Package: rayplot (via r-universe)

November 23, 2026

Type Package

Title Toy Interactive Plotting with raylib

Version 0.0.1

Description A toy package that opens a raylib window from R and draws an interactive scatter plot with mouse pan and scroll zoom. The render loop is driven cooperatively from R's event loop via 'later', so the R console stays usable while the window is open. Not production ready; a playground for wiring raylib into R.

License GPL-3

Encoding UTF-8

Imports later, ggplot2, grDevices, stats, jsonlite, utils

Suggests shiny, htmltools

SystemRequirements A C toolchain and OpenGL + X11 development headers on Linux (e.g. libgl1-mesa-dev, xorg-dev). raylib itself is vendored in src/raylib (raylib 5.5, zlib licence) and built with the package; no system raylib install is needed.

NeedsCompilation yes

Config/pak/sysreqs libgl1-mesa-dev libx11-dev zlib1g-dev

Repository https://konrad1991.r-universe.dev

Date/Publication 2026-10-26 15:19:08 UTC

RemoteUrl https://github.com/Konrad1991/rayplot

RemoteRef HEAD

RemoteSha ad0b320801b2438eeffeef0ecfe1de5093e4c31f

Contents

rayplot 2
rayplot_web 3
rayplot3D_scatter 5
rayplotCanvas 6

Index **8**

`rayplot` *Interactive raylib plot from a ggplot object*

Description

Open a non-blocking **raylib** window that renders a plot with mouse pan, scroll-wheel zoom and hover tooltips. The render loop is driven from R's event loop via **later**, so the console stays usable while the window is open.

Usage

```
rayplot(p, ..., width = 800L, height = 600L, title = "rayplot", fps = 60)

rayplot_close(handle = NULL)
```

Arguments

`p`	Either a `ggplot` object, or the first of one or more layer specifications (see 'Details').
`...`	Further layer specifications, used only when `p` is itself a layer specification rather than a `ggplot` object.
`width, height`	Window size in pixels. Positive integers.
`title`	Window title.
`fps`	Target frames per second for the redraw loop.
`handle`	The external-pointer handle returned by `rayplot()`. When `NULL`, `rayplot_close()` closes whichever window is currently open.

Details

ggplot input (the common case). When `p` inherits from `"ggplot"`, `rayplot()` calls `ggplot_build` and translates each built layer into a typed specification that the C renderer consumes. Aesthetics are resolved by ggplot first, so scales, positions (`position_dodge`, `position_jitterdodge`, ...) and computed statistics all apply. Each layer carries its own colour palette, taken from that layer's `fill` or `colour` column.

Supported geoms are `geom_point` and `geom_boxplot`, in cartesian coordinates with a single panel. Any other geom, coord or facet raises an error.

3D. If a single `geom_point` layer maps a `z` aesthetic (`aes(x, y, z = )`), the whole plot opens in the orbit-camera 3D renderer instead (see `rayplot3D_scatter`). The 3D view ignores colour and draws every point the same; point size comes from the `size` aesthetic.

Direct layer input. `p` (and `...`) may instead be `lists` of the form `list(type = "scatter", x = , y = , ...)` or `list(type = "box", ...)`. This bypasses ggplot entirely and is used mainly for testing the renderer.

Interaction. Drag to pan, scroll to zoom, hover a point or box for a tooltip, press `Esc` or close the window to quit. Everything is in data coordinates.

One window at a time. raylib has a single global GL context, so only one rayplot window (2D or 3D) can be open. Calling `rayplot()` or `rayplot3D_scatter` again closes the current window first.

Value

Invisibly, an external-pointer `handle` identifying the window. Pass it to `rayplot_close()` to close that window; `rayplot_close()` with no argument closes whichever window is open.

See Also

`ggplot_build`.

Examples

```
## Not run:
library(ggplot2)

## scatter
rayplot(ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) + geom_point())

## dodged boxplot + points, different palette per layer
p <- ggplot(CO2, aes(
    x = factor(conc), y = uptake,
    group = interaction(conc, Treatment, Type)
  )) +
  geom_boxplot(aes(fill = interaction(Treatment, Type)),
               position = position_dodge(width = 0.8)) +
  geom_point(aes(colour = interaction(Treatment, Type)),
             position = position_dodge(width = 0.8)) +
  scale_fill_brewer(palette = "Set2") +
  scale_colour_grey(start = 0.1, end = 0.5) +
  theme(legend.position = "none")

h <- rayplot(p, title = "CO2 uptake")

## keep working in the console, then:
rayplot_close(h)

## End(Not run)
```

`rayplot_web` *Export a plot as a WebAssembly page*

Description

Render a plot in the browser instead of a native window. R builds the same per-layer specification that `rayplot` feeds the C renderer, serialises it to a compact binary blob, and writes a self-contained HTML page that draws it with an Emscripten build of the raylib render core. Pan, zoom and hover tooltips work client-side; R is not involved once the page is open.

Usage

```
rayplot_web(p, file, ..., width = 800L, height = 600L, title = "rayplot",
            assets = system.file("web", package = "rayplot"))

rayplot_spec(p, ...)

rayplot_serialize(layers, width = 800L, height = 600L)
```

Arguments

p	A ggplot object, or layer specifications, exactly as for rayplot.
file	Path of the .html page to write.
...	Further layer specifications when p is one.
width, height	Canvas size in pixels.
title	Page <title>.
assets	Directory containing rayplot.js, rayplot.wasm and template.html. The default is the installed inst/web. The WebAssembly module is *not* built by R CMD INSTALL; build it once with inst/web/build.sh (needs the Emscripten SDK).
layers	A list of layer specifications, e.g. from rayplot_spec().

Details

rayplot_spec() returns the list of per-layer specifications. rayplot_serialize() turns that list into the little-endian binary blob consumed by inst/web/rayplot_web.c (format documented at the top of that file). rayplot_web() base64-encodes the blob into the HTML page and, if rayplot.js / rayplot.wasm are present in assets, copies them next to file.

Only geom_point and geom_boxplot layers are supported, as for rayplot.

Value

rayplot_web() returns the output path, invisibly. rayplot_spec() a list of layer specifications. rayplot_serialize() a raw vector.

See Also

rayplot

Examples

```
## Not run:
library(ggplot2)
p <- ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) + geom_point()

rayplot_web(p, "mtcars.html")
# then serve the folder and open mtcars.html, e.g.:
#   python3 -m http.server
```

```
## End(Not run)
```

`rayplot3D_scatter` *Interactive 3D scatter plot (experimental)*

Description

Open a non-blocking **raylib** window showing a 3D point cloud with an orbit camera. This entry point does not use **ggplot2**; supply the coordinate vectors directly. The render loop is driven from R's event loop via **later**.

Usage

```
rayplot3D_scatter(x, y, z, width = 800L, height = 600L,
                  title = "rayplot3D", point_radius = 0.1, fps = 60)

rayplot3D_close(handle = NULL)
```

Arguments

`x`, `y`, `z`	Numeric vectors of equal length giving the point coordinates.
`width`, `height`	Window size in pixels.
`title`	Window title.
`point_radius`	Sphere radius for each point, in data units.
`fps`	Target frames per second for the redraw loop.
`handle`	The external-pointer handle returned by `rayplot3D_scatter()`. When `NULL`, `rayplot3D_close()` closes whichever window is currently open.

Details

Interaction: **left-drag** orbits the camera, **right-drag** pans, and the **scroll wheel** zooms. Press `Esc` or close the window to quit.

This is experimental: the API is minimal (a single scatter geometry, no axes labels or legends) and may change.

raylib has a single global GL context, so only one rayplot window can be open; calling this or `rayplot` again closes the current window first.

Value

Invisibly, an external-pointer `handle` identifying the window. Pass it to `rayplot3D_close()` to close that window.

See Also

`rayplot` for 2D plots from `ggplot` objects.

Examples

```
## Not run:
h <- rayplot3D_scatter(
  CO2$conc, CO2$uptake, as.numeric(CO2$Treatment),
  point_radius = 0.1
)

rayplot3D_close(h)

## End(Not run)
```

`rayplotCanvas` *Embed a rayplot renderer in a Shiny app*

Description

Draw plots inside a Shiny app with the WebAssembly build of the raylib render core, updating them over Shiny's websocket without reloading the page or using an `iframe`. `rayplotCanvas()` places the drawing surface in the UI; `rayplot_send()` pushes a plot to it from the server.

Usage

```
rayplotCanvas(id = "rayplot", width = 800L, height = 600L)

rayplot_send(p, id = "rayplot", ..., width = 800L, height = 600L,
             session = NULL)
```

Arguments

`id`	Element id shared by the canvas and the `rayplot_send()` calls that target it.
`width, height`	Drawing-buffer size in pixels. The value passed to `rayplotCanvas()` and to `rayplot_send()` must match.
`p`	A `ggplot` object (or layer specifications), as for `rayplot`.
`...`	Further layer specifications when `p` is one.
`session`	The Shiny session; defaults to the current reactive domain.

Details

The WebAssembly module (`inst/web/rayplot.js` + `rayplot.wasm`) is *not* built by `R CMD INSTALL`. Build it once with `inst/web/build.sh` (needs the Emscripten SDK); `rayplotCanvas()` serves it via an `htmlDependency`.

The first `rayplot_send()` for a canvas creates the renderer; later calls swap in the new data (each resets pan/zoom to fit). Only one canvas per page is supported (raylib uses a single GL context). `shiny` and `htmltools` are needed only for these helpers.

Only `geom_point` and `geom_boxplot` layers are supported.

Value

`rayplotCanvas()` a `tagList` for the UI. `rayplot_send()` the `id`, invisibly.

See Also

`rayplot_web` for a standalone HTML page; `rayplot` for the desktop window.

Examples

```
## Not run:
library(shiny)
library(ggplot2)

ui <- fluidPage(
  selectInput("pal", "Palette", c("Set2", "Dark2", "Paired")),
  rayplotCanvas("plot", 900, 620)
)

server <- function(input, output, session) {
  observe({
    p <- ggplot(CO2, aes(factor(conc), uptake,
                         group = interaction(conc, Treatment, Type))) +
      geom_boxplot(aes(fill = interaction(Treatment, Type)),
                   position = position_dodge(width = 0.8)) +
      geom_point(aes(colour = interaction(Treatment, Type)),
                 position = position_dodge(width = 0.8)) +
      scale_fill_brewer(palette = input$pal) +
      theme(legend.position = "none")
    rayplot_send(p, "plot", width = 900L, height = 620L)
  })
}

shinyApp(ui, server)

## End(Not run)
```

Index

ggplot_build, *2, 3*

htmlDependency, *6*

rayplot, 2, *3–7*
rayplot3D_close *(*rayplot3D_scatter*)*, 5
rayplot3D_scatter, *2, 3*, 5
rayplot_close *(*rayplot*)*, 2
rayplot_send *(*rayplotCanvas*)*, 6
rayplot_serialize *(*rayplot_web*)*, 3
rayplot_spec *(*rayplot_web*)*, 3
rayplot_web, 3, *7*
rayplotCanvas, 6

tagList, *7*