1 Input Pajek file description

Program PajekToSvgAnim reads Pajek project (.PAJ) file and generates output .SVG, .SVG.GZ and .HTML files (.SVG.GZ file is compressed .SVG file in GZIP format for faster loading generated objects in the SVG window). Input Pajek file must have the following structure (in the multiple network case with *N* vertices, *R* relations and *M* time points):

***Network** *network name*
***Vertices** *N*
1 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
2 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
…
N "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
***Arcs :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Arcs :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Arcs :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Edges :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Network** *network name in time point* T_1
***Vertices** N_1
1 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
2 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
…
$\mathbf{N_1}$ "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
***Arcs :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Arcs :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Arcs :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…

***Edges :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Edges :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Network** *network name in time point* T_2
***Vertices** N_2
1 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
2 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
…
$\mathbf{N_2}$ "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
***Arcs :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Arcs :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Arcs :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Edges :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Network** *network name in time point* T_M
***Vertices** N_M
1 "*lab*" *coordX koordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
2 "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
…
$\mathbf{N_M}$ "*lab*" *coordX coordY value shape* **x_fact** *factX* **y_fact** *factY* **ic** *color* **[***activ_int***]**
***Arcs :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Arcs :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Arcs :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :1** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
***Edges :2** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…
…
***Edges :R** "*relation name*"
init_vertex term_vertex value **w** *width* **c** *color* **[***activ_int***]**
…

***Partition** "*partition name*"
***Vertices** *N*
1 / 0
1 / 0 } *N*
…
1 / 0

First, the input Pajek file specifies the data of the whole temporal network with all the vertices and lines and their properties. Then, time slices of network for each time point are listed. Time slices contain just the vertices and lines which are active in time points.
The input Pajek file is ended by a special vertex partition, which specifies for each vertex whether its label is visible or not in the network layout (value 1: label is visible, value 0: label is not visible).

Vertex and line properties in the individual networks are standard properties, defined in the program package Pajek. A vertex can have the following properties:

- label (required),
- coordinates x and y (required),
- value (required for the first vertex; if the first vertex is the only vertex with specified value, all subsequent vertices have the same value, too),
- shape (required for the first vertex; see analogous explanation for the vertex value),
- factor in x direction,
- factor in y direction (program considers factor in x direction also for y direction),
- color,
- activity interval (required for the first vertex; see analogous explanation for the vertex value).

A line can have the following properties:

- initial vertex (required),
- terminal vertex (required),
- value (required)
- width (equal to value, if not specified),
- color,
- activity interval (required).

There is no need for preparing the whole input Pajek file, containing the temporal network and all subsequent time slices. It is enough to prepare just the first part that is the data of the temporal network with vertices, lines and their properties. The subsequent time slices are generated automatically using the Pajek command *Net → Transform → Generate in Time → All*, which takes three arguments: the first time point, the last time point and the step.

After generating time slices, we use program package Pajek also to find an optimal dynamic layout of the temporal network. In Pajek, several commands for optimal static layout are implemented, for example the energy command *Layout → Energy → Kamada-Kawai → Free* (Draw screen), which move vertices to locations that minimize the variations in line length. Using this command will produce aesthetic static layout.

To achieve optimal dynamic layout, however, it is necessary to consider additional aesthetic criterion known as "preserving the mental map" or dynamic stability. The term mental map refers to the abstract information a user forms by looking at the layout of a network. In the context of dynamic layout changes to this map should be minimal, in other words dynamic layout should preserve the mental map. Technically, the horizontal and vertical order of vertices should stay the same when changing positions.

Most common approach is to find the optimal static layout just for the whole network and then use that layout for subsequent time slices. Thus, each vertex has just one position in all time points in which it is present.

In Pajek, this can be done for example by applying the energy command *Kamada-Kawai* on the layout of the whole network and then using the command *Next* to produce the layouts for the subsequent time slices. Additionally, we can use the command *Append to Pajek Project file* to save fixed positions or minor movements of vertices (in the current time slice layout) to Pajek project file. In this way we gradually build up dynamically stable layout of the network before using the program PajekToSvgAnim, which actually generates the dynamic layout.

2 PajekToSvgAnim GUI

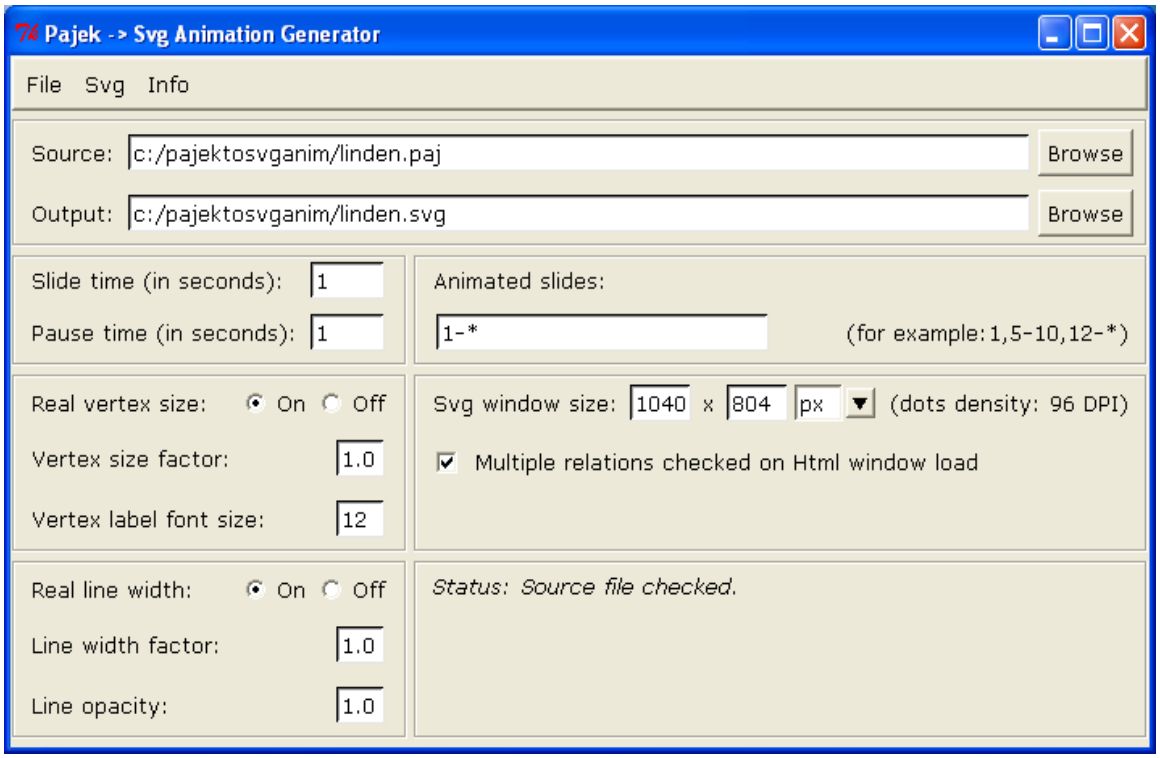
At startup of the program PajekToSvgAnim the Main window opens. The user can select input Pajek file and output .SVG file and run the SVG objects generation command (Picture 1). The program offers some additional choices for adjusting the dynamic layout. These choices are explained in detail in the next chapter.

The menu in the Main window contains the following commands:

- File → Source: for selecting input Pajek file.
- File → Output: for selecting output .SVG file.
- File → Exit: for exiting the program.
- Svg → Generate: for generating the .SVG, .SVG.GZ and .HTML files.
- Info → About: information about current version of the program.

On Picture 1 the Main window is shown. When input Pajek file is selected, the name and the location of the output .SVG file is automatically set to the input Pajek file name with .SVG extension and the same location. In the bottom right of the window the Status line shows whether the network structure in the input file is syntactically correct.

The size of SVG window is also automatically set and can be changed.



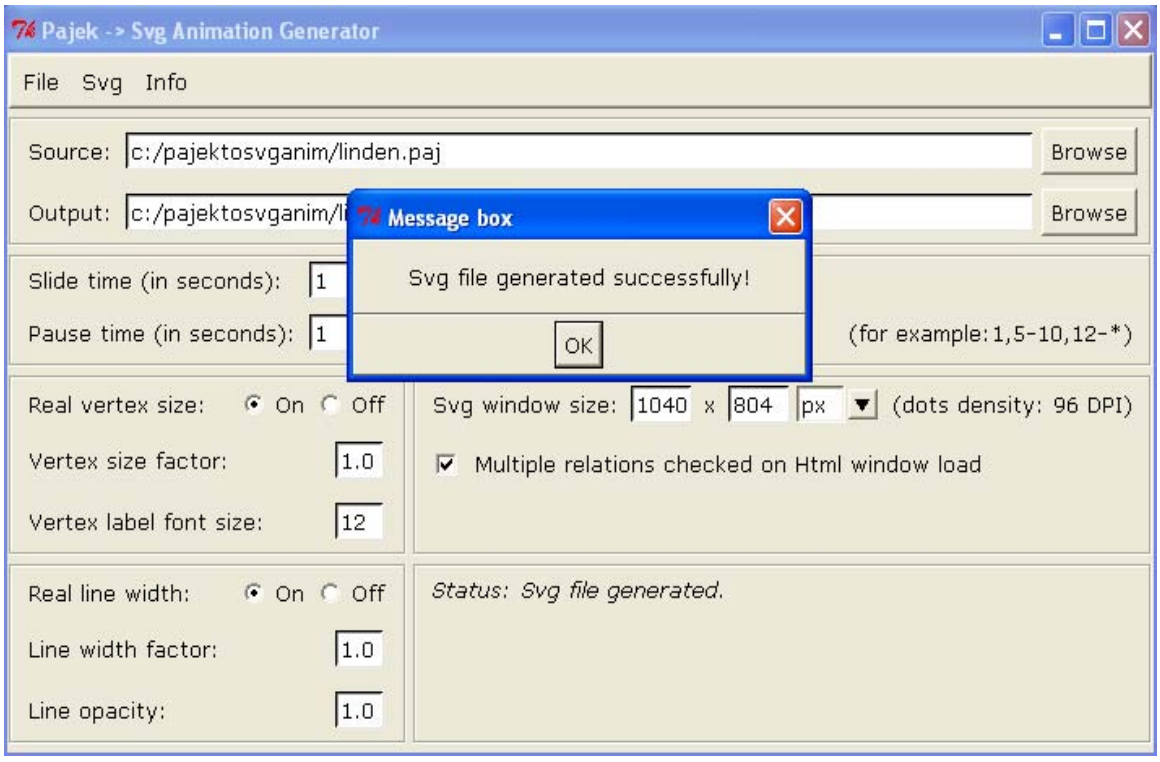
Picture 1: Input Pajek file selection in the Main window of the program PajekToSvgAnim.

When the input Pajek file is selected and checked, the command Svg → Generate from the menu can be used to run the generation of the .SVG, .SVG.GZ and .HTML files. If the output files are generated successfully, the confirmative message appears. (Picture 2). Opening the output .HTML file results in dynamic layout of the network from the input file. HTML window contains SVG window and the checkboxes for showing the vertex labels and the relations in the multiple network case (Picture 3a and Picture 3b). The output .HTML file is linked to compressed .SVG.GZ file for faster loading generated objects in the SVG window.

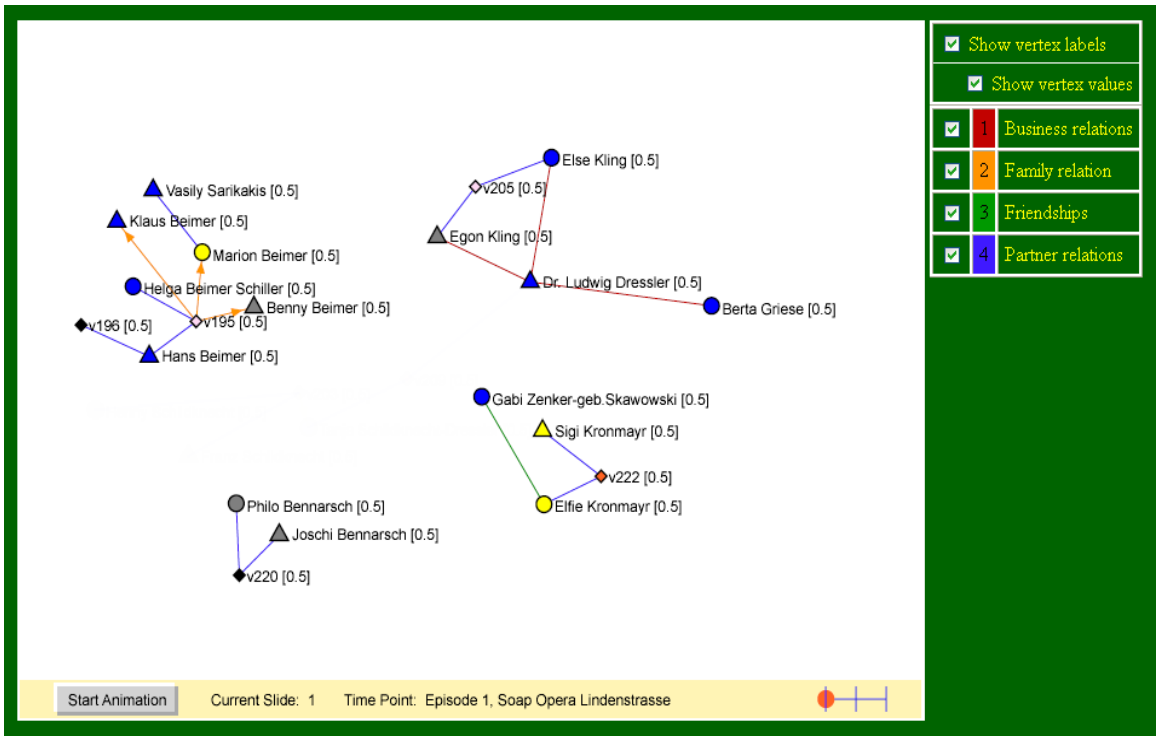
On Picture 3a and Picture 3b the usage of additional functionality and additional feature of the program PajekToSvgAnim are shown. First, the Status line contains the description of actual time point. In order to use this functionality it is necessary to create a tab delimited data file TIMES.TXT in the same folder as input Pajek data file, which should look like this:

1 T*he description of the first time point*
2 T*he description of the second time point*
3 T*he description of the third time point*
….

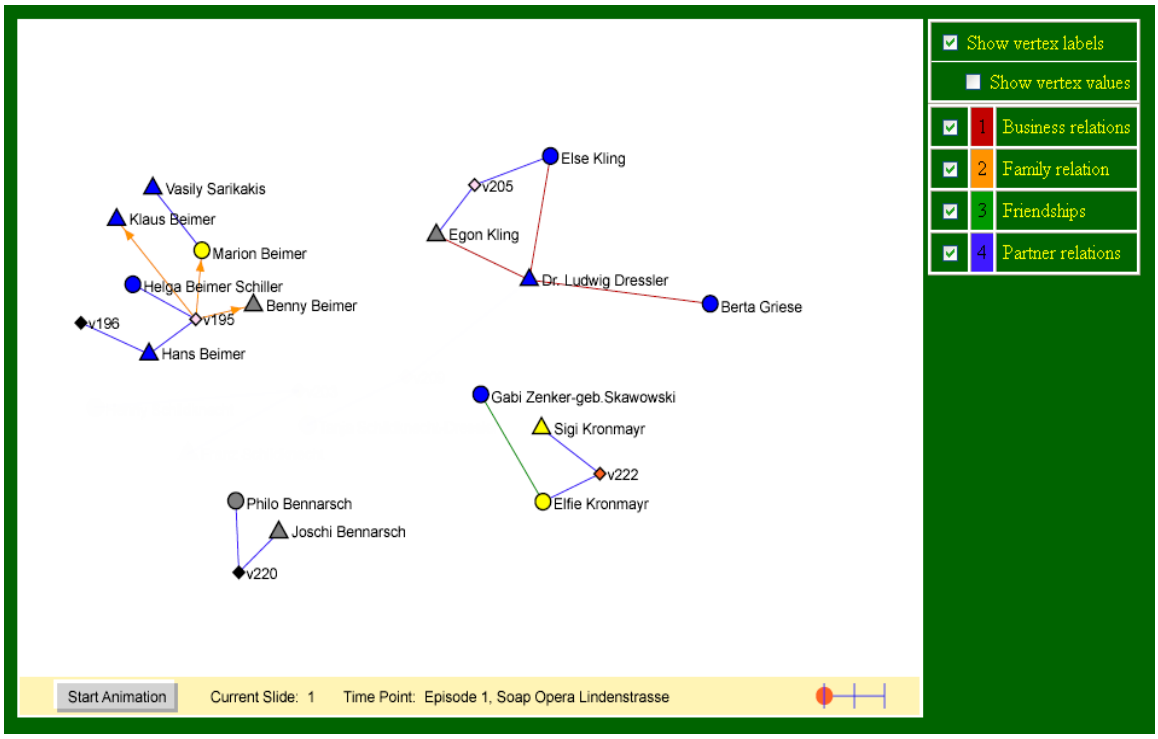
Second, the Status line contains also the progress bar for showing the duration length of dynamic passes between time points.



Picture 2: Confirmative message after the successful generation of the SVG objects in the program PajekToSvgAnim.



Picture 3a: SVG window inside HTML window shows dynamic layout of the selected network (vertices are shown with labels and values).



Picture 3b: Upper HTML window without showing vertex values (only labels are shown).

3 Instructions for using program PajekToSvgAnim

PajekToSvgAnim GUI contains several choices for satisfying user needs and preferences. After selecting the input Pajek file and the name and path of the output .SVG file the user can adjust the following choices:

- The duration length of dynamic layout of network passing from one time point to another (Slide time).
- The duration length of static layout of network in time point before passing to the next time point (Pause time).
- Numerical list of dynamic passes of network between time points to be shown. (Animated slides).

  Example: Animated slides: 1, 5-10, 12-*.

  In this case the layouts of the network go by self in the following order:

  1. Dynamic layout of network passing from first to second time point with corresponding static layouts in the first and second time point.
  2. Static layouts in the 3rd and 4th time point.
  3. Dynamic layouts of network passing from 5th to 10th time point with corresponding static layouts.
  4. Static layout in the 11th time point.
  5. Dynamic layouts of network passing from 12h time point to the last time point with corresponding static layouts.

- Choice between two sizes of the vertex shapes: actual size, defined in the input file, and the unified size for all vertices (Real vertex size On/Off).
- Multiplying factor for controlling vertex size (Vertex size factor).
- Font size of the vertex labels (Vertex label font size).
- The size of the SVG window (Svg window size).
- Selection of relations in the HTML window in the multiple network case (Multiple relations checked on Html window load).
- Choice between two line widths: actual width, defined in the input file, and the unified width for all lines (Real line width On/Off).
- Multiplying factor for controlling line width (Line width factor).
- The opacity of lines (Line opacity).

Let's take a look at some examples, showing the usage of the choices listed above:

1. To examine dynamic layouts or static layouts of a temporal network more precisely, increase the duration length of dynamic layout or the duration length of static layout respectively (1 second by default).

2. To examine a temporal network with many time points, it is easier to focus on some specific dynamic layouts. Such focus can be made by setting the numerical list of dynamic passes of network between time points to be shown. The default list is 1-* which means that all dynamic passes of network should be shown.

3. Choice between the actual vertex size, defined in the input file, and the unified size for all vertices, is known from the program package Pajek. The second option is used when we are not interested in actual vertex sizes or when the vertices are too large to display.

4. In order to make all vertices in the layout larger or smaller, increase or reduce vertex size factor. Default vertex size factor is 1.0.

5. Setting the font size of the vertex labels is also an option known from program package Pajek. Default font size is 12.

6. Dynamic layout of the network can be used as part of a web page. To define the size of the SVG window which shows the dynamic layout choose between the following units: cm, inch, px. Default size of the SVG window is the whole display.

7. In a multiple network case with many relations the dynamic layout might not be clear enough to research the network visually. Use the option 'Multiple relations checked on Html window load' in order to reduce the number of relations to be shown. By default all relations are shown.

8. Choice between the actual line width, defined in the input file, and the unified width for all lines, is known from the program package Pajek. The second option is used when we are not interested in actual line widths or when the lines are too thick to display.

9. In order to make all lines in the layout thicker or thinner, increase or reduce line width factor. Default line width factor is 1.0.

10. If the network contains multiple lines some of the lines might not be visible in the layout, because they are covered by other lines, which connect the same vertices.
    In the program package Pajek this problem can be solved by using the command *Net* → *Transform* → *SortLines* → *LineValues* → *Descending*, which sorts the lines by values in descending order. To solve this problem in the program PajekToSvgAnim, reduce the level of line opacity using the following scale:

    1 - lines are completely opaque,
    0.75 - lines are partly transparent,
    0.5 - lines are medium transparent,
    0.25 - lines are very transparent,
    0 - lines are completely transparent or invisible.

    We can choose any value between 0 and 1, not just a value from the scale. Default value is 1.