Home

Edit · New Page · Jump to bottom

Roy van Schaijk edited this page on Apr 29, 2019 · 28 revisions

Introduction

Sigma is a JavaScript library to display graphs. It has been designed as an engine that you can customize and use to develop highly interactive Web applications that show graph visualizations. Here is a short list of the most important features:

- **Custom rendering**: You can use the Canvas or WebGL built-in renderers, or even write your own. And the built-in renderers also provide a lot of ways to already customize the rendering.
- **Interactivity oriented**: You can catch when the users clicks or rolls the mouse over a node. You can catch when the user drags the graph or zoom in, and always know the position of the graph relatively to the screen. And much more.
- **Powerful graph model**: Sigma is just a rendering engine, but you might want to do more, like running your own graph algorithms. For that, sigma's graph model is customizable, and you can add your own custom indexes on the data.
- **Extendable**: It is easy to develop plugins or simple snippets to augment sigma's features. Some are already available in the main repository to read some popular graph file formats, or to run complex layout algorithms, for instance.
- **Compatibility**: Sigma runs on all modern browsers that support Canvas, and works faster on browser with WebGL support.

Getting started

To get sigma's code locally, you first need to clone the repository and install development dependencies. Ensure that node.js is already installed on your computer.

```
$ git clone https://github.com/jacomyal/sigma.js.git
$ cd sigma.js
$ npm install
```

For windows users please ensure the directory 'C:\Users\UserName\AppData\Roaming\npm' exists. If not, manually create it.

Sigma provides many code examples to show you what it can do and how to use it. Some of these examples load external data files, and you need to access them through a local server to see the examples working. You can use `npm start` to start a node.js based local HTTP server, included in the development dependencies:

```
$ cd sigma.js
$ npm start
```

Then, you will have access to the examples list at http://localhost:8000/examples.

Overview

Any instance of sigma is basically a **graph model** and a **controller**. The graph model is the part of sigma that helps manipulating the data, and the controller provides some useful methods to interface the rendering process, the data and your application.

Also, it is possible to bind any instance of sigma to one or more **cameras** and **renderers**. The renderers are the components that actually **render the graph**. The cameras work exactly as in video games, they are the components that make possible to move in the graph with your mouse or the zoom in with your mouse wheel, for instance. Each renderer works with one and exactly one camera, but it is possible to bind several renderers to the same camera.

In most cases, you will just need to instantiate sigma with one camera and one controller. And sigma provides simple ways to be initialized for the most common usages. Here is a basic example:

```
<!DOCTYPE html>
<html>
<head>
  <title>Basic sigma.js example</title>
  <style type="text/css">
    body {
      margin: 0;
    }
    #container {
      position: absolute;
      width: 100%;
      height: 100%;
    }
  </style>
</head>
<body>
  <div id="container"></div>
  <script src="./sigma.min.js"></script>
  <script>
    // Let's first initialize sigma:
    var s = new sigma('container');

    // Then, let's add some data to display:
    s.graph.addNode({
      // Main attributes:
      id: 'n0',
      label: 'Hello',
      // Display attributes:
      x: 0,
      y: 0,
      size: 1,
      color: '#f00'
    }).addNode({
      // Main attributes:
      id: 'n1',
      label: 'World !',
      // Display attributes:
      x: 1,
      y: 1,
      size: 1,
      color: '#00f'
    }).addEdge({
      id: 'e0',
      // Reference extremities:
      source: 'n0',
      target: 'n1'
    });

    // Finally, let's ask our sigma instance to refresh:
    s.refresh();
  </script>
</body>
</html>
```

Anytime, it is easy to customize how sigma renders your graph by overriding some settings. The available settings, their default values and what they do is described here.

For instance, here is how to draw the edges with a specific color instead of the color of their source:

```
s.settings({
  edgeColor: 'default',
  defaultEdgeColor: '#999'
});

// Refresh the graph to see the changes:
s.refresh();
```

Angular Installation Instructions

Steps

- npm install sigma -save
- npm install sigma-webpack -save
- Open angular.json file in a text editor

- Look for "architect" -> "build" -> "options" -> "assets". Assets takes and array of settings. Add {"glob": "**/*", "input":"./node_modules/sigma/build", "output": "/assets/lib" }
- Look for "architect" -> "build" -> "options" -> "scripts". Scripts tells angular to include the files when it compiles the application. Add "src/assets/lib/sigma/sigma.conf.js","node_modules/sigma/build/sigma.min.js"
- Save the file
- Create the sigma.conf.js file and add in line to it: mxBasePath = 'assets/lib/sigma';
- Save sigma.conf.js
- Start your angular app with ng serve
- Start Chrome and open the debugger
- Go to the source tab and you should see sigma.min.js in the assets/lib folder

To Test sigma really works, edit one of your component .ts files.

- Add import { sigma } from 'sigma';
- Add var sigmaJs = new sigma();

When you inspect the object in Chrome debugger, it should be a valid sigma object. If you get sigma undefined resource error, it means sigma wasn't installed correctly. Remove sigma and repeat the installation steps again.