The Journal of Open Source Software

Makie.jl: Flexible high-performance data visualization for Julia

Simon Danisch[1] **and Julius Krumbiegel**[2]

1 Beacon Biosignals **2** Department of Systems Neuroscience, University Medical Center Hamburg-Eppendorf

DOI: 10.21105/joss.03349

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @fverdugo
- @gaelforget

Submitted: 25 April 2021
Published: 18 June 2021

Summary

`Makie.jl` is a cross-platform plotting ecosystem for the Julia programming language (Bezanson et al., 2012) , which enables researchers to create high-performance, GPU-powered, interactive visualizations, as well as publication-quality vector graphics with one unified interface. The infrastructure based on `Observables.jl` allows users to express how a visualization depends on multiple parameters and data sources, which can then be updated live, either programmatically, or through sliders, buttons and other GUI elements. A sophisticated layout system makes it easy to assemble complex figures. It is designed to avoid common difficulties when aligning nested subplots of different sizes, or placing colorbars or legends freely without spacing issues. `Makie.jl` leverages the Julia type system to automatically convert many kinds of input arguments which results in a very flexible API that reduces the need to manually prepare data. Finally, users can extend every step of this pipeline for their custom types through Julia's powerful multiple dispatch mechanism, making Makie a highly productive and generic visualization system.

Statement of need

Good data visualization is crucial for the work of every scientist around the world. To effectively understand and communicate results, different disciplines ranging from publication-quality static vector graphics, through animated movies, to interactive data exploration tools, require flexible and powerful plotting software. Most tools available today lack one or more of the following attributes: High performance for the responsive rendering of large datasets, interactive visualizations with flexible dynamic languages, the ability to extend the plotting pipeline to handle user-defined data structures and types, implementations of both 2D and 3D rendering, and the power to construct complex figures without having to tweak positions after the fact in image editing software. Therefore, researchers have to switch between tools which means they have to spend more time to learn unfamiliar syntax and redo work if one software turns out to lack critical abilities for the task at hand.

`Makie.jl` is a new plotting package which is built from the ground up to leverage the power of `Julia`, a relatively young programming language which excels at technical computing and has seen steady growth of its user base since reaching the 1.0 milestone in 2018. Julia users have historically often used plotting software from other ecosystems, such as `matplotlib` (Hunter, 2007) or `ggplot2` (Wickham, 2011) through interface packages to Python and R like `PyCall.jl` and `RCall.jl`. But these wrapper packages cannot take full advantage of Julia's type system and multiple dispatch paradigm, so they leave both performance and flexibility on the table. `Makie.jl` aims to fill this gap in the Julia ecosystem.

The Journal of Open Source Software

Example

The following figure illustrates a few key features of `Makie.jl`: The map example augments a dataset of the world's airports to 100 million points, a size that many other plotting frameworks would struggle with.

The mandelbrot fractal heatmap on the right is plotted by simply passing an x- and y-range, as well as the mandelbrot function, which is then automatically evaluated for all xy combinations. Julia's multiple dispatch mechanism is, in very simple terms, a way to overload functions with different versions or methods, that are chosen depending on the set of input arguments. This allows Makie to offer very flexible input argument alternatives, which get the user to their visualization faster, by reducing common redundant data preprocessing steps.

The volumetric contour and mesh plots in the bottom row showcase Makie's 3D ability. Complex meshes can be visualized and interacted with effortlessly.

The figure layout is iteratively built by simply specifying object positions like `f[1, 1]` for "first row, first column of the figure," or even nested like `f[1:2, 2][2, 1:2]` for "first two rows, second column, and there in a nested layout second row, first two columns." Users do not have to set up grid structures of the correct sizes a priori, but can simply add objects where they are required and the layouts grow automatically. The figure title is another example of this, it is placed at `f[0, :]` which means "one row above the current first row, across all columns." Adding a zero-th row, simply shifts all other rows downwards, and is a very simple and intuitive way of adding elements to an existing structure. While other packages offer functions like `supertitle`, these are usually limited by their assumptions about the figure layout and can't handle more unusual requirements. In Makie, text can be placed anywhere in a layout and it's trivial to construct complex figures with multiple grids and subheaders this way.

```
using GLMakie, GLMakie.FileIO
using LinearAlgebra: norm
using DelimitedFiles

f = Figure(resolution = (1400, 1000), font = "Helvetica")

a = readdlm(assetpath("airportlocations.csv"))
a_rep = repeat(a, 100_000_000 ÷ size(a, 1), 1) .+ randn.()
scatter(f[1, 1], airports_rep, color = (:black, 0.01), markersize = 0.5,
    strokewidth = 0, axis = (title = "Airports (100 Million points)",
    limits = (-200, 200, -70, 80)))

r = LinRange(-5, 5, 100)
volume = [sin(x) + sin(y) + 0.1z^2 for x = r, y = r, z = r]
ax, c = contour(f[2, 1][1, 1], volume, levels = 8, colormap = :viridis,
    axis = (type = Axis3, viewmode = :stretch, title = "3D contour"))
Colorbar(f[2, 1][1, 2], c, label = "intensity")

function mandelbrot(x, y)
    z = c = x + y*im
    for i in 1:30.0; abs(z) > 2 && return i; z = z^2 + c; end; 0
end
ax2, hm = heatmap(f[1:2, 2][1, 2], -2:0.005:1, -1.1:0.005:1.1, mandelbrot,
    colormap = Reverse(:deep), axis = (title = "Mandelbrot set",))
hidedecorations!(ax2)
Colorbar(f[1:2, 2][1, 1], hm, flipaxis = false,
  label = "Iterations", height = 300)
```

```
Axis3(f[1:2, 2][2, 1:2], aspect = :data, title = "Brain mesh")
brain = load(assetpath("brain.stl"))
color = [-norm(x[1] .- Point(-40, 10, 45)) for x in brain for i in 1:3]
mesh!(brain, color = color, colormap = :thermal)

Label(f[0, :], "Makie.jl Example Figure")

save("paper_example.png", f)
```

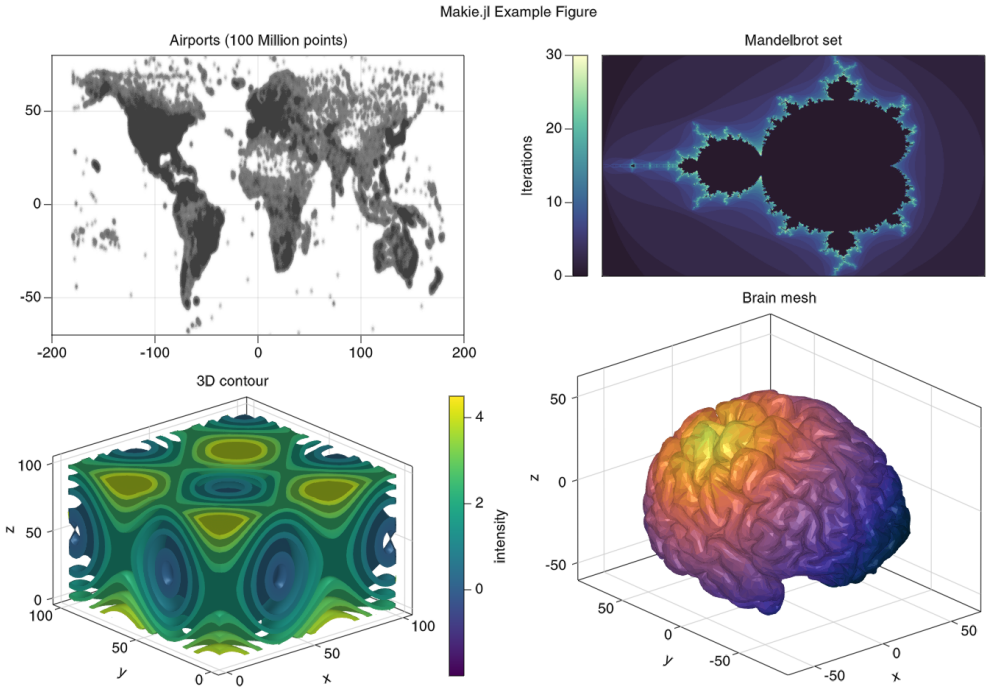

Figure 1: Makie can visualize data with at least 100 million points interactively and render three-dimensional volumes or meshes. Axes and colorbars can be placed freely in nested grids and aligned in a visually pleasing way. The mandelbrot fractal heatmap demonstrates one use of Julia's multiple dispatch, as the mandelbrot function can be directly evaluated on a two-dimensional grid without manually preparing arrays.

Overview

Makie's core package is `AbstractPlotting.jl`, which contains the infrastructure that plots are built out of. It defines primitive plot types such as lines, scatters, heatmaps and text, which can be freely combined to build up higher-level recipes such as density plots, histograms, and contour plots.

There are currently three different backends for AbstractPlotting, which are `GLMakie.jl`, `CairoMakie.jl` and `WGLMakie.jl`. Each backend package has a different set of strengths: GLMakie excels at high-performance plotting with large datasets, CairoMakie creates beautiful vector graphics and WGLMakie can be run in the browser. Users can switch back and forth between backends easily, allowing them to, for example, build up a figure piece by piece in GLMakie, explore the data interactively, and then save a publication-quality version with CairoMakie.

Makie is built around the idea of a reactive workflow using `Observables.jl`. Observables are wrappers for arbitrary objects which can trigger actions when their content is changed. Makie's plotting functions all accept observables as input arguments in addition to normal datatypes like numbers or arrays. If a plotted observable changes, this is reflected immediately in the display. Observables can be chained together, so that changes to many plot objects can flow from a few steering observables. A common example is a single *time* observable, which multiple visualizations depend on simultaneously. The observable approach frees the user from having to update plot objects manually to achieve interactive visualizations, which is usually required in most other plotting software. Makie also offers a range of GUI elements such as sliders, buttons and menus, which seamlessly fit into the observable workflow. For example, a slider over an integer range can be used directly to pick a 2D slice from a 3D dataset, like an fMRI image. This can then be plotted using a heatmap, and will update whenever the slider or 3D data are changed.

A common problem when creating figures for publication is that multiple elements in the figure are overlapping, leaving too much empty space, or simply cannot be placed exactly where the user wants without a lot of trial and error. To alleviate these issues, Makie has a powerful layout system, which makes it easy to build up very flexible and complex figures without a lot of boilerplate setup. Each figure has a top-level grid layout, in which layoutable objects or further nested grid layouts can be placed at arbitrary positions. Grid layouts are built around the assumption that objects like axes usually have an inner important area, which is what other objects should be aligned with, and outer decorations which are visually less important and are treated as part of the gap between rows and columns. This avoids the well-known issue of incorrectly aligned axes between subplots, as they are not aligned along their outer bounding boxes, which depend on unpredictable things like exact tick label lengths, but their visually leading lines, the axis spines. In other plotting software, it is a common issue to place a legend not inside or outside a single axis, but centered below multiple facets or in an entirely different position. In Makie, objects like legends or colorbars are not tied to specific axes and can therefore be placed wherever the user desires. Flexible colorbar or legend placement is a common wish but still surprisingly hard to achieve in many plotting packages, which require manual specification of precise bounding box coordinates for such cases. This often leads to a cycle of readjustments whenever any other figure parameters change. In Makie, the layout algorithm guarantees that no other objects collide or overlap, which can otherwise be the cause for long-winded and non-reproducible sessions in image editing software. Finally, grid layouts can be nested to arbitrary depths, which means that multiple different subfigures can be combined easily by assembling their top-level layouts into a new parent layout.

Its ability to build responsive data visualizations with complex but clear layouts makes `Makie.jl` suitable for a wide range of work with high demands regarding both performance and aesthetics.

Acknowledgements

S.D. has been supported by the German Federal Ministry of Education and Research, grant number 01IS10S27 2020, as well as NSF Grant OAC-1835443. The Makie project is glad to have an enthusiastic community that has helped us to improve a lot over the years. The authors want to thank everybody who has opened issues, reported bugs, contributed ideas or code and has supported us in driving `Makie.jl` forward.

References

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv Preprint arXiv:1209.5145*.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *IEEE Annals of the History of Computing*, *9*(03), 90–95.

Wickham, H. (2011). ggplot2. *Wiley Interdisciplinary Reviews: Computational Statistics*, *3*(2), 180–185.