trevorldavis.com/piecepackr/configuration-lists.html for more details about piecepackr configuration lists.

**Examples**

cfg <- pp_cfg(list(invert_colors=TRUE))
as.list(cfg)
is_pp_cfg(cfg)
as_pp_cfg(list(suit_color="darkred,black,darkgreen,darkblue,grey"))
cfg$|get_suit_color(suit=3)
cfg$annotation_color
cfg$|has_matchsticks
cfg$has_matchsticks <- TRUE
cfg$has_matchsticks
cfg$|get_width("tile_back")
cfg$|get_height("die_face")
cfg$|get_depth("coin_face")

cfg <- list()
system.time(replicate(100, grid.piece("tile_face", 4, 4, cfg)))
cfg <- pp_cfg(list())
system.time(replicate(100, grid.piece("tile_face", 4, 4, cfg)))
pp_shape

Shape object for generating various grobs

Description

pp_shape() creates an R6 object with methods for creating various shape based grobs.

Usage

pp_shape(label = "rect", theta = 90, radius = 0.2, back = FALSE)

Arguments

label Label of the shape. One of
“circle” Circle.
“convexN” An N-sided convex polygon. theta controls which direction the first vertex is drawn.
“concaveN” A “star” (concave) polygon with N “points”. theta controls which direction the first point is drawn. radius controls the distance of the “inner” vertices from the center.
“halma” A 2D outline of a “Halma pawn”.
“kite” “Kite” quadrilateral shape.
“meeple” A 2D outline of a “meeple”.
“oval” Oval.
“pyramid” An “Isosceles” triangle whose base is the bottom of the viewport. Typically used to help draw the face of the “pyramid” piece.
“rect” Rectangle.
“roundrect” “Rounded” rectangle. radius controls curvature of corners.
theta convex and concave polygon shapes use this to determine where the first point is drawn.
radius concave polygon and roundrect use this to control appearance of the shape.
back Whether the shape should be reflected across a vertical line in the middle of the viewport.

Details

pp_shape objects serve the following purposes:

1. Make it easier for developers to customize game piece appearances either through a "grob_fn" or "op_grob_fn" styles in pp_cfg() or manipulate a piece post drawing via functions like grid::grid.edit().
2. Used internally to generate piecepackr’s built-in game piece grobs.
**pp_shape**

**R6 Class Method Arguments**

- `mat_width` Numeric vector of mat widths.
- `clip` “clip grob” to perform polyclip operation with. See `gridGeometry::grid.polyclip()` for more info.
- `op` Polyclip operation to perform. See `gridGeometry::grid.polyclip()` for more info.
- `pattern` Pattern to fill in shape with. See `gridpattern::patternGrob()` for more info.
- `...` Passed to `gridpattern::patternGrob()`.
- `name` Grid grob name value.
- `gp` Grid gpar list. See `grid::gpar()` for more info.
- `vp` Grid viewport or NULL.

**pp_shape R6 Class Methods**

- `checkers(name = NULL, gp = gpar(), vp = NULL)` Returns a grob of checkers for that shape.
- `gridlines(name = NULL, gp = gpar(), vp = NULL)` Returns a grob of gridlines for that shape.
- `hexlines(name = NULL, gp = gpar(), vp = NULL)` Returns a grob of hexlines for that shape.
- `mat(mat_width = 0, name = NULL, gp = gpar(), vp = NULL)` Returns a grob for a matting “mat” for that shape.
- `pattern(pattern = "stripe", ..., name = NULL, gp = gpar(), vp = NULL)` Fills in the shape’s `npc_coords` with a pattern. See `gridpattern::patternGrob()` for more information.
- `polyclip(clip, op = "intersection", name = NULL, gp = gpar(), vp = NULL)` Returns a grob that is an “intersection”, “minus”, “union”, or “xor” of another grob. Note unlike `gridGeometry::polyclipGrob` it can directly work with a pieceGrob “clip grob” argument.
- `shape(name = NULL, gp = gpar(), vp = NULL)` Returns a grob of the shape.

**pp_shape R6 Class Active Bindings**

- `label` The shape’s label.
- `theta` The shape’s theta.
- `radius` The shape’s radius.
- `back` A boolean of whether this is the shape’s “back” side.
- `npc_coords` A named list of “npc” coordinates along the perimeter of the shape.

**Examples**

```r
if (require("grid")) {
  gp <- gpar(col="black", fill="yellow")
  rect <- pp_shape(label="rect")
  convex6 <- pp_shape(label="convex6")
  circle <- pp_shape(label="circle")

  pushViewport(viewport(x=0.25, y=0.75, width=1/2, height=1/2))
  grid.draw(rect$shape(gp=gp))
  grid.draw(rect$gridlines(gp=gpar(col="blue", lex=4)))
  ```
grid.draw(rect$hexlines(gp=gpar(col="green")))
popViewport()

pushViewport(viewport(x=0.75, y=0.75, width=1/2, height=1/2))
grid.draw(convex6$shape(gp=gp))
grid.draw(convex6$checkers(gp=gpar(fill="blue")))
popViewport()

pushViewport(viewport(x=0.25, y=0.25, width=1/2, height=1/2))
grid.draw(circle$shape(gp=gp))
grid.draw(circle$mat(mat_width=0.2, gp=gpar(fill="blue")))
popViewport()

pushViewport(viewport(x=0.75, y=0.25, width=1/2, height=1/2))
grid.draw(rect$shape(gp=gp))
grid.draw(rect$mat(mat_width=c(0.2, 0.1, 0.3, 0.4), gp=gpar(fill="blue")))
popViewport()

grid.newpage()
gp <- gpar(col="black", fill="yellow")

vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("halma")$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("pyramid")$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("kite")$shape(gp=gp, vp=vp))
vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("meeple")$shape(gp=gp, vp=vp))

grid.newpage()
vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("convex3", 0)$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("convex4", 90)$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("convex5", 180)$shape(gp=gp, vp=vp))
vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("convex6", 270)$shape(gp=gp, vp=vp))

grid.newpage()
vp <- viewport(x=1/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("concave3", 0, 0.1)$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=1/4, width=1/2, height=1/2)
grid.draw(pp_shape("concave4", 90, 0.2)$shape(gp=gp, vp=vp))
vp <- viewport(x=3/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("concave5", 180, 0.3)$shape(gp=gp, vp=vp))
vp <- viewport(x=1/4, y=3/4, width=1/2, height=1/2)
grid.draw(pp_shape("concave6", 270)$shape(gp=gp, vp=vp))

if (require("gridpattern")) {
  grid.newpage()
  hex <- pp_shape("convex6")
}
Description

cleave converts a delimiter separated string into a vector. inch(x) is equivalent to unit(x, "in").
is_color_invisible tells whether the color is transparent (and hence need not be drawn).

Usage

is_color_invisible(col)

inch(inches)

cleave(s, sep = ",", float = FALSE, color = FALSE)

file2grob(file, distort = TRUE)

Arguments

col | Color
---|---
inches | Number representing number of inches
s | String to convert
sep | Delimiter (defaults to ",")
float | If TRUE cast to numeric
color | if TRUE convert empty strings to "transparent"
file | Filename of image
distort | Logical value of whether one should preserve the aspect ratio or distort to fit the area it is drawn in

Examples

to_x(90, 1)
to_y(180, 0.5)
to_t(0, -1)
to_r(0.5, 0)

cleave("0.5,0.2,0.4,0.5", float=TRUE)
render_piece

Render image of game pieces

Description

render_piece() renders an image of game pieces to a file or graphics device. It is a wrapper around `pmap_piece()` that can auto-size files and graphic devices, apply axes offsets, annotate coordinates, and set up rayrender/rayvertex scenes.

Usage

```r
render_piece(
  df,
  file = NULL,
  ...,
  .f = piecepackr::grid.piece,
  cfg =getOption("piecepackr.cfg", NULL),
  envir =getOption("piecepackr.envir", game_systems("sans")),
  width = NULL,
  height = NULL,
  ppi = 72,
  bg = "white",
  xoffset = NULL,
  yoffset = NULL,
  new_device = TRUE,
  dev = NULL,
  dev.args = list(res = ppi, bg = bg, units = "in"),
  annotate = FALSE,
  annotation_scale = NULL
)
```

Arguments

df A data frame of game piece information with (at least) the named columns "piece_side", "x", and "y".

render_piece

**Arguments to `pmap_piece()`**

- `file` Filename to save animation unless **NULL** in which case it either uses the current graphics device or opens a new device (depending on `new_device` argument).
- `...` Arguments to `pmap_piece()`
- `.f` Low level graphics function to use e.g. `grid.piece()`, `piece3d()`, `piece_mesh()`, or `piece()`.
- `cfg` A piecepackr configuration list
- `envir` Environment (or named list) of piecepackr configuration lists
- `width` Width of animation (in inches). Inferred by default.
- `height` Height of animation (in inches). Inferred by default.
- `ppi` Resolution of animation in pixels per inch.
- `bg` Background color (use "transparent" for transparent)
- `xoffset` Number to add to the x column in `df`. Inferred by default.
- `yoffset` Number to add to the y column in `df`. Inferred by default.
- `new_device` If `file` is **NULL** should we open up a new graphics device?
- `dev` Graphics device function to use. If **NULL** infer a reasonable choice.
- `dev.args` Additional arguments to pass to `dev` (besides `filename`, `width`, and `height`). Will filter out any names that aren't in `formals(dev)`.
- `annotate` If **TRUE** or "algebraic" annotate the plot with "algebraic" coordinates, if **FALSE** or "none" don't annotate, if "cartesian" annotate the plot with "cartesian" coordinates.
- `annotation_scale` Multiplicative factor that scales (stretches) any annotation coordinates. By default uses `attr(df, "scale_factor") %||% 1`.

**Value**

An invisible list of the dimensions of the image, as a side effect saves a graphic

**See Also**

This function is a wrapper around `pmap_piece()`.

**Examples**

```r
df_board <- data.frame(piece_side = "board_face", suit = 3, rank = 8,
                        x = 4.5, y = 4.5, stringsAsFactors = FALSE)
df_w <- data.frame(piece_side = "bit_face", suit = 6, rank = 1,
                    x = rep(1:8, 2), y = rep(1:2, each=8),
                    stringsAsFactors = FALSE)
df_b <- data.frame(piece_side = "bit_face", suit = 1, rank = 1,
                    x = rep(1:8, 2), y = rep(7:8, each=8),
                    stringsAsFactors = FALSE)
df <- rbind(df_board, df_w, df_b)
df$cfg <- "checkers1"
render_piece(df)
```
Alternative Wavefront OBJ file generators

**Description**

These are alternative Wavefront OBJ generators intended to be used as a \texttt{obj.fn} attribute in a \texttt{pp.cfg()} "configuration list". \texttt{save_ellipsoid_obj} saves an ellipsoid with a color equal to that piece’s \texttt{background.color}. \texttt{save_peg_doll_obj} saves a "peg doll" style doll with a color equal to that piece’s \texttt{edge.color} with a "pawn belt" around it’s waste from that suit’s and rank’s \texttt{belt.face}.

**Usage**

```r
save_ellipsoid_obj(
  piece_side = "bit_face",
  suit = 1,
  rank = 1,
  cfg = getOption("piecepackr.cfg", pp.cfg()),
  ...
)
```

```r
save_peg_doll_obj(
  piece_side = "pawn_top",
  suit = 1,
  rank = 1,
  cfg = getOption("piecepackr.cfg", pp.cfg()),
  ...
)
```
save_ellipsoid_obj

y = 0,
z = 0,
angle = 0,
axis_x = 0,
axis_y = 0,
width = NA,
height = NA,
depth = NA,
filename = tempfile(fileext = ".obj"),
res = 72
)

Arguments

piece_side A string with piece and side separated by a underscore e.g. "coin_face"
suit Number of suit (starting from 1).
rank Number of rank (starting from 1)
cfg Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base:::dynGet()
...
Ignored.
x Where to place piece on x axis of viewport
y Where to place piece on y axis of viewport
z z-coordinate of the piece. Has no effect if op_scale is 0.
angle Angle (on xy plane) to draw piece at
axis_x First coordinate of the axis unit vector.
axis_y Second coordinate of the axis unit vector.
width Width of piece
height Height of piece
depth Depth (thickness) of piece. Has no effect if op_scale is 0.
filename Name of Wavefront OBJ object.
subdivide Increasing this value makes for a smoother ellipsoid (and larger OBJ file and slower render). See ellipse3d.
res Resolution of the faces.

See Also

See pp_cfg() for a discussion of “configuration lists”. Wavefront OBJ file generators are used by save_piece_obj() and (by default) piece3d() (rgl wrapper), piece() (rayrender wrapper), and piece_mesh() (rayvertex wrapper).
save_piece_images  Save piecepack images

Description

Saves images of all individual piecepack pieces.

Usage

```r
save_piece_images(
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  directory = tempdir(),
  format = "svg",
  angle = 0
)
```

Arguments

- `cfg`: Piecepack configuration list
- `directory`: Directory where to place images
- `format`: Character vector of formats to save images in
- `angle`: Numeric vector of angles to rotate images (in degrees)

Examples

```r
if (all(capabilities(c("cairo", "png")))) {
  cfg <- pp_cfg(list(suit_color="darkred,black,darkgreen,darkblue,gray"))
  save_piece_images(cfg, directory=tempdir(), format="svg", angle=0)
  save_piece_images(cfg, directory=tempdir(), format="png", angle=90)
}
```

save_piece_obj  Save Wavefront OBJ files of board game pieces

Description

`save_piece_obj` saves Wavefront OBJ files (including associated MTL and texture image) of board game pieces.
Usage

```r
save_piece_obj(
  piece_side = "tile_face",
  suit = 1,
  rank = 1,
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  ...,
  x = 0,
  y = 0,
  z = 0,
  angle = 0,
  axis_x = 0,
  axis_y = 0,
  width = NA,
  height = NA,
  depth = NA,
  filename = tempfile(fileext = ".obj"),
  scale = 1,
  res = 72
)
```

Arguments

- **piece_side**: A string with piece and side separated by a underscore e.g. "coin_face"
- **suit**: Number of suit (starting from 1).
- **rank**: Number of rank (starting from 1)
- **cfg**: Piecepack configuration list or pp_cfg object, a list of pp_cfg objects, or a character vector referring to names in envir or a character vector referring to object names that can be retrieved by base::dynGet().
- **...**: Ignored.
- **x**: Where to place piece on x axis of viewport
- **y**: Where to place piece on y axis of viewport
- **z**: z-coordinate of the piece. Has no effect if op_scale is 0.
- **angle**: Angle (on xy plane) to draw piece at
- **axis_x**: First coordinate of the axis unit vector.
- **axis_y**: Second coordinate of the axis unit vector.
- **width**: Width of piece
- **height**: Height of piece
- **depth**: Depth (thickness) of piece. Has no effect if op_scale is 0.
- **filename**: Name of Wavefront OBJ object.
- **scale**: Multiplicative scaling factor to apply to width, height, and depth.
- **res**: Resolution of the faces.
save_print_and_play

Value

A list with named elements "obj", "mtl", "png" with the created filenames.

See Also

See geometry_utils for a discussion of the 3D rotation parameterization.

Examples

```r
if (all(capabilities(c("cairo", "png")))) {
  cfg <- game_systems("sans3d")$dominoes
  files <- save_piece_obj("tile_face", suit = 3+1, rank=6+1, cfg = cfg)
  print(files)
}
```

---

save_print_and_play  Save piecepack print-and-play (PnP) file

Description

Save piecepack print-and-play (PnP) file

Usage

```r
save_print_and_play(
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  output_filename = "piecepack.pdf",
  size = c("letter", "A4", "A5", "4x6"),
  pieces = NULL,
  arrangement = c("single-sided", "double-sided"),
  dev = NULL,
  dev.args = list(family = cfg$fontfamily, onefile = TRUE, units = "in", bg = "white",
                  res = 72),
  quietly = FALSE,
  ...
)
```

Arguments

- `cfg`  
  Piecepack configuration list or pp_cfg object
- `output_filename`  
  Filename for print-and-play file
- `size`  
  PnP output size (currently either "letter", "A4", "A5", or "4x6") Size "4x6" currently only supports pieces = "piecepack" and doesn’t support bleed = TRUE.
**Description**

`scale_x_piece()` and `scale_y_piece()` are wrappers around `ggplot2::scale_x_continuous()` and `ggplot2::scale_y_continuous()` with "better" defaults for board game diagrams. `label_letter()` labels breaks with letters and `label_counting()` labels breaks with positive integers to more easily generate (i.e. chess) algebraic notation coordinates. `breaks_counting()` generates breaks of just the positive integers within the limits.
scale_x_piece

scale_x_piece(
  ..., 
  name = NULL, 
  breaks = breaks_counting(), 
  minor_breaks = NULL, 
  labels = label_letter()
)

scale_y_piece(
  ..., 
  name = NULL, 
  breaks = breaks_counting(), 
  minor_breaks = NULL, 
  labels = label_counting()
)

label_letter()

label_counting()

breaks_counting()

Arguments

... Passed to ggplot2::scale_x_continuous() or ggplot2::scale_y_continuous().

name The name of the scale. Used as the axis or legend title. If waiver(), the default, the name of the scale is taken from the first mapping used for that aesthetic. If NULL, the legend title will be omitted.

breaks One of:
  • NULL for no breaks
  • waiver() for the default breaks computed by the transformation object
  • A numeric vector of positions
  • A function that takes the limits as input and returns breaks as output (e.g., a function returned by scales::extended_breaks()). Also accepts rlang lambda function notation.

minor_breaks One of:
  • NULL for no minor breaks
  • waiver() for the default breaks (one minor break between each major break)
  • A numeric vector of positions
  • A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation.

labels One of:
  • NULL for no labels
spdx_license_list

- waiver() for the default labels computed by the transformation object
- A character vector giving labels (must be same length as breaks)
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

Value

close_x_piece() and close_y_piece() return ggplot2 scale objects. label_letter() and label_counting() return functions suitable for use with the labels scale argument. breaks_counting() returns a function suitable for use with the breaks scale argument.

Examples

```r
if (require("ggplot2") && require("tibble")) {
  envir <- game_systems("sans")
  df_board <- tibble(piece_side = "board_face", suit = 3, rank = 8,
                     x = 4.5, y = 4.5)
  df_w <- tibble(piece_side = "bit_face", suit = 6, rank = 1,
                  x = rep(1:8, 2), y = rep(1:2, each=8))
  df_b <- tibble(piece_side = "bit_face", suit = 1, rank = 1,
                  x = rep(1:8, 2), y = rep(7:8, each=8))
  df <- rbind(df_board, df_w, df_b)

  # 'cfg' must be a character vector for 'geom_piece'
  ggplot(df, aes_piece(df)) +
  geom_piece(cfg = "checkers1", envir = envir) +
  coord_fixed() +
  scale_x_piece() +
  scale_y_piece() +
  theme_minimal(28) +
  theme(panel.grid = element_blank())
}
```

spdx_license_list  

SPDX License List data

Description

spdx_license_list is a data frame of SPDX License List data.

Usage

spdx_license_list
Format

a data frame with eight variables:

- **id**: SPDX Identifier.
- **name**: Full name of license. For Creative Commons licenses these have been tweaked from the SPDX version to more closely match the full name used by Creative Commons Foundation.
- **url**: URL for copy of license located at spdx.org
- **fsf**: Is this license considered Free/Libre by the FSF?
- **osi**: Is this license OSI approved?
- **deprecated**: Has this SPDX Identifier been deprecated by SPDX?
- **badge**: Filename of appropriate “button mark” badge (if any) located in system.file("extdata/badges", package = "piecepackr").
- **url_alt**: Alternative URL for license. Manually created for a subset of Creative Commons licenses. Others taken from https://github.com/sindresorhus/spdx-license-list.

See Also

See https://spdx.org/licenses/ for more information.
Index

* datasets
  spdx_license_list, 47
AA_to_R, 5
aabb_piece, 3
aes(), 14
aes_piece (geom_piece), 14
animate_piece, 7
as_pp_cfg (pp_cfg), 31
base::options(), 3
basicPieceGrob (basicPieceGrobs), 9
basicPieceGrobs, 9
borders(), 15
breaks_counting (scale_x_piece), 45
checkersGrob (piecepackr-defunct), 27
cleave (pp_utils), 37
concaveGrobFn (piecepackr-defunct), 27
cvxGrobFn (piecepackr-defunct), 27
cropmarkGrob (grid.cropmark), 16
eclipse3d, 41
file2grob (pp_utils), 37
font_utils, 10
fortify(), 14
game_systems, 11
game_systems(), 33
gggplot2::scale_x_continuous(), 16, 45, 46
ggplot2::scale_y_continuous(), 16, 45, 46
grid.cropmark, 16
grid.piece, 18
grid.piece(), 7, 30, 39
grid::gpar(), 35
grid::unit(), 18, 21
gridGeometry::grid.polyclip(), 35
gridlinesGrob (piecepackr-defunct), 27
gridpattern::patternGrob(), 35
halmaGrob (piecepackr-defunct), 27
has_font (font_utils), 10
hexlinesGrob (piecepackr-defunct), 27
inch (pp_utils), 37
is_color_invisible (pp_utils), 37
is_pp_cfg (pp_cfg), 31
kiteGrob (piecepackr-defunct), 27
label_counting (scale_x_piece), 45
label_letter (scale_x_piece), 45
lambda, 46, 47
matGrob (piecepackr-defunct), 27
op_transform, 22
picturePieceGrobFn (basicPieceGrobs), 9
piece, 23
piece(), 7, 30, 39, 41
piece3d, 25
piece3d(), 7, 30, 39, 41
piece_mesh, 28
piece_mesh(), 7, 39, 41
pieceGrob (grid.piece), 18
pieceGrob(), 15, 16, 30
piecepackr (piecepackr-package), 3
piecepackr-defunct, 27
piecepackr-package, 3
pmap_piece, 29
pmap_piece(), 21, 39, 45
pp_cfg, 13, 31
pp_cfg(), 41
pp_shape, 34
pp_utils, 37
previewLayoutGrob (basicPieceGrobs), 9
pyramidGrob (piecepackr-defunct), 27
pyramidTopGrob (basicPieceGrobs), 9

R_to_AA (AA_to_R), 5
R_x (AA_to_R), 5
R_y (AA_to_R), 5
R_z (AA_to_R), 5
render_piece, 38
render_piece(), 30
rgl::material3d(), 26

save_ellipsoid_obj, 40
save_peg_doll_obj (save_ellipsoid_obj), 40

save_piece_images, 42
save_piece_obj, 42
save_piece_obj(), 41
save_print_and_play, 44
scale_x_piece, 45
scale_x_piece(), 16
scale_y_piece (scale_x_piece), 45
scale_y_piece(), 16
scales::extended_breaks(), 46
spdx_license_list, 47

to_degrees (AA_to_R), 5
to_hexpack (game_systems), 11
to_r (AA_to_R), 5
to_radians (AA_to_R), 5
to_subpack (game_systems), 11
to_t (AA_to_R), 5
to_x (AA_to_R), 5
to_y (AA_to_R), 5
transformation object, 46
Trig, 6