r2d3 - R domain specific language for D3.js

Filip Stachura, Marek Rogala, Olga Mierzwa-Sulima, Krzysztof Wróbel

2017-02-09

R2D3 Proposal - review version

r2d3 - R domain specific language for D3.js

Note: *We are aware that package with a same name already exists (https://github.com/hadley/r2d3), however what we are proposing is a completely different thing and this particular name will suit it the most. Hadley's library is no longer maintained and is only a D3 wrapper for ggplot2.*

Applicants: Filip Stachura, Marek Rogala, Olga Mierzwa-Sulima

Supporting authors: Krzysztof Wróbel

on behalf of Appsilon Data Science company. Contact email: hello@appsilondatascience.com; contact number: +48 509 125 362

Short Description

Data driven documents in R without predefined wrappers. With this package you can create any visualisation possible in D3 directly with R language.

The problem and motivation

It's natural for a data scientist to look for a better data visualisation methods in R. Although there are many greatly supported R packages that allow to create plots (ggplot2, plotly, rbokeh), when it comes to non-standard visualisations like network graphs, bubble or radial charts you have to look for something created by the community.

If you know JavaScript (JS) you might think about wrapping a D3 component into **htmlwidget** and use it in your RMarkdown report or Shiny app. In fact there are many libraries that were already written to wrap a specific D3 components, for instance: NetworkGraphs(http://christophergandrud.github.io/networkD3/), BubbleChart (https://github.com/jcheng5/bubbles), CoffeeWheel (https://github.com/armish/coffeewheel), etc. .

However only some of them generate truly interactive visualisations, other only allow for image or SVG generation. And when it comes to customization they are very limited, time-consuming and often difficult to do without an appropriate JS knowledge. Since not many data scientists know JS, for the sake of their work, they often resort to manual editing of produced SVG files or static images in Photoshop. In other case they order visualisations from D3 specialists. Either way it's difficult, time and cost consuming to use what's currently available on the market.

Value Proposition

So in order to tackle the problems of customized, interactive and pleasant data visualisations we are proposing a standardized package wrapping D3, which will allow data scientists to recreate any data visualisation possible in D3 directly with R language. With a D3-like syntax anyone would be able to create visualisations like those below.

In a longer term it will lead to much cleaner code in R scripts and Shiny apps, due to not mixing too much JS with R. Also it will be less likely to run into compatibility issues, which you can have when integrating various R packages based on D3. Hopefully it will also be possible to unify current standards for wrapping

Bubble Chart

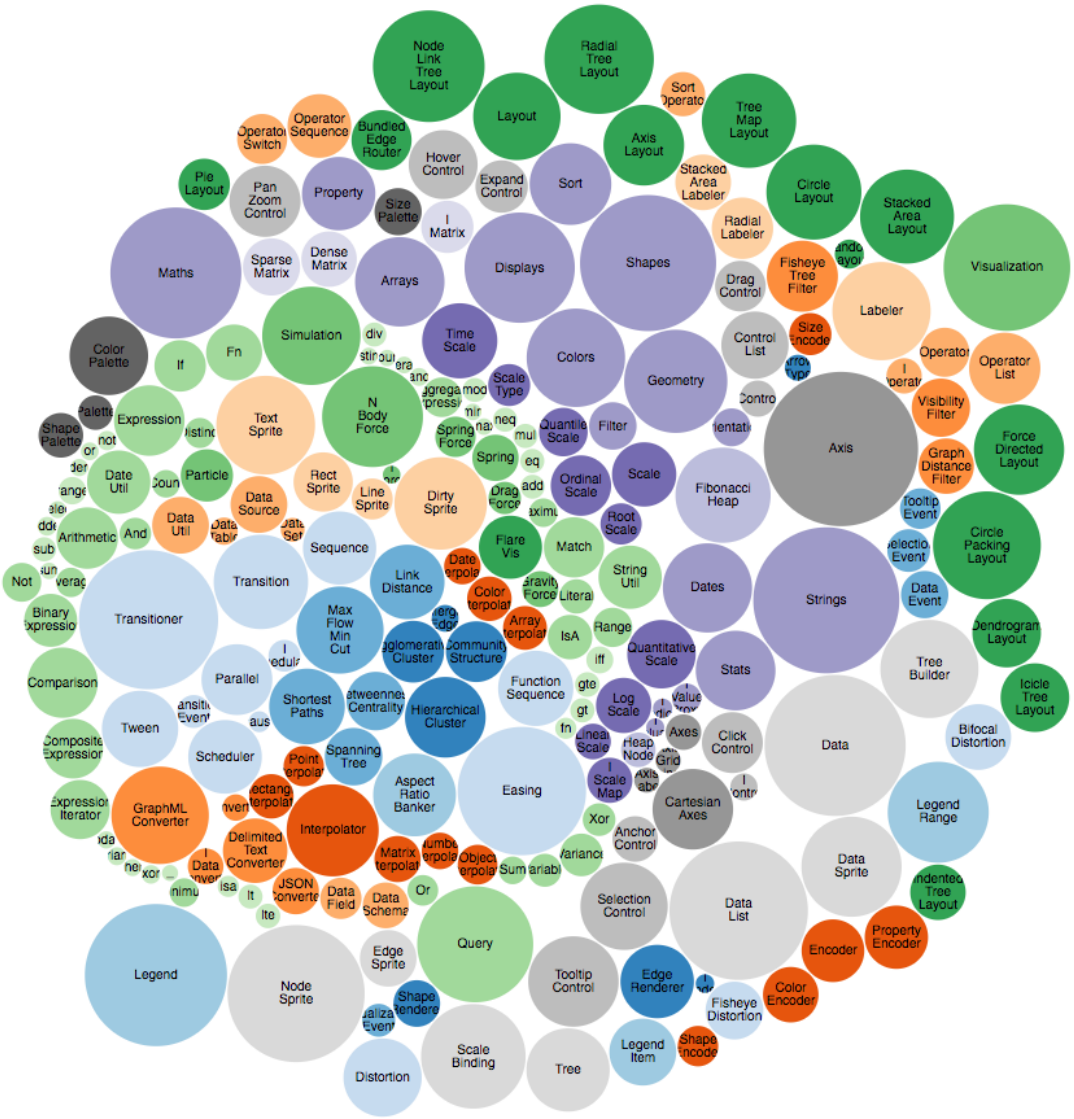

Figure 1:

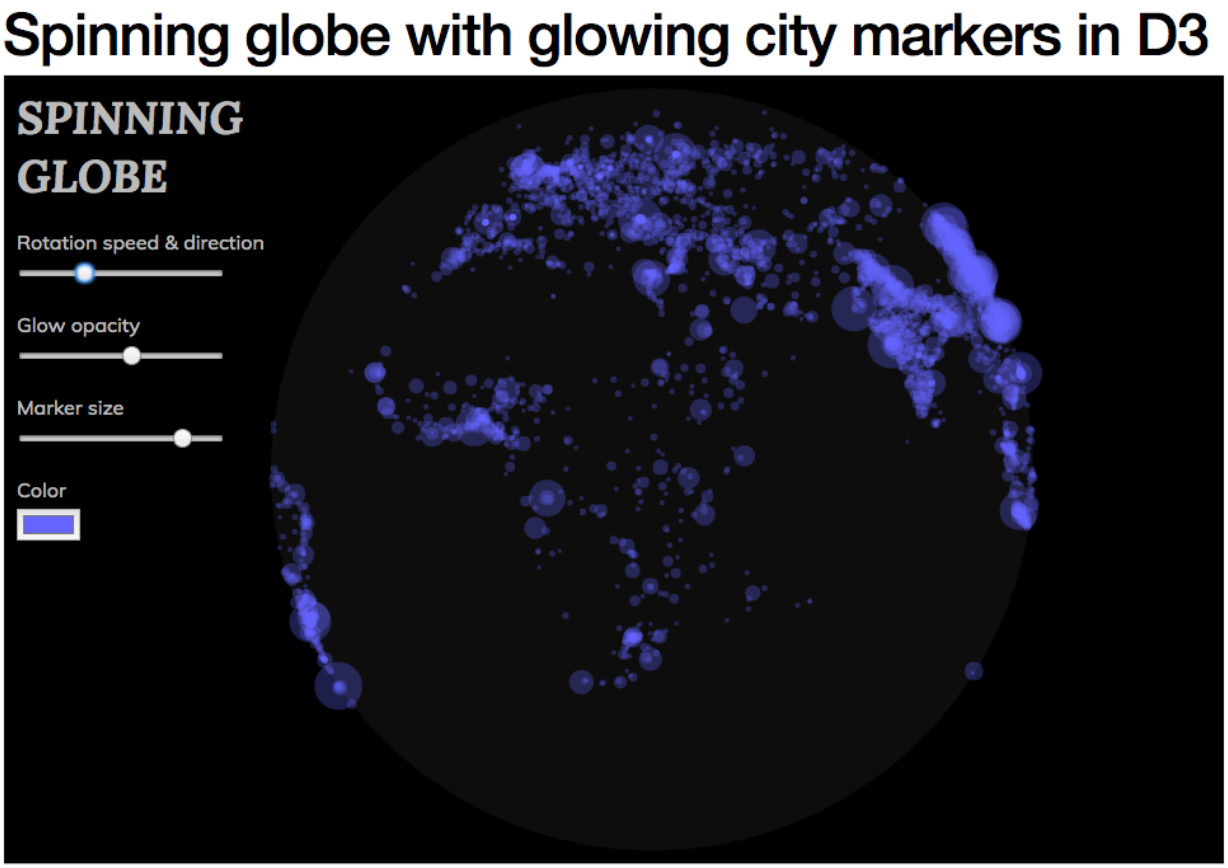
Spinning globe with glowing city markers in D3
SPINNING GLOBE
Rotation speed & direction
Glow opacity
Marker size
Color

Figure 2:

Hierarchical Edge Bundling

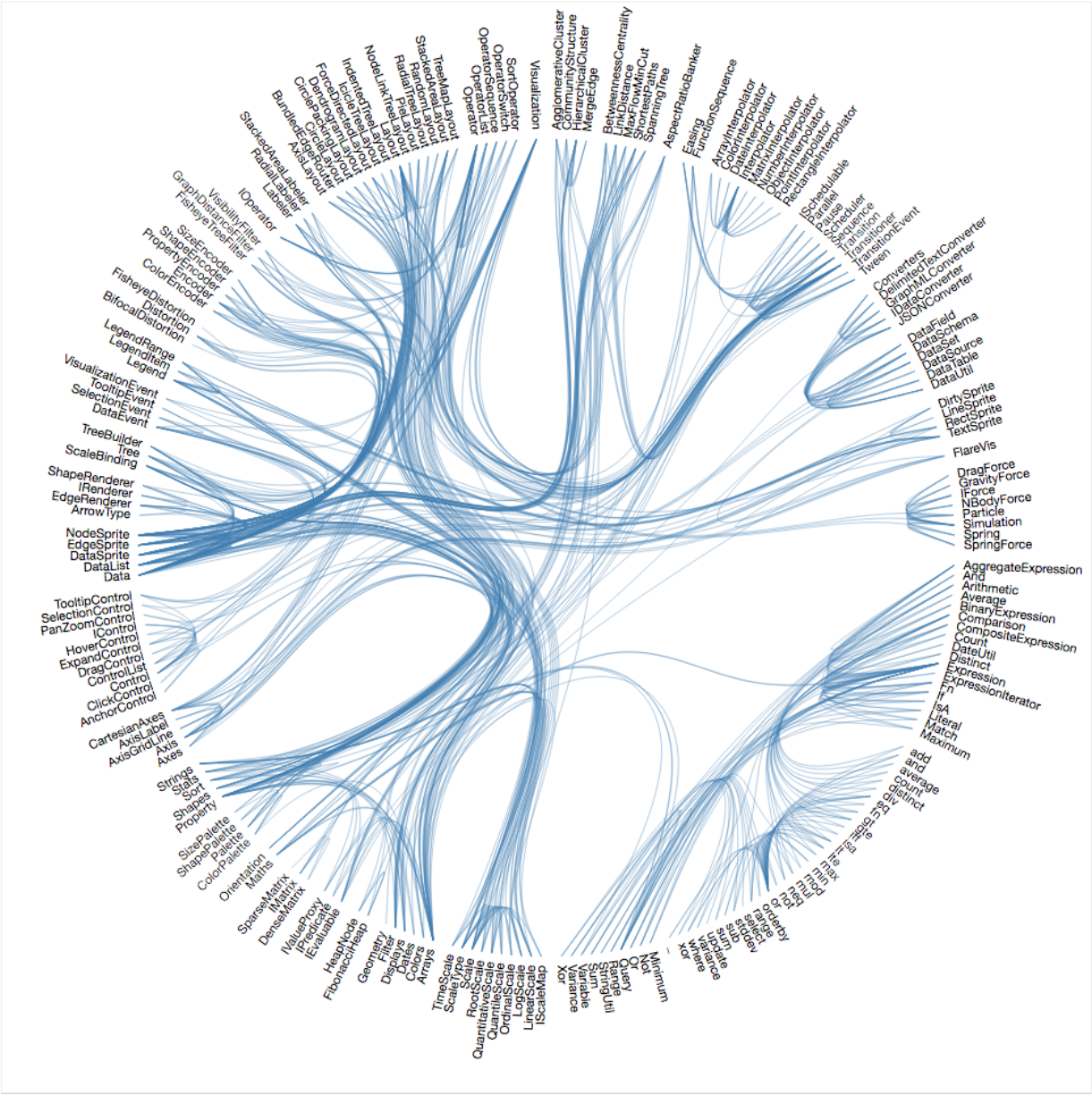

Figure 3:

D3 plugins and components for using them where it's necessary: especialy in RMarkdown reports and Shiny apps.

Last, but not least all the necessary ingredients for nice visualisations in one R package will be unified under a single package with well designed standards. That gives a better chance of strong community support through years of its development.

At Appsilon Data Science we have a broad experience of both JS and R and in our day-to-day job we often had to tackle those problems. Our customers often require non-standard data visualisations and most of them we had to recreate in D3. Since it's also a common problem in R community we have decided to wrap up our experiences and release it as an open source solution. In order to do it with highest possible standards and quality we are applying for R Consortium grant.

We think this project aligns well with the R Consortium's goals:

- it will create a D3 powered data visualisation framework for R and Shiny and will greatly benefit all of the users.
- it will create and promote the best practices for the data visualisation in R language code and Shiny applications.
- encourage and increase R user adoption, especially among those who were unsatisfied with current R visualisations methods.

The Plan: How are you going to solve the problem?

We solve this using **htmlwidget** . Thanks to that users can use D3 in RMarkdown reports and reactive Shiny apps. The structure of the visualisation (plot, graph, etc.) is described in R and later on interpreted in the browser. As the code is interpreted it might be adapted for compatibility with different API's of D3 (there are breaking changes between v3 and v4 [https://github.com/d3/d3/blob/master/CHANGES.md]).

We have implemented a Proof of Concept (POC) that allows for creation of various plots and animations, which is hosted on target Github page: [https://github.com/appsilon/r2d3]. Creating this has already eliminated biggest risks of this project. We managed to prove that general implementation concept is possible. We already know that 90% of D3 plots can be easily implemented purely in R and we believe that we can make it 100%. At the same time we want to make our Domain Specific Language (DSL) usage as convenient as possible. We understand that this is going to play crucial role in adoption of this library.

The biggest challenge that will have to be faced is **debugging.**

Currently debugging in POC is not easy. We want to provide a way of exception reporting and handling. Hopefully we will be able to report an error with an appropriate link to documentation explaining the proper usage of methods (like it's done in React currently).

We do not want to play with d3 internals, so d3.js related problems will be redirected to it's Github Issue page. However, in case of unexpected problems during project's time-frame we will be in touch with a d3 community.

During POC we have created a few design assumptions for future package development:

- In POC r2d3 package we've generated D3 functions automatically. Thanks to that we can get later on interpret abstract structure within JavaScript.
- we've made our functions pipes friendly as it seems like a feasible alternative to Object Oriented (OO) design of D3. Pipes feels more natural in R in our opinion. Still it's open for discussion and $ can be used in general syntax.
- For some complex D3 instructions we've added js specific functions (js_prop, js_fun, save_var, get_var, etc.). We've already got feedback from R users that it seems too complicated. One alternative would be to generate that code automatically, this for sure can work for simple expressions, lambdas etc.
- we want to allow for purrr like lambdas usage.
- seamless usage of D3 plugins in r2d3.

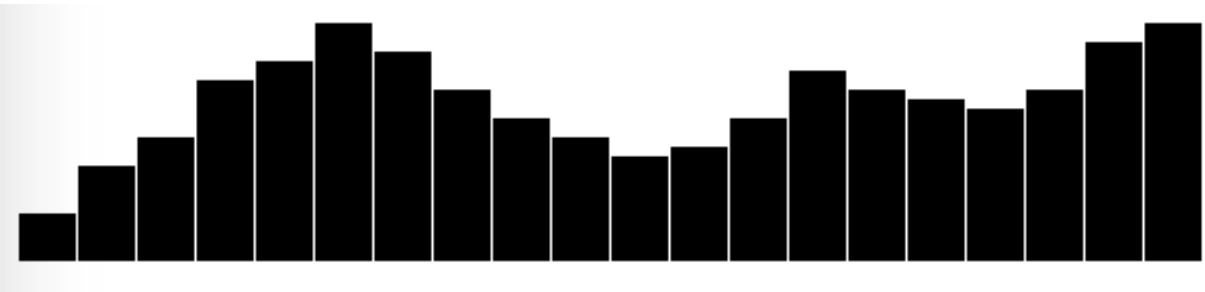

Figure 4:

- debugging via an appropriate exception handling: code position and documentation link.

All those assumptions are going to be discussed with R community in the 1st phase of this project before going into 2nd phase of implementation. This will bring the best standards and practices to this library.

Example 1: Basic bar chart

D3.js

```
d3_bar_chart <- function(dataset) {
 var svg = d3
 .select("#r2d3")
 .append("svg")
 .attr("width", width)
 .attr("height", height);

 svg
 .selectAll("rect")
 .data(dataset)
 .enter()
 .append("rect")
 .attr("x", function(d, i) {
 return i * (width / dataset.length);
 })
 .attr("y", function(d) {
 return height - (d * 4);
 })
 .attr("width", width / dataset.length - barPadding)
 .attr("height", function(d) {
 return d * 4;
 })
        .style("fill", "black");
}
```

Proof of Concept

```
d3_bar_chart <- function(v) {
 dataset <- data.frame(
 v = v,
 height = v * 5,
```

```
 x = 30 * 1:length(v),
 y = 150 - v * 5
 )
 i <- list()
 i$i1 <- d3() %>%
 select("#r2d3") %>%
 append("svg") %>%
 attr("width", 500) %>%
 attr("height", 200) %>%
 save_var("svg")
 i$i2 <- get_var("svg") %>%
 selectAll("rect") %>%
 data(dataset) %>%
 enter() %>%
 append("rect") %>%
 attr("x", r2js::compile(function(i) i$x)) %>%
 attr("y", r2js::compile(function(i) i$y)) %>%
 attr("width", 25) %>%
 attr("height", r2js::compile(function(i) i$height)) %>%
 style("fill", "black")
 execute(i, height = "200px")
```

Our goal: r2d3

```
d3_bar_chart <- function(dataset) {
 r2d3({
 svg <- d3() %>%
 select("#r2d3") %>%
 append("svg") %>%
 attr("width", width) %>%
 attr("height", height)

 svg %>%
 selectAll("rect") %>%
 data(dataset) %>%
 enter() %>%
 append("rect") %>%
 # One can use purrr like lambdas:
 attr("x", ~ .y *(width / length(dataset)) %>%
 attr("y", ~ height - (.x * 4)) %>%
 attr("width", width / length(dataset)) %>%
 # Classic functions works as well:
 attr("height", function (i) i * 4) %>%
 style("fill", "black")
 })
}
```

Example 2: Sphere animation

Using r2d3 creating animations feels natural as it does in D3. No need to animate frames. Examples in this gist is completely created with R and usable: https://cdn.rawgit.com/filipstachura/0ec130d0d8e478882d626ae94c77a117/raw/73b8a21adf9ef63ae9699ea242964db6a7f84be2/sphere-r2d3.html

Current POC code is not perfect, but still we have created fairly complicated animations directly in RStudio watching them within the Viewer. It feels awesome and we want more.

D3.js

```
var canvas = d3.select("canvas"),
 canvasNode = canvas.node(),
 context = canvasNode.getContext("2d");

var width = canvasNode.width,
 height = canvasNode.height,
 distance = 1.4,
 scale = 500;

var graticule = d3.geo.graticule()(),
 sphere = {type: "Sphere"};

d3.timer(function(elapsed) {
 context.clearRect(0, 0, width, height);
 context.save();

 var tilt = (-89 + Math.abs((elapsed / 30) % 358 - 179)) * Math.PI / 180,
 alpha = Math.acos(distance * Math.cos(tilt) * .99);

 var render = d3.geo.pipeline()
 .source(d3.geo.jsonSource)
 .pipe(d3.geo.rotate, 0, 0, elapsed / 3000)
 .pipe(d3.geo.clipCircle, Math.acos(1 / distance) - 1e-6);

 if (alpha) render
 .pipe(d3.geo.rotate, 0, Math.PI + tilt, 0)
 .pipe(d3.geo.clipCircle, Math.PI - alpha)
 .pipe(d3.geo.rotate, 0, -Math.PI - tilt, 0);

 render = render
 .pipe(d3.geo.project, d3.geo.satellite(distance, tilt), .5 / scale)
 .sink(d3.geom.contextSink, context);

 context.translate(width / 2, height / 2);
 context.scale(scale, -scale);
 context.lineWidth = 1 / scale;

 context.beginPath();
 render(graticule);
 context.stroke();

 context.beginPath();
 render(sphere);
 context.lineWidth = 2 / scale;
 context.stroke();

 context.restore();
});
```

Proof of Concept

```
library(r2d3)
library(magrittr)

i <- list()

i$i1 <- d3() %>%
 select(".r2d3") %>%
 append("canvas") %>%
 attr("width", 960) %>%
 attr("height", 500)
i$i2 <- d3() %>%
 select("canvas") %>%
 save_var("canvas")
i$i3 <- get_var("canvas") %>% node() %>% save_var("canvasNode")
i$i4 <- get_var("canvasNode") %>% getContext("2d") %>% save_var("context")
i$i5 <- d3() %>% js_prop("geo") %>% graticule() %>% call() %>% save_var("graticule")

# This could be avoided but we don't have time for that at this point.
# This functionality allows to gradually refactor code from javascript to R.
timer_function <- js_func(
 evaluate("
 env.context.clearRect(0, 0, 960, 500);
 env.context.save();

 var tilt = (-89 + Math.abs((env['.x'] / 30) % 358 - 179)) * Math.PI / 180,
 alpha = Math.acos(1.4 * Math.cos(tilt) * .99);

 env.context.beginPath();

 env.context.translate(960 / 2, 500 / 2);
 env.context.scale(500, -500);
 env.context.lineWidth = 1 / 500;

 var render = d3.geo.pipeline()
 .source(d3.geo.jsonSource)
 .pipe(d3.geo.rotate, 0, 0, env['.x'] / 3000)
 .pipe(d3.geo.clipCircle, Math.acos(1 / 1.4) - 1e-6)

 if (alpha) render
 .pipe(d3.geo.rotate, 0, Math.PI + tilt, 0)
 .pipe(d3.geo.clipCircle, Math.PI - alpha)
 .pipe(d3.geo.rotate, 0, -Math.PI - tilt, 0);

 render = render
 .pipe(d3.geo.project, d3.geo.satellite(1.4, tilt), .5 / 500)
 .sink(d3.geom.contextSink, env.context)(env.graticule);

 env.context.stroke();
 env.context.restore();")
 )

 i$i6 <- d3() %>% timer(timer_function)
```

```
 execute(i)
```

Our goal: r2d3

```
r2d3({
 canvas <- d3() %>% select("canvas")
 canvasNode <- canvas %>% node()
 context <- canvasNode %>% getContext("2d")
 width <- canvasNode$width
 height <- canvasNode$height
 distance <- 1.4
 scale <- 500
 graticule <- d3()$geo %>% graticule()()
 sphere <- list(type = "Sphere")

 d3() %>% timer(function(elapsed) {
 context %>% clearRect(0, 0, width, height)
 context %>% save()
 tilt <- (abs((elapsed / 30) %% 358 - 179) - 89) * pi / 180
 alpha <- acos(distance * cos(tilt)) * 0.99
 render <- d3()$geo %>%
    pipeline() %>%
 source(d3()$geo$jsonSource) %>%
 pipe(d3()$geo$rotate, 0, 0, elapsed / 3000) %>%
 pipe(d3()$geo$clipCircle, acos(1 / distance) - 1e-6)

 ...
 })
})
```

Example 3: Easily extending D3 with plugins

One of crucial requirements we have in mind is to allow for easy extending of visualizations through D3 plugins. As an example we choose awesome Textures.js library. Below you can see how we've implemented interactive scatter plot showing mtcars data using r2d3 extended with textures.

Important requirement here is to track right version of libraries dependencies and resolve them correctly.

R source: *below*

Html source: https://gist.github.com/filipstachura/c926abd54ff7225043d5ee7ca354af6e

Proof of Concept

```
library(r2d3)
library(texturer)
library(magrittr)

i <- list()
i$i1 <- d3() %>%
 select(".r2d3") %>%
 append("svg") %>%
```

```
 attr("width", 500) %>%
 attr("height", 500) %>%
 save_var("svg")
i$i2 <- textures() %>%
 lines() %>%
 thicker() %>%
 save_var("dashed")
i$i3 <- textures() %>%
 circles() %>%
 size(5) %>%
 radius(2) %>%
 save_var("dots")
i$i4 <- get_var("svg") %>%
 call(get_var("dashed"))
i$i5 <- get_var("svg") %>%
 call(get_var("dots"))
i$i6 <- get_var("svg") %>%
 selectAll("circle") %>%
 data(mtcars) %>%
 enter() %>%
 append("circle") %>%
 attr("cx", js_func(get_var(".") %>% js_prop("hp"))) %>%
 attr("cy", js_func(get_var(".") %>% js_prop("disp"))) %>%
 attr("r", js_func(get_var(".") %>% js_prop("mpg"))) %>%
 attr('pointer-events', 'all') %>%
 style("fill", url(get_var("dots"))) %>%
 on("mouseover", js_func(d3() %>% select(this()) %>% style("fill", url(get_var("dashed"))))) %>%
 on("mouseout", js_func(d3() %>% select(this()) %>% style("fill", url(get_var("dots")))))
execute(i)
```

Our goal: r2d3

```
library(r2d3)
library(texturer)
library(magrittr)

r2d3({
 svg <- d3() %>%
 select("body") %>%
 append("svg") %>%
 attr("width", 500) %>%
 attr("height", 500)

 dashed <- textures() %>%
 lines() %>%
 thicker()
 dots <- textures() %>%
 circles() %>%
 size(5) %>%
 radius(2)

 svg$call(dashed)
 svg$call(dots)
```

```
 svg %>%
 selectAll("circle") %>%
 data(mtcars) %>%
 enter() %>%
 append("circle") %>%
 attr("cx", ~ .$hp) %>%
 attr("cy", ~ .$disp) %>%
 attr("r", ~ .$mpg) %>%
 attr('pointer-events', 'all') %>%
 style("fill", dots) %>%
 on("mouseover", ~ d3() %>% select(this) %>% style("fill", dashed)) %>%
 on("mouseout", ~ d3() %>% select(this) %>% style("fill", dots))
})
```

The plan

We are going to divide this project in 3 basic phases: **Phase 1 Design and validation - approximately. 1 month**

We are going to start a discussion on Github, when it comes to the most important design aspects mentioned above:

- modulartiy
- syntax: pipes (%>%) vs dollar sign ($)
- using R specific functions in R2D3
- debugging

We are going to release information about our project's design discussion on multiple R-related social channels like:

- FB groups (TODO name two)
- Slack channels
- R Bloggers
- R Consortium
- Hacker news

We will also try to implement 10 examples D3 specific visulations with current POC version of R2D3 to potentially identify other undetected issues.

Phase 2 - Core package implementation - approximately 3-4 months

During this time we are going to finalise the implemention of this package and release it to CRAN:

- translating basic R expressions to JS
- polishing D3 like syntax R-to-JS runtime
- support for other D3 based plugins
- handling exceptions with nice Documentation Urls(e.g. using R specific functions in R2D3)
- create a documentation
- write unity tests and basic examples coverage tets
- CRAN release

Phase 3 - Package maturity tests - approximately 1-2 months

During this phase we are going to find 2-4 beta testing partners, who are willing to use our library commercially. We are going to support them during integration process testing the maturity of our package in the process. Gathered feedback will be later used to improve the package.

To ease learning curve for less experienced R users we are also going to create few working samples, which will be ready-to-paste for usage in theirs projects and publish them in our repository.

How Can The ISC Help? :

We estimate scope of the project to take about 6 months of work. We are going to need some additional time resources and thus we are going to hire data scientist, who will be dedicating his/her efforts solely to this project (a funding of $12400). During 3rd phase to support beta testers phase we will need extra $2000. This funding will allow us to make sure that this open source project have same priority as other commercial projects we work on and have the best possible quality.

At the same time we want to spark the adoption of this package through conferences. If accepted we believe it would make sense to present r2d3 package during EARL conference in San Francisco (US location) and during UseR in Brussels (Europe location). Our estimations are that both of them are going to cost $3000 ($2000 and $1000 respectively).

We also apply for separate, smaller budget for marketing of this package. This would cover the costs of series of blog posts showing potential of using D3 straight from the R. We want to write 6 separate articles with total cost of $800.

In our opinion the biggest impact ISC can have on this work would be to contact us directly with data scientists from R Consortium member companies. Thanks to that we can take 'customer development' approach, gathering requirements from that and going through beta tests directly with them. This can also lead to very successful case studies we can share with others on how to use r2d3 to deliver business results.

Summing it up:

- Development and support : $14,400
- EARL and useR: $3,000
- Marketing: $800
- ISC help: priceless

Total: $18,200 (can be paid in few installments)

Dissemination : How will you ensure that your work is available to the widest number of people?

We are going to open-source this project on Github using MIT license: [https://github.com/appsilon/r2d3]. We will be using **r-hub** for testing compatibility on different operating systems and machines. As stated before, by the end of the 2nd phase package will be released to CRAN.

Besides two mandatory blog posts to R Consortium blog, we are going to use our Appsilon Data Science Blog to report significant progress in this package. During the 3rd phase we want to publish there a few posts about creating nice visualisations with R2D3 (Radial charts, 3D earth visualisations etc.). It's worth mentioning that our blog's RSS feed is also connected to R-Bloggers, which gives it quite high community visibility. Extra adoption boost could give us a hosted post on RStudio and Microsoft's VSTR social channels, which are members of R Consortium. It would be nice to use social media channels of other members as well, if this package sides well with their business or marketing strategies.

We are active on a few R-specific Data Science groups on Slack and FB, so we are going to publish updates there as well. From our past experiences we know that can get a lot of valuable feedback from community there.

Every year we are participating in global R conferences like useR! and EARL, so depending on proposal acceptance date we might be able to give r2d3 presentation during this years editions. Otherwise we are going to give a talk on useR! and EARLs in 2018, both in Europe and USA. Obviously we are going to promote this project in local R events in Poland.