

Package: glyph (via r-universe)

July 6, 2026

Title A Next-Generation Grammar of Interactive Graphics

Version 0.1.0

Description A modern visualization grammar that treats interactivity, animation, and composable layouts as first-class concepts rather than afterthoughts. Designed to address key limitations of existing grammars: native hover, click, and zoom events, 'WebGL'-accelerated rendering for large datasets, built-in multi-plot composition, and a token-based theming system. Renders to interactive HTML widgets via 'D3.js' or static SVG from a single declarative specification.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

RoxygenNote 7.3.3

Config/testthat/edition 3

Imports htmlwidgets (>= 1.6.0), jsonlite (>= 1.8.0), rlang (>= 1.1.0), cli (>= 3.6.0), grDevices, utils, tools

Suggests testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL <https://github.com/josh45-source/glyph>

BugReports <https://github.com/josh45-source/glyph/issues>

NeedsCompilation no

Author Joash Joshua Ayo [aut, cre]

Maintainer Joash Joshua Ayo <joashjoshua789@gmail.com>

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

Date/Publication 2026-07-06 14:39:58 UTC

RemoteUrl <https://github.com/cran/glyph>

RemoteRef HEAD

RemoteSha 0b39bf6f0107cf237f83ccf855790b2cb9888154

Contents

animate	2
animation	3
compile	4
compose	5
export	6
facet	6
glyph	7
inset	7
interact	8
interactivity	9
layout	9
marginals	10
mark_area	10
mark_bar	11
mark_line	11
mark_link	12
mark_point	12
mark_ribbon	13
mark_rule	13
mark_text	14
marks	15
print.glyph_spec	15
render	16
rendering	16
scale	17
scales	18
selection	19
theme_tokens	20
theming	21
titles	21
to_vegalite	22
Index	23

 animate

Add animation/transition behavior

Description

Add animation/transition behavior

Usage

```
animate(
  spec,
  transition = "morph",
  duration = 500,
  stagger = 0,
  by = NULL,
  easing = "ease-in-out",
  loop = FALSE
)
```

Arguments

spec	A <code>glyph_spec</code>
transition	Transition type: "morph", "fade", "slide", "none"
duration	Duration in milliseconds
stagger	Delay between successive marks in ms (for entrance effects)
by	Column that defines animation keyframes (like <code>gganimate</code> 's <code>transition_states</code>). e.g. <code>by = year</code> plays through years.
easing	Easing function: "linear", "ease-in", "ease-out", "ease-in-out", "bounce", "elastic"
loop	Repeat animation (TRUE, FALSE, or number of times)

Value

Modified `glyph_spec`

Examples

```
# Entrance animation: bars grow from baseline
glyph(mtcars, x = cyl, y = mpg) |>
  mark_bar() |>
  animate(transition = "slide", stagger = 50)

# Keyframe animation: morph between groups
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(size = hp, color = cyl) |>
  animate(by = gear, transition = "morph", duration = 800)
```

Description

Animations in `glyph` are declarative state transitions, not frame-by-frame rendering. You describe *what changes* and *how it should interpolate*, and the renderer handles the rest.

Details

Two animation paradigms:

- **Transitions:** animate between data states (e.g. year 2020 → 2021). Marks morph smoothly, entering/exiting as data changes.
- **Entrances:** animate marks appearing on first render (stagger bars growing from zero, points fading in).

compile

Compile a glyph spec to a resolved representation

Description

Compile a glyph spec to a resolved representation

Usage

```
compile(spec, engine = "auto", width = NULL, height = NULL)
```

```
## S3 method for class 'glyph_spec'
compile(spec, engine = "auto", width = NULL, height = NULL)
```

```
## S3 method for class 'glyph_layout'
compile(spec, engine = "auto", width = NULL, height = NULL)
```

Arguments

spec	A <code>glyph_spec</code> or <code>glyph_layout</code>
engine	Rendering backend: "auto", "html", "svg", "canvas", "webgl"
width	Width in pixels (NULL for auto)
height	Height in pixels (NULL for auto)

Value

A `glyph_compiled` object (a list with resolved data + JSON)

A `glyph_compiled` object containing the resolved spec, JSON string, and engine selection

A `glyph_compiled` object containing the resolved layout spec and JSON string

`compose`*Compose multiple glyph specs into a layout*

Description

Compose multiple glyph specs into a layout

Usage

```
compose(  
  ...,  
  type = "hstack",  
  widths = NULL,  
  heights = NULL,  
  gap = 10,  
  shared_scales = FALSE,  
  linked_selections = FALSE,  
  title = NULL  
)
```

Arguments

<code>...</code>	glyph_spec objects or layout objects (for nesting)
<code>type</code>	Layout type: "hstack", "vstack", "grid", "wrap"
<code>widths</code>	Relative widths for columns (e.g. <code>c(2, 1)</code> for 2:1 split)
<code>heights</code>	Relative heights for rows
<code>gap</code>	Gap between plots in px
<code>shared_scales</code>	Share axis scales across plots: TRUE, FALSE, "x", "y", or "both"
<code>linked_selections</code>	Share interaction selections across plots
<code>title</code>	Overall layout title

Value

A `glyph_layout` object

Examples

```
p1 <- glyph(mtcars, x = wt, y = mpg) |> mark_point()  
p2 <- glyph(mtcars, x = wt, y = hp) |> mark_point()  
p3 <- glyph(mtcars, x = hp, y = mpg) |> mark_line()  
  
# Horizontal stack  
compose(p1, p2, type = "hstack")  
  
# Grid with linked brushing
```

```
compose(p1, p2, p3, type = "wrap",
        linked_selections = TRUE, shared_scales = "x")
```

export *Export to various formats*

Description

Export to various formats

Usage

```
export(spec, file, width = 800, height = 600)
```

Arguments

spec	A <code>glyph_spec</code> or <code>glyph_compiled</code>
file	Output file path. Extension determines format: <code>.html</code> , <code>.svg</code> , <code>.png</code> , <code>.pdf</code> , <code>.json</code> (exports the raw spec)
width	Width in pixels
height	Height in pixels

Value

Invisibly returns the output file path

facet *Facet a plot by one or two variables*

Description

Like `ggplot2`'s `facet_wrap`/`facet_grid` but with a key improvement: each facet can have independent geom parameters and scales by default (instead of forced uniformity).

Usage

```
facet(spec, rows = NULL, cols = NULL, free_scales = "none", wrap = NULL)
```

Arguments

spec	A <code>glyph_spec</code>
rows	Row faceting variable (bare name or <code>NULL</code>)
cols	Column faceting variable (bare name or <code>NULL</code>)
free_scales	"none", "x", "y", "both"
wrap	If only one variable, wrap into a grid with this many columns

Value

Modified `glyph_spec`

glyph	<i>Create a Glyph Visualization</i>
-------	-------------------------------------

Description

Entry point for the `glyph` grammar. Creates a specification object that describes a visualization declaratively. Unlike `ggplot2`, the `spec` is a pure data structure (nested list) that can be serialized, inspected, and compiled to multiple backends.

Usage

```
glyph(data = NULL, ...)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>tibble</code> . Unlike <code>ggplot2</code> , <code>glyph</code> also accepts lists-of-lists, URLs to CSV/JSON, or arrow tables (planned).
<code>...</code>	Global aesthetic mappings using bare column names (no <code>aes()</code> needed). e.g. <code>glyph(mtcars, x = wt, y = mpg)</code>

Value

A `glyph_spec` object

Examples

```
spec <- glyph(mtcars, x = wt, y = mpg) |>
  mark_point()
summary(spec)
```

inset	<i>Add an inset plot</i>
-------	--------------------------

Description

Add an inset plot

Usage

```
inset(spec, inset, position = "top-right")
```

Arguments

spec	The main glyph_spec
inset	A glyph_spec to place as an inset
position	Where to place: "top-right", "top-left", "bottom-right", "bottom-left", or a list(x, y, width, height) with proportions 0-1

Value

Modified glyph_spec

interact	<i>Add interactive behaviors to a glyph spec</i>
----------	--

Description

Add interactive behaviors to a glyph spec

Usage

```
interact(
  spec,
  tooltip = FALSE,
  zoom = FALSE,
  brush = FALSE,
  hover = NULL,
  click = NULL,
  crossfilter = FALSE,
  nearest = FALSE
)
```

Arguments

spec	A glyph_spec
tooltip	Show values on hover. TRUE for auto-generated, or a glue-style template string like "{x}: {y} ({color})".
zoom	Enable scroll-to-zoom and pan
brush	Enable rectangular brush selection
hover	Highlight mark on hover ("enlarge", "brighten", "outline", or NULL)
click	Action on click: "select", "filter", "url", or a callback name
crossfilter	Link this plot's selections to other plots in a layout
nearest	Snap selection to nearest point (useful for line charts)

Value

Modified glyph_spec

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  interact(
    tooltip = "{cyl} cylinders\n{mpg} mpg at {wt} tons",
    zoom = TRUE,
    hover = "enlarge",
    brush = TRUE,
    crossfilter = TRUE
  )
```

interactivity

First-Class Interactivity

Description

Unlike `ggplot2` where interactivity is bolted on via `ggplotly()`, `glyph` treats interaction as part of the visualization grammar. Interactions are declared in the spec and compiled to the appropriate backend (D3 events for HTML, Shiny bindings for server-side).

Details

Design philosophy: interactions are *selections* that filter or highlight marks. This follows Vega-Lite's insight that most interactions are really about defining subsets of the data. A tooltip is "select the nearest point, show its values." Brushing is "select points in a rectangle, highlight them."

layout

Layout Composition

Description

Built-in multi-plot composition. No external packages needed. Layouts can express grids, stacks, insets, marginal plots, and arbitrary nesting. Composed plots can share selections for cross-filtering (linked brushing).

marginals	<i>Add a marginal plot (histogram/density on axes)</i>
-----------	--

Description

Adds marginal distributions to a plot's axes — a common pattern that requires `ggExtra` or manual grid manipulation in `ggplot2`.

Usage

```
marginals(spec, x = "histogram", y = "histogram", size = 0.15)
```

Arguments

spec	A <code>glyph_spec</code>
x	Marginal type for x-axis: "histogram", "density", "boxplot", NULL
y	Marginal type for y-axis: "histogram", "density", "boxplot", NULL
size	Proportion of plot area for marginals (0.0 to 0.4)

Value

Modified `glyph_spec`

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  marginals(x = "histogram", y = "density")
```

mark_area	<i>Add an area mark</i>
-----------	-------------------------

Description

Add an area mark

Usage

```
mark_area(spec, ..., data = NULL, style = list())
```

Arguments

spec	A <code>glyph_spec</code>
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified glyph_spec object with the area mark added

mark_bar	<i>Add a bar mark</i>
----------	-----------------------

Description

Add a bar mark

Usage

```
mark_bar(spec, ..., data = NULL, style = list(), orient = "vertical")
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
orient	"vertical" or "horizontal"

Value

Modified glyph_spec object with the bar mark added

mark_line	<i>Add a line mark</i>
-----------	------------------------

Description

Add a line mark

Usage

```
mark_line(spec, ..., data = NULL, style = list(), interpolate = "monotone")
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
interpolate	Interpolation method: "linear", "monotone", "step", "basis"

Value

Modified glyph_spec object with the line mark added

mark_link	<i>Add a link/edge mark (for networks, sankeys, slope graphs)</i>
-----------	---

Description

Add a link/edge mark (for networks, sankeys, slope graphs)

Usage

```
mark_link(spec, ..., data = NULL, style = list())
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified glyph_spec object with the link mark added

mark_point	<i>Add a point mark (scatterplot)</i>
------------	---------------------------------------

Description

Add a point mark (scatterplot)

Usage

```
mark_point(spec, ..., data = NULL, style = list())
```

Arguments

spec	A glyph_spec
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified `glyph_spec`

Examples

```
spec <- glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl, size = hp, tooltip = "{cyl} cyl, {mpg} mpg")
```

mark_ribbon	<i>Add a ribbon/band mark (for confidence intervals, ranges)</i>
-------------	--

Description

Add a ribbon/band mark (for confidence intervals, ranges)

Usage

```
mark_ribbon(spec, ..., data = NULL, style = list())
```

Arguments

spec	A <code>glyph_spec</code>
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties

Value

Modified `glyph_spec` object with the ribbon mark added

mark_rule	<i>Add a rule (reference line) mark</i>
-----------	---

Description

Add a rule (reference line) mark

Usage

```
mark_rule(
  spec,
  ...,
  data = NULL,
  style = list(),
  x_intercept = NULL,
  y_intercept = NULL
)
```

Arguments

spec	A <code>glyph_spec</code>
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
x_intercept	Fixed x position for vertical rule
y_intercept	Fixed y position for horizontal rule

Value

Modified `glyph_spec` object with the rule mark added

mark_text	<i>Add a text/label mark</i>
-----------	------------------------------

Description

Add a text/label mark

Usage

```
mark_text(spec, ..., data = NULL, style = list(), smart_repel = TRUE)
```

Arguments

spec	A <code>glyph_spec</code>
...	Aesthetic mappings (x, y, color, size, shape, alpha, tooltip)
data	Optional per-mark data override
style	Named list of fixed visual properties
smart_repel	Automatically avoid label overlaps (TRUE by default). This is a first-class feature, not an extension package.

Value

Modified `glyph_spec` object with the text mark added

 marks

Visual Marks

Description

Marks are the visual encodings in `glyph` — analogous to `ggplot2`'s `geoms` but with key differences:

1. **Mappings without `aes()`**: pass bare column names directly.
2. **Per-mark data**: each mark can have its own data source, enabling multi-dataset plots without awkward data parameter overrides.
3. **Built-in interaction hints**: marks carry interaction metadata (what happens on hover, click, drag) as part of the spec.
4. **Transition-aware**: marks know how to interpolate between states for animation.

`print.glyph_spec`
Auto-render when printed (like `ggplot2`)

Description

Auto-render when printed (like `ggplot2`)

Usage

```
## S3 method for class 'glyph_spec'
print(x, ...)
```

Arguments

<code>x</code>	A <code>glyph_spec</code> object
<code>...</code>	Additional arguments (ignored)

Value

Invisibly returns the `glyph_spec` object

render	<i>Render a glyph spec as an htmlwidget</i>
--------	---

Description

Render a glyph spec as an htmlwidget

Usage

```
render(spec, width = NULL, height = NULL)
```

Arguments

spec	A <code>glyph_spec</code> , <code>glyph_layout</code> , or <code>glyph_compiled</code>
width	Widget width
height	Widget height

Value

An `htmlwidget` object that renders the visualization in an HTML viewer

rendering	<i>Rendering Pipeline</i>
-----------	---------------------------

Description

The rendering pipeline compiles a `glyph_spec` into output. The spec is first resolved (evaluate quosures, compute stats, merge defaults), then serialized to JSON, then handed to a backend:

- **"html"** (default): `htmlwidgets` + `D3.js` for interactive viewing
- **"svg"**: Static SVG file (publication quality)
- **"canvas"**: HTML5 Canvas for large-data performance
- **"webgl"**: WebGL via `regl/deck.gl` for 100K+ points (planned)
- **"pdf"**: Direct PDF output via R's `pdf()` device (planned)

Details

Key architectural difference from `ggplot2`: the spec is a pure data structure. The `compile()` step resolves it into a concrete render tree. This means you can:

- Inspect the compiled spec as JSON (for debugging or export)
- Serialize it and render on a different machine
- Export it to Vega-Lite JSON (near 1:1 mapping)
- Compile to multiple backends from one spec

scale	<i>Define a scale for an aesthetic channel</i>
-------	--

Description

Define a scale for an aesthetic channel

Usage

```
scale(
  spec,
  aesthetic,
  type = "auto",
  domain = NULL,
  range = NULL,
  nice = TRUE,
  zero = FALSE,
  reverse = FALSE,
  label = NULL,
  format = NULL
)
```

Arguments

spec	A <code>glyph_spec</code>
aesthetic	Which channel: "x", "y", "color", "size", "shape", "alpha"
type	Scale type: "linear", "log", "sqrt", "time", "ordinal", "band", "quantize", "threshold"
domain	Explicit domain (data range). NULL for auto.
range	Explicit output range. For color: a palette name or vector.
nice	Round domain to nice values (TRUE/FALSE)
zero	Force zero in domain (TRUE/FALSE)
reverse	Reverse the scale
label	Axis/legend label (NULL for auto from column name)
format	Format string for tick labels (e.g. "%.2f", "%b %Y")

Value

Modified `glyph_spec`

Examples

```
glyph(mtcars, x = wt, y = mpg) |>
  mark_point(color = cyl) |>
  scale("y", "linear", zero = TRUE, label = "Miles per gallon") |>
  scale("color", "ordinal", range = "Set2")
```

scales

Scales

Description

Scales map data values to visual properties. Glyph simplifies ggplot2's `scale_<aesthetic>_<type>` explosion into a composable system: `scale(<aesthetic>, <type>, ...)`.

Usage

```
scale_color(spec, palette = "Tableau10", ...)
```

```
scale_log(spec, aesthetic = "y", base = 10, ...)
```

```
scale_time(spec, aesthetic = "x", ...)
```

Arguments

<code>spec</code>	A <code>glyph_spec</code>
<code>palette</code>	A named palette: "viridis", "Set2", "Tableau10", "Blues", etc.
<code>...</code>	Additional arguments passed to <code>scale()</code>
<code>aesthetic</code>	Which channel: "x", "y", "color", "size", "shape", "alpha"
<code>base</code>	Log base (default 10)

Value

Modified `glyph_spec` object

Modified `glyph_spec` object

Modified `glyph_spec` object

Functions

- `scale_color()`: Set color palette by name
- `scale_log()`: Log-transform an axis
- `scale_time()`: Time/date axis

selection	<i>Add a selection parameter (advanced)</i>
-----------	---

Description

For complex interactions: define a named selection that can be referenced by marks and scales. This is how you build linked views, conditional encoding, and interactive legends.

Usage

```
selection(  
  spec,  
  name,  
  type = "point",  
  on = "click",  
  fields = NULL,  
  resolve = "global"  
)
```

Arguments

spec	A <code>glyph_spec</code>
name	Selection name (referenced in conditional encodings)
type	"point", "interval", or "legend"
on	Event trigger: "click", "mouseover", "drag"
fields	Which data fields the selection projects onto
resolve	For composed views: "global", "union", "intersect"

Value

Modified `glyph_spec`

Examples

```
# Interactive legend: click legend entries to filter  
glyph(mtcars, x = wt, y = mpg) |>  
  mark_point(color = cyl) |>  
  selection("legend_filter", type = "legend", fields = "cyl")
```

 theme_tokens

Set theme tokens

Description

Set theme tokens

Usage

```
theme_tokens(
  spec,
  preset = NULL,
  font = NULL,
  font_size = NULL,
  bg = NULL,
  fg = NULL,
  accent = NULL,
  grid = NULL,
  border = NULL,
  padding = NULL,
  title_size = NULL
)
```

Arguments

spec	A <code>glyph_spec</code>
preset	A named preset: "light" (default), "dark", "minimal", "publication", "presentation". Presets set all tokens to coherent defaults.
font	Font family for all text
font_size	Base font size in px (axis labels scale relative to this)
bg	Background color
fg	Primary text/axis color
accent	Primary accent color (used for single-series marks)
grid	Grid line visibility: TRUE, FALSE, "x", "y"
border	Plot border: TRUE/FALSE
padding	Padding around the plot area in px
title_size	Title font size multiplier (relative to font_size)

Value

Modified `glyph_spec`

Examples

```

glyph(mtcars, x = wt, y = mpg) |>
  mark_point() |>
  theme_tokens(preset = "dark")

glyph(mtcars, x = wt, y = mpg) |>
  mark_point() |>
  theme_tokens(font = "IBM Plex Sans", bg = "#fafafa", grid = "y")

```

theming	<i>Token-Based Theming</i>
---------	----------------------------

Description

Instead of ggplot2's 90+ theme() arguments, glyph uses a design-token system. A small set of semantic tokens (font, colors, spacing, sizes) cascade through the entire visualization. Think of it like CSS custom properties for plots.

Details

Tokens cascade: setting bg changes the background, but also automatically adjusts text color for contrast, grid line opacity, and tooltip styling. You override only what you want; everything else adapts.

titles	<i>Add a title, subtitle, or caption</i>
--------	--

Description

Add a title, subtitle, or caption

Usage

```
titles(spec, title = NULL, subtitle = NULL, caption = NULL)
```

Arguments

spec	A glyph_spec
title	Main title
subtitle	Subtitle (below title)
caption	Caption (bottom of plot, for source attribution)

Value

Modified glyph_spec object with updated title metadata

to_vegalite	<i>Export the spec as Vega-Lite JSON (interop)</i>
-------------	--

Description

Because glyph's spec is structurally similar to Vega-Lite, we can export to Vega-Lite JSON for use in Python (Altair), JavaScript, or the Vega Editor.

Usage

```
to_vegalite(spec)
```

Arguments

spec	A glyph_spec
------	--------------

Value

A JSON string in Vega-Lite format

Index

animate, [2](#)
animation, [3](#)

compile, [4](#)
compose, [5](#)

export, [6](#)

facet, [6](#)

glyph, [7](#)

inset, [7](#)
interact, [8](#)
interactivity, [9](#)

layout, [9](#)

marginals, [10](#)
mark_area, [10](#)
mark_bar, [11](#)
mark_line, [11](#)
mark_link, [12](#)
mark_point, [12](#)
mark_ribbon, [13](#)
mark_rule, [13](#)
mark_text, [14](#)
marks, [15](#)

print.glyph_spec, [15](#)

render, [16](#)
rendering, [16](#)

scale, [17](#)
scale_color (scales), [18](#)
scale_log (scales), [18](#)
scale_time (scales), [18](#)
scales, [18](#)
selection, [19](#)

theme_tokens, [20](#)

theming, [21](#)
titles, [21](#)
to_vegalite, [22](#)