

# Package ‘piecepackr’

March 22, 2022

**Encoding** UTF-8

**Type** Package

**Title** Board Game Graphics

**Version** 1.10.3

**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), systemfonts, testthat, tweenr, vdiff

**SystemRequirements** ghostscript

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Author** Trevor L Davis [aut, cre],  
Linux Foundation [dct] (Uses some data from the ``SPDX License List''  
<<https://github.com/spdx/license-list-XML>>),  
Delapouite <<https://delapouite.com/>> [ill] (Meeple shape extracted from  
``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** 2022-03-22 21:50:02 UTC

## R topics documented:

piecepackr-package	2
aabb_piece	3
AA_to_R	4
animate_piece	6
basicPieceGrobs	8
font_utils	10
game_systems	11
geom_piece	13
grid.piece	16
op_transform	19
piece	21
piece3d	22
piecepackr-defunct	24
piece_mesh	25
pmap_piece	27
pp_cfg	28
pp_shape	31
pp_utils	34
render_piece	35
save_ellipsoid_obj	37
save_piece_images	39
save_piece_obj	39
save_print_and_play	41
scale_x_piece	42
spx_license_list	44
<b>Index</b>	<b>45</b>

---

piecepackr-package      *piecepackr: Board Game Graphics*

---

## 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()`:

**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

<code>df</code>	A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.
<code>cfg</code>	Piecepack configuration list or <code>pp_cfg</code> object, a list of <code>pp_cfg</code> objects, or a character vector referring to names in <code>envir</code> or a character vector referring to object names that can be retrieved by <code>base::dynGet()</code> .

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  $(x + \lambda * z * \cos(\alpha), y + \lambda * z * \sin(\alpha))$  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**

```
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**

pp\_cfg 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

[https://en.wikipedia.org/wiki/Axis-angle\\_representation](https://en.wikipedia.org/wiki/Axis-angle_representation) for more details about the Axis-angle representation of 3D rotations. See [Trig](#) for R's built-in trigonometric functions.

### 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. <code>grid.piece()</code> , <code>piece3d()</code> , <code>piece()</code> , or <code>piece_mesh()</code> .
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.
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 <code>attr(df, "scale_factor") %  % 1</code> .

## Value

Nothing, as a side effect creates an animation.

## Examples

```

# 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

```
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**

```

is_mac <- tolower(Sys.info()[["sysname"]]) == "darwin"
if (require("grid") && capabilities("cairo") && !is_mac) {
  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()

  grid.newpage()
  pushViewport(viewport(width=unit(6, "in"), height=unit(6, "in")))
  grid.draw(previewLayoutGrob("preview_layout", suit=5, rank=0, cfg=cfg))
  popViewport()

  grid.newpage()
  pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
  grid.draw(pyramidTopGrob("pyramid_top", suit=3, rank=5))
  popViewport()

  directory <- tempdir()
  save_piece_images(cfg, directory=directory, format="svg", angle=0)
  cfg2 <- pp_cfg(list(grob_fn=picturePieceGrobFn(directory)))

  grid.newpage()
  pushViewport(viewport(width=unit(0.75, "in"), height=unit(0.75, "in")))
  grid.draw(pyramidTopGrob("pyramid_top", suit=3, rank=5, cfg=cfg2))
  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).

### 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

```
if ((Sys.which("pdffonts") != "") && capabilities("cairo")) {
  chars <- c("a", "\u2666")
  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**

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

**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 piped dice in six color schemes (color controlled by their suit).

**dominoes, dominoes\_black, dominoes\_blue, dominoes\_green, dominoes\_red, dominoes\_white, dominoes\_yellow**

Traditional piped 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 saucers** A public domain accessory developed by Karol M. Boyle at Mesomorph Games. See <https://web.archive.org/web/20190719155827/http://www.piecepack.org/Accessories.html>.

**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-ppn.html>.

**playing\_cards\_expansion** A piecepack with the standard dQuoteFrench playing card suits. See <https://www.ludism.org/ppwiki/PlayingCardsExpansion>.

**hexpack** 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")) {
  # 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),
                        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=c(1,4,1,4,4,1,4,1,2,3,2,3,3,2,3,2),
                        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**

```
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
)

aes_piece(df)
```

**Arguments**

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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). <code>0</code> (the default) leads to an “orthographic” projection, <code>0.5</code> is the most common scale used in the “cabinet” projection, and <code>1.0</code> is the scale used in the “cavalier” projection.
op_angle	What is the angle of the oblique projection? Has no effect if <code>op_scale</code> is <code>0</code> .
inherit.aes	If <code>FALSE</code> , 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. <code>borders()</code> .
df	A data frame of game piece information with (at least) the named columns “piece_side”, “x”, and “y”.

## Details

`geom_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

`geom_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

```
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.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

```

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
)

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 (plus bleed if bleed is TRUE)
height	Height of piece (plus bleed if bleed is TRUE)
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	Type of grid grob to use. Either "normal" (default), "picture", or "raster". "picture" exports to (temporary) svg and re-exports as a <code>grImport2::pictureGrob</code> . "raster" exports to (temporary) png and re-exports as a <code>grid::rasterGrob</code> . The latter two can be useful if drawing pieces really big or small and don't want to mess with re-configuring font sizes and linewidths.
bleed	Whether to add a "bleed" zone around the piece. If width or height is NA we will add 1/8 inch bleeds. <code>bleed = TRUE</code> is incompatible with <code>op_scale &gt; 0</code> .
draw	A logical value indicating whether graphics output should be produced.

**Value**

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")) {
  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)
pt_thickness	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).
as_top	Character vector of components whose "side" should be converted to "top" e.g. c("pawn_face").
cfg_class	Either "list" (default) or "character". Desired class of the cfg column in the returned tibble. "list" is more efficient for use with pmap_piece() but geom_piece() needs "character".

**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 op\_angle or the dimensions of pieces.

**Value**

A tibble with extra columns added and re-sorted rows

**See Also**

<https://trevorldavis.com/piecepackr/3d-projections.html> for more details and examples of oblique projections in piecepackr.

**Examples**

```
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")
```

---

piece *Create rayrender board game piece objects*

---

### Description

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

### Usage

```
piece(
  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
)
```

### 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 (plus bleed if bleed is TRUE)

height	Height of piece (plus bleed if bleed is TRUE)
depth	Depth (thickness) of piece. Has no effect if <code>op_scale</code> 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> for more information about the rayrender package. See [geometry\\_utils](#) for a discussion of the 3D rotation parameterization.

**Examples**

```
if (require("rayrender")) {
  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))
}
```

---

piece3d

*Render board game pieces with rgl*

---

**Description**

piece3d draws board games pieces using the rgl package.

**Usage**

```
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 (plus bleed if bleed is TRUE)
height	Height of piece (plus bleed if bleed is TRUE)
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.material](#) for more info about setting rgl material properties. See [geometry\\_utils](#) for a discussion of the 3D rotation parameterization.

**Examples**

```
if (require("rgl")) {
  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.)
}
```

---

piecepackr-defunct      *Defunct functions*

---

**Description**

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 (plus bleed if bleed is TRUE)
height	Height of piece (plus bleed if bleed is TRUE)
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 rayvertex object.

**See Also**

See <https://www.rayvertex.com> for more information about the rayvertex package. See [geometry\\_utils](#) for a discussion of the 3D rotation parameterization.

**Examples**

```
if (require("rayvertex")) {
  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**

```
pmap_piece(
  .l,
  .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 <code>grid.piece()</code> , <code>pieceGrob()</code> , <code>piece3d()</code> , or <code>piece()</code> .
...	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 <code>base::dynGet()</code> .
envir	Environment (or named list) containing configuration list(s).
trans	Function to modify .l before drawing. Default (NULL) is to not modify .l. <code>op_transform</code> can help with using an oblique projection (i.e. <code>op_scale</code> 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,
               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)
  pmap_piece(df, cfg=cfg, op_scale=0.5, default.units="in")
}
```

---

pp\_cfg

*Configuration list R6 object*

---

**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**

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

```
is_pp_cfg(cfg)
```

```
as_pp_cfg(cfg = list())
```

**Arguments**

cfg List of configuration options

**Details**

pp\_cfg R6 class objects serve the following purposes:

- 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 a 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", or "raster".

**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  
 sdx\_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 <b>“circle”</b> Circle. <b>“convexN”</b> An N-sided convex polygon. theta controls which direction the first vertex is drawn. <b>“concaveN”</b> 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. <b>“halma”</b> A 2D outline of a “Halma pawn”. <b>“kite”</b> “Kite” quadrilateral shape. <b>“meeple”</b> A 2D outline of a “meeple”. <b>“oval”</b> Oval. <b>“pyramid”</b> An “Isosceles” triangle whose base is the bottom of the viewport. Typically used to help draw the face of the “pyramid” piece. <b>“rect”</b> Rectangle. <b>“roundrect”</b> “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**

```

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")
  gp <- gpar(fill = c("blue", "yellow", "red"), col = "black")
  grid.draw(hex$pattern("polygon_tiling", gp = gp, spacing = 0.1,
                       type = "truncated_trihexagonal"))
  gp <- gpar(fill = "black", col = NA)
  grid.draw(hex$mat(mat_width = 0.025, gp = gp))
}
}

```

pp\_utils

*Miscellaneous piecepackr utility functions***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**

<code>col</code>	Color
<code>inches</code>	Number representing number of inches
<code>s</code>	String to convert
<code>sep</code>	Delimiter (defaults to ",")
<code>float</code>	If TRUE cast to numeric
<code>color</code>	if TRUE convert empty strings to "transparent"
<code>file</code>	Filename of image
<code>distort</code>	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)
cleave("black,darkred,#050EAA,,", color=TRUE)

if (require("grid")) {
  grid.rect(width=inch(1), height=inch(3), gp=gpar(fill="blue"))
}

is_color_invisible("transparent")
is_color_invisible(NA)
is_color_invisible("blue")
is_color_invisible("#05AE9C")

```

---

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**

```

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".
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 <a href="#">pmap_piece()</a>
.f	Low level graphics function to use e.g. <a href="#">grid.piece()</a> , <a href="#">piece3d()</a> , <a href="#">piece_mesh()</a> , or <a href="#">piece()</a> .
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 <code>formals(dev)</code> .
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 <code>attr(df, "scale_factor") %  % 1</code> .

**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**

```
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)
render_piece(df, op_scale = 0.5, trans = op_transform, annotate = "algebraic")
## Not run: # May takes a while to render
if (require(rayvertex)) {
  envir3d <- game_systems("sans3d")
  render_piece(df, .f = piece_mesh, envir = envir3d,
               op_scale = 0.5, trans = op_transform)
}

## End(Not run)

```

---

save\_ellipsoid\_obj      *Alternative Wavefront OBJ file generators*

---

## Description

These are alternative Wavefront OBJ generators intended to be used as a `obj_fn` attribute in a `pp_cfg()` “configuration list”. `save_ellipsoid_obj` saves an ellipsoid with a color equal to that piece’s `background_color`. `save_peg_doll_obj` saves a “peg doll” style doll with a color equal to that piece’s `edge_color` with a “pawn belt” around it’s waste from that suit’s and rank’s `belt_face`.

## Usage

```

save_ellipsoid_obj(
  piece_side = "bit_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"),
  subdivide = 3
)

save_peg_doll_obj(
  piece_side = "pawn_top",
  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"),
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 <code>envir</code> or a character vector referring to object names that can be retrieved by <code>base::dynGet()</code> .
...	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 <code>op_scale</code> 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 (plus bleed if <code>bleed</code> is TRUE)
height	Height of piece (plus bleed if <code>bleed</code> is TRUE)
depth	Depth (thickness) of piece. Has no effect if <code>op_scale</code> 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 <a href="#">ellipse3d</a> .
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	<i>Save piecepack images</i>
-------------------	------------------------------

---

### Description

Saves images of all individual piecepack pieces.

### Usage

```
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

```
is_mac <- tolower(Sys.info()[["sysname"]]) == "darwin"
if (all(capabilities(c("cairo", "png"))) && !is_mac) {
  cfg <- pp_cfg(list(suit_color="darkred,black,darkgreen,darkblue,grey"))
  save_piece_images(cfg, directory=tempdir(), format="svg", angle=0)
  save_piece_images(cfg, directory=tempdir(), format="png", angle=90)
}
```

---

save_piece_obj	<i>Save Wavefront OBJ files of board game pieces</i>
----------------	--

---

### Description

save\_piece\_obj saves Wavefront OBJ files (including associated MTL and texture image) of board game pieces.

**Usage**

```

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 (plus bleed if bleed is TRUE)
height	Height of piece (plus bleed if bleed is TRUE)
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.



**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**

```
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**

```
save_print_and_play(
  cfg = getOption("piecepackr.cfg", pp_cfg()),
  output_filename = "piecepack.pdf",
  size = c("letter", "A4", "A5", "4x6"),
  pieces = c("piecepack", "matchsticks", "pyramids"),
  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")
pieces	Character vector of desired PnP pieces. Supports "piecepack", "matchsticks", "pyramids", "subpack", or "all".
arrangement	Either "single-sided" or "double-sided". Ignored if size = "4x6".
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 <code>formals(dev)</code> .
quietly	Whether to hide messages about missing metadata in the provided configuration.

## Examples

```
is_mac <- tolower(Sys.info()[["sysname"]]) == "darwin"
if (capabilities("cairo") && !is_mac) {
  cfg <- pp_cfg(list(invert_colors.suited=TRUE))
  cfg$description <- 'Piecepack with an "inverted" color scheme.'
  cfg$title <- '"Inverted" piecepack'
  cfg$copyright <- "\u00a9 2022 Trevor L Davis. Some Right Reserved."
  cfg$spdx_id <- "CC-BY-4.0"
  cfg$credit <- ""
  save_print_and_play(cfg, "my_pnp_file.pdf")
  save_print_and_play(cfg, "my_pnp_file_ds.pdf", arrangement="double-sided")
  save_print_and_play(cfg, "my_pnp_file_A4.pdf", size="A4", pieces="all")
  save_print_and_play(cfg, "my_pnp_file_A5.pdf", size="A5")
  unlink("my_pnp_file.pdf")
  unlink("my_pnp_file_ds.pdf")
  unlink("my_pnp_file_A4.pdf")
  unlink("my_pnp_file_A5.pdf")
}
```

---

scale\_x\_piece

*ggplot2 game diagram scales*

---

## 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.

## Usage

```
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 <code>ggplot2::scale_x_continuous()</code> or <code>ggplot2::scale_y_continuous()</code> .
name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
breaks	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no breaks</li> <li>• <code>waiver()</code> for the default breaks computed by the <a href="#">transformation object</a></li> <li>• A numeric vector of positions</li> <li>• A function that takes the limits as input and returns breaks as output (e.g., a function returned by <code>scales::extended_breaks()</code>). Also accepts <code>rlang</code> <a href="#">lambda</a> function notation.</li> </ul>
minor_breaks	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no minor breaks</li> <li>• <code>waiver()</code> for the default breaks (one minor break between each major break)</li> <li>• A numeric vector of positions</li> <li>• A function that given the limits returns a vector of minor breaks. Also accepts <code>rlang</code> <a href="#">lambda</a> function notation.</li> </ul>
labels	One of: <ul style="list-style-type: none"> <li>• <code>NULL</code> for no labels</li> <li>• <code>waiver()</code> for the default labels computed by the transformation object</li> <li>• A character vector giving labels (must be same length as breaks)</li> <li>• A function that takes the breaks as input and returns labels as output. Also accepts <code>rlang</code> <a href="#">lambda</a> function notation.</li> </ul>

## Value

`scale_x_piece()` and `scale_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

```

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 SPDFX 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, 44
- AA\_to\_R, 4
- aabb\_piece, 3
- aes(), 14
- aes\_(), 14
- aes\_piece (geom\_piece), 13
- animate\_piece, 6
- as\_pp\_cfg (pp\_cfg), 28
  
- base::options(), 3
- basicPieceGrob (basicPieceGrobs), 8
- basicPieceGrobs, 8
- borders(), 14
- breaks\_counting (scale\_x\_piece), 42
  
- checkersGrob (piecepackr-defunct), 24
- cleave (pp\_utils), 34
- concaveGrobFn (piecepackr-defunct), 24
- convexGrobFn (piecepackr-defunct), 24
  
- ellipse3d, 38
  
- file2grob (pp\_utils), 34
- font\_utils, 10
- fortify(), 14
  
- game\_systems, 11
- game\_systems(), 30
- geom\_piece, 13
- geometry\_utils, 22, 24, 26, 41
- geometry\_utils (AA\_to\_R), 4
- get\_embedded\_font (font\_utils), 10
- get\_shape\_grob\_fn (piecepackr-defunct), 24
- ggplot(), 14
- ggplot2::scale\_x\_continuous(), 15, 42, 43
- ggplot2::scale\_y\_continuous(), 15, 42, 43
  
- grid.piece, 16
- grid.piece(), 7, 27, 36
- grid::gpar(), 32
- gridGeometry::grid.polyclip(), 32
- gridlinesGrob (piecepackr-defunct), 24
- gridpattern::patternGrob(), 32
  
- halmaGrob (piecepackr-defunct), 24
- has\_font (font\_utils), 10
- hexlinesGrob (piecepackr-defunct), 24
  
- inch (pp\_utils), 34
- is\_color\_invisible (pp\_utils), 34
- is\_pp\_cfg (pp\_cfg), 28
  
- kiteGrob (piecepackr-defunct), 24
  
- label\_counting (scale\_x\_piece), 42
- label\_letter (scale\_x\_piece), 42
- lambda, 43
  
- matGrob (piecepackr-defunct), 24
  
- op\_transform, 19
  
- picturePieceGrobFn (basicPieceGrobs), 8
- piece, 21
- piece(), 7, 27, 36, 38
- piece3d, 22
- piece3d(), 7, 27, 36, 38
- piece\_mesh, 25
- piece\_mesh(), 7, 36, 38
- pieceGrob (grid.piece), 16
- pieceGrob(), 15, 27
- piecepackr (piecepackr-package), 2
- piecepackr-defunct, 24
- piecepackr-package, 2
- pmap\_piece, 27
- pmap\_piece(), 18, 36
- pp\_cfg, 13, 28
- pp\_cfg(), 38

pp\_shape, 31  
pp\_utils, 34  
previewLayoutGrob (basicPieceGrobs), 8  
pyramidGrob (piecepackr-defunct), 24  
pyramidTopGrob (basicPieceGrobs), 8

R\_to\_AA (AA\_to\_R), 4  
R\_x (AA\_to\_R), 4  
R\_y (AA\_to\_R), 4  
R\_z (AA\_to\_R), 4  
render\_piece, 35  
render\_piece(), 28  
rgl.material, 24

save\_ellipsoid\_obj, 37  
save\_peg\_doll\_obj (save\_ellipsoid\_obj),  
37  
save\_piece\_images, 39  
save\_piece\_obj, 39  
save\_piece\_obj(), 38  
save\_print\_and\_play, 41  
scale\_x\_piece, 42  
scale\_x\_piece(), 15  
scale\_y\_piece (scale\_x\_piece), 42  
scale\_y\_piece(), 15  
scales::extended\_breaks(), 43  
spdx\_license\_list, 44

to\_degrees (AA\_to\_R), 4  
to\_hexpack (game\_systems), 11  
to\_r (AA\_to\_R), 4  
to\_radians (AA\_to\_R), 4  
to\_subpack (game\_systems), 11  
to\_t (AA\_to\_R), 4  
to\_x (AA\_to\_R), 4  
to\_y (AA\_to\_R), 4  
transformation object, 43  
Trig, 6