Package 'ivue'

November 6, 2026

Title Interactive 3D Visualization of Data and Graphs

Version 0.1.1

Description Creates interactive three-dimensional point clouds and embedded weighted graphs with numerical or categorical annotations. Provides reusable continuous and categorical color scales, matching legends, highlighting, and geometric edge, path, and label layers. Supports multiple weighted-graph formats and optional layouts through 'igraph', with explicit distance or strength weight semantics. Renders browser widgets using 'rgl' without requiring a native graphics window. Plays recorded coordinate frames with interactive controls and exports orthographic graph animations to GIF.

License GPL (>= 3)

URL https://pgajer.github.io/ivue/, https://github.com/pgajer/ivue

BugReports https://github.com/pgajer/ivue/issues

Encoding UTF-8

Language en-US

Depends R (>= 4.1.0)

Imports htmlwidgets, htmltools, grDevices, graphics, stats, utils, methods

Suggests grip, rgl, magick, shiny, geometry, igraph, Matrix, callr, pkgload, testthat (>= 3.1.7), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Pawel Gajer [aut, cre]

Maintainer Pawel Gajer <pgajer@gmail.com>

Repository CRAN

Date/Publication 2026-11-06 20:50:18 UTC

Contents

ivue-package . . . 2
animate.frames . . . 4
camera.zup . . . 6
color.scale.cont . . . 7
layer3D.axes . . . 9
layer3D.callback . . . 11
layer3D.edges . . . 12
layer3D.mesh . . . 13
layer3D.surface . . . 14
plot3D.graph . . . 16
plot3D.plain . . . 18
prepare.graph . . . 21
print.ivue_graph . . . 23
write.animation.gif . . . 23

Index **26**

`ivue-package` *ivue: Interactive 3D Visualization of Data and Graphs*

Description

Explore three-dimensional point clouds and embedded graphs with numerical or categorical annotations, reusable color scales, and geometric layers.

Start here

Open Finding your way around ivue to choose a task, or Example data and recipes for reproducible inputs. The small example below constructs a point cloud with a named numerical annotation; print `view` interactively to display it. Plotting requires the optional rgl package: `install.packages("rgl")`. Color mapping and `prepare.graph()` work without it. No XQuartz setup is required.

`plot3D.plain()` shows a point cloud, `plot3D.cont()` maps numerical values, and `plot3D.groups()` maps categorical annotations. Each returns a browser widget. Printing it in RStudio opens the Viewer; in an interactive R console it opens the browser. Function execution itself does not launch a browser. Use `htmlwidgets::saveWidget()` to export HTML for later viewing or sharing.

`prepare.graph()` exposes IDs, edge order, and attributes before rendering. `plot3D.graph()` accepts that object with supplied coordinates or an explicit layout. Weights do not automatically control edge width or color.

Guides

Five installed vignettes provide complementary starting points:

- Finding your way around ivue: task-oriented function catalog.
- Example data and recipes: reproducible inputs and reusable recipes.

- Point clouds and weighted graphs: detailed plotting controls.
- Retinal development: data provenance and case study.
- Coordinate animations: recorded frames and export.

From the console, use `vignette("function-guide", package = "ivue")`, `vignette("example-data", package = "ivue")`, or `vignette(package = "ivue")` to list all five. Built package distributions include these guides for offline reading.

Rendering

Widgets use private rgl null-device scenes. The rgl namespace is loaded only when rendering; caller options and the previous device are restored. See `vignette("ivue-introduction", package = "ivue")` for worked examples.

Author(s)

Maintainer: Pawel Gajer <pgajer@gmail.com>

Authors:

- Pawel Gajer <pgajer@gmail.com>

See Also

Useful links:

- https://pgajer.github.io/ivue/
- https://github.com/pgajer/ivue
- Report bugs at https://github.com/pgajer/ivue/issues

Examples

```
library(ivue)
X <- rbind(a = c(0, 0, 0), b = c(1, 1, 1), c = c(2, 0, 0))
height <- c(c = 0, a = 0, b = 1)
if (nzchar(system.file(package = "rgl"))) {
  view <- plot3D.cont(X, height, legend.title = "Height")
  # Print view interactively to display it; construction opens no window.
}
```

`animate.frames` *Play Recorded Coordinate Frames*

Description

Display a sequence of point clouds or embedded graphs with browser playback controls. No layout algorithm, coordinate alignment, or interpolation is applied. The camera can be rotated while playback is paused or running.

Usage

```
animate.frames(
  frames,
  edges = NULL,
  labels = NULL,
  frame.index = NULL,
  max.frames = 100L,
  fps = 6,
  loop = TRUE,
  col = "#197A68",
  point.size = 5,
  edge.col = "gray65",
  edge.width = 1,
  camera = NULL,
  width = NULL,
  height = 600L,
  background.color = "white",
  mapping = NULL,
  legend.title = "Color",
  caption = NULL,
  description = NULL,
  controls = TRUE
)
```

Arguments

`frames`	List of at least two numeric n-by-2 or n-by-3 matrices with identical dimensions. A row is one vertex throughout the sequence. Each row must be entirely finite or entirely missing (NA or NaN, an inactive vertex). If row names are supplied, every frame must have the same unique names in the same order. Two-dimensional coordinates are embedded in the z=0 plane.
`edges`	Optional two-column matrix of one-based vertex indices, shared across frames. An edge is visible only when both endpoints are active.
`labels`	Optional character labels, one per original frame.
`frame.index`	Optional strictly increasing original frame indices to retain. Selection is explicit and takes precedence over max.frames.

max.frames	Maximum frames retained by evenly spaced subsampling, including the first and last. NULL keeps all frames. Subsampling reports a message; original indices remain in the timeline and returned metadata.
fps	Frames per second at the initial playback speed, from 0.1 to 100.
loop	Repeat browser playback.
col	Point colors, length one or n. Alpha components are preserved.
point.size	Point diameter in screen pixels.
edge.col	Edge colors, length one or the number of edges.
edge.width	Positive edge width in screen units.
camera	Initial camera specification, as in plot3D.plain(). The default is orthographic: face-on for 2D and camera.zup() for 3D.
width, height	Widget dimensions, as in plot3D.plain().
background.color	Canvas background color.
mapping	Optional result of map.colors(), with one color per vertex in frame row order. Mutually exclusive with col. Carries a fixed numerical or categorical legend into saved interactive HTML. Colors do not change with the frames; describe their meaning with caption.
legend.title	Title for the mapping's color legend.
caption	Optional plain-text interpretation retained below the widget when saved as HTML; for example, 'Color: final saddle height; positions: current frame.' For GIF output, request annotations = TRUE in write.animation.gif().
description	Optional plain-text scene description for readers who cannot see or manipulate the canvas. NULL describes the point count. Also shown below the widget, including when scripts or WebGL are unavailable.
controls	Show keyboard-operable view controls: rotate, zoom, reset, and download current view settings as an R recipe. The recipe contains camera, bounds, and aspect; use source("ivue-view.R"), then pass view$camera, view$limits, and view$aspect to a new plot. Browser interaction never changes the original R object. Match widget dimensions as well as settings for equal screen scale. Reset restores the initial view.

Details

Playback starts paused and steps between recorded frames. All frames have equal duration, even after subsampling; the timeline does not represent solver wall time. Bounds are fitted once to all original frames, including omitted frames. Inactive rows may appear or disappear at any step; missing positions are never interpolated. Supplied colors retain their association with vertex rows and edge rows. Numeric point, edge, and background palette indices are resolved when the widget is created, so later palette changes do not alter playback or GIF export.

Large traces increase widget size approximately with the product of frame count and the number of vertices plus edge endpoints. Use max.frames or frame.index to limit output size. A frame can be empty, but the full sequence must contain at least one finite point.

Only points and optional straight edges are animated. Use col with map.colors() to reuse numerical or categorical color scales; passing mapping instead also preserves its legend. Ordinary static plotting and validation retain their stricter finite-coordinate requirements. rgl is loaded only when a widget is constructed.

Value

An rglwidget/htmlwidget with an attached player. Save interactive output using htmlwidgets::saveWidget(). In Shiny, return the complete animation from shiny::renderUI() into shiny::uiOutput() so its separate player and caption are included; rgl::renderRglwidget() returns only the scene and is suitable for static views. attr(widget, "ivue.animation") contains the retained frames, original frame indices, labels, active masks, edges, fixed bounds, styles, fps, and initial camera for GIF export.

See Also

write.animation.gif()

Examples

```
X <- rbind(c(0, 0), c(1, 0), c(0, 1))
first <- X; first[3, ] <- NA
frames <- list(first, X, X * 1.5)
edges <- rbind(c(1, 2), c(2, 3), c(3, 1))
if (nzchar(system.file(package = "rgl"))) {
  w <- animate.frames(frames, edges, fps = 2)
}
```

---

camera.zup *A Z-Up Initial Camera*

Description

Construct a camera without opening a graphics device or loading rgl.

Usage

```
camera.zup(elevation = 20, turn = -135, fov = 0, zoom = 0.8)
```

Arguments

elevation	Viewing elevation in degrees above the xy plane, from -90 to 90. At either pole the z axis points along the viewing direction.
turn	Rotation about the data's z axis, in degrees. Zero puts positive x to the right and positive y away from the viewer. The default -135 puts positive x down-left and positive y down-right at positive elevation.
fov	Field of view in degrees, from 0 to 179. Zero gives orthographic projection without perspective foreshortening.

zoom — Positive rgl zoom parameter: smaller values enlarge the scene, larger values show a wider view. Browser and GIF export use this convention.

Details

This sets the initial view, not an interactive rotation constraint. Away from the poles, positive z projects upward. Interactive dragging can subsequently tilt it. The rotation is Rx(elevation - 90) times Rz(turn).

Value

A list with userMatrix, fov, and zoom, accepted by the camera argument of every plot3D function.

See Also

layer3D.axes(), plot3D.plain()

Examples

```
camera.zup()
camera.zup(elevation = 30, turn = 0)
```

---

color.scale.cont *Reusable Color Scales*

Description

Construct a scale once to keep colors comparable across scenes. Scale construction and mapping do not load rgl or create a graphics device.

Usage

```
color.scale.cont(
  values,
  mode = c("continuous", "binned"),
  palette = NULL,
  color.map = NULL,
  limits = NULL,
  center = NULL,
  breaks = NULL,
  n.bins = 10L,
  method = c("uniform", "quantile"),
  winsor.p = 0,
  oob = c("squish", "censor", "error"),
  na.color = "gray80",
  digits = NULL
)
```

```
color.scale.groups(
  groups,
  colors = NULL,
  na.color = "gray80",
  unknown = c("error", "missing")
)

map.colors(x, scale)
```

Arguments

values	Numeric reference values. Missing values are allowed; infinity is rejected. Automatic limits are fitted to these reference values only.
mode	Continuous interpolation (default) or explicit bins.
palette	R colors or a function of the requested number of colors. NULL uses the Viridis HCL palette. Numeric palette indices are resolved when fitting the scale, so later session palette changes have no effect.
color.map	Optional function of numeric values returning R colors. Mutually exclusive with palette. Must be deterministic and pointwise: a value's color cannot depend on other values, their order, or call count. Receives data-unit values after out-of-bounds handling, not normalized palette positions. Observations, legend ticks, and the ramp are mapped in separate calls. The caller is responsible for this contract.
limits	Two finite, nondecreasing numeric limits. Equal limits are permitted for constant data.
center	Optional reference value for a continuous diverging scale. Supply an appropriate diverging palette explicitly. Automatic limits are symmetric about center; explicit limits must contain it strictly. With palette, center maps to the palette midpoint. With color.map, center can affect automatic limits but does not transform the values passed to the callback.
breaks	Strictly increasing numerical bin boundaries, or NULL.
n.bins	Positive number of requested bins.
method	Uniform or quantile bin boundaries.
winsor.p	Explicit tail probability used when fitting uniform bins. Zero (default) disables winsorization; must be less than 0.5.
oob	Out-of-bounds handling: squish to limits, use missing color, or error.
na.color	Color for missing values and censored out-of-bounds values.
digits	NULL (default) increases significant digits from 3 up to 17 until distinct legend boundaries/ticks have distinct labels. An explicit integer fixes precision and warns if labels become ambiguous. Does not change bin calculations.
groups	Reference group labels or a factor. Factors retain level order; other inputs use first-occurrence order. Missing factor levels and values use na.color, distinct from the literal group "NA". Empty strings are valid groups. Legend labels are quoted when empty strings or a group named "Missing" occur, keeping them distinct from the missing-value legend entry.

colors	Optional named group colors, covering all nonmissing reference levels. Names must be unique and nonmissing; an empty name identifies the empty-string group. Numeric colors are fixed when the scale is fitted.
unknown	Whether unseen groups raise an error or use the missing color.
x	Values or groups to map, according to the scale type.
scale	A scale constructed by color.scale.cont() or color.scale.groups().

Details

Prefer a palette and fixed limits for comparable views. A custom color.map must apply the same rule to each value regardless of the batch. For example, function(x) ifelse(x < 0, "blue", "red") is pointwise. A mapper that computes range(x), ranks, or quantiles from the current batch to choose its colors is not supported: it can disagree with its own legend and assign different colors to the same value in different views, even without mutable external state. Fit such reference quantities once and capture them in a fixed mapping function instead.

Value

A scale of class ivue_color_scale. map.colors() returns a list containing row-aligned colors, a legend data frame (label, color, count), and the scale. Continuous legend ticks have NA counts; binned and group counts describe the mapped input. A Missing entry is added as needed.

See Also

plot3D.cont(), plot3D.groups(), ivue-package, Finding your way around ivue, Shared-scale recipe.

Examples

```
sc <- color.scale.cont(c(-1, 0, 1))
map.colors(c(-1, 0.5, NA), sc)
groups <- factor(c("low", "high", "low"), levels = c("low", "high"))
group.scale <- color.scale.groups(groups, c(low = "blue", high = "red"))
map.colors(groups, group.scale)
fixed.map <- function(x) ifelse(x < 0, "blue", "red")
custom <- color.scale.cont(c(-2, 4), color.map = fixed.map, limits = c(-2, 4))
map.colors(c(-1, 0, 2), custom)$colors
```

---

layer3D.axes *Coordinate Axes Through an Origin*

Description

Add three coordinate axes with positive-end arrowheads, rather than a bounding box. Use axes = FALSE in the plotting call to suppress its ordinary axes. This layer never changes the camera; camera.zup() supplies a complementary initial view.

Usage

```
layer3D.axes(
  origin = c(0, 0, 0),
  limits = NULL,
  padding = 0.2,
  labels = c("x", "y", "z"),
  col = "black",
  width = 2,
  head.length = 0.04,
  head.angle = pi/8,
  cex = 1.2,
  label.offset = 0.04
)
```

Arguments

origin	Three finite coordinates at which the axes intersect.
limits	NULL for automatic limits, or a finite numeric 3-by-2 matrix with rows x, y, z and columns lower, upper. Each row must strictly contain the corresponding origin coordinate. Row and column names are optional.
padding	Nonnegative fraction added to automatic half-lengths. Ignored when limits are supplied.
labels	Three axis labels. Empty strings suppress individual labels.
col	Axis colors, length one or three, also used for heads and labels.
width	Positive shaft line widths, length one or three, in screen units.
head.length	Arrowhead length as a fraction of each full axis span, from 0 to 0.25. Zero omits heads. Length is capped at 80 percent of the positive arm to keep the head beyond the origin.
head.angle	Cone half-angle in radians, strictly between 0 and pi/2.
cex	Positive label size multiplier.
label.offset	Nonnegative gap beyond each positive tip, as a fraction of that axis's full span.

Details

Automatic limits are symmetric about origin and enclose all rows of X. A coordinate with no extent uses the largest other half-length, or one data unit if all points coincide with origin. Arrowheads are solid cones in data coordinates, not screen-facing decorations, so they rotate with the scene. Their proportions assume aspect = "equal"; independently normalizing coordinate axes can distort them. Limits specify axis endpoints, not clipping bounds for the data. There are no tick marks.

Value

An ivue_layer specification for the layers argument.

See Also

camera.zup(), layer3D.edges(), plot3D.cont()

Examples

```
axes <- layer3D.axes(head.length = 0.04)
if (nzchar(system.file(package = "rgl"))) {
  X <- rbind(c(-1, -1, 0), c(1, 0, 1), c(0, 1, -1))
  w <- plot3D.plain(X, axes = FALSE, layers = list(axes),
                    camera = camera.zup())
}
```

---

layer3D.callback *Callback Layers*

Description

An advanced escape hatch for drawing with rgl. The callback must not open, close, or switch devices. Its context contains X, row.ids (integer positions), observation.ids (coordinate row names, or NULL), colors, highlight, and draw.ids (row, object, index). Captured object IDs are not live devices.

Usage

```
layer3D.callback(fun, args = list())
```

Arguments

fun	Function called with context as its first argument.
args	Named list of additional arguments to fun.

Value

An ivue_layer specification.

Examples

```
labels <- layer3D.callback(function(ctx) {
  rgl::text3d(ctx$X[1, , drop = FALSE], texts = "First point")
})
if (nzchar(system.file(package = "rgl"))) {
  w <- plot3D.plain(matrix(1:9, ncol = 3), layers = list(labels))
}
```

layer3D.edges *Geometric Layers for a 3D Scene*

Description

Layers are evaluated on the scene's private device before widget capture. They never open devices themselves. Row indices refer to the original X.

Usage

```
layer3D.edges(edges, col = "gray65", width = 1)

layer3D.path(path, col = "red3", width = 2)

layer3D.labels(
  rows,
  labels,
  col = "black",
  cex = 1,
  adj = c(0.5, 0.5),
  offset = c(0, 0, 0)
)
```

Arguments

edges	Two-column matrix of one-based endpoint indices. Empty edges are allowed. Self-loops are rejected; duplicate edges retain their order.
col	Color, scalar or one per edge/path segment/label.
width	Positive line width, scalar or one per edge/path segment.
path	Ordered vector of row indices. Fewer than two indices draws nothing.
rows	Row indices for labels.
labels	Text for each selected row.
cex	Positive text size multiplier.
adj	Two finite label-adjustment values.
offset	Three finite offsets added to label positions in data units.

Value

An ivue_layer specification for the layers argument.

Examples

```
edges <- matrix(c(1, 2, 2, 3), ncol = 2, byrow = TRUE)
layer3D.edges(edges)
layer3D.path(c(1, 3, 2), col = "red", width = 2)
layer3D.labels(c(1, 2), c("Start", "End"))
```

`layer3D.mesh` *A Triangular Surface Layer*

Description

Draw supplied triangular faces using the plotting coordinates. This layer does not construct a triangulation or change a graph used for analysis.

Usage

```
layer3D.mesh(
  triangles,
  col = "gray75",
  alpha = 0.2,
  edges = TRUE,
  edge.col = "gray45",
  edge.alpha = 0.35,
  edge.width = 1,
  lit = FALSE
)
```

Arguments

`triangles`	Numeric matrix with three columns of one-based vertex indices, one face per row. Indices refer to the plot's X, after vertex-ID alignment for graph plots. Empty matrices are allowed. Repeated vertices within a face and duplicate faces (including reversed faces) are rejected.
`col`	Face colors, length one or one per triangle. Colors are constant within each face; they do not inherit the point color scale.
`alpha`	Face opacity multiplier, length one or one per triangle, in `[0, 1]`. Multiplies any alpha already present in col. Zero hides the faces.
`edges`	Draw mesh edges. Each undirected edge is drawn once, even when shared by two faces.
`edge.col`	Single mesh-edge color.
`edge.alpha`	Mesh-edge opacity multiplier in `[0, 1]`, independent of face opacity and multiplied by the alpha in edge.col.
`edge.width`	Positive mesh-edge width in screen units.
`lit`	Apply rgl lighting to faces. FALSE keeps face colors independent of orientation. Both sides are drawn; consistent face winding is advisable when enabling lighting.

Details

The same layer can be reused with different coordinates as long as vertex identities and row order are preserved. Connectivity is never recomputed after embedding. Geometrically collapsed or collinear triangles are retained: the layer does not repair folds, degeneracies, intersections, or inconsistent orientation. It does not require a manifold mesh.

Faces are planar interpolations between vertices, not an exact smooth surface or a new shortest-path graph. Rendering requires rgl; constructing the layer does not. A polygon offset reduces interference between faces and their edge overlay. Transparency is handled by the renderer and can have ordering artifacts for intersecting surfaces.

Value

An ivue_layer specification for the layers argument.

See Also

layer3D.surface(), layer3D.edges(), layer3D.axes(), plot3D.cont()

Examples

```
triangles <- rbind(c(1, 2, 3), c(1, 3, 4))
surface <- layer3D.mesh(triangles, alpha = 0.2)
if (nzchar(system.file(package = "rgl"))) {
  X <- rbind(c(-1, -1, 0), c(1, -1, 0), c(1, 1, 1), c(-1, 1, 0))
  w <- plot3D.plain(X, layers = list(surface, layer3D.axes()),
                    camera = camera.zup())
}
```

---

layer3D.surface *An Independently Positioned Gridded Surface*

Description

Add a reference surface with its own coordinates to any plot3D scene. Unlike layer3D.mesh(), the surface does not use the plotted observations as its vertices and stays fixed when reused with another configuration.

Usage

```
layer3D.surface(
  x,
  y,
  z,
  col = "gray75",
  alpha = 0.2,
  edges = FALSE,
  edge.col = "gray45",
```

```
  edge.alpha = 0.35,
  edge.width = 1,
  lit = FALSE
)
```

Arguments

x, y	Finite numeric coordinate vectors, each of length at least two, strictly increasing or strictly decreasing.
z	Finite numeric matrix with length(x) rows and length(y) columns. Entry z[i, j] is the height at (x[i], y[j]); use outer(x, y, fun) to evaluate a height function on the grid. Missing values are not supported.
col	Face color, length one or one per grid cell. Cell order has the x index varying fastest, then the y index. Both triangles in a cell have the same color; colors do not inherit the plot's point color scale.
alpha	Face opacity multiplier, length one or one per grid cell, in [0, 1]. Multiplies any opacity in col. Zero hides the faces.
edges	Draw grid lines, without the triangulation diagonals.
edge.col	Single grid-line color.
edge.alpha	Grid-line opacity multiplier in [0, 1].
edge.width	Positive grid-line width in screen units.
lit	Apply lighting to faces. FALSE keeps colors independent of orientation; TRUE helps reveal surface shape. Both sides are drawn.

Details

Each rectangular parameter cell is split along the diagonal from (i, j) to (i+1, j+1). The result is a piecewise-planar approximation, not an exact smooth surface. A finer grid improves the approximation. The surface contributes to the scene bounds, but automatic layer3D.axes limits are based on the plotted observations; supply explicit axis limits if needed. No alignment or rescaling of either set of coordinates is done. Align an embedding to the reference coordinates before interpreting their spatial agreement. Transparent intersecting surfaces can have rendering order artifacts. Construction requires neither rgl nor geometry; rendering uses rgl on the plot's private device.

Value

An ivue_layer specification for the layers argument.

See Also

layer3D.mesh(), layer3D.axes(), plot3D.cont()

Examples

```
x <- y <- seq(-1, 1, length.out = 31)
z <- outer(x, y, function(x, y) 0.8 * (x^2 - y^2))
reference <- layer3D.surface(x, y, z, col = "lightblue", alpha = 0.3)
```

```
if (nzchar(system.file(package = "rgl"))) {
  X <- rbind(c(-0.5, 0, 0.2), c(0, 0.5, -0.2), c(0.5, 0.5, 0))
  w <- plot3D.plain(X, point.type = "sphere", sphere.radius = 0.03,
      layers = list(reference, layer3D.axes()), camera = camera.zup())
}
```

`plot3D.graph` *Draw a Weighted Embedded Graph*

Description

Graph input is normalized independently of the rendering engine. Supplied coordinates never trigger layout computation or igraph construction.

Usage

```
plot3D.graph(
  graph,
  X = NULL,
  layout = NULL,
  vertices = NULL,
  directed = NULL,
  weight.type = NULL,
  seed = 1L,
  edge.col = "gray65",
  edge.width = 1,
  values = NULL,
  groups = NULL,
  layers = list(),
  ...
)
```

Arguments

`graph`	A named list with adj.list and weight.list (neighbors are row indices); a list with edges and vertices; a weighted data frame with from, to, weight columns plus the vertices argument; a numerical square adjacency matrix; a Matrix sparse adjacency matrix; or an igraph object. Matrices use zero for absent edges; use lists/tables to represent zero-weight edges. A graph returned by `prepare.graph()` can be reused directly.
`X`	Finite n-by-3 coordinates. Unnamed rows follow vertex order; named rows must match vertex IDs exactly and are aligned to graph vertex order.
`layout`	NULL when X is supplied, otherwise "fr", "kk", or a function of the normalized graph returning n-by-3 coordinates. Supply exactly one of X and layout. Custom functions receive vertices (id plus attributes), edges (integer from/to indices, weight, attributes), directed, weight.type.

`vertices` Explicit vertex IDs or a data frame with a unique id column. Required for edge-table input, including isolated vertices. Adjacency lists default to list names, or character row numbers when unnamed.

`directed` Logical directedness; NULL uses stored directedness or FALSE. This release rejects directed rendering, self-loops, and parallel edges. Undirected adjacency lists must be reciprocal with equal weights.

`weight.type` "distance", "strength", or "unweighted". Required for weighted layout algorithms but not supplied-coordinate drawing. The fr algorithm requires strengths; kk requires distances. No inversion occurs. Unweighted mode only accepts missing or unit weights.

`seed` Seed used locally for layout computation, without changing the caller's random-number state.

`edge.col`, `edge.width` Explicit visual attributes, scalar or one per normalized edge. They are not automatically inferred from weights.

`values`, `groups` Optional vertex coloring. Unnamed vectors follow graph vertex order. Named vectors must match vertex IDs exactly and are reordered to that order; partial, duplicate, missing, or extra names are rejected. Supply at most one; scales and legends use the corresponding point family. Default categorical colors use factor levels or first occurrence in the supplied annotation vector, before ID alignment, as in `plot3D.groups()`.

`layers` Additional layer3D specifications.

`...` Named controls for the selected point family. Legacy graph-layout and basin arguments are not supported.

Details

Only fr and kk need igraph. Negative and zero finite weights can be stored/drawn, but these layout algorithms require positive weights. Explicit sparse zeros are rejected because zero-edge semantics would be ambiguous. Duplicate/asymmetric adjacency entries are rejected, not averaged. igraph vertex names supply canonical IDs; a pre-existing id attribute is retained as .igraph.id (an existing .igraph.id attribute is a conflict). Named per-vertex `col`, highlight-style color vectors, and logical `highlight` masks use the same exact-ID alignment as values/groups. Unnamed scalar colors are recycled. Numeric highlight indices always refer to graph vertex order, not the supplied coordinate row order.

Value

A widget with normalized graph data in attr(widget, "ivue")$graph. Its observation.ids are graph vertex IDs; row.ids remain integer positions in graph vertex order.

See Also

`prepare.graph()`, `plot3D.plain()`, `layer3D.edges()`

Examples

```
g <- list(adj.list = list(2L, c(1L, 3L), 2L, integer()),
          weight.list = list(2, c(2, 4), 4, numeric()))
X <- rbind(c(0, 0, 0), c(1, 1, 0), c(2, 0, 1), c(0, 2, 1))
if (nzchar(system.file(package = "rgl"))) w <- plot3D.graph(g, X = X)
```

`plot3D.plain` *Interactive 3D Point Clouds*

Description

These functions always return a browser widget. They use a private null device and restore caller graphics options and the previous device. Loading ivue does not load rgl. No XQuartz or native display is required.

Usage

```
plot3D.plain(
  X,
  col = "gray55",
  point.type = c("point", "sphere"),
  point.size = 3,
  sphere.radius = NULL,
  alpha = 1,
  highlight = NULL,
  highlight.style = list(),
  non.highlight.style = list(col = "gray80", alpha = 0.4),
  axes = FALSE,
  xlab = "",
  ylab = "",
  zlab = "",
  aspect = c("equal", "normalized"),
  camera = list(),
  width = NULL,
  height = 600L,
  background.color = "white",
  layers = list(),
  shiny.brush = NULL,
  limits = NULL,
  description = NULL,
  controls = TRUE
)

plot3D.cont(
  X,
  values,
  scale = NULL,
```

```
  legend.show = TRUE,
  legend.title = "Value",
  legend.position = c("left", "right"),
  legend.font.size = 12,
  legend.width = 240,
  ...
)

plot3D.groups(
  X,
  groups,
  scale = NULL,
  legend.show = TRUE,
  legend.title = "Group",
  legend.position = c("left", "right"),
  legend.font.size = 12,
  legend.width = 240,
  ...
)
```

Arguments

X	Numeric matrix or all-numeric data frame with exactly three columns and at least one row. Coordinates must be finite; rows are never dropped. Explicit row names are unique, nonempty, nonmissing observation IDs. Automatic data-frame row numbers are not IDs. Point plots keep row order.
col	Plain point colors, length one or nrow(X).
point.type	Draw screen-space points or data-space spheres.
point.size	Positive point size in screen pixels.
sphere.radius	Positive radius in data units. NULL uses 1 percent of the largest coordinate span, with a minimum of 1e-8. Does not choose type.
alpha	Opacity multiplier in [0, 1], preserving alpha in supplied colors.
highlight	NULL (all), a logical mask, or one-based row indices.
highlight.style, non.highlight.style	Named style overrides: point.type, point.size, sphere.radius, col, alpha. Color vectors must align to all rows. Highlighting changes styling, never the fitted color scale or row identity. A style's alpha replaces the global alpha multiplier for that subset; it still multiplies the alpha component of the selected colors.
axes	Show axes.
xlab, ylab, zlab	Axis labels.
aspect	Equal data-unit scales (default), or normalized axis lengths. Normalization distorts relative distances when coordinate spans differ.
camera	Named list of theta, phi, fov, zoom, or a 4-by-4 userMatrix. Downloaded recipes also include observer (three finite eye coordinates, positive depth), which is tied to that scene's framing. Omit observer to fit the eye distance automatically when transferring an orientation. With no orientation supplied, defaults to

camera.zup(): z upward, elevation 20 degrees, turn -135 degrees, orthographic projection, and zoom 0.8. A list containing only fov or zoom retains this orientation. Explicit theta, phi, or userMatrix selects an rgl camera instead; omitted controls then retain the previous defaults theta = 35, phi = 20, fov = 30, and zoom = 0.8. Use camera.zup() to customize a z-up view.

width, height — Widget dimensions in pixels; NULL width fills its container.

background.color — Canvas background color.

layers — List of layer3D specifications, evaluated before widget capture.

shiny.brush — Optional rgl brush configuration passed as shinyBrush.

limits — Optional finite 3-by-2 matrix: rows x, y, z; columns lower, upper. Nondecreasing ranges must contain all point coordinates. These fix framing, not clipping planes: spheres and layers cannot expand the range and may extend outside the visible viewport. NULL fits automatically. Equal endpoints are accepted for constant axes. Use the same limits, camera, aspect, and widget dimensions for spatial comparisons. Equal aspect preserves data-unit distances; normalized aspect stretches axes according to these ranges. Limits never add observations or change IDs.

description — Optional plain-text scene description for readers who cannot see or manipulate the canvas. NULL describes the point count. Also shown below the widget, including when scripts or WebGL are unavailable.

controls — Show keyboard-operable view controls: rotate, zoom, reset, and download current view settings as an R recipe. The recipe contains camera, bounds, and aspect; use source("ivue-view.R"), then pass view$camera, view$limits, and view$aspect to a new plot. Browser interaction never changes the original R object. Match widget dimensions as well as settings for equal screen scale. Reset restores the initial view.

values — Numeric values, one per row. Missing values use the scale's NA color.

scale — Reusable scale, or NULL to fit a default scale to all values/groups.

legend.show — Show the HTML color legend.

legend.title — Legend title.

legend.position — Side of the scene for the legend.

legend.font.size — Legend font size in pixels.

legend.width — Legend maximum width in pixels, constrained by the container.

... — Named scene controls from plot3D.plain, excluding X and col. Unknown names and legacy argument spellings are rejected.

groups — Group labels or factor, one per row; groups need not be clusters.

Details

Named values, groups, per-point col, logical highlight, and style color vectors are matched to rownames(X), using the same exact-ID rule as plot3D.graph(). Their names must cover every

observation exactly once. Missing, empty, duplicate, partial, or extra names cause errors, as do named annotations without explicit coordinate row names. Only unnamed scalar colors are recycled. Unnamed vectors follow coordinate row order; use `unname()` explicitly if annotation names are not observation IDs. Numeric highlight indices and indexed layers always use coordinate row positions, regardless of names attached to those indices. Plotting never reorders point coordinates or infers IDs from an annotation vector.

Without a supplied categorical scale, factor levels set the color order; otherwise groups use first occurrence in the supplied annotation vector, before ID alignment. The same named vector therefore gives matching group colors in point and graph views even if their coordinate orders differ. Reuse a scale to keep colors fixed when annotation order or membership changes.

Value

An rglwidget/htmlwidget. `attr(widget, "ivue")` contains coordinates, row.ids (integer row positions), observation.ids (explicit coordinate row names, or NULL), mapped colors, highlight, draw.ids (row, object, index), camera, aspect, captured scene, and (for colored plots) mapping data. Object IDs describe the captured scene, not an open device. Save separately with `htmlwidgets::saveWidget()`. Mapped colors describe the base scale before highlight and opacity overrides. Legends reflect the scale and global alpha, not highlight styles.

See Also

`color.scale.cont()`, `map.colors()`, `plot3D.graph()`, ivue-package, Finding your way around ivue, Example data and recipes.

Examples

```
set.seed(1)
xs <- runif(250, -1, 1)
ys <- runif(250, -1, 1)
X <- cbind(xs, ys, 1.2 * (xs^2 - ys^2))
if (nzchar(system.file(package = "rgl"))) w <- plot3D.plain(X, axes = TRUE)
if (nzchar(system.file(package = "rgl"))) {
  sc <- color.scale.cont(X[, 3])
  continuous <- plot3D.cont(X, X[, 3], scale = sc)
  grouped <- plot3D.groups(X, ifelse(X[, 3] >= 0, "positive", "negative"))
  positions <- rbind(a = c(0, 0, 0), b = c(1, 1, 1), c = c(2, 0, 0))
  annotation <- c(c = 10, a = 0, b = 5)
  by.id <- plot3D.cont(positions, annotation) # a gets 0, b gets 5, c gets 10
}
```

---

prepare.graph *Prepare Graph Data for Visualization*

Description

Validate graph input and expose vertex IDs, edge order, weights, and attributes without loading rgl, opening a device, or computing a layout. The prepared object can be inspected and reused by `plot3D.graph()`.

Usage

```
prepare.graph(graph, vertices = NULL, directed = NULL, weight.type = NULL)
```

Arguments

graph	A named list with adj.list and weight.list (neighbors are row indices); a list with edges and vertices; a weighted data frame with from, to, weight columns plus the vertices argument; a numerical square adjacency matrix; a Matrix sparse adjacency matrix; or an igraph object. Matrices use zero for absent edges; use lists/tables to represent zero-weight edges. A graph returned by `prepare.graph()` can be reused directly.
vertices	Explicit vertex IDs or a data frame with a unique id column. Required for edge-table input, including isolated vertices. Adjacency lists default to list names, or character row numbers when unnamed.
directed	Logical directedness; NULL uses stored directedness or FALSE. This release rejects directed rendering, self-loops, and parallel edges. Undirected adjacency lists must be reciprocal with equal weights.
weight.type	"distance", "strength", or "unweighted". Required for weighted layout algorithms but not supplied-coordinate drawing. The fr algorithm requires strengths; kk requires distances. No inversion occurs. Unweighted mode only accepts missing or unit weights.

Details

Vertex order is preserved. Edge-table input retains edge row order; reciprocal undirected adjacency input retains one copy of each edge, from the lower vertex index. Matrix inputs follow that adjacency convention. `edge.col` and `edge.width` follow this prepared edge order and are never inferred automatically from weights. Isolates remain in the vertex table. Prepared objects are revalidated on reuse; editing them does not bypass validation. Directed data can be prepared but cannot currently be rendered. Missing weights require explicit `weight.type = "unweighted"`.

Value

An `ivue_graph` list with `vertices` (a data frame with canonical character `id` plus supplied attributes), `edges` (a data frame with integer `from` and `to` row indices into vertices, numerical `weight`, and supplied attributes), `directed`, and `weight.type`.

Examples

```
edges <- data.frame(from = "a", to = "b", weight = 2)
graph <- prepare.graph(edges, vertices = c("a", "b", "isolate"),
                       weight.type = "strength")
graph$vertices
graph$edges
widths <- 1 + graph$edges$weight
X <- rbind(b = c(1, 0, 0), isolate = c(0, 1, 0), a = c(0, 0, 0))
values <- c(isolate = 3, a = 1, b = 2)
if (nzchar(system.file(package = "rgl"))) {
  w <- plot3D.graph(graph, X = X, values = values, edge.width = widths)
```

```
}
```

`print.ivue_graph` *Inspect Prepared Graphs and Color Scales*

Description

Compact console summaries retain ordinary list access through `$`. Printing does not load a graphics backend, evaluate a custom color function, or change the object. Use `x$vertices`, `x$edges`, or `x$levels` for the full data.

Usage

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

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

Arguments

`x`	A prepared graph or color scale.
`...`	Reserved for compatibility with the print generic.

Value

The original object, invisibly.

See Also

`prepare.graph()`, `color.scale.cont()`, `color.scale.groups()`

`write.animation.gif` *Export Recorded Frames to GIF*

Description

Render the retained frames of an `animate.frames()` widget to an animated GIF. Export uses a separate orthographic raster renderer and requires the optional magick package; it does not launch a browser or native 3D window.

Usage

```
write.animation.gif(
  animation,
  file,
  fps = NULL,
  width = 600L,
  height = 600L,
  final.hold = 2,
  loop = TRUE,
  labels = TRUE,
  overwrite = FALSE,
  annotations = FALSE
)
```

Arguments

animation	A widget returned by animate.frames().
file	Destination ending in .gif. Its parent directory must exist.
fps	Frames per second, from 0.1 to 100; NULL uses the widget's initial speed.
width, height	GIF dimensions in pixels.
final.hold	Additional seconds to hold the last frame, from zero to 600.
loop	Repeat the GIF indefinitely; FALSE plays once.
labels	Draw the retained frame labels above the image.
overwrite	Allow replacing an existing destination.
annotations	Include the animation's mapping legend and plain-text caption. FALSE preserves the unannotated layout. TRUE reserves space beside and below the scene within width and height; enlarge these dimensions if the text does not fit. No annotation is drawn over observations.

Details

GIF export uses the widget's retained coordinates, visibility masks, colors, edge widths, and initial camera orientation. Camera rotations or speed changes made later in the browser are not returned to R. Download view settings and supply their camera when constructing a new animation to reuse its orientation and zoom. With annotations = TRUE, a raster legend and caption are drawn from the retained mapping and caption, including category counts and missing values. Arbitrary HTML is not rasterized. Create a widget with an explicit camera to export that view. Perspective cameras (fov greater than zero) are rejected; use camera.zup(fov = 0).

The raster renderer projects points and straight edges orthographically, with fixed bounds and equal coordinate scales across all frames. Zoom has the rgl convention: smaller values enlarge the scene. The observer position and bounds determine its orthographic scale. Annotations reduce the available scene area; compare exports using the same dimensions and annotation layout. Text wraps at a fixed readable size; layouts that cannot fit are rejected. Edges are painted before points, ordered within each group from back to front. This is a diagram renderer, not a pixel-identical WebGL screenshot or a depth-buffered rendering of intersecting 3D geometry. Point sizes can differ slightly

between browser and raster output. No alignment, recentering of individual frames, or interpolation is performed.

GIF delays are rounded to centiseconds, with a minimum of one centisecond. The additional final hold is applied once per loop. Export works from an R widget object, not from a saved HTML file. Temporary images and graphics devices are cleaned up on success and failure.

Value

The normalized output path, invisibly.

See Also

`animate.frames()`

Examples

```
if (nzchar(system.file(package = "rgl")) &&
    requireNamespace("magick", quietly = TRUE)) {
  X <- rbind(c(0, 0), c(1, 0), c(0, 1))
  w <- animate.frames(list(X, X * 1.5), fps = 2)
  path <- tempfile(fileext = ".gif")
  write.animation.gif(w, path, width = 240, height = 240)
  unlink(path)
}
```

Index

∗ **package**
    ivue-package, 2

animate.frames, 4
animate.frames(), *23–25*

camera.zup, 6
camera.zup(), *5*, *9*, *10*, *20*
color.scale.cont, 7
color.scale.cont(), *21*, *23*
color.scale.groups *(*color.scale.cont*)*, 7
color.scale.groups(), *23*

ivue *(*ivue-package*)*, 2
ivue-package, 2, *9*, *21*

layer3D.axes, 9
layer3D.axes(), *7*, *14*, *15*
layer3D.callback, 11
layer3D.edges, 12
layer3D.edges(), *10*, *14*, *17*
layer3D.labels *(*layer3D.edges*)*, 12
layer3D.mesh, 13
layer3D.mesh(), *14*, *15*
layer3D.path *(*layer3D.edges*)*, 12
layer3D.surface, 14
layer3D.surface(), *14*

map.colors *(*color.scale.cont*)*, 7
map.colors(), *5*, *6*, *21*

plot3D.cont *(*plot3D.plain*)*, 18
plot3D.cont(), *2*, *9*, *10*, *14*, *15*
plot3D.graph, 16
plot3D.graph(), *2*, *20*, *21*
plot3D.groups *(*plot3D.plain*)*, 18
plot3D.groups(), *2*, *9*, *17*
plot3D.plain, 18
plot3D.plain(), *2*, *5*, *7*, *17*
prepare.graph, 21
prepare.graph(), *2*, *16*, *17*, *22*, *23*
print.ivue_color_scale
        *(*print.ivue_graph*)*, 23
print.ivue_graph, 23

write.animation.gif, 23
write.animation.gif(), *5*, *6*