

Package: skiagd (via r-universe)

October 19, 2026

Title Creative Coding Pipeline for R

Version 0.1.8.6

Description A toy R wrapper for 'rust-skia'
<<https://github.com/rust-skia/rust-skia>> (the Rust crate
'skia_safe' <https://rust-skia.github.io/doc/skia_safe/>, a
binding for 'Skia' <<https://skia.org/>>).

License MIT + file LICENSE

Depends R (>= 4.2)

Imports cli, colorfast, grid, purrr, rlang

Suggests base64enc, dplyr, htmltools, knitr, quarto, testthat (>= 3.0.0), vdiff

VignetteBuilder quarto

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

OS_type unix

Roxygen list(markdown = TRUE)

SystemRequirements GNU make, Cargo (Rust's package manager), rustc,
fontconfig, freetype2

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev make libclang-dev

Repository <https://paithiov909.r-universe.dev>

Date/Publication 2026-09-07 08:59:36 UTC

RemoteUrl <https://github.com/paithiov909/skiagd>

RemoteRef HEAD

RemoteSha ad289e35c3d24eba77cf9278982af5643350d3be

Contents

add_arc	3
add_atlas	5
add_circle	7
add_diff_rect	8
add_line	11
add_path	12
add_png	14
add_point	15
add_rect	17
add_text	19
add_vertices	21
as_nativeraster	23
as_png	24
BlendMode	25
BlurStyle	26
canvas	27
Cap	27
col2rgba	28
create_mapping	28
dev_size	29
draw_img	30
embed_img	30
enable_autocomplete	31
FillType	32
FontStyle	32
freeze	33
ImageFilter	34
Join	35
list_font_families	36
paint	36
PathEffect	37
pictures	39
PointMode	39
RuntimeEffect	40
Shader	41
Style	42
svg_bounds	43
svg_interpolate	44
svg_transform	45
text_info	45
TileMode	46
transform-matrix	47
VertexMode	47

Index

49

add_arc	<i>Add arcs</i>
---------	-----------------

Description

Adds one or more elliptical arcs to an existing picture.

Usage

```
add_arc(
  img,
  ltrb,
  rsx_trans = matrix(c(1, 0, 0, 0, 0, 0), nrow(ltrb), 6, byrow = TRUE),
  angle = matrix(c(0, 360), nrow(ltrb), 2, byrow = TRUE),
  use_center = TRUE,
  ...,
  props = paint()
)
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>ltrb</code>	A numeric matrix (or a data-frame-like object) with 4 numeric columns (left, top, right, bottom), where each row defines the bounding box of the oval that contains the arc.
<code>rsx_trans</code>	A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to: • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction • offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction
<code>angle</code>	A numeric matrix (or a data-frame-like object) with 2 numeric columns giving the sweep angles (in degrees) for each arc.
<code>use_center</code>	A logical value indicating whether to draw the wedge edges, i.e., lines from the oval center to the arc end points (like a "Pac-Man" wedge).
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. • <code>sigma</code> must be a numeric vector of blur sigmas for each shape. • <code>width</code> must be a numeric vector of stroke widths for each shape. • <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Details

The number of arcs is determined by `nrow(ltrb)`. The number of rows of `rsx_trans` and `angle`, and the lengths of `sigma` and `width` (and the number of columns of `color`) must match this number.

The drawing attributes `sigma`, `width`, and `color` can be supplied via `...`. If they are not supplied, they are taken from props and recycled to all arcs.

When `use_center = TRUE` and the sweep in `angle` describes a full circle (e.g., `c(0, 360)`), the geometry includes the wedge edges from the oval's rightmost point toward the center. Depending on style, this may be filled and thus not visually apparent.

Value

A raw vector containing a serialized picture.

Note

- Arcs are drawn from an oval specified by a rounded rectangle internally, and this geometry does not support rotation. If the rotation angle in `rsx_trans` is not zero, this function will error. To draw rotated arcs or more general shapes, use [add_path\(\)](#) instead.

Examples

```
## Not run:
# Arrange arcs by shifting coordinates in `ltrb`
arcs <-
  dplyr::tibble(
    l = c(60, 260, 60),
    t = c(60, 60, 240),
    r = c(220, 420, 220),
    b = c(200, 200, 380),
    angle_start = c(0, 45, 0),
    angle_end = c(270, 270, 360)
  )

canvas("white") |>
  add_arc(
    ltrb = dplyr::select(arcs, l, t, r, b),
    angle = dplyr::select(arcs, angle_start, angle_end),
    use_center = TRUE
  ) |>
  draw_img()

# Arrange arcs by translating the same `ltrb` via `rsx_trans`
arcs <-
  dplyr::tibble(
    l = 60,
    t = 60,
    r = 220,
    b = 200,
    angle_start = c(0, 0, 0),
    angle_end = c(270, 180, 360)
```

```

    )
    rsx_trans <-
      dplyr::tibble(
        sc = 1,
        rot = 0,
        tx = c(0, 200, 0),
        ty = c(0, 0, 180),
        ax = 0,
        ay = 0
      )

    canvas("white") |>
      add_arc(
        ltrb = dplyr::select(arcs, l, t, r, b),
        rsx_trans = rsx_trans,
        angle = dplyr::select(arcs, angle_start, angle_end),
        use_center = TRUE
      ) |>
      draw_img()

    ## End(Not run)

```

add_atlas

Add atlas

Description

Draws a PNG sprite multiple times (as an atlas) onto an existing picture.

Usage

```
add_atlas(img, png, rsx_trans, ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
png	A raw vector of a PNG image to be used as a sprite. This can be created by as_png() from another picture, or read from a .png file using readBin() .
rsx_trans	A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to: • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction • offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction

... For some drawing functions, you can specify `sigma`, `width`, and `color` as named arguments.

- `sigma` must be a numeric vector of blur sigmas for each shape.
- `width` must be a numeric vector of stroke widths for each shape.
- `color` must be an integer matrix with 4 rows (RGBA) and N columns (shapes).

If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.

props A list of painting attributes created by `paint()`.

Details

The number of sprites drawn is determined by `nrow(rsx_trans)`. Each row of `rsx_trans` specifies an RSX transform applied to the sprite (scale, rotation, translation, and anchor offsets).

Value

A raw vector containing a serialized picture.

Note

- To create sprites (or canvases) of a specific size, you may need to fix the size of the current graphics device (e.g., using a dummy device) or supply `canvas_size` via `props = paint(canvas_size = ...)`, because drawing functions use `props[["canvas_size"]]` at each call.

Examples

```
## Not run:
# To fix the canvas size to 48x16, open a dummy device
png(nullfile(), width = 48, height = 16)

# Create a simple arrow sprite (48x16) as a PNG
arrow_png <-
  canvas("transparent") |>
  add_line(
    from = matrix(c(4, 8), ncol = 2),
    to = matrix(c(44, 8), ncol = 2)
  ) |>
  add_line(
    from = matrix(c(
      36, 4,
      44, 8,
      36, 12
    ), ncol = 2, byrow = TRUE),
    to = matrix(c(
      44, 8,
      36, 12,
      44, 8
    ), ncol = 2, byrow = TRUE)
  ) |>
```

```

    as_png()

# Close the dummy device
dev.off()

# Place the sprite with different rotations and positions
# (anchor offsets set to the sprite center: 24, 8)
rsx_trans <-
  dplyr::tibble(
    sc = 1,
    rot = c(0, pi / 6, pi / 3, pi / 2, pi * 3 / 4),
    tx = c(100, 200, 300, 200, 100),
    ty = c(100, 100, 100, 200, 200),
    ax = 24,
    ay = 8
  )

# Open a dummy device again
png(nullfile(), width = 400, height = 300)

img <-
  canvas("white") |>
  add_atlas(arrow_png, rsx_trans = rsx_trans)

# Close the dummy device
dev.off()

# Draw the picture
draw_img(img)

## End(Not run)

```

add_circle

Add circles

Description

Adds one or more circles to an existing picture.

Usage

```
add_circle(img, center, radius, ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
center	A numeric matrix (or a data-frame-like object) with 2 columns (x and y), where each row represents the center of a circle.
radius	A numeric vector of radii, one for each circle.

...

For some drawing functions, you can specify sigma, width, and color as named arguments.

- sigma must be a numeric vector of blur sigmas for each shape.
- width must be a numeric vector of stroke widths for each shape.
- color must be an integer matrix with 4 rows (RGBA) and N columns (shapes).

If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.

props A list of painting attributes created by `paint()`.

Details

The number of circles is determined by `nrow(center)`. The length of radius must match this number.

The drawing attributes sigma, width, and color can be supplied via If they are not supplied, they are taken from props and recycled to all circles.

Value

A raw vector containing a serialized picture.

Examples

```
## Not run:
circles <-
  dplyr::tibble(
    x = c(100, 300),
    y = c(100, 200),
    radius = c(40, 60)
  )

canvas("white") |>
  add_circle(
    center = dplyr::select(circles, x, y),
    radius = dplyr::pull(circles, radius)
  ) |>
  draw_img()

## End(Not run)
```

add_diff_rect

Add difference rectangles

Description

Adds one or more difference rectangles (outer minus inner), optionally with rounded corners.

Usage

```
add_diff_rect(
  img,
  outer,
  inner,
  rsx_trans = matrix(c(1, 0, 0, 0, 0, 0), nrow(outer), 6, byrow = TRUE),
  outer_radii = matrix(0, nrow(outer), 2),
  inner_radii = matrix(0, nrow(inner), 2),
  ...,
  props = paint()
)
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>outer, inner</code>	A numeric matrix (or a data-frame-like object) with 4 numeric columns (left, top, right, bottom), where each row represents a rectangle.
<code>rsx_trans</code>	A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to: • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction • offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction
<code>outer_radii, inner_radii</code>	A numeric matrix (or data-frame-like object) with 2 numeric columns (x and y), where each row represents the corner radii of a rounded rectangle.
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. • <code>sigma</code> must be a numeric vector of blur sigmas for each shape. • <code>width</code> must be a numeric vector of stroke widths for each shape. • <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Details

The number of shapes is determined by `nrow(outer)`. The numbers of rows of `inner`, `rsx_trans`, `outer_radii`, and `inner_radii`, and the lengths of `sigma` and `width` (and the number of columns of `color`) must match this number.

The drawing attributes `sigma`, `width`, and `color` can be supplied via `...`. If they are not supplied, they are taken from `props` and recycled to all shapes.

Value

A raw vector containing a serialized picture.

Note

- As with `add_rect()`, Skia's rounded rectangle does not support rotation. If the rotation angle in `rsx_trans` is not zero, this function will error. To draw rotated rectangles or more general shapes, use `add_path()`.

Examples

```
## Not run:
# Shift rectangles by modifying coordinates directly
ltrb <-
  dplyr::tibble(
    lo = c(60, 280),
    to = c(60, 60),
    ro = c(240, 460),
    bo = c(200, 200),
    li = c(100, 320),
    ti = c(100, 100),
    ri = c(200, 420),
    bi = c(160, 160)
  )

canvas("white") |>
  add_diff_rect(
    outer = dplyr::select(ltrb, lo, to, ro, bo),
    inner = dplyr::select(ltrb, li, ti, ri, bi),
    outer_radii = matrix(c(16, 16), nrow(ltrb), 2, byrow = TRUE),
    inner_radii = matrix(c(8, 8), nrow(ltrb), 2, byrow = TRUE)
  ) |>
  draw_img()

# Use the same geometry and translate via `rsx_trans`
ltrb <-
  dplyr::tibble(
    lo = rep_len(60, 2),
    to = rep_len(60, 2),
    ro = rep_len(240, 2),
    bo = rep_len(200, 2),
    li = rep_len(100, 2),
    ti = rep_len(100, 2),
    ri = rep_len(200, 2),
    bi = rep_len(160, 2)
  )
rsx_trans <-
  dplyr::tibble(
    sc = 1,
    rot = 0,
    tx = c(0, 220),
    ty = c(0, 0),
```

```

    ax = 0,
    ay = 0
  )

  canvas("white") |>
    add_diff_rect(
      outer = dplyr::select(ltrb, lo, to, ro, bo),
      inner = dplyr::select(ltrb, li, ti, ri, bi),
      rsx_trans = rsx_trans
    ) |>
    draw_img()

## End(Not run)

```

add_line

*Add lines***Description**

Adds one or more line segments to an existing picture.

Usage

```
add_line(img, from, to, ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
from	A numeric matrix (or a data-frame-like object) with 2 columns (x and y) where each row represents the start point of a line segment.
to	A numeric matrix (or a data-frame-like object) with 2 columns (x and y), where each row represents the end point of a line segment.
...	For some drawing functions, you can specify sigma, width, and color as named arguments. • sigma must be a numeric vector of blur sigmas for each shape. • width must be a numeric vector of stroke widths for each shape. • color must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.
props	A list of painting attributes created by paint() .

Details

The number of line segments is determined by `nrow(from)`. The number of rows of `to`, and the lengths of `sigma` and `width` (and the number of columns of `color`) must match this number.

The drawing attributes `sigma`, `width`, and `color` can be supplied via `...`. If they are not supplied, they are taken from `props` and recycled to all line segments.

Value

A raw vector containing a serialized picture.

Examples

```
## Not run:
lines <-
  dplyr::tibble(
    x0 = c(50, 50),
    y0 = c(50, 250),
    x1 = c(350, 350),
    y1 = c(50, 250)
  )

canvas("white") |>
  add_line(
    from = dplyr::select(lines, x0, y0),
    to = dplyr::select(lines, x1, y1)
  ) |>
  draw_img()

## End(Not run)
```

add_path

Add SVG paths

Description

Adds one or more SVG paths to an existing picture.

Each path is given as a string in the SVG d attribute syntax, and can be positioned by providing an RSX transform per path via `rsx_trans`.

Usage

```
add_path(
  img,
  path,
  rsx_trans = matrix(c(1, 0, 0, 0, 0, 0), length(path), 6, byrow = TRUE),
  ...,
  props = paint()
)
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>path</code>	A character vector of SVG path notations (the d attribute syntax), e.g. "M45 10 H55 V45 H90 V55 H55 V90 H45 V55 H10 V45 H45 Z".

rsx_trans	A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to: • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction • offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction
...	For some drawing functions, you can specify sigma, width, and color as named arguments. • sigma must be a numeric vector of blur sigmas for each shape. • width must be a numeric vector of stroke widths for each shape. • color must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.
props	A list of painting attributes created by <code>paint()</code> .

Details

The number of paths is determined by `length(path)`. The number of rows of `rsx_trans`, and the lengths of `sigma` and `width` (and the number of columns of `color`) must match this number.

The drawing attributes `sigma`, `width`, and `color` can be supplied via If they are not supplied, they are taken from props and recycled to all paths.

The `FillType` for closed paths is taken from `props[["fill_type"]]` and applied to all paths.

Value

A raw vector containing a serialized picture.

Examples

```
## Not run:
# Single path
canvas("white") |>
  add_path("M45 10 H55 V45 H90 V55 H55 V90 H45 V55 H10 V45 H45 Z") |>
  draw_img()

# Place the same path multiple times using `rsx_trans`
path <- rep("M45 10 H55 V45 H90 V55 H55 V90 H45 V55 H10 V45 H45 Z", 3)
rsx_trans <-
  dplyr::tibble(
    sc = 1,
    rot = 0,
    tx = c(0, 120, 0),
    ty = c(0, 0, 120),
    ax = 1,
```

```

    ay = 1
  )

  canvas("white") |>
    add_path(path, rsx_trans = rsx_trans) |>
    draw_img()

## End(Not run)

```

add_png

Add PNG image to canvas

Description

Draws a PNG image onto an existing picture.

Usage

```
add_png(img, png, left = 0, top = 0, ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
png	A raw vector of a PNG image.
left	A numeric scalar giving the horizontal offset (in pixels) of the PNG's top-left corner from the canvas origin. Negative values are allowed.
top	A numeric scalar giving the vertical offset (in pixels) of the PNG's top-left corner from the canvas origin. Negative values are allowed.
...	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. • <code>sigma</code> must be a numeric vector of blur sigmas for each shape. • <code>width</code> must be a numeric vector of stroke widths for each shape. • <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
props	A list of painting attributes created by <code>paint()</code> .

Value

A raw vector containing a serialized picture.

add_point

*Add points***Description**

Draws grouped point sequences using Skia's batched point API.

Usage

```
add_point(img, point, group = rep_len(1, nrow(point)), ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
point	A numeric matrix (or a data-frame-like object) with two numeric columns (x and y), where each row is a point.
group	A vector of grouping indices for point. Points are split into groups and drawn group-by-group. Note that grouping is based on consecutive runs of the same value (see Details).
...	For some drawing functions, you can specify sigma, width, and color as named arguments. • sigma must be a numeric vector of blur sigmas for each shape. • width must be a numeric vector of stroke widths for each shape. • color must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.
props	A list of painting attributes created by <code>paint()</code> .

Details

This function draws one shape per group, where each group is a consecutive run in group (because grouping is implemented via `rle()`). If the same group id appears in multiple non-consecutive runs, they are treated as separate groups.

The drawing behavior for each group is controlled by `props[["point_mode"]]`:

- `PointMode$Points`: draws the group's points.
- `PointMode$Lines`: draws line segments through the group's points.
- `PointMode$Polygon`: draws a polygon from the group's points.

The attributes sigma, width, and color can be supplied via They are applied per group, so their lengths must match the number of groups (i.e., `length(rle(group)$values)`). If not supplied, they are taken from props and recycled to all groups.

Value

A raw vector containing a serialized picture.

Examples

```
## Not run:
rad <- \(\deg) deg * (pi / 180)

cv_size <- dev_size()

# Generate coordinates for a rose curve
rose <-
  dplyr::tibble(
    i = seq_len(360),
    r = 120 * abs(sin(rad(4 * i)))
  ) |>
  dplyr::reframe(
    id = i,
    x = r * cos(rad(360 * i / 360)) + cv_size[1] / 2,
    y = r * sin(rad(360 * i / 360)) + cv_size[2] / 2,
    .by = i
  )

# Points (color per group; here, one point per group => color per point)
canvas("white") |>
  add_point(
    dplyr::select(rose, x, y),
    group = dplyr::pull(rose, id),
    color = seq(0, 1, length.out = nrow(rose)) |>
      grDevices::hsv(1, 1, 1) |>
      col2rgba(),
    props = paint(width = 3)
  ) |>
  draw_img()

# Lines (one polyline per group)
rose2 <-
  dplyr::reframe(
    rose,
    x = c(cv_size[1] / 2, x),
    y = c(cv_size[2] / 2, y),
    .by = id
  )
canvas("white") |>
  add_point(
    dplyr::select(rose2, x, y),
    group = dplyr::pull(rose2, id),
    color = unique(dplyr::pull(rose2, id) / nrow(rose)) |>
      grDevices::hsv(1, 1, 1) |>
      col2rgba(),
    props = paint(point_mode = PointMode$Lines, width = 1)
  ) |>
```

```

draw_img()

# Polygon (a single polygon from all points)
canvas("white") |>
  add_point(
    dplyr::select(rose, x, y),
    group = rep_len(1, nrow(rose)),
    props = paint(point_mode = PointMode$Polygon, width = 3, color = "hotpink")
  ) |>
  draw_img()

## End(Not run)

```

add_rect

Add rectangles

Description

Adds one or more rounded rectangles to an existing picture.

Usage

```

add_rect(
  img,
  ltrb,
  rsx_trans = matrix(c(1, 0, 0, 0, 0, 0), nrow(ltrb), 6, byrow = TRUE),
  radii = matrix(0, nrow(ltrb), 2),
  ...,
  props = paint()
)

```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>ltrb</code>	A numeric matrix (or a data-frame-like object) with 4 columns (left, top, right, bottom), where each row represents a rectangle.
<code>rsx_trans</code>	A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to: • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction • offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction
<code>radii</code>	A numeric matrix (or data-frame-like object) with 2 columns (x and y), where each row represents the corner radii of a rounded rectangle.

... For some drawing functions, you can specify `sigma`, `width`, and `color` as named arguments.

- `sigma` must be a numeric vector of blur sigmas for each shape.
- `width` must be a numeric vector of stroke widths for each shape.
- `color` must be an integer matrix with 4 rows (RGBA) and N columns (shapes).

If they are not provided as named arguments, they will be taken from `props`. If the function does not matter them, ... is simply ignored.

`props` A list of painting attributes created by `paint()`.

Details

The number of rectangles is determined by `nrow(ltrb)`. The number of rows of `rsx_trans` and `radii`, and the lengths of `sigma` and `width` (and the number of columns of `color`) must match this number.

The drawing attributes `sigma`, `width`, and `color` can be supplied via If they are not supplied, they are taken from `props` and recycled to all rectangles.

Value

A raw vector containing a serialized picture.

Note

- Skia's rounded rectangle does not support rotation. If the rotation angle in `rsx_trans` is not zero, this function will error. To draw rotated rectangles or more general shapes, use `add_path()` instead.

Examples

```
## Not run:
# Arrange rectangles by shifting coordinates in `ltrb`
ltrb <-
  dplyr::tibble(
    l = c(60, 220, 60),
    t = c(60, 60, 180),
    r = c(180, 340, 180),
    b = c(140, 140, 260)
  )

canvas("white") |>
  add_rect(
    ltrb = ltrb,
    radii = matrix(c(12, 12), nrow(ltrb), 2, byrow = TRUE)
  ) |>
  draw_img()

# Arrange rectangles by translating the same `ltrb` via `rsx_trans`
ltrb <-
```

```

    dplyr::tibble(
      l = rep_len(60, 3),
      t = rep_len(60, 3),
      r = rep_len(180, 3),
      b = rep_len(140, 3)
    )
    rsx_trans <-
      dplyr::tibble(
        sc = 1,
        rot = 0,
        tx = c(0, 160, 0),
        ty = c(0, 0, 120),
        ax = 0,
        ay = 0
      )

    canvas("white") |>
      add_rect(
        ltrb = ltrb,
        rsx_trans = rsx_trans,
        radii = matrix(c(12, 12), nrow(ltrb), 2, byrow = TRUE)
      ) |>
      draw_img()

    ## End(Not run)

```

add_text

*Add text***Description**

Draws text strings as text blobs.

Usage

```
add_text(img, text, rsx_trans, freeze = TRUE, ..., props = paint())
```

Arguments

- | | |
|-----------|--|
| img | A raw vector of a serialized picture. |
| text | A character vector of text strings to be drawn. <code>NA_character_</code> is not allowed. Each element of <code>text</code> is handled as one text blob. |
| rsx_trans | <p>A numeric matrix (or a data-frame-like object) with 6 columns where each row represents an RSX transform. Each column of the matrix corresponds to:</p> <ul style="list-style-type: none"> • scale • angle of rotation (in radians) • amount of translation in the X-axis direction • amount of translation in the Y-axis direction |

	• offset for the anchor point in the X-axis direction • offset for the anchor point in the Y-axis direction
freeze	A logical value indicating whether to freeze the picture after drawing text. If TRUE, the result is rasterized and re-added to a new canvas (like <code>freeze()</code>).
...	For some drawing functions, you can specify sigma, width, and color as named arguments. • sigma must be a numeric vector of blur sigmas for each shape. • width must be a numeric vector of stroke widths for each shape. • color must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.
props	A list of painting attributes created by <code>paint()</code> .

Details

The placement of glyphs is controlled by `rsx_trans`. For `add_text()`, `rsx_trans` must have the same number of rows as the total number of characters to be drawn, i.e. `sum(nchar(text))`, not `length(text)`.

In contrast, `sigma` and `color` provided via ... (or from props) are matched to `length(text)` (one value per text element). If you need per-character values for `sigma` or `color`, split your string into single characters and pass them as a character vector.

Value

A raw vector containing a serialized picture.

Note

- Text blobs do not have a font fallback mechanism. Characters not supported by the specified font may not render correctly.
- If `freeze = FALSE`, the returned picture can become large because font data may be embedded. In most cases, it is recommended to keep `freeze = TRUE`.

Examples

```
## Not run:
if ("Noto Sans Mono" %in% list_font_families()[["family"]]) {
  info <- text_info(
    "Hello, skiagd!",
    props = paint(family = "Noto Sans Mono", fontsize = 36)
  )
  offset <- seq(
    0, info$n_chars * (info$width / info$n_chars),
    length.out = info$n_chars
  )
  rsx_trans <-
    dplyr::tibble(
```

```

    sc = 1,
    rot = 0,
    x = (dev_size()[1] - info$width) / 2 + offset,
    y = dev_size()[2] / 2,
    ax = 0,
    ay = 0
  )
  canvas("white") |>
  add_text(
    "Hello, skiagd!",
    rsx_trans = rsx_trans,
    props = paint(family = "Noto Sans Mono", fontsize = 36)
  ) |>
  draw_img()
}

## End(Not run)

```

add_vertices

*Add vertices***Description**

Adds a vertex mesh to an existing picture.

Usage

```
add_vertices(img, vertices, ..., props = paint())
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>vertices</code>	A numeric matrix (or a data-frame-like object) with 2 numeric columns (x and y), where each row is a vertex position. Vertices are consumed in groups of three (three rows per triangle). If <code>nrow(vertices)</code> is not a multiple of 3, the last <code>nrow(vertices) %% 3</code> vertices are ignored.
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. <code>sigma</code> must be a numeric vector of blur sigmas for each shape. <code>width</code> must be a numeric vector of stroke widths for each shape. <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Details

The vertex mode is taken from `props[["vertex_mode"]]`. The blur sigma is taken from `props[["sigma"]]`.

Vertex colors are per-vertex attributes. If `color` is supplied via `...`, it must be an RGBA integer matrix with 4 rows and `nrow(vertices)` columns (one color per vertex). If `color` is not supplied, `props[["color"]]` is recycled to all vertices.

Value

A raw vector containing a serialized picture.

Note

When vertex colors are present, the paint's blend source is determined by:

- `props[["shader"]]` if a shader is set, otherwise
- an opaque version of `props[["color"]]`.

This source is then combined with the interpolated vertex colors. As a result, to paint each vertex with a different visible color, you typically need to set a [Shader](#) (see the examples below).

Examples

```
## Not run:
# A single triangle with per-vertex colors.
# To make the vertex colors visible as-is, set a shader.
canvas("white") |>
  add_vertices(
    dplyr::tibble(
      x = c(128, 0, 128),
      y = c(256, 256, 0)
    ),
    color = col2rgba(c("#61dafb", "#fb61da", "#dafb61")),
    props = paint(
      shader = Shader$from_picture(
        canvas("ffffff00"),
        TileMode$Repeat,
        dev_size(),
        diag(3)
      )
    )
  ) |>
  draw_img()

## End(Not run)
```

as_nativeraster	<i>Convert picture into native raster</i>
-----------------	---

Description

Converts `img` to an integer matrix with class `nativeRaster`.

Usage

```
as_nativeraster(img, ..., props = paint())
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. <code>sigma</code> must be a numeric vector of blur sigmas for each shape. <code>width</code> must be a numeric vector of stroke widths for each shape. <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Details

In base R, a `nativeRaster` object conventionally stores colors with **non-premultiplied** alpha (RGB channels are independent of the alpha channel).

In contrast, `skiagd` renders internally with **premultiplied** alpha, and the resulting pixel values are returned as-is.

When such a raster is drawn onto an R graphics device (e.g., via `grid::grid.raster()`), the device composites the image with its background color. If the alpha channel is not fully opaque (< 255), this extra alpha blending can change the apparent colors compared with what you would expect from the output of `as_png()`.

Value

A `nativeRaster` object.

Examples

```
## Not run:
img <- canvas("navy") |>
  as_nativeraster()

dev.hold(2)
```

```

grid::grid.newpage()
grid::grid.raster(img, interpolate = FALSE)
dev.flush(2)

## End(Not run)

```

as_png

Convert picture into PNG image

Description

Renders a serialized picture to a PNG image.

Usage

```
as_png(img, ..., props = paint())
```

Arguments

img	A raw vector of a serialized picture.
...	For some drawing functions, you can specify sigma, width, and color as named arguments. • sigma must be a numeric vector of blur sigmas for each shape. • width must be a numeric vector of stroke widths for each shape. • color must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from props. If the function does not matter them, ... is simply ignored.
props	A list of painting attributes created by paint() .

Value

A raw vector of a PNG image.

Examples

```

## Not run:
png <-
  canvas("navy") |>
  as_png()

# Write the PNG image to a file
writeBin(png, "navy.png")

## End(Not run)

```

BlendMode	<i>BlendMode (0-28)</i>
-----------	-------------------------

Description

BlendMode determines how source and destination colors are combined.

Usage

BlendMode

Details

The following blend modes are available in Skia:

1. Clear
2. Src
3. Dst
4. SrcOver
5. DstOver
6. SrcIn
7. DstIn
8. SrcOut
9. DstOut
10. SrcATop
11. DstATop
12. Xor
13. Plus
14. Modulate
15. Screen
16. Overlay
17. Darken
18. Lighten
19. ColorDodge
20. ColorBurn
21. HardLight
22. SoftLight
23. Difference
24. Exclusion
25. Multiply

- 26. Hue
- 27. Saturation
- 28. Color
- 29. Luminosity

See Also

[BlendMode in skia_safe - Rust](#)

Other paint-attributes: [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

BlurStyle

BlurStyle (0-3)

Description

BlurStyle controls how a blur mask filter is applied to the shape.

Usage

BlurStyle

Details

The following BlurStyle are available:

- Normal: Normal blur.
- Solid: Solid blur.
- Outer: Outer blur.
- Inner: Inner blur.

See Also

[BlurStyle in skia_safe - Rust](#)

Other paint-attributes: [BlendMode](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

canvas	<i>Create new canvas</i>
--------	--------------------------

Description

Creates a new serialized picture (a single frame) with a canvas filled with the specified color.

The coordinate system follows the usual image convention: the origin is at the top-left, the X axis increases to the right, and the Y axis increases downward. Units are pixels.

Usage

```
canvas(fill = "transparent", canvas_size = paint()[["canvas_size"]])
```

Arguments

fill	An RGBA color specification for the background fill. You can also provide a named color or a hexadecimal color code, which is converted internally using <code>colorfast::col_to_rgb()</code> .
canvas_size	An integer vector of length 2 specifying canvas width and height, in pixels.

Value

A raw vector containing a serialized Skia picture.

Examples

```
## Not run:
canvas("navy") |> draw_img()

canvas(c(255, 0, 0, 255)) |> draw_img() # filled with red

## End(Not run)
```

Cap	<i>Cap (0-2)</i>
-----	------------------

Description

Cap determines the stroke cap (the geometry drawn at the beginning and end of strokes).

Usage

```
Cap
```

Details

The following caps are available:

- Butt: Butt cap.
- Round: Round cap.
- Square: Square cap.

See Also

Cap in [skia_safe::paint](#) - Rust

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

col2rgba	<i>Convert colors to matrix of RGBA integers</i>
----------	--

Description

A wrapper of [colorfast::col_to_rgb\(\)](#).

Usage

col2rgba(color)

Arguments

color col for [colorfast::col_to_rgb\(\)](#).

Value

An integer matrix with 4 rows (RGBA) and N columns (the same length as color).

create_mapping	<i>Create transformation matrix from points pairs</i>
----------------	---

Description

Creates a transformation matrix from sets of points src and dst. If mapping is not possible, throws an error.

Usage

create_mapping(src, dst)

Arguments

src	A numeric matrix (or a data-frame-like object) with 2 columns (x and y) and 4 rows, where each row is a point.
dst	A numeric matrix (or a data-frame-like object) with 2 columns (x and y) and 4 rows, where each row is a point.

Value

A numeric matrix of size 3x3.

Examples

```
## Not run:
cv_size <- dev_size()

# Create an affine matrix that fits a 768x576 picture to current device size
create_mapping(
  src = matrix(
    c(0, 0, 768, 0, 768, 576, 0, 576),
    ncol = 2,
    byrow = TRUE
  ),
  dst = matrix(
    c(0, 0, cv_size[1], 0, cv_size[1], cv_size[2], 0, cv_size[2]),
    ncol = 2,
    byrow = TRUE
  )
)

## End(Not run)
```

dev_size

*Get the size of the current graphics device***Description**

Just returns the size of the current graphics device as integers.

Usage

```
dev_size(units = "px")
```

Arguments

units units for `grDevices::dev.size()`.

Value

An integer vector of length 2 (width, height).

draw_img	<i>Plot picture as raster image</i>
----------	-------------------------------------

Description

Converts `img` to a `nativeRaster` and draws it using `grid::grid.raster()` with `interpolate = TRUE`.

Linear interpolation is therefore applied by default. This produces smoother edges (e.g., for circles and diagonal shapes), at the cost of slightly slower rendering. If you prefer a faster but more blocky result, you can manually call `grid::grid.raster()` with `interpolate = FALSE` on the output of `as_nativeraster()`.

See the examples in `as_nativeraster()` for a comparison.

Usage

```
draw_img(img, ..., props = paint())
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. <code>sigma</code> must be a numeric vector of blur sigmas for each shape. <code>width</code> must be a numeric vector of stroke widths for each shape. <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code>. If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Value

`img` is returned invisibly.

embed_img	<i>Embed an image as HTML</i>
-----------	-------------------------------

Description

Embeds an image file as a base64-encoded HTML `<img>` tag.

For string input, the image file is read from disk, encoded as a data URI, and wrapped in an HTML object. For raw input, the image data is directly embedded in an HTML object, assuming it is a valid PNG image data.

Usage

```
embed_img(x, max_size = 4 * 1024^2, browsable = TRUE, ...)
```

```
data_uri(x, max_size = 4 * 1024^2, ...)
```

Arguments

<code>x</code>	File path or raw image data.
<code>max_size</code>	Maximum allowed file size in bytes. If <code>NULL</code> , size checking is disabled.
<code>browsable</code>	Logical. Whether to wrap the result in <code>htmltools::browsable()</code> .
<code>...</code>	Additional attributes passed to <code>htmltools::tags\$img()</code> .

Details

This function is designed as a lightweight helper for visualization and documentation purposes. It does not perform any image decoding itself and relies on the input being readable by `base64enc::dataURI()`.

Value

- `embed_img()` returns an HTML object.
- `data_uri()` returns a data URI string.

<code>enable_autocomplete</code>	<i>Enable autocomplete for painting attributes</i>
----------------------------------	--

Description

Enables autocomplete for painting attributes. To do this, this function simply assigns them to the current environment.

Usage

```
enable_autocomplete(env = parent.frame())
```

Arguments

<code>env</code>	Environment to assign objects to.
------------------	-----------------------------------

Value

Called for its side-effect.

FillType	<i>FillType (0-3)</i>
----------	-----------------------

Description

FillType determines how paths are drawn. This is for [add_path\(\)](#) only. Not used in other functions.

Usage

FillType

Details

The following FillType are available:

- Winding
- EvenOdd
- InverseWinding
- InverseEvenOdd

See Also

[FillType in skia_safe::path - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

FontStyle	<i>FontStyle (0-3)</i>
-----------	------------------------

Description

FontStyle determines the font style.

Usage

FontStyle

Details

The following styles are available:

- Normal: Normal (plain).
- Bold: Bold (bold).
- Italic: Italic (italic).
- BoldItalic: BoldItalic (bold.italic).

See Also

[FontStyle in skia_safe - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

 freeze

Freeze picture

Description

Rasterizes a picture into a PNG and adds it to a new canvas.

This is equivalent to `as_png(img, props = props)` and then drawing the resulting PNG onto a fresh canvas using `add_png()` with the default blend mode (`BlendMode$SrcOver`). It can be used to flatten a picture before continuing to draw.

Usage

```
freeze(img, left = 0, top = 0, fill = "transparent", ..., props = paint())
```

Arguments

<code>img</code>	A raw vector of a serialized picture.
<code>left</code>	A numeric scalar giving the horizontal offset (in pixels) where the rasterized image is drawn on the new canvas. Negative values are allowed.
<code>top</code>	A numeric scalar giving the vertical offset (in pixels) where the rasterized image is drawn on the new canvas. Negative values are allowed.
<code>fill</code>	An RGBA color specification for the background fill of the new canvas. You can provide a named color or a hexadecimal color code, which is converted using <code>col2rgba()</code> .
<code>...</code>	For some drawing functions, you can specify <code>sigma</code> , <code>width</code> , and <code>color</code> as named arguments. <code>sigma</code> must be a numeric vector of blur sigmas for each shape. <code>width</code> must be a numeric vector of stroke widths for each shape. <code>color</code> must be an integer matrix with 4 rows (RGBA) and N columns (shapes). If they are not provided as named arguments, they will be taken from <code>props</code> . If the function does not matter them, <code>...</code> is simply ignored.
<code>props</code>	A list of painting attributes created by <code>paint()</code> .

Value

A raw vector containing a serialized picture.

ImageFilter

*ImageFilter***Description**

ImageFilter is a struct that offers a reference to `skia_safe::ImageFilter`. You can apply an image filter to canvas via `paint()`.

Concatenating image filters with `c()` is equivalent to sequentially compose them into a single filter using `ImageFilter$compose()`. For blending image filters, use `ImageFilter$blend()` explicitly.

Arguments

<code>img</code>	A raw vector of picture.
<code>crop_rect</code>	Numerics of length 4 for cropping the filtered image.
<code>dst, src</code>	ImageFilter objects.
<code>coef</code>	Numerics that represents the coefficients $c(k_1, k_2, k_3, k_4)$. Each output pixel is the result of combining the corresponding dst and src pixels using these values.
<code>mode</code>	BlendMode .
<code>sigma</code>	Numerics of length 2 for blur sigma.
<code>tile_mode</code>	TileMode .
<code>color_mat</code>	A 4x5 row-major numeric matrix that represents a color matrix. Every pixel's color value is multiplied by this matrix in the same way as the feColorMatrix SVG filter. A playground to build color matrices is available here .
<code>outer, inner</code>	ImageFilter objects.
<code>radius</code>	Numerics of length 2; radius of ellipse for dilation and erosion.
<code>channels</code>	Numerics of length 2 in range of $[0, 3]$ (corresponding to R, G, B, or A channel); color channels to be used along X and Y axes within the source image.
<code>scale</code>	A numeric scalar; displacement scale factor to be used.
<code>displacement</code>	An ImageFilter object that displaces the source image.
<code>offset</code>	Numerics of length 2 for X and Y offsets.
<code>source</code>	A RuntimeEffect object.
<code>uniforms</code>	A named list of numerics to be assigned to uniforms in source.

Details

The following filters are available:

- `no_filter()`: does not apply any image filter. This is the default image filter for `paint()`.
- `from_picture(img, crop_rect)`: creates an image filter from a picture.
- `arithmetic(dst, src, coef, crop_rect)`: applies an arithmetic operation to two image filters.

- `blend(dst, src, mode, crop_rect)`: blends two image filters with a given blend mode.
- `blur(sigma, tile_mode, crop_rect)`: creates a blur image filter.
- `color_matrix(color_mat)`: creates an image filter from a color matrix.
- `compose(outer, inner)`: composes two image filters.
- `crop(crop_rect, tile_mode)`: crops the source image.
- `dilate(radius, crop_rect)`: dilates the source image.
- `displacement_map(channels, scale, displacement, crop_rect)`: creates a displacement map.
- `drop_shadow(offset, sigma, color, crop_rect)`: creates a drop shadow image filter.
- `erode(radius, crop_rect)`: erodes the source image.
- `offset(offset, crop_rect)`: creates an offset image filter.
- `runtime_shader(source, uniforms)`: creates an image filter from a [RuntimeEffect](#).

Value

An `ImageFilter` object.

See Also

- [Image Filters | React Native Skia](#)
- [skia_safe::image_filters - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

Join

Join (0-2)

Description

Join determines the stroke join (the geometry drawn at the corners of strokes) for shapes.

Usage

Join

Details

The following joins are available:

- Miter: Miter join.
- Round: Round join.
- Bevel: Bevel join.

See Also

Join in `skia_safe::paint` - Rust

Other paint-attributes: `BlendMode`, `BlurStyle`, `Cap`, `FillType`, `FontStyle`, `ImageFilter`, `PathEffect`, `PointMode`, `Shader`, `Style`, `VertexMode`

<code>list_font_families</code>	<i>List available font families</i>
---------------------------------	-------------------------------------

Description

Returns font families available on the system.

Since `skia_gd` can only access fonts installed on the system, font families in the font registry or local fonts registered by the `systemfonts` package cannot be specified as the family in `paint()`.

Usage

```
list_font_families()
```

Value

A tibble containing family.

<code>paint</code>	<i>Define painting attributes</i>
--------------------	-----------------------------------

Description

The `paint()` function allows users to specify various painting attributes for drawing shapes on the canvas, such as color and stroke width.

Usage

```
paint(...)
```

Arguments

... `<dynamic-dots>` Named arguments specifying painting attributes. See details.

Details

The following painting attributes can be specified:

- `canvas_size`: Integers of length 2 (width, height).
- `color`: An RGBA color specification (integers of length 4). This can be also specified using named colors or hexadecimal color codes, which are converted using `col2rgba()`.
- `style`: Paint style. See [Style](#).
- `join`: Stroke join. See [Join](#).
- `cap`: Stroke cap. See [Cap](#).
- `width`: A numeric scalar (stroke width).
- `miter`: A numeric scalar (stroke miter).
- `fontsize`: A numeric scalar (font size).
- `family`: Font family name. You can list available font families using `list_font_families()`.
- `fontface`: Font face. See [FontStyle](#).
- `sigma`: A numeric scalar. Default value for blur sigma.
- `blur_style`: [BlurStyle](#) for a blur mask filter applied to the shape.
- `blend_mode`: See [BlendMode](#).
- `path_effect`: See [PathEffect](#).
- `shader`: See [Shader](#).
- `image_filter`: See [ImageFilter](#).
- `point_mode`: [PointMode](#) for `add_point()`.
- `vertex_mode`: [VertexMode](#) for `add_vertices()`.
- `fill_type`: [FillType](#) for `add_path()`.

Value

A list containing the specified painting attributes, merged with default values.

PathEffect

PathEffect

Description

`PathEffect` is a struct that offers a reference to `skia_safe::PathEffect`. You can apply a path effect to shapes via `paint()`.

Concatenating path effects with `c()` is equivalent to sum them sequentially into a single effect using `PathEffect$sum()`.

Arguments

first	A PathEffect object.
second	A PathEffect object.
start	A numeric scalar in the range [0, 1].
end	A numeric scalar in the range [0, 1].
length	A numeric scalar; length of the subsegments.
deviation	A numeric scalar; limit of the movement of the endpoints.
seed	An integer scalar; random seed.
intervals	A numeric vector; even number of entries with even indices specifying the length of the "on" intervals, and the odd index specifying the length of "off".
phase	A numeric scalar; offset into the intervals array (for dash()), or distance (mod advance) along the path for its initial position (for path_1d()).
radius	A numeric scalar; radius of the rounded corners.
path	A string scalar of SVG notation to replicate.
advance	A numeric scalar; space between instances of path.
style	A string scalar; how to transform path at each point. Can be "translate", "rotate", or "morph".
transform	Numerics of length 9; see transform-matrix .
width	A numeric scalar; width of the path to be stamped.

Details

The following effects are available:

- `no_effect()`: does not apply any path effect. This is the default effect for `paint()`.
- `sum(first, second)`: applies two effects in sequence.
- `trim(start, end)`: trims the start and end of the path. Note that you can't trim nothing at all, i.e., setting `start = 0` and `end = 1` does nothing.
- `discrete(length, deviation, seed)`: applies discrete path effect.
- `dash(intervals, phase)`: applies dash path effect.
- `corner(radius)`: applies corner path effect.
- `path_1d(path, advance, phase, style)`: applies 1D path effect.
- `path_2d(path, transform)`: applies 2D path effect.
- `line_2d(width, transform)`: applies 2D line path effect.

Value

A PathEffect object.

See Also

[Path Effects | React Native Skia](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PointMode](#), [Shader](#), [Style](#), [VertexMode](#)

pictures

Pictures

Description

In Skia, a picture is a prerecorded list of drawing operations on a canvas. The drawing functions of `skia` take it as their first argument, add new shapes onto it, and return a serialized picture as a raw vector again.

A serialized picture is a binary format containing a single frame, which can be saved to a `.skp` file using `writeBin()`, and reused by any drawing functions as long as it is compatible with the version of Skia used to create it.

You can review contents of `.skp` files with the [Skia debugger](#) if they are compatible with the version.

See Also

[Pictures | React Native Skia](#)

PointMode

PointMode (0-2)

Description

`PointMode` determines how points are drawn. This is for `add_point()` only. Not used in other functions.

Usage

`PointMode`

Details

The following `PointMode` are available:

- **Points:** Draws each point as a point. The shape of point drawn depends on props.
- **Lines:** Each pair of point draws a line segment. One line is drawn for every two points; each point is used once. If count is odd, the final point is ignored.
- **Polygon:** Each adjacent pair of point draws a line segment. count minus one lines are drawn; the first and last point are used once.

See Also

[PointMode in skia_safe::canvas - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [Shader](#), [Style](#), [VertexMode](#)

RuntimeEffect

RuntimeEffect

Description

RuntimeEffect is a struct that wraps `skia_safe::RuntimeEffect`.

Skia provides a shading language called SkSL. The syntax is similar to GLSL, but differs in minor details. A quick overview of their differences can be found [here](#).

You can compile an SkSL source into a RuntimeEffect using `RuntimeEffect$make()`, and apply it as an [ImageFilter](#) or a [Shader](#).

Arguments

`sksl` A string scalar of an SkSL source. For [ImageFilter](#), the fragment shader must receive the currently filtered image as shader uniform.

Details

RuntimeEffect as an R environment exposes the following method:

- `make(sksl)`: Takes an SkSL source and compiles it into a RuntimeEffect.

A RuntimeEffect object has the following method:

- `source()`: Returns the original SkSL source as a string scalar.

Value

For `make()`, a RuntimeEffect object is returned if the SkSL source is successfully compiled. Otherwise, an error is thrown with the compilation error message.

Examples

```
## Not run:
effect <-
  RuntimeEffect$make(R"{
    uniform shader image;
    uniform vec2 resolution;
    vec4 main(vec2 fragCoord) {
      vec2 uv = fragCoord / resolution;
      return vec4(uv.x, uv.y, .6, 1.0);
    }
  }")

cv_size <- dev_size()
imgf <-
  ImageFilter$runtime_shader(
    effect,
    list(resolution = as.double(cv_size))
```

```

)
canvas() |>
  add_rect(
    matrix(c(0, 0, cv_size), ncol = 4),
    props = paint(image_filter = imgf)
  ) |>
  draw_img()

## End(Not run)

```

 Shader

Shader

Description

Shader is a struct that offers a reference to `skia_safe::Shader`. You can apply a shader to shapes via `paint()`.

Concatenating shaders with `c()` is equivalent to blend them all into a single shader using `Shader$blend()` with the default `BlendMode`. You can pass mode explicitly for `c()` to change the blend mode.

Arguments

<code>img</code>	A raw vector of picture.
<code>mode</code>	For <code>blend()</code> , <code>BlendMode</code> . For others, <code>TileMode</code> .
<code>tile_size</code>	Numerics of length 2; tile size (width, height).
<code>transform</code>	Numerics of length 9; see transform-matrix .
<code>png</code>	A raw vector of PNG image.
<code>source</code>	A <code>RuntimeEffect</code> object.
<code>uniforms</code>	A named list of numerics to be assigned to uniforms in source.
<code>color</code>	An integer matrix in range <code>[0, 255]</code> where each column is an RGBA color.
<code>dst</code>	A Shader object; destination shader.
<code>src</code>	A Shader object; source shader.
<code>freq</code>	Numerics of length 2; frequencies.
<code>octaves</code>	A numeric scalar; number of octaves.
<code>seed</code>	Integer scalar; random seed.
<code>start</code>	Numerics of length 2; starting point (x, y).
<code>end</code>	Numerics of length 2; ending point (x, y).
<code>flags</code>	A logical scalar; typically, you can leave this as <code>FALSE</code> . See here for details.
<code>radii</code>	Numerics of length 2; radii of start and end circles.
<code>center</code>	Numerics of length 2; center of the gradient.
<code>start_angle</code>	A numeric scalar in range <code>[0, 360]</code> ; starting angle. For default, set 0.
<code>end_angle</code>	A numeric scalar in range <code>[0, 360]</code> ; ending angle. For default, set 360.

Details

The following shaders are available:

- `no_shader()`: does not apply any shader. This is the default shader for `paint()`.
- `from_picture(img, mode, tile_size, transform)`: takes a picture and returns an image shader.
- `from_png(png, mode, transform)`: takes a PNG image and returns an image shader.
- `from_runtime_effect(source, uniforms)`: takes a [RuntimeEffect](#) and returns a shader.
- `color(color)`: takes a color and returns a color shader.
- `blend(mode, dst, src)`: returns a shader where the given shaders are combined with [Blend-Mode](#).
- `fractal_noise(freq, octaves, seed, tile_size)`: fractal perlin noise shader.
- `turbulence(freq, octaves, seed, tile_size)`: turbulence noise shader.
- `linear_gradient(start, end, color, mode, flags, transform)`: linear gradient shader.
- `radial_gradient(center, radius, color, mode, flags, transform)`: radial gradient shader.
- `conical_gradient(start, end, radii, color, mode, flags, transform)`: conical gradient shader.
- `sweep_gradient(center, start_angle, end_angle, color, mode, flags, transform)`: sweep gradient shader.

Value

A Shader object.

See Also

- [Image Shaders | React Native Skia](#)
- [Gradients | React Native Skia](#)
- [Perlin Noise Shaders | React Native Skia](#)
- [Blending and Colors | React Native Skia](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Style](#), [VertexMode](#)

Style	<i>Style (0-2)</i>
-------	--------------------

Description

Style determines the stroke style of shapes.

Usage

Style

Details

The following styles are available:

- `StrokeAndFill`: Stroke and fill.
- `Stroke`: Stroke only.
- `Fill`: Fill only.

See Also

[Style in `skia\_safe::paint` - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [VertexMode](#)

svg_bounds

Retrieve bounding boxes of SVG paths

Description

Computes axis-aligned bounding boxes for one or more SVG paths.

Usage

```
svg_bounds(path)
```

Arguments

path	A character vector of SVG path notations (the <code>d</code> attribute syntax). Each element is treated as a separate path.
------	---

Value

A tibble containing columns `id`, `left`, `top`, `right` and `bottom` for each path. `id` is a 1-based index corresponding to the input order of path.

See Also

Other path-utils: [svg_interpolate\(\)](#), [svg_transform\(\)](#)

Examples

```
svg_bounds("M45 10 H55 V45 H90 V55 H55 V90 H45 V55 H10 V45 H45 Z")
```

svg_interpolate	<i>Interpolate between two SVG paths</i>
-----------------	--

Description

Interpolates between two SVG paths that are compatible for morphing.

Usage

```
svg_interpolate(t, first, second)
```

Arguments

t	A numeric vector of interpolation weights. Values between 0 and 1 produce intermediate paths; values outside this range are wrapped.
first	A string scalar of an SVG path notation (the d attribute syntax).
second	A string scalar of an SVG path notation (the d attribute syntax).

Details

This function returns intermediate paths between `first` and `second` using weights `t`. Paths must be interpolatable (e.g., compatible command sequences); otherwise an error is thrown.

Value

A character vector of SVG path notations interpolated.

See Also

Other path-utils: [svg_bounds\(\)](#), [svg_transform\(\)](#)

Examples

```
trans <- matrix(c(1, 0, -50, 0, 1, -50, 0, 0, 1), 3, 3)
first <- svg_transform(R"(
  M10 18 H40 M18 10 V40
  M90 18 H60 M82 10 V40
  M10 82 H40 M18 90 V60
  M90 82 H60 M82 90 V60
)", trans)
second <- svg_transform(R"(
  M12 22 H44 M22 12 V44
  M88 22 H56 M78 12 V44
  M12 78 H44 M22 88 V56
  M88 78 H56 M78 88 V56
)", trans)
svg_interpolate(seq(-2, 2, length.out = 10), first, second)
```

svg_transform	<i>Transform SVG paths</i>
---------------	----------------------------

Description

Applies an affine transformation to SVG path notations.

This is useful for translating, scaling, or skewing paths written in the SVG d attribute syntax before placing them with [add_path\(\)](#).

Usage

```
svg_transform(path, transform)
```

Arguments

path	A character vector of SVG path notations (the d attribute syntax). Each element is treated as a separate path.
transform	Numerics of length 9; see transform-matrix .

Value

A character vector of transformed SVG path notations.

See Also

Other path-utils: [svg_bounds\(\)](#), [svg_interpolate\(\)](#)

Examples

```
trans <- matrix(c(1, 0, -50, 0, 1, -50, 0, 0, 1), 3, 3)
svg_transform("M45 10 H55 V45 H90 V55 H55 V90 H45 V55 H10 V45 H45 Z", trans)
```

text_info	<i>Get width, bounding box, and number of characters</i>
-----------	--

Description

Returns metrics for text strings when they are shaped and drawn as a text blob with the given props.

Usage

```
text_info(text, props = paint())
```

Arguments

text	A character vector of text strings.
props	A list of painting attributes created by <code>paint()</code> .

Value

A tibble containing one row per element of text, with columns: id (1-based index), n_chars, width (advance width), and left, top, right, bottom for the bounding box.

TileMode	<i>TileMode (0-3)</i>
----------	-----------------------

Description

TileMode determines how the source is tiled for shaders. This is not a painting attribute. To specify TileMode, directly pass these pointers to shader functions.

Usage

TileMode

Details

The following TileMode are available:

- Clamp
- Repeat
- Mirror
- Decal

See Also

[TileMode in skia_safe - Rust](#)

transform-matrix	<i>Affine transformation matrix</i>
------------------	-------------------------------------

Description

Several skiagd APIs accept transform as numerics of length 9. This value is interpreted as a 3x3 affine transformation matrix.

Typical uses include transforming:

- shader coordinate systems such as gradients and image shaders,
- path effect patterns such as `PathEffect$path_2d()` and `PathEffect$line_2d()`,
- SVG path data via [svg_transform\(\)](#).

Details

The matrix is read in the following layout:

$$\begin{bmatrix} \text{scale}_x & \text{skew}_y & \text{persp}_0 \\ \text{skew}_x & \text{scale}_y & \text{persp}_1 \\ \text{trans}_x & \text{trans}_y & \text{persp}_2 \end{bmatrix}$$

For ordinary affine transforms, the last column is typically `c(0, 0, 1)`, while the other entries control scaling, skewing, and translation.

skiagd passes this matrix through to the corresponding Skia API, so the exact effect depends on where transform is used.

See Also

[Matrix in skia_safe::matrix - Rust](#)

VertexMode	<i>VertexMode (0-2)</i>
------------	-------------------------

Description

VertexMode determines how vertices are drawn. This is for [add_vertices\(\)](#) only. Not used in other functions.

Usage

VertexMode

Details

The following VertexMode are available:

- Triangles
- TriangleStrip
- TriangleFan

See Also

[VertexMode in skia_safe::vertices - Rust](#)

Other paint-attributes: [BlendMode](#), [BlurStyle](#), [Cap](#), [FillType](#), [FontStyle](#), [ImageFilter](#), [Join](#), [PathEffect](#), [PointMode](#), [Shader](#), [Style](#)

Index

- * **paint-attributes**
 - BlendMode, 25
 - BlurStyle, 26
 - Cap, 27
 - FillType, 32
 - FontStyle, 32
 - ImageFilter, 34
 - Join, 35
 - PathEffect, 37
 - PointMode, 39
 - Shader, 41
 - Style, 42
 - VertexMode, 47
- * **path-utils**
 - svg_bounds, 43
 - svg_interpolate, 44
 - svg_transform, 45
- add_arc, 3
- add_atlas, 5
- add_circle, 7
- add_diff_rect, 8
- add_line, 11
- add_path, 12
- add_path(), 4, 10, 18, 32, 37, 45
- add_png, 14
- add_png(), 33
- add_point, 15
- add_point(), 37, 39
- add_rect, 17
- add_rect(), 10
- add_text, 19
- add_vertices, 21
- add_vertices(), 37, 47
- as_nativeraster, 23
- as_nativeraster(), 30
- as_png, 24
- as_png(), 5, 23
- BlendMode, 25, 26, 28, 32–39, 41–43, 48
- BlurStyle, 26, 26, 28, 32, 33, 35–39, 42, 43, 48
- canvas, 27
- Cap, 26, 27, 32, 33, 35–39, 42, 43, 48
- col2rgba, 28
- col2rgba(), 33, 37
- colorfast::col_to_rgb(), 27, 28
- create_mapping, 28
- data_uri (embed_img), 30
- dev_size, 29
- draw_img, 30
- embed_img, 30
- enable_autocomplete, 31
- FillType, 13, 26, 28, 32, 33, 35–39, 42, 43, 48
- FontStyle, 26, 28, 32, 32, 35–39, 42, 43, 48
- freeze, 33
- freeze(), 20
- grDevices::dev.size(), 29
- grid::grid.raster(), 23, 30
- ImageFilter, 26, 28, 32, 33, 34, 36–40, 42, 43, 48
- Join, 26, 28, 32, 33, 35, 35, 37–39, 42, 43, 48
- list_font_families, 36
- list_font_families(), 37
- paint, 36
- paint(), 3, 6, 8, 9, 11, 13–15, 18, 20, 21, 23, 24, 30, 33, 34, 36–38, 41, 42, 46
- PathEffect, 26, 28, 32, 33, 35, 36, 37, 37, 39, 42, 43, 48
- picture (pictures), 39
- pictures, 39
- PointMode, 26, 28, 32, 33, 35–38, 39, 42, 43, 48

`readBin()`, [5](#)
`rle()`, [15](#)
`RuntimeEffect`, [34](#), [35](#), [40](#), [41](#), [42](#)

`Shader`, [22](#), [26](#), [28](#), [32](#), [33](#), [35–40](#), [41](#), [43](#), [48](#)
`Style`, [26](#), [28](#), [32](#), [33](#), [35–39](#), [42](#), [42](#), [48](#)
`svg_bounds`, [43](#)
`svg_bounds()`, [44](#), [45](#)
`svg_interpolate`, [44](#)
`svg_interpolate()`, [43](#), [45](#)
`svg_transform`, [45](#)
`svg_transform()`, [43](#), [44](#), [47](#)

`text_info`, [45](#)
`TileMode`, [34](#), [41](#), [46](#)
`transform-matrix`, [38](#), [41](#), [45](#), [47](#)

`VertexMode`, [26](#), [28](#), [32](#), [33](#), [35–39](#), [42](#), [43](#), [47](#)

`writeBin()`, [39](#)