Package: mypaintr (via r-universe)

October 22, 2026

Type Package

Title Plot R Graphics Like a Human

Version 0.1.0

Author David Hugh-Jones [aut, cre]

Maintainer David Hugh-Jones <david@example.com>

Description R graphics device for human-like, sketched plotting. Uses libmypaint brushes to draw lines and polygons. Includes functions for plotting ``rough" lines and polygons. Integrates with base and ggplot graphics.

License MIT + file LICENSE

URL https://github.com/hughjonesd/mypaintr, https://hughjonesd.github.io/mypaintr/

BugReports https://github.com/hughjonesd/mypaintr/issues

Encoding UTF-8

SystemRequirements cairo, json-c, pkg-config. Optional system packages: libmypaint, mypaint-brushes

Imports grDevices, ggplot2

NeedsCompilation yes

Roxygen list(markdown = TRUE)

Config/Needs/website pkgdown

Suggests knitr, png, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Config/pak/sysreqs libcairo2-dev pkg-config

Repository https://hughjonesd.r-universe.dev

Date/Publication 2026-06-03 02:48:26 UTC

RemoteUrl https://github.com/hughjonesd/mypaintr

RemoteRef v0.1.0

RemoteSha 2c3a19834f6c8792b76ca0c5c3bcfb75deb822f9

Contents

brush_dirs . . . 2
brush_inputs . . . 3
brush_settings . . . 3
brushes . . . 4
crosshatch . . . 4
geom_mypaint_bar . . . 5
geom_mypaint_col . . . 6
hand . . . 7
hatch . . . 9
jumble . . . 10
knitr_mypaint_hook . . . 11
load_brush . . . 12
mypaint_device . . . 12
mypaint_wrap . . . 14
pressure_flat . . . 15
rough_arrows . . . 16
rough_lines . . . 17
rough_points . . . 18
rough_polygons . . . 19
rough_polypath . . . 20
rough_rect . . . 21
rough_segments . . . 22
set_brush . . . 23
set_hand . . . 24
speed_flat . . . 25
tweak_brush . . . 26
zigzag . . . 32

Index **34**

`brush_dirs` *Discover installed mypaint brush directories*

Description

Discover installed mypaint brush directories

Usage

```
brush_dirs()
```

Value

A character vector of directories containing `.myb` brushes.

See Also

Other brush management: brush_inputs(), brush_settings(), brushes(), load_brush(), set_brush(), tweak_brush()

Examples

```
brush_dirs()
```

`brush_inputs` *libmypaint brush input metadata*

Description

libmypaint brush input metadata

Usage

```
brush_inputs()
```

See Also

Other brush management: brush_dirs(), brush_settings(), brushes(), load_brush(), set_brush(), tweak_brush()

Examples

```
head(brush_inputs())
```

`brush_settings` *libmypaint brush setting metadata*

Description

libmypaint brush setting metadata

Usage

```
brush_settings()
```

See Also

Other brush management: brush_dirs(), brush_inputs(), brushes(), load_brush(), set_brush(), tweak_brush()

Examples

```
head(brush_settings())
```

brushes *List installed mypaint brushes*

Description

List installed mypaint brushes

Usage

```
brushes(paths = default_mypaint_brush_dirs())
```

Arguments

paths	Optional brush directories. Defaults to locally discovered `mypaint-brushes` locations.

Value

A character vector of brush names, relative to the brush root.

See Also

Other brush management: `brush_dirs()`, `brush_inputs()`, `brush_settings()`, `load_brush()`, `set_brush()`, `tweak_brush()`

Examples

```
head(brushes())
```

crosshatch *Cross-hatch fill pattern*

Description

Cross-hatch fill pattern

Usage

```
crosshatch(angle = 45, density = 7, clip = TRUE, padding = 0)
```

Arguments

angle	One or two hatch angles in degrees. If a single angle is supplied, the second pass defaults to `angle + 90`.
density	Approximate line density in lines per inch. Larger values give denser fills.
clip	When `TRUE`, hatch endpoints stay on the shape boundary to reduce overshoot.
padding	Inset from the polygon edge in inches. Positive values leave a small gap between the fill pattern and the boundary.

Value

A fill-pattern object for `draw_rough_*()` helpers and mypaint geoms.

See Also

Other fill patterns: `hatch()`, `jumble()`, `zigzag()`

Examples

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 10))
draw_rough_rect(
  2, 2, 8, 8,
  col = "grey90",
  fill_pattern = crosshatch(angle = c(30, 120), density = 9)
)
```

---

geom_mypaint_bar *Draw rough, brush-rendered bars in ggplot2*

Description

Draw rough, brush-rendered bars in ggplot2

Usage

```
geom_mypaint_bar(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  ...,
  just = 0.5,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fill_pattern = NULL,
  brush = NULL,
  fill_brush = NULL,
  hand = NULL,
  stroke_hand = hand,
  fill_hand = hand,
  auto_solid_bg = NULL
)
```

Arguments

mapping, data, position, just, lineend, linejoin, na.rm, show.legend, inherit.aes	As for ggplot2::geom_col().
stat	The statistical transformation to use. Defaults to "count".
...	Other arguments passed to ggplot2::layer().
fill_pattern	Optional fill pattern created with hatch(), crosshatch(), zigzag(), or jumble().
brush	Stroke brush specification created with tweak_brush(), an installed mypaint brush name, .myb file path, JSON brush string, or NULL for solid borders.
fill_brush	Fill brush specification created with tweak_brush(), an installed mypaint brush name, .myb file path, JSON brush string, or NULL for solid fills.
hand	Optional hand-drawn geometry applied to both outline and hatch by default.
stroke_hand	Optional hand-drawn geometry for the outline.
fill_hand	Optional hand-drawn geometry for the hatch strokes.
auto_solid_bg	Reserved for future parity with device-level style controls.

Value

A ggplot layer.

---

geom_mypaint_col *Draw rough, brush-rendered columns in ggplot2*

Description

This geom owns both the bar outline and the hatch fill, so the shading lines follow the same rough outline rather than the underlying true rectangle.

Usage

```
geom_mypaint_col(
  mapping = NULL,
  data = NULL,
  position = "stack",
  ...,
  just = 0.5,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fill_pattern = NULL,
  brush = NULL,
  fill_brush = NULL,
```

```
  hand = NULL,
  stroke_hand = hand,
  fill_hand = hand,
  auto_solid_bg = NULL
)
```

Arguments

mapping, data, position, just, lineend, linejoin, na.rm, show.legend, inherit.aes	As for ggplot2::geom_col().
...	Other arguments passed to ggplot2::layer().
fill_pattern	Optional fill pattern created with hatch(), crosshatch(), zigzag(), or jumble().
brush	Stroke brush specification created with tweak_brush(), an installed mypaint brush name, .myb file path, JSON brush string, or NULL for solid borders.
fill_brush	Fill brush specification created with tweak_brush(), an installed mypaint brush name, .myb file path, JSON brush string, or NULL for solid fills.
hand	Optional hand-drawn geometry applied to both outline and hatch by default.
stroke_hand	Optional hand-drawn geometry for the outline.
fill_hand	Optional hand-drawn geometry for the hatch strokes.
auto_solid_bg	Reserved for future parity with device-level style controls.

Value

A ggplot layer.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl))) +
    geom_mypaint_bar(fill_pattern = hatch())
}
```

---

hand *Hand-drawn geometry settings*

Description

human_hand() is the same as hand(), but starts from rougher defaults with bow, wobble, width jitter, and human-style pressure and speed profiles enabled.

Usage

```
hand(
  seed = NULL,
  bow = 0,
  wobble = 0,
  multi_stroke = 1L,
  width_jitter = 0,
  endpoint_jitter = 0,
  pressure = pressure_flat(),
  speed = speed_flat(),
  xtilt = 0,
  ytilt = 0,
  barrel_rotation = 0
)

human_hand(
  seed = NULL,
  bow = 0.004,
  wobble = 0.004,
  multi_stroke = 1L,
  width_jitter = 0.08,
  endpoint_jitter = 0,
  pressure = pressure_human(),
  speed = speed_human(),
  xtilt = 0,
  ytilt = 0,
  barrel_rotation = 0
)
```

Arguments

`seed`	Optional random seed used for repeatable geometry.
`bow`	Typical bowing of long strokes as a proportion of segment length.
`wobble`	Low-frequency path wobble as a proportion of segment length.
`multi_stroke`	Number of overdrawn strokes to use.
`width_jitter`	Relative variation in line width between overdrawn strokes.
`endpoint_jitter`	Relative endpoint jitter as a proportion of segment length.
`pressure`	Pressure profile function, typically created with `pressure_flat()`, `pressure_smooth()`, or `pressure_human()`. A single number is treated as `pressure_flat(pressure)`.
`speed`	Speed profile function, typically created with `speed_flat()` or `speed_human()`. A single positive number is treated as `speed_flat(speed)`. Speed profiles affect `mypaint_device()` brush rendering only.
`xtilt, ytilt`	Stylus tilt inputs passed to libmypaint, in its normalized `-1` to `1` range.
`barrel_rotation`	Stylus barrel rotation, in degrees.

Details

hand() defaults to plain, base-R-like geometry with no bowing, wobble, or jitter and flat pressure and speed. human_hand() has different, more human-like defaults, including pressure_human() and speed_human().

Pressure and speed profile functions are called with t, turn, and length, where length is the total stroke length in device units.

The speed, xtilt, ytilt, and barrel_rotation arguments affect only brush rendering on mypaint_device(). They are ignored by standard graphics devices.

Value

An object describing how rough geometry should be generated.

An object describing how rough geometry should be generated.

Examples

```
plot.new()
plot.window(c(0, 10), c(0, 10))
draw_rough_lines(c(0, 10), c(8, 8), lwd = 4, hand = hand())
draw_rough_lines(c(0, 10), c(6, 6), lwd = 4, hand = human_hand())
draw_rough_lines(c(0, 10), c(4, 4), lwd = 4,
                 hand = human_hand(seed = 1,
                   bow = 0.02, wobble = 0.01))
draw_rough_lines(c(0, 10), c(2, 2), lwd = 4,
                 hand = human_hand(seed = 1,
                   pressure = pressure_smooth(0.7, taper = 0.5)))
draw_rough_lines(c(0, 10), c(1, 1), lwd = 4,
                 hand = human_hand(seed = 1,
                   pressure = pressure_human()))
```

---

hatch *Hatch fill pattern*

Description

Hatch fill pattern

Usage

```
hatch(angle = 45, density = 8, clip = TRUE, padding = 0)
```

Arguments

angle	Base hatch angle in degrees.
density	Approximate line density in lines per inch. Larger values give denser fills.
clip	When TRUE, hatch endpoints stay on the shape boundary to reduce overshoot.
padding	Inset from the polygon edge in inches. Positive values leave a small gap between the fill pattern and the boundary.

Value

A fill-pattern object for `draw_rough_*()` helpers and mypaint geoms.

See Also

Other fill patterns: `crosshatch()`, `jumble()`, `zigzag()`

Examples

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 10))
draw_rough_rect(2, 2, 8, 8, col = "red",
                fill_pattern = hatch(density = 10))
```

`jumble` *Jumble fill pattern*

Description

Jumble fill pattern

Usage

```
jumble(
  angle = 0,
  density = 5,
  radius = 0.76/density,
  wobble = 0.2,
  clip = TRUE,
  padding = 0
)
```

Arguments

`angle`	Base angle in degrees for the underlying guide lines.
`density`	Approximate line density in lines per inch. Larger values give denser fills.
`radius`	Loop radius in inches. Defaults to `0.76 / density`, so the loops are sized as a fraction of the line spacing.
`wobble`	Amount of irregularity in the loop shapes, spacing, and size. Larger values give less even, more varied loops.
`clip`	When `TRUE`, split loop paths at the shape boundary.
`padding`	Inset from the polygon edge in inches. Positive values leave a small gap between the fill pattern and the boundary.

Value

A fill-pattern object for `draw_rough_*()` helpers and mypaint geoms.

See Also

Other fill patterns: crosshatch(), hatch(), zigzag()

Examples

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 10))
draw_rough_rect(2, 2, 8, 8, col = "red", fill_pattern = jumble())
```

---

knitr_mypaint_hook *Create a knitr chunk hook for live mypaint rendering*

Description

The returned hook opens mypaint_device() before chunk evaluation and injects the generated PNG files afterward. This avoids knitr's normal plot replay path, which does not preserve device-local style changes such as set_hand() and set_brush().

Usage

```
knitr_mypaint_hook(...)
```

Arguments

...	Default arguments passed through to mypaint_device() when the hook opens a device. Chunk-specific overrides can be supplied in the chunk option mypaint.args as a named list. Because these arguments are applied when the device opens, use this hook or mypaint.args to set chunk defaults; set_brush() and set_hand() still work within the chunk after the device is open.

Details

Register it with knitr::knit_hooks$set(mypaint = knitr_mypaint_hook(...)) and then enable it for chunks with mypaint = TRUE. Chunks should also set fig.keep = "none" and fig.ext = "png". If a chunk explicitly sets dev=, the hook is skipped and knitr's normal device handling is used.

Value

A function suitable for knitr::knit_hooks$set().

Examples

```
if (requireNamespace("knitr", quietly = TRUE)) {
  hook <- knitr_mypaint_hook(brush = "deevad/2B_pencil")
  print(is.function(hook))
}
```

load_brush *Load an installed mypaint brush*

Description

Load an installed mypaint brush

Usage

```
load_brush(brush, paths = default_mypaint_brush_dirs(), normalize = "all")
```

Arguments

brush	Brush name like "classic/pencil" or a path to a .myb file.
paths	Optional brush directories. Defaults to locally discovered mypaint-brushes locations.
normalize	One of "all", "size", "tracking", or "none". Defaults to "all"; use "none" to explicitly bypass normalization.

Value

A reusable brush specification object.

See Also

Other brush management: brush_dirs(), brush_inputs(), brush_settings(), brushes(), set_brush(), tweak_brush()

Examples

```
if (length(brushes())) {
  x <- load_brush(brushes()[[1]])
  stopifnot(inherits(x, "mypaintr_brush"))
}
```

mypaint_device *Open a libmypaint-backed graphics device*

Description

Open a libmypaint-backed graphics device

Usage

```
mypaint_device(
  filename = NULL,
  width = 7,
  height = 7,
  res = 144,
  pointsize = 12,
  bg = "white",
  brush = NULL,
  fill_brush = NULL,
  hand = NULL,
  stroke_hand = NULL,
  fill_hand = NULL,
  auto_solid_bg = TRUE
)
```

Arguments

filename	Output PNG filename. If it contains \%d, pages are numbered.
width, height	Device size in inches.
res	Resolution in pixels per inch.
pointsize	Base pointsize.
bg	Background colour.
brush	Stroke brush specification created with tweak_brush(), an installed mypaint brush name, .myb file path, JSON brush string, or NULL for solid strokes.
fill_brush	Optional fill brush spec. Defaults to brush when not supplied. Use explicit NULL for solid fills.
hand	Optional hand-drawn geometry spec applied to both stroke and fill primitives by default.
stroke_hand	Optional hand-drawn geometry spec for strokes.
fill_hand	Optional hand-drawn geometry spec for fills.
auto_solid_bg	Draw large fills that match the device background using normal Cairo rendering even when fill_style = "brush".

Value

Opens a graphics device and returns NULL invisibly.

Examples

```
out <- tempfile("mypaint.png")
mypaint_device(out, width = 4, height = 3, bg = "ivory")
try(set_brush("classic/pen"), silent = TRUE)
plot(
  1:10,
  col = "steelblue",
```

```
  pch = 16,
  cex = 1.4
)
dev.off()
unlink(out)
```

`mypaint_wrap` *Wrap a grid grob, ggplot layer, or ggplot theme element with scoped mypaint styling*

Description

`mypaint_wrap()` applies temporary mypaintr brush and hand settings while the wrapped object is drawn, then restores the previous device style. It can wrap grid grobs, ggplot2 layers, and ggplot2 theme elements. This makes it useful for direct `grid::grid.draw()` workflows, for ggplot calls such as `ggplot(...) + mypaint_wrap(geom_line(...), ...)`, and for theme elements such as `theme(panel.grid = mypaint_wrap(element_line(), ...))`.

Usage

```
mypaint_wrap(
  object,
  brush = NULL,
  fill_brush = NULL,
  hand = NULL,
  stroke_hand = hand,
  fill_hand = hand,
  auto_solid_bg = NULL
)
```

Arguments

`object`	A grid grob, ggplot2 layer, or ggplot2 theme element.
`brush`	Stroke brush specification created with `tweak_brush()`, an installed mypaint brush name, `.myb` file path, JSON brush string, or `NULL` for solid strokes.
`fill_brush`	Fill brush specification created with `tweak_brush()`, an installed mypaint brush name, `.myb` file path, JSON brush string, or `NULL` for solid fills.
`hand`	Optional hand-drawn geometry applied to both stroke and fill by default.
`stroke_hand`	Optional hand-drawn geometry for strokes.
`fill_hand`	Optional hand-drawn geometry for fills.
`auto_solid_bg`	Optional override for background-like fills.

Value

An object of the same general kind as `object`.

Examples

```
line <- grid::linesGrob(c(0.1, 0.9), c(0.2, 0.8))
if ("classic/pen" %in% brushes()) {
  wrapped <- mypaint_wrap(line, brush = "classic/pen", hand = hand())
}

if (requireNamespace("ggplot2", quietly = TRUE) &&
    "classic/pen" %in% brushes()) {
  ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    mypaint_wrap(ggplot2::geom_line(), brush = "classic/pen", hand = hand())

  ggplot2::theme(
    panel.grid = mypaint_wrap(ggplot2::element_line(), brush = "classic/pen")
  )
}
```

---

pressure_flat *Pressure profiles for hand-drawn strokes*

Description

Pressure profiles for hand-drawn strokes

Usage

```
pressure_flat(value = 1)

pressure_smooth(value = 1, taper = 1, turn_taper = 0.35)

pressure_human(
  value = 1,
  taper = 0.6,
  start = 0.35,
  end = 0.55,
  peak = 0.45,
  turn_taper = 0.35
)

pressure_dashed(value = 1, pattern = c(24, 12))

pressure_dashed_smooth(value = 1, pattern = c(24, 12), taper = 1)
```

Arguments

value	Maximum pressure supplied to the brush, in the range 0 to 1.
taper	How strongly pressure changes over the stroke. 0 keeps the pressure flat; 1 applies the full profile shape.

turn_taper	How strongly pressure is reduced at sharp turns.
start, end	Relative pressure at the start and end of a human-style stroke when taper = 1.
peak	Position of peak pressure along the stroke, in the range 0 to 1.
pattern	Alternating on/off dash lengths in device units. The default uses a 2:1 on/off ratio like base R's dashed line, at a moderate brush-scale length.

Value

A pressure-profile function for the pressure argument of hand() and human_hand(). Custom functions can also be supplied directly; they must accept t, normalized stroke progress in the range 0 to 1, turn in the range 0 to 1, and length, the total stroke length in device units. turn describes local path curvature: 0 is straight, larger values are sharper corners, and values near 1 are near reversals. Custom functions must be vectorized over t and turn, and return either length 1 or length(t).

Examples

```
plot.new()
plot.window(c(0, 10), c(0, 10))
draw_rough_lines(c(1, 9), c(8, 8), lwd = 5,
                 hand = hand(pressure = pressure_flat(0.5)))
draw_rough_lines(c(1, 9), c(5, 5), lwd = 5,
                 hand = hand(pressure = pressure_smooth()))
draw_rough_lines(c(1, 9), c(2, 2), lwd = 5,
                 hand = hand(pressure = pressure_human()))
```

---

rough_arrows *Compute or draw rough arrows*

Description

Compute or draw rough arrows

Usage

```
rough_arrows(x0, y0, x1, y1, length = 0.25, angle = 30, code = 2, hand = NULL)

draw_rough_arrows(
  x0,
  y0,
  x1,
  y1,
  length = 0.25,
  angle = 30,
  code = 2,
  hand = NULL,
  ...
)
```

Arguments

x0, y0	Arrow starts.
x1, y1	Arrow ends.
length	Arrowhead length in inches, as in graphics::arrows().
angle	Arrowhead angle in degrees.
code	Integer code indicating where heads are drawn: 0 for none, 1 at the start, 2 at the end, 3 at both ends.
hand	Hand-drawn geometry settings created with hand().
...	Graphics parameters passed to graphics::lines().

Value

A list with x, y, and id components describing roughened polyline geometry for arrow shafts and heads.

See Also

Other rough drawing helpers: rough_lines(), rough_points(), rough_polygons(), rough_polypath(), rough_rect(), rough_segments()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_arrows(8, 2, 2, 8, hand = human_hand())
```

---

rough_lines *Compute or draw rough connected lines*

Description

Compute or draw rough connected lines

Usage

```
rough_lines(x, y = NULL, hand = NULL)

draw_rough_lines(x, y = NULL, hand = NULL, ...)
```

Arguments

x, y	Coordinates as for graphics::lines().
hand	Hand-drawn geometry settings created with hand().
...	Graphics parameters passed to graphics::lines().

Value

A list with x, y, and id components describing roughened polyline geometry for each connected run.

See Also

Other rough drawing helpers: rough_arrows(), rough_points(), rough_polygons(), rough_polypath(), rough_rect(), rough_segments()

Examples

```
y <- c(2, 5, 4, 7, 6, 8)
plot(1:6, y, type = "n")
draw_rough_lines(1:6, y, hand = human_hand(multi_stroke = 2))
```

---

rough_points *Compute or draw rough points*

Description

Compute or draw rough points

Usage

```
rough_points(x, y = NULL, hand = NULL)

draw_rough_points(x, y = NULL, hand = NULL, ...)
```

Arguments

x, y	Point coordinates as for graphics::points().
hand	Hand-drawn geometry settings created with hand().
...	Graphics parameters passed to graphics::points().

Value

A list with jittered x and y point locations.

See Also

Other rough drawing helpers: rough_arrows(), rough_lines(), rough_polygons(), rough_polypath(), rough_rect(), rough_segments()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_points(1:10, 1:10,
                  hand = human_hand(),
                  pch = 16, cex = 1.4)
```

rough_polygons *Compute or draw rough polygons*

Description

Compute or draw rough polygons

Usage

```
rough_polygons(x, y = NULL, hand = NULL)

draw_rough_polygons(
  x,
  y = NULL,
  hand = NULL,
  col = NA,
  border = graphics::par("fg"),
  fill_pattern = NULL,
  ...
)
```

Arguments

x, y	Polygon coordinates.
hand	Hand-drawn geometry settings created with hand().
col	Fill colour. When visible and fill_pattern is NULL, a solid fill is drawn.
border	Border colour.
fill_pattern	Optional fill pattern created with hatch(), crosshatch(), zigzag(), or jumble().
...	Graphics parameters passed to graphics::lines().

Value

A list with x and y components containing a roughened closed outline suitable for plotting with graphics::lines().

See Also

Other rough drawing helpers: rough_arrows(), rough_lines(), rough_points(), rough_polypath(), rough_rect(), rough_segments()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_polygons(c(2, 5, 8, 3), c(2, 7, 5, 1),
                    hand = human_hand(),
                    col = "red",
                    fill_pattern = zigzag())
```

rough_polypath *Compute or draw a rough multipath*

Description

Compute or draw a rough multipath

Usage

```
rough_polypath(
  x,
  y = NULL,
  id = NULL,
  rule = c("winding", "evenodd"),
  hand = NULL
)

draw_rough_polypath(
  x,
  y = NULL,
  id = NULL,
  rule = c("winding", "evenodd"),
  hand = NULL,
  col = NA,
  border = graphics::par("fg"),
  fill_pattern = NULL,
  ...
)
```

Arguments

x, y	Coordinates as for graphics::polypath().
id	Optional path ids. Consecutive points with the same id belong to one closed ring.
rule	Fill rule, "winding" or "evenodd".
hand	Hand-drawn geometry settings created with hand().
col	Fill colour. When visible and fill_pattern is NULL, a solid fill is drawn.
border	Border colour.
fill_pattern	Optional fill pattern created with hatch(), crosshatch(), zigzag(), or jumble().
...	Graphics parameters passed to graphics::lines().

Value

A list with x, y, id, and rule components describing roughened closed rings.

See Also

Other rough drawing helpers: rough_arrows(), rough_lines(), rough_points(), rough_polygons(), rough_rect(), rough_segments()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_polypath(c(2, 8, 8, 2, 4, 6, 6, 4),
                    c(2, 2, 8, 8, 4, 4, 6, 6),
                    id = c(rep(1, 4), rep(2, 4)),
                    rule = "evenodd",
                    hand = human_hand(),
                    col = "red",
                    fill_pattern = hatch(density = 9))
```

---

rough_rect *Compute or draw a rough rectangle*

Description

Compute or draw a rough rectangle

Usage

```
rough_rect(x0, y0, x1, y1, hand = NULL)

draw_rough_rect(
  x0,
  y0,
  x1,
  y1,
  hand = NULL,
  col = NA,
  border = graphics::par("fg"),
  fill_pattern = NULL,
  ...
)
```

Arguments

x0, y0	Rectangle corner.
x1, y1	Opposite rectangle corner.
hand	Hand-drawn geometry settings created with hand().
col	Fill colour. When visible and fill_pattern is NULL, a solid fill is drawn.
border	Border colour.
fill_pattern	Optional fill pattern created with hatch(), crosshatch(), zigzag(), or jumble().
...	Graphics parameters passed to graphics::lines().

Value

A list with x and y components containing a roughened closed outline suitable for plotting with graphics::lines().

See Also

Other rough drawing helpers: rough_arrows(), rough_lines(), rough_points(), rough_polygons(), rough_polypath(), rough_segments()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_rect(2, 2, 8, 7,
                hand = human_hand(),
                col = "red",
                fill_pattern = crosshatch(padding = 0.05))
```

---

rough_segments *Compute or draw rough segments*

Description

Compute or draw rough segments

Usage

```
rough_segments(x0, y0, x1, y1, hand = NULL)

draw_rough_segments(x0, y0, x1, y1, hand = NULL, ...)
```

Arguments

x0, y0	Segment starts.
x1, y1	Segment ends.
hand	Hand-drawn geometry settings created with hand().
...	Graphics parameters passed to graphics::lines().

Value

A list with x, y, and id components describing roughened polyline geometry for each segment.

See Also

Other rough drawing helpers: rough_arrows(), rough_lines(), rough_points(), rough_polygons(), rough_polypath(), rough_rect()

Examples

```
plot(1:10, 1:10, type = "n")
draw_rough_segments(1:3, 2:4, 4:6, c(8, 5, 7), hand = human_hand())
```

`set_brush` *Set the active mypaintr brush*

Description

Set the active mypaintr brush

Usage

```
set_brush(
  brush = NULL,
  type = c("both", "stroke", "fill"),
  auto_solid_bg = NULL
)
```

Arguments

`brush`	Brush specification created with `tweak_brush()`, an installed brush name, `.myb` file path, JSON brush string, or `NULL` to switch the selected type back to solid rendering.
`type`	Which rendering channel to update: `"both"`, `"stroke"`, or `"fill"`.
`auto_solid_bg`	Optional override for background-like fills.

Value

`NULL`, invisibly. If the active graphics device is not `mypaint_device()`, this emits a warning and has no effect.

See Also

Other brush management: `brush_dirs()`, `brush_inputs()`, `brush_settings()`, `brushes()`, `load_brush()`, `tweak_brush()`

Examples

```
ex_file <- tempfile(fileext = ".png")
mypaint_device(ex_file)

plot.new()
plot.window(c(0, 10), c(0, 10))
brushes <- c("classic/pen", "classic/charcoal", "classic/ink_blot",
             "ramon/2B_pencil")
for (idx in seq_along(brushes)) {
  set_brush(brushes[idx])
```

```
  lines(c(1, 9), c(2 * idx, 2 * idx), lwd = 2)
}

dev.off()
img <- png::readPNG(ex_file)
grid::grid.raster(img)
```

---

set_hand *Set the active hand*

Description

Set the active hand

Usage

```
set_hand(hand = NULL, type = c("both", "stroke", "fill"))
```

Arguments

hand	Hand-drawn geometry created with hand(), or NULL to disable it for the selected type. This disables rough path perturbation only; it does not disable the active brush, and note that some brushes have their own internal wobbly pathing! Use set_brush() as well if you want fully plain, solid rendering.
type	Which rendering channel to update: "both", "stroke", or "fill".

Value

NULL, invisibly. If the active graphics device is not mypaint_device(), this emits a warning and has no effect.

Examples

```
ex_file <- tempfile(fileext = ".png")
mypaint_device(ex_file)

plot.new()
plot.window(c(0, 10), c(0, 10))
set_hand(hand())
rect(1, 1, 5, 5, col = "darkred", density = 5)
set_hand(human_hand())
rect(5, 5, 9, 9, col = "darkgreen", density = 5)

dev.off()
img <- png::readPNG(ex_file)
grid::grid.raster(img)
```

speed_flat *Speed profiles for hand-drawn strokes*

Description

Speed profiles for hand-drawn strokes

Usage

```
speed_flat(value = 1)

speed_human(
  value = 1,
  taper = 0.65,
  start = 0.55,
  end = 0.65,
  peak = 1.4,
  peak_at = 0.45,
  turn_slowdown = 0.45,
  min = 0.05
)
```

Arguments

value	Base speed multiplier. 1 preserves the default distance-based timing heuristic, values greater than 1 draw faster, and values below 1 draw slower.
taper	How strongly speed changes over the stroke. 0 keeps the speed flat; 1 applies the full profile shape.
start, end	Relative speed at the start and end of a human-style stroke when taper = 1.
peak	Relative peak speed reached during the stroke when taper = 1.
peak_at	Position of peak speed along the stroke, in the range 0 to 1.
turn_slowdown	How strongly speed is reduced at sharp turns.
min	Minimum speed multiplier returned by the profile.

Value

A speed-profile function for the speed argument of hand() and human_hand(). Custom functions can also be supplied directly; they must accept t, normalized stroke progress in the range 0 to 1, turn in the range 0 to 1, and length, the total stroke length in device units. Speed profiles return positive speed multipliers. Custom functions must be vectorized over t and turn, and return either length 1 or length(t).

Examples

```
plot.new()
plot.window(c(0, 10), c(0, 10))
draw_rough_lines(c(1, 9), c(8, 8), lwd = 5,
                 hand = hand(speed = speed_flat(0.5)))
draw_rough_lines(c(1, 9), c(5, 5), lwd = 5,
                 hand = hand(speed = speed_human()))
```

---

tweak_brush *Tweak a brush specification*

Description

Tweak a brush specification

Usage

```
tweak_brush(
  brush,
  normalize = "all",
  opaque,
  opaque_multiply,
  opaque_linearize,
  radius_logarithmic,
  hardness,
  anti_aliasing,
  dabs_per_basic_radius,
  dabs_per_actual_radius,
  dabs_per_second,
  radius_by_random,
  speed1_slowness,
  speed2_slowness,
  speed1_gamma,
  speed2_gamma,
  offset_by_random,
  offset_by_speed,
  offset_by_speed_slowness,
  slow_tracking,
  slow_tracking_per_dab,
  tracking_noise,
  color_h,
  color_s,
  color_v,
  restore_color,
  change_color_h,
  change_color_l,
  change_color_hsl_s,
```

```
  change_color_v,
  change_color_hsv_s,
  smudge,
  smudge_length,
  smudge_radius_log,
  eraser,
  stroke_threshold,
  stroke_duration_logarithmic,
  stroke_holdtime,
  custom_input,
  custom_input_slowness,
  elliptical_dab_ratio,
  elliptical_dab_angle,
  direction_filter,
  lock_alpha,
  colorize,
  snap_to_pixel,
  pressure_gain_log,
  gridmap_scale,
  gridmap_scale_x,
  gridmap_scale_y,
  smudge_length_log,
  smudge_bucket,
  smudge_transparency,
  offset_y,
  offset_x,
  offset_angle,
  offset_angle_asc,
  offset_angle_view,
  offset_angle_2,
  offset_angle_2_asc,
  offset_angle_2_view,
  offset_angle_adj,
  offset_multiplier,
  posterize,
  posterize_num,
  paint_mode
)
```

Arguments

brush	Installed brush name, `.myb` file path, JSON brush string, or another `tweak_brush()` object.
normalize	One of `"all"`, `"size"`, `"tracking"`, or `"none"`.
opaque	0 means brush is transparent, 1 fully visible (also known as alpha or opacity)
opaque_multiply	This gets multiplied with opaque. You should only change the pressure input of this setting. Use 'opaque' instead to make opacity depend on speed. This

setting is responsible to stop painting when there is zero pressure. This is just a convention, the behaviour is identical to 'opaque'.

`opaque_linearize`
: Correct the nonlinearity introduced by blending multiple dabs on top of each other. This correction should get you a linear ("natural") pressure response when pressure is mapped to opaque_multiply, as it is usually done. 0.9 is good for standard strokes, set it smaller if your brush scatters a lot, or higher if you use dabs_per_second. 0.0 the opaque value above is for the individual dabs 1.0 the opaque value above is for the final brush stroke, assuming each pixel gets (dabs_per_radius*2) brushdabs on average during a stroke

`radius_logarithmic`
: Basic brush radius (logarithmic) 0.7 means 2 pixels 3.0 means 20 pixels

`hardness`
: Hard brush-circle borders (setting to zero will draw nothing). To reach the maximum hardness, you need to disable Pixel feather.

`anti_aliasing`
: This setting decreases the hardness when necessary to prevent a pixel staircase effect (aliasing) by making the dab more blurred. 0.0 disable (for very strong erasers and pixel brushes) 1.0 blur one pixel (good value) 5.0 notable blur, thin strokes will disappear

`dabs_per_basic_radius`
: How many dabs to draw while the pointer moves a distance of one brush radius (more precise: the base value of the radius)

`dabs_per_actual_radius`
: Same as above, but the radius actually drawn is used, which can change dynamically

`dabs_per_second`
: Dabs to draw each second, no matter how far the pointer moves

`radius_by_random`
: Alter the radius randomly each dab. You can also do this with the by_random input on the radius setting. If you do it here, there are two differences: 1) the opaque value will be corrected such that a big-radius dabs is more transparent 2) it will not change the actual radius seen by dabs_per_actual_radius

`speed1_slowness`
: How slow the input fine speed is following the real speed 0.0 change immediately as your speed changes (not recommended, but try it)

`speed2_slowness`
: Same as 'fine speed filter', but note that the range is different

`speed1_gamma`
: This changes the reaction of the 'fine speed' input to extreme physical speed. You will see the difference best if 'fine speed' is mapped to the radius. -8.0 very fast speed does not increase 'fine speed' much more +8.0 very fast speed increases 'fine speed' a lot For very slow speed the opposite happens.

`speed2_gamma`
: Same as 'fine speed gamma' for gross speed

`offset_by_random`
: Add a random offset to the position where each dab is drawn 0.0 disabled 1.0 standard deviation is one basic radius away <0.0 negative values produce no jitter

`offset_by_speed`
: Change position depending on pointer speed = 0 disable > 0 draw where the pointer moves to < 0 draw where the pointer comes from

`offset_by_speed_slowness`
: How slow the offset goes back to zero when the cursor stops moving

`slow_tracking`
: Slowdown pointer tracking speed. 0 disables it, higher values remove more jitter in cursor movements. Useful for drawing smooth, comic-like outlines.

`slow_tracking_per_dab`
: Similar as above but at brushdab level (ignoring how much time has passed if brushdabs do not depend on time)

`tracking_noise`
: Add randomness to the mouse pointer; this usually generates many small lines in random directions; maybe try this together with 'slow tracking'

`color_h`
: Color hue

`color_s`
: Color saturation

`color_v`
: Color value (brightness, intensity)

`restore_color`
: When selecting a brush, the color can be restored to the color that the brush was saved with. 0.0 do not modify the active color when selecting this brush 0.5 change active color towards brush color 1.0 set the active color to the brush color when selected

`change_color_h`
: Change color hue. -0.1 small clockwise color hue shift 0.0 disable 0.5 counter-clockwise hue shift by 180 degrees

`change_color_l`
: Change the color lightness using the HSL color model. -1.0 blacker 0.0 disable 1.0 whiter

`change_color_hsl_s`
: Change the color saturation using the HSL color model. -1.0 more grayish 0.0 disable 1.0 more saturated

`change_color_v`
: Change the color value (brightness, intensity) using the HSV color model. HSV changes are applied before HSL. -1.0 darker 0.0 disable 1.0 brigher

`change_color_hsv_s`
: Change the color saturation using the HSV color model. HSV changes are applied before HSL. -1.0 more grayish 0.0 disable 1.0 more saturated

`smudge`
: Paint with the smudge color instead of the brush color. The smudge color is slowly changed to the color you are painting on. 0.0 do not use the smudge color 0.5 mix the smudge color with the brush color 1.0 use only the smudge color

`smudge_length`
: This controls how fast the smudge color becomes the color you are painting on. 0.0 immediately update the smudge color (requires more CPU cycles because of the frequent color checks) 0.5 change the smudge color steadily towards the canvas color 1.0 never change the smudge color

`smudge_radius_log`
: This modifies the radius of the circle where color is picked up for smudging. 0.0 use the brush radius -0.7 half the brush radius (fast, but not always intuitive) +0.7 twice the brush radius +1.6 five times the brush radius (slow performance)

`eraser`
: how much this tool behaves like an eraser 0.0 normal painting 1.0 standard eraser 0.5 pixels go towards 50% transparency

`stroke_threshold`
: How much pressure is needed to start a stroke. This affects the stroke input only. MyPaint does not need a minimum pressure to start drawing.

`stroke_duration_logarithmic`
: How far you have to move until the stroke input reaches 1.0. This value is logarithmic (negative values will not invert the process).

`stroke_holdtime`
: This defines how long the stroke input stays at 1.0. After that it will reset to 0.0 and start growing again, even if the stroke is not yet finished. 2.0 means twice as long as it takes to go from 0.0 to 1.0 9.9 or higher stands for infinite

`custom_input`
: Set the custom input to this value. If it is slowed down, move it towards this value (see below). The idea is that you make this input depend on a mixture of pressure/speed/whatever, and then make other settings depend on this 'custom input' instead of repeating this combination everywhere you need it. If you make it change 'by random' you can generate a slow (smooth) random input.

`custom_input_slowness`
: How slow the custom input actually follows the desired value (the one above). This happens at brushdab level (ignoring how much time has passed, if brushdabs do not depend on time). 0.0 no slowdown (changes apply instantly)

`elliptical_dab_ratio`
: Aspect ratio of the dabs; must be >= 1.0, where 1.0 means a perfectly round dab.

`elliptical_dab_angle`
: Angle by which elliptical dabs are tilted 0.0 horizontal dabs 45.0 45 degrees, turned clockwise 180.0 horizontal again

`direction_filter`
: A low value will make the direction input adapt more quickly, a high value will make it smoother

`lock_alpha`
: Do not modify the alpha channel of the layer (paint only where there is paint already) 0.0 normal painting 0.5 half of the paint gets applied normally 1.0 alpha channel fully locked

`colorize`
: Colorize the target layer, setting its hue and saturation from the active brush color while retaining its value and alpha.

`snap_to_pixel`
: Snap brush dab's center and its radius to pixels. Set this to 1.0 for a thin pixel brush.

`pressure_gain_log`
: This changes how hard you have to press. It multiplies tablet pressure by a constant factor.

`gridmap_scale`
: Changes the overall scale that the GridMap brush input operates on. Logarithmic (same scale as brush radius). A scale of 0 will make the grid 256x256 pixels.

`gridmap_scale_x`
: Changes the scale that the GridMap brush input operates on - affects X axis only. The range is 0-5x. This allows you to stretch or compress the GridMap pattern.

`gridmap_scale_y`
: Changes the scale that the GridMap brush input operates on - affects Y axis only. The range is 0-5x. This allows you to stretch or compress the GridMap pattern.

`smudge_length_log`
: Logarithmic multiplier for the "Smudge length" value. Useful to correct for high-definition/large brushes with lots of dabs. The longer the smudge length the more a color will spread and will also boost performance dramatically, as the canvas is sampled less often

`smudge_bucket`
: There are 256 buckets that each can hold a color picked up from the canvas. You can control which bucket to use to improve variability and realism of the brush. Especially useful with the "Custom input" setting to correlate buckets with other settings such as offsets.

`smudge_transparency`
: Control how much transparency is picked up and smudged, similar to lock alpha. 1.0 will not move any transparency. 0.5 will move only 50% transparency and above. 0.0 will have no effect. Negative values do the reverse

`offset_y`
: Moves the dabs up or down based on canvas coordinates.

`offset_x`
: Moves the dabs left or right based on canvas coordinates.

`offset_angle`
: Follows the stroke direction to offset the dabs to one side.

`offset_angle_asc`
: Follows the tilt direction to offset the dabs to one side. Requires Tilt.

`offset_angle_view`
: Follows the view orientation to offset the dabs to one side.

`offset_angle_2`
: Follows the stroke direction to offset the dabs, but to both sides of the stroke.

`offset_angle_2_asc`
: Follows the tilt direction to offset the dabs, but to both sides of the stroke. Requires Tilt.

`offset_angle_2_view`
: Follows the view orientation to offset the dabs, but to both sides of the stroke.

`offset_angle_adj`
: Change the Angular Offset angle from the default, which is 90 degrees.

`offset_multiplier`
: Logarithmic multiplier for X, Y, and Angular Offset settings.

`posterize`
: Strength of posterization, reducing number of colors based on the "Posterization levels" setting, while retaining alpha.

`posterize_num`
: Number of posterization levels (divided by 100). 0.05 = 5 levels, 0.2 = 20 levels, etc. Values above 0.5 may not be noticeable.

`paint_mode`
: Subtractive spectral color mixing mode. 0.0 no spectral mixing 1.0 only spectral mixing

Value

A reusable brush specification object.

See Also

Other brush management: `brush_dirs()`, `brush_inputs()`, `brush_settings()`, `brushes()`, `load_brush()`, `set_brush()`

Examples

```
ex_file <- tempfile(fileext = ".png")
mypaint_device(ex_file)

plot.new()
plot.window(c(0, 10), c(0, 10))
rect(2, 0, 4, 10, col = "orange")
pen <- load_brush("classic/pen")
set_brush(pen)
abline(h = 9, lwd = 3)
set_brush(tweak_brush(pen, dabs_per_actual_radius = 0.5))
abline(h = 7, lwd = 3)
set_brush(tweak_brush(pen, radius_logarithmic = 1.5))
abline(h = 5, lwd = 3)
set_brush(tweak_brush(pen, opaque = 0.5))
abline(h = 3, lwd = 3)
set_brush(tweak_brush(pen, radius_by_random = 0.2))
abline(h = 1, lwd = 3)

dev.off()
img <- png::readPNG(ex_file)
grid::grid.raster(img)
```

---

zigzag *Zigzag fill pattern*

Description

Zigzag fill pattern

Usage

```
zigzag(angle = 45, density = 6, clip = TRUE, padding = 0)
```

Arguments

angle	Base hatch angle in degrees.
density	Approximate line density in lines per inch. Larger values give denser fills.
clip	When TRUE, zigzag endpoints stay on the shape boundary to reduce overshoot.
padding	Inset from the polygon edge in inches. Positive values leave a small gap between the fill pattern and the boundary.

Value

A fill-pattern object for `draw_rough_*()` helpers and mypaint geoms.

See Also

Other fill patterns: `crosshatch()`, `hatch()`, `jumble()`

Examples

```
plot.new()
plot.window(xlim = c(0, 10), ylim = c(0, 10))
draw_rough_rect(2, 2, 8, 8, col = "red",
                fill_pattern = zigzag(density = 7))
```

Index

∗ **brush management**
  brush_dirs, 2
  brush_inputs, 3
  brush_settings, 3
  brushes, 4
  load_brush, 12
  set_brush, 23
  tweak_brush, 26
∗ **fill patterns**
  crosshatch, 4
  hatch, 9
  jumble, 10
  zigzag, 32
∗ **pressure profiles**
  pressure_flat, 15
∗ **rough drawing helpers**
  rough_arrows, 16
  rough_lines, 17
  rough_points, 18
  rough_polygons, 19
  rough_polypath, 20
  rough_rect, 21
  rough_segments, 22
∗ **speed profiles**
  speed_flat, 25

brush_dirs, 2
brush_dirs(), *3, 4*, *12*, *23*, *31*
brush_inputs, 3
brush_inputs(), *3, 4*, *12*, *23*, *31*
brush_settings, 3
brush_settings(), *3, 4*, *12*, *23*, *31*
brushes, 4
brushes(), *3*, *12*, *23*, *31*

crosshatch, 4
crosshatch(), *6, 7*, *10, 11*, *19–21*, *32*

draw_rough_arrows *(*rough_arrows*)*, 16
draw_rough_lines *(*rough_lines*)*, 17
draw_rough_points *(*rough_points*)*, 18
draw_rough_polygons *(*rough_polygons*)*, 19
draw_rough_polypath *(*rough_polypath*)*, 20
draw_rough_rect *(*rough_rect*)*, 21
draw_rough_segments *(*rough_segments*)*, 22

geom_mypaint_bar, 5
geom_mypaint_col, 6
ggplot2::geom_col(), *6, 7*
ggplot2::layer(), *6, 7*
graphics::arrows(), *17*
graphics::lines(), *17*, *19–22*
graphics::points(), *18*
graphics::polypath(), *20*

hand, 7
hand(), *7*, *16–22*, *24, 25*
hatch, 9
hatch(), *5–7*, *11*, *19–21*, *32*
human_hand *(*hand*)*, 7
human_hand(), *9*, *16*, *25*

jumble, 10
jumble(), *5–7*, *10*, *19–21*, *32*

knitr_mypaint_hook, 11

load_brush, 12
load_brush(), *3, 4*, *23*, *31*

mypaint_device, 12
mypaint_device(), *8, 9*, *11*, *23, 24*
mypaint_wrap, 14

pressure_dashed *(*pressure_flat*)*, 15
pressure_dashed_smooth *(*pressure_flat*)*, 15
pressure_flat, 15
pressure_flat(), *8*
pressure_human *(*pressure_flat*)*, 15
pressure_human(), *8, 9*

pressure_smooth *(*pressure_flat*)*, 15
pressure_smooth(), *8*

rough_arrows, 16
rough_arrows(), *18, 19*, *21, 22*
rough_lines, 17
rough_lines(), *17–19*, *21, 22*
rough_points, 18
rough_points(), *17–19*, *21, 22*
rough_polygons, 19
rough_polygons(), *17, 18*, *21, 22*
rough_polypath, 20
rough_polypath(), *17–19*, *22*
rough_rect, 21
rough_rect(), *17–19*, *21, 22*
rough_segments, 22
rough_segments(), *17–19*, *21, 22*

set_brush, 23
set_brush(), *3, 4*, *11, 12*, *24*, *31*
set_hand, 24
set_hand(), *11*
speed_flat, 25
speed_flat(), *8*
speed_human *(*speed_flat*)*, 25
speed_human(), *8, 9*

tweak_brush, 26
tweak_brush(), *3, 4*, *6, 7*, *12–14*, *23*, *27*

zigzag, 32
zigzag(), *5–7*, *10, 11*, *19–21*