Architecture

# Definitions

A **plot** is an algorithm that maps data elements onto a set of graphic elements such as lines, markers or raster images. Location, size and other specific visual properties of the graphic elements could be either bound to the data in some way or have some fixed value.

## Transformations

Each plot requires **a coordinate transform** , a function that transforms data values to a screen point . Coordinate transforms in are always orthogonal, i.e. and axes transform independently of each other: .

Each axis transform is actually a composition of **data transform**  and **plot transform** .

Data transform for a point maps a point on plot plane . Plot plane can be thought of as an abstract sheet of graph paper which size is limited by representation of floating values only.

A domain of a given data transform is a possibly infinite segment on which the data transform function is defined and monotonically increases, and thus it is invertible, too:

In implementation, if a data transform is a partial function, i.e. , we can make it total by assuming that values outside of the must be transformed by a data transform to System.Double.NaN ():

The same rule can be applied to the reverse function.

Each plot occupies rectangular area on a screen and defines a window that maps portion of plot plane to this screen area:

(thus it is possible that ).

**Plot transform** is a transform from plot coordinates to screen coordinates: . Plot transform is

always a composition of Translate and Scale transforms.

Let visible region in the plot pane is

Plot transform maps to :

It can be rewritten:

And shorter:

where

Thus, the plot transformation can be found from visible region and screen region .

If we have a screen region and a plot transform defined by , we can find a visible region in a plot pane that is mapped by to :

If sizes of the screen region change, but scale factors of the pair remains same, sizes of the visible region change proportionally.

## Aspect ratio

Aspect ratio of the plot transformation is:

If , the transformation is uniform by both axes.

It is possible to fix the aspect ratio, i.e. guarantee that

From point of view of visible regions, since width and height of the screen area are fixed, we might need to adjust the visible region to make the actual aspect ratio correct. Let we have a visible region with same left/top postion, and sizes ; so that its aspect ratio is

We build an **actual visible region with fixed aspect ratio** as a visible region with same left/top and sizes :

Note that actual visible region can be only enlarged along one of the axes, so that . Then, if , we build the plot transformation function as a map of to .

From point of view of a plot transformation defined using scale factors and offsets, the problem statement is: let we have a plot transformation defined using , and we need to alter it so that corresponding visible region would be defined as above to guarantee the aspect ratio. The altered plot transform can be found using following formulas:

The latter plot transform:

* Keeps aspect ratio, i.e. .
* Visible region in the plot plane for the transform includes or equal to original visible region and enlarges along one of the axes.

## Plot

The **plot**  is defined as follows:

For a point of data space we can find a point on the screen using the coordinate transformation function which is a composition of data and plot transformations:

## Navigation

**Navigation** is an element which can change visible region in the plot plane. These changes affect the plot transformation function.

### Pan

Pan gesture is describes shift of a point on a screen. The coordinate transform functions changes as follows:

This doesn’t change the aspect ratio of the transform.

### Move To

Move to gesture is moves a center of the visible region so that new center point in the screen is at .

### Zoom

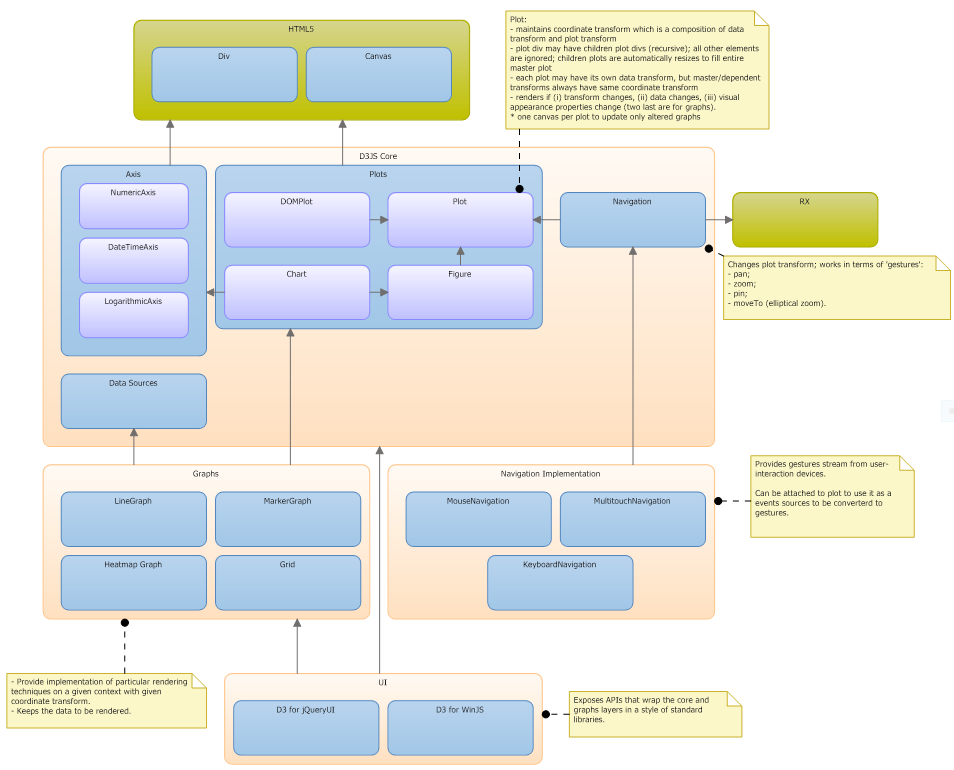
Zoom gesture is describes scaling of a visible region with invariant point which is a center point of the region and width and height are scaled by the factor .

This doesn’t change the aspect ratio of the transform.

### Zoom to a point

Zoom to a point is :

# Components



## Plot

Plot maintains *a coordinate transform* which is a composition of data transform and plot transform. The plot transform determines the visible region in the plot plane.

Plot is *hosted in a* DIV element (only one plot can be hosted in a single DIV element) which is used as a visual output of the plot. It determines the screen region for the plot. Plot can place canvas or svg element into the DIV and render in it, or it can place any DOM element within the DIV and position it in the plot plane.

Plot can have *children plots*; in this case it is *a master plot*. DIV elements of children plots must belong to the DIV element of their master plot, but may have tree-structure with the constraints:

* Only DIVs included to another DIV (but not any other HTML element) can be considered as children.
* Inner DIV element can be marked as a master plot; in this case it is not considered as a child of the outer plot.

Children plots can be grouped into a DIV element; in this case they can be shown/hidden/animated altogether.

Master plot *layouts* its children plots so that they all have same output screen region.

*Figure* plot is a plot with predefined layout which contains children plots and axes with user-defined placement.

All children plots have same plot transform as their master, but may have different data transforms.

Each plot (if needs) should an individual canvas element so that it could be updated separately from others, if only its data changes.

Properties of a plot are listed in the table.

|  |  |
| --- | --- |
| Property | Description |
| Host | A reference (?) to a DIV element which hosts the plot. |
| IsMaster  Master | If the plot is master, its Master points itself. |
| Children |  |
| XDataTransform {get;set;}  YDataTransform {get;set;} | A function which converts a point in the data plane to a point in the plot plane.  { dataToPlot(), domain }  Domain restricts allowed region of the plot plane so that visible region must belong to this set.  If undefined, . |
| CoordinateTransform {get;} | A function which converts a point in the plot plane to a point in the screen plane.  Children plots return the function of their master plot.  { scaleX, scaleY, offsetX, offsetY, plotToScreenX(x), plotToScreenY(y), screenToPlotX(x), screenToPlotY(y) } |
| IsInAnimation | True, if someone from outside of the plot will call at least once the Render method soon. |
| RequestsRender | True, if the plot wants the Render to be called to update its output. |
| IsAutoFitEnabled |  |
| AspectRatio | If defined, makes the SetVisibleRegion keep the aspect ratio. When set to a child plot, propagated to the master plot and to all other children. |
| IsVisible | Determines whether the plot (with its children plots) should be rendered or not. |
|  |  |

*Methods* of a plot are listed in the table.

|  |  |
| --- | --- |
| Method | Description |
| ctor(div)  ctor(div, master) | Initializes for the given DIV element; sets up the CSS styles. Prohibits double initialization.  If master plot is provided, new plot is a child plot.  Builds Children collection.  Recursively checks the inner DIV elements.  Figure can be only a master plot.  Ignores already initialized plots or marked as master plot.  Figure also finds DIV elements marked as axes.  This method may create a canvas element and put it into the host DIV element. |
| UpdateLayout() | When called at a child plot, should be redirected to the master plot.  Master plot re-arranges its content elements. There are different implementation for common plot and figure, which also makes layout of axes and titles.  Update layout is usually called when:   * screen size of the master element is changed (called from outside); * new elements are added or existing are removed from the master plot (called from the master plot); * one of the elements of the master plot changes its size thus affecting the overall layout (e.g. an axis shows longer values during navigation) (initiated by a child).   UpdateLayout invalidates output of all children plots, making them re-render.  Master plot takes size of its host div as an output screen region and all children plots are to be fitted into the region.  (RequestUpdateLayout?) |
| SetVisibleRegion(region) | Sets the visible region (in the plot plane). This method is usually called by navigation.    Takes the AspectRatio value into account.  Figure: New visible region may cause update layout, so the method should call UpdateLayout.  All: re-render data. |
| GetVisibleRegion() | Builds a visible region in the plot plane from the current plot transformation and screen region of the plot. |
| RequestNextFrame(plot) | Only master plot should accept this call. It notifies that the given plot needs another render call at the next frame (to allow other system events to be handled between the renders).  The plot is marked as ‘RequestsRender = true’.  If the master plot IsInAnimation is true, returns  Otherwise, checks whether the timer for the next frame is set or not. If not set, sets the timer.  When the timer fires, finds all children plots with ‘RequestsRender’ and calls Render method.  (We can also provide duration from the last frame) |
| Render(coordinateTransform) | Updates the output screen region. It can draw in the corresponding canvas, or put/move DOM elements in the host DIV element. Some plots may not have this method (e.g. figure).  The plot transform is taken from the master plot. Visible region can be determined from the plot transform.  (We also might need to provide a context for the method to keep DOM elements).  The method is usually called when:   * layout of the master plot is updated (called by the master); * this plot previously requested next frame (called by the master); * plot appearance properties are changed (called via requesting next frame or directly by the plot; if fit to view is switched off); * plot data has changed (called via next frame or directly by the plot).   If Render uses animation or batches it may request next frame to complete the work. |
| Plot(data) | This method is to be called by a user to render the data. It may be missed in some plot objects (e.g. a figure); particular type of argument depend on a plot object, e.g. marker.Plot() may take an object:  { x: [1,2,3], y: [0,2,10], color: [0.1, 0.2, 0.01], palette: ’red,green,blue’, size: ’10px’, shape: ‘square’ }    { x: [1,2,3], y: [0,2,10], color: [0.1, 0.2, 0.01], palette: ’red,green,blue’, size: ’10px’, shape: { draw(context2d, marker), measure(marker) } }  The method changes the inner state of the plot (e.g. store the given data and properties) and calls the Render method of this plot instance.  If the plot uses batches to render the data, it can finish the previous batch and then render the most recent data.  If auto fit is enable it should check the bounding box and probably update the layout.  If properties change, the plot should fire the Change event to update a legend. |
| GetBoundingBox()  GetPadding() | Implementation depends on particular type of the plot. Returns the bounding box required to visualize all the data using current properties values.  Padding is a quad which is a screen bounding rectangle for a plot – minimal axis aligned rectangle that contains all screen representations of graphics elements produced by plot. In some cases screen bounding rectangle is larger than plot bounding rectangle transformed from figure to screen coordinates. For example, line graph drawn by stroke with thickness 5pt or points displayed with radius 10pt circles. |
| FitToView() | Children delegate this call to the master. Master builds the total bounding box for all children and if it is not equal to the visible region, changes the visible region. |
| AddPlot(plot)  AddDOM(element)  Line(data, properties):Plot  Markers(data, properties)  Heatmap(data, properties) |  |
| RemovePlot(plot)  RemoveDOM(element) |  |
|  |  |

(Mouse support: tooltips, highlighting)

### Update layout & rendering algorithms

D3.Plot exposes API to update layout and rendering of a plot:

* updateLayout (requestUpdateLayout) updates screen positions and sizes of children DOM elements of the plot in accordance with the current screen size of the master plot’s host DIV element and the current visible rectangle in plot coordinates. The async version of the method allows to perform only most recent update, and thus has better interactivity for low-frame rate rendering.
* requestNextFrame(plot) invalidates the given plot and asynchronously requests rendering for it in current visible rectangle.
* requestNextFrameOrUpdate() is intended to be called from derived plots. This method efficiently invalidates the plot;
  + If auto-fit mode is enabled, and bounding box or padding changed, layout update is requested.
  + Otherwise, next frame is requested.

Private method of the D3.Plot setVisibleRegion disables auto-fit (if it is enabled), takes a desired visible rectangle, and saves it in the private field \_plotRect, and asynchronously updates the layout which will make the saved rectangle actual. This method is called during the navigation’s animation and to synchronize visible rectangles of bound plots (see Bound plots). For the latter, the method takes the second argument which suppresses notifications for bound plots in the coming update layout (to avoid echo in the binding).



Update layout algorithm takes the current screen size of the master plot’s host DIV element and the desired visible rectangle in plot coordinates (depends on whether auto fit mode is enabled or not), then measure new layout and arranges the children elements:

* computes visible plot plane region from bounding boxes of dependent plots (if auto fit is enabled),
* computes and updates the physical layout (incl. size) of the plot’s inner HTML elements (this may concern visible plot region),
* forces rendering for each plot.

Stage which computes physical layout of elements of the master plot is called *measure*.

Layout update is done by plot.arrange() method which gets the actual screen region for the plot and the plot is to be arranged in accordance. Default Plot adjusts the host div to take the given place. But particular plot implementations should override this method to alter their inner elements, too (e.g. plot’s canvas should also be resized).

Visible plot plane region is computed (aka “fit to view”) as follows:

* for each plot we call plot.getBoundingBox(); plot returns a bounding box in the plot plane or special value (undefined), indicating that the plot cannot determine the bounding box (probably, this plot depends on other plots, e.g. map plot or function-based plot);
* building a total bounding box as a union of bounding boxes returned by the plots (ignoring undefined boxes);
* providing the total bounding box to the plot those returned undefined previously, so that they can extend or do not change it;
* the final bounding box is a bounding box for all plots, but we also must consider thickness of plot’s elements (e.g. line thickness or marker size), which should be also visible even for all elements when we make fit to view, so we do the following:
* for each plot we call plot.getPadding(); plot returns 4 values representing margins in screen pixels to be arranged around the bounding box.

(See algorithm in the CEES DEV-Documents\Plot Update Layout.pdf)

Extra padding is an extra value in pixels which is added to the padding computed for the plots. It’s goal is to improve visual quality of the picture. The value can be configured using D3.padding constant.

### Deferred update layout

Plot.requestUpdateLayout() sets the timeout to invoke Plot.updateLayout(). Such request cancels all previously requested renders (via Plot.requestNextFrame) because the update will include renders. If render is requested after update was requested, it is ignored.

### Declarative

Declarative definition of a figure with plots, axes and legend:

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2px”

data-d3-datatransform=”new LogTransform();”>

<table>

<tr><th>x</th><th>y</th></tr>

<tr><td>0.1</td><td>20</td></tr>

<tr><td>0.2</td><td>10</td></tr>

</table>

</div>

Name of a plot by default is value of the name attribute, if it exists, or a value of its id.

Names of plots can duplicate or be empty. Plot.get(name) method returns undefined or one of the plots having the given name.

Should we make auto-initialization of all plots in the page?

* For “No”: because we might want to initialize plot in a specific order.

### Imperative

Var plot = new D3.Plot($(“div”));

plot.polyline({x: xvalues, y: yvalues, stroke: ‘Red’, thickness: 2 });

var polyline = plot.polyline({x: xvalues, y: yvalues2, stroke: ‘Blue, thickness: 2 });

…

For(…) {

polyline.plot({x: xvalues, y: yvalues3, stroke: ‘Blue, thickness: 2 });  
}

### Canvas-based plot

Some D3 plots can use private canvas element to render its output (e.g. markers). Use of private canvas instead of a shared canvas allows to invalidate and render plots individually (for example, if plot’s data or markers color change), thus increases performance on updating the composite plot.

To facilitate development of canvas-based plots, D3 has the CanvasPlot object to be used as a prototype. When initializing, it adds a canvas element to the host div and sets the classes of the element. Also, it can destroy the plot clearing the added elements and attributes.

At the level of the CanvasPlot can be implemented fast navigation based on moving/scaling already rendered image instead of re-rending real data.

CanvasPlot exposes getContext() method to get canvas context to draw.

To support a large number of canvas-based plots we need to enable canvas-sharing, since a number of canvases is limited by browser and machine capabilities (appr. 50-100 canvases on a good machine). The solution is two enable 2 modes:

* *navigation*, when user actively navigates through the plot space. In this case all the plots are to be re-rendered so we have no benefits from multiple canvases. Hence, use a single canvas for all.
* *update*, when data of individual plots update, and user mostly doesn't navigate the chart. We have benefits from individual canvases since we can re-render only updated plots.

### Multiple canvas vs single canvas

Master plot supports a style “flatRendering: true|false” and exposes property FlatRendering: ‘true|’false’.

|  |  |
| --- | --- |
| Flat rendering (single canvas) | Multiple canvases |
| Minimizes number of <canvas> elements on a page (critical for Chrome). | Fast updates of single plots, when auto fit is disabled. |
| Suitable for chart with many plots (>= 50). | Suitable for chart with small amount of plots. |

Flat rendering mode uses single canvas for all CanvasPlots; therefore it may corrupt order of layers, if between two canvas plots there is a non-canvas plot. In this case, the latter will be placed above the canvas plots.

A canvas of first canvas plot (in rendering order) is shared by all canvas plots.

This mode is implemented at the level of the D3.CanvasPlot object, which returns a canvas context for derived canvas plots. In the flat mode, it returns same canvas for all canvas plots.

### Data sources

Data source is an object provided as an argument for the D3.Plot.plot() method. Actually required signature of the object depends on the particular plot object. For instance, markers plot can take the following object: { x: [1,2,3], y: [0,2,10], color: [0.1, 0.2, 0.01], palette: ’red,green,blue’, size: ’10px’, shape: ‘square’ }.

Guidelines for the data source specification:

* Data source is an object which contains a number of mandatory (e.g. y) and optional (x, color) data series and optional styles (palette, size, shape).
* Data series can be either an array of primitive values (e.g. numbers), or a function.

If data changes, the plot can be update through another call of plot() with the updated data source object (i.e. there is no way to update a single point of the data series), but a plot object *can* provide properties to update visual styles (e.g. marker shape) without calling plot.

Data source can be defined declaratively in the HTML source of the page within the plot’s DIV. If presented, data is read from the HTML element either by default supplied with the plot implementation, or defined by a developer in the plot’s HTML definition using data-d3-datasource attribute:

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2px” data-d3-datasource=”parseMyData” data-d3-datatransform=”new LogTransform();”>

<table>

<tr><th>x</th><th>y</th></tr>

<tr><td>0.1</td><td>20</td></tr>

<tr><td>0.2</td><td>10</td></tr>

</table>

</div>

Default style of the plot can be changed by providing it using data-d3-style attribute in the declarative declaration.

### Events

Plot object exposes following events (see D3.Event):

|  |  |
| --- | --- |
| Event | Description |
| appearanceChanged | Occurs when a plot’s property affecting its visual appearance changes.  Event parameter contains name of the property that is changed. If there is a significant change in several properties, this property is undefined.  E.g. stroke, thickness etc.  This event is useful for the legend implementation. |
| childrenChanged | Occurs at master plot when collection of children of the plot or one of its descendant plots is changed. |
| visibleChanged | Occurs at a plot when its actual visible rectangle is changed. Handler gets (senderPlot, visibleRect) (in plot coordinates).  Actual visible rectangle can differ from the rectangle provided in Navigation.SetVisibleRect(), since it may be constrained by aspect ratio. |

The events are attached to the master plot.

### Mouse support

Master plot subscribes to the “mousemove” and “click” events. When the event occurs, master plot computes plot coordinates for the screen mouse position and calls onMouseMove and onClick methods (if they exist) providing the point both in screen and plot coordinate systems.

Note: these methods are not part of API and therefore should not be overridden by developers; if a developer needs to subscribe to mouse events, he can use standard approaches, e.g. jquery’s .mousemove for the plot’s central part.

Figure places dummy div element to its central part to provide events for the mouse within that part both for this case and for navigation. Other plots use their host element as the events source. For this in the plot API there is a property CentralPart that returns the div.

### Add and remove plots





Master plot allows to add/remove plots in different ways.

First, pair of methods D3.Plot.**addChild**(plot) and D3.Plot.**removeChild**(plot) logically adds/removes a plot from its container plot (child may be not immediate). The added plot changes its master to the master of the new container plot. Remove method first recursively looks up for the plot to remove and, if succeeds, removes it from the container.

These methods are designed to be used by developers creating new types of plots.

D3.Plot.addChild (plot) accepts an object which has an instance of D3.Plot as its prototype. This functions adds new plot to the list of the children and causes request layout update. From one side this allows to add multiple plots one by one without updating layout immediately after each new plot is added; from other size, this eventually cause new layout.

Method D3.Plot.removeChild(plot) removes the plot identified either by its name (attribute div-d3-name) or by a plot instance itself. If there is no given plot, no exception fires and the method returns false. A plot removed in this way can then be added once again.

Next, method D3.Plot.**remove**() physically removes this plot from its container. This includes invocation of the removeChild method and removing of the plot’s host element from the DOM tree. The D3.Plot.remove() method is to be used by developers to remove plots imperatively. A plot removed in this way cannot be added once again or used to plot something.

All these methods cause asynchronous layout update. This allows multiple consequent add/remove calls eventually followed by the layout update.

Next, there are specific functions to add new plots of particular types.

D3.Plot.**polyline**(name, data) creates or updates a plot with the given name. If there is no plot with the name in, it creates new polyline plot and adds it to the master plot. Otherwise, if the plot exists, it must be polyline and the plot method is called for this with data passed as an argument. Therefore this method can be called in a loop, so that first call creates the plot and next calls just update existing plot. The plot can be removed using the D3.Plot.remove() method.

### Finding an object in a Plot

Plot.get(p) finds and returns an object for a parameter p, which can be one of following types:

* Name of a plot, which is set through data-d3-name attribute or the Plot.name property.
* ID of a plot’s (or an axis for a figure) host element.
* Host element of a plot (or an axis for a figure).

If an object is not found, returns undefined.

### Plot Factory and new plot registration

To register new data-d3-plot key, use

D3.register(‘myplot’, factory);

D3 maintains a table “key”-“factory”, where key is a string, factory is a function .

If key already existed, it is replaced. Key “plot” cannot be replaced.

## Bound plots

Plots binding means binging of their visible rectangles; i.e. if one of the plot is navigated, the bound plot shows the same visible rectangle. Note that screen sizes of the plot still may differ; only visible rectangle in plot coordinates is same.

Binding is always two-way; but if you need one-way binding, you can disable navigation at one of the plots.

It is possible to bind just vertical or horizontal ranges of the visible rectangles, or both.

API:

var binding = D3.bindPlots(plot1, plot2);

var binding = D3.bindPlots(plot1, plot2, filter);

Where the optional filter is either “vh” (default), “v”, “h”.

Note: only master plots are bound; if a dependent plot is given, its master plot is used instead. Order of arguments can be arbitrary.

The resulting object is of type D3.PlotsBinding.

To remove the binding, call

binding.destroy();

After binding is destroyed, it is no more usable.

## Figure and Chart

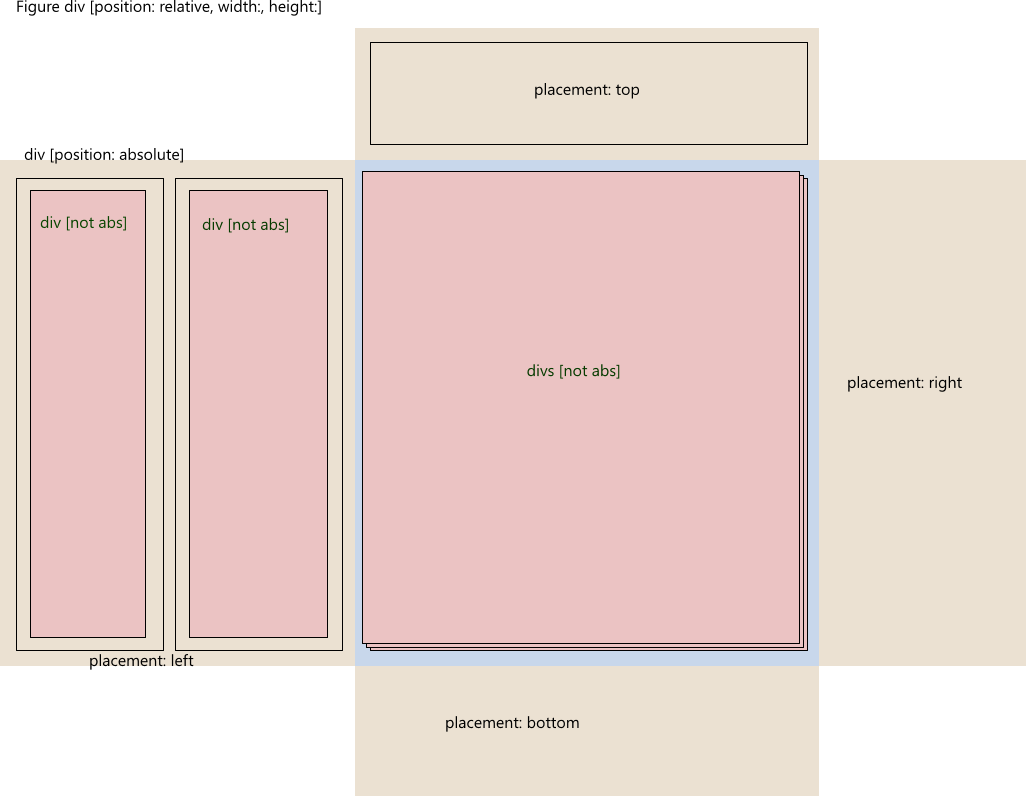
*Figure* is a master plot which, in addition to children plots, maintains and layouts a number of DIV elements with user-defined placement.

Figure overrides the default plot layout algorithm and makes a layout of inner plots, axes and other in accordance with the *data-d3-placement* attribute in HTML and placement argument in imperative API. (Placement cannot be changed after adding an element to the figure.)

Supported placements are: left, right, top, bottom, and center.

Usually plots are placed in the center of a Figure, axes and titles are placed in the side slots. Figure class provides two-pass algorithm that prevents well-known loop occurring on resize of figure with fixed aspect ratio: figure resize forces update of plot-to-screen transform which adjusts labels on the axes. Change of label size may result in change of central part size which again updates plot-to-screen transform which in turn leads to axes label updates and so on.

Figure positions child DIV elements with attribute data-d3-placement. All other elements are positioned by default within the host element of the figure.



Layout rules:

1. Figure is DIV with relative position and fixed width and height.
2. If some DOM element is a child of the figure and :
   1. If : layout algorithm computes its position so that such elements are positioned sequentially in a certain side part, and its width (for top/bottom parts) or height (for left/right parts). For this, element is placed into *a figure element container*, which is a DIV positioned as absolute in coordinates relative to the figure. The order of elements within the pairs left/right, top/bottom is inversed.
   2. If , it is also placed into a figure element container which fills the entire central parts. Thus, elements in the center are positioned as a stack one over another in the order as they are given in the HTML or added to the figure.
3. If , the element is positioned within the figure.

To position element within the central part, it is possible to add a div with placement “center” and position “relative” and put inside it another div with position “absolute” and set the left/top attributes. This is the way how a draggable legend can be implemented.

There is no way to position an element within side parts of the figure.

Method figure.getAxes() returns all axes of the figure, figure.getAxes(placement) returns axes with given placement, which is left, top, right, or bottom.

### Chart

*Chart* element is a prepackaged figure with *axis*, *grid lines*, *legend* and *title*.

Figure can be identified in HTML by data-d3-plot="figure". Example of declarative definition of a figure:

<div data-d3-plot=”figure” id=”figure”>

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2px”

data-d3-datatransform=”new LogTransform();”>

<table>

<tr><th>x</th><th>y</th></tr>

<tr><td>0.1</td><td>20</td></tr>

<tr><td>0.2</td><td>10</td></tr>

</table>

</div>

<div data-d3-plot=”markers”>

<div data-d3-plot=”heatmap”>

</div>

</div>

<div data-d3-axis=”numeric” data-d3-placement=”bottom”>

</div>

<div data-d3-axis=”numeric” data-d3-placement=”left”>

</div>

</div>

<div data-d3-legendForPlot=”figure”>

</div>

…

plot = D3.InitializePlot($(“figure”));

Chart is figure which contains vertical and horizontal axes and a number of plots:

We claim that:

* Data transform:

1. . We propagate new data transform when it is changed through the chart’s API to its dependant plots; but not vice versa. If a user changes data transform at some dependent plot, the axes and other plots will still have the chart’s data transform. (Note that Figure does not propagate data transforms.)
2. , therefore we need to specify a set of axes supported by chart (i.e. numeric, date time, logarithmic). Change of the data transform through the chart’s API leads to potential replacement of the axes or to an exception if the new data transform is unsupported.

* Plot transform: as any other plot.

Chart’s plots should not have data transform different than chart’s data transform. User still is able to change the data transform of an individual dependent plot, but (i) though the plot will be rendered using its own data transform, it will not be reflected in the chart’s axes, (ii) if then it changes chart’s data transform, the plot’s data transform will be replaced.

### Title

Title is a div element with an attribute data-d3-placement in a figure:

<div data-d3-placement="top">

The Title

</div>

## Layers

Plot and a central part of a figure is a composition of layers. These layers must be carefully arrange using z-index to enable mouse events be properly propagated. Z-indices are now configurable in the D3 namespace.

The layers below are listed so that first have lower z-index and last have higher z-index:

* Plots, in the order as they are defined in the HTML or added using API.
* Navigation layer used to handle mouse events. (This is a “central part”; in fact it is an empty plot object.)
* DOM Markers (see DOMMarkersPlot).
* Tooltip.

(Another possible solution is to move all plots and other elements from central part into the center plot which is used as a navigation layer. Then all its dependent plots’ events are automatically propagated to the navigation layer. Why not to use <div> as a central part, not a plot?)

## Axis

(see D3JS Modeling project)

Axes controls show how numeric values of different nature correspond to screen coordinates. Labels are drawn for each major tick. Shorter minor ticks divide space between major ticks.

Axis control initializes for div element and adds a canvas element to render lines and ticks. Labels are rendered as div elements added to the host div element.

Axis control renders given range in data coordinates within actual screen region determined by the host element. When constructed, axis needs to be initialized as either horizontal or vertical.

If data range or screen size of the host element change, the Update() method should be called to update the axis on the screen.

Different implementations of axis should be used for different data transforms used in a correspondent plot.

Axis has orientation which is either horizontal or vertical; labels are arranged based on the given placement (bottom/top, or left/right, respectively for the orientations).

### Supported types of axes

|  |  |
| --- | --- |
| Axis type | Description |
| numeric | Standard numeric axis with range , dynamic tick selection and support for data transform. |
| log | Logarithmic axis with plot range , data range . Data transform is fixed (log10). |
| labels | Shows fixed number of ticks and labels given in initialization as arrays labels and ticks (optional).  Ticks are numeric values in plot coordinates.  Labels are objects of any type; they will be converted to strings or placed as HTML on the axis.  There are two modes:   1. Ticks is omitted, or length of ticks equals length of labels. Each tick has a corresponding index; if there are no ticks, labels are numbered sequentially. 2. Length of ticks is one more than length of labels. Labels are placed between two ticks.   Axis shows only ticks given in input array.  Supports data transform. |

When an axis’ screen size or plot range changes, the Update() method is to be invoked. It does the following:



## Navigation



NavigationController is an object which produces a sequence of coordinate transforms and has an input stream as a source of so called gestures. A gesture is an object describing one of supported navigation action (pan, zoom etc.); now we implement a limited set of gestures sources (mouse-based and multitouch), but it can be extended later.

List of the supported gestures:

* Pan changes the visible region keeping the zoom level.
* Zoom changes the zoom level keeping the visible region.
* ZoomTo changes both the visible region and the zoom level. This is implemented using elliptical zoom animation algorithm.
* Pin stops the animated navigation process (started previously).

(See mathematical background for the gestures in the Navigation section (page 4))

### Implementation details

*Gesture source* is an observable sequence of gestures.

*Navigation* subscribes to a gesture source (assigned externally), and for each gesture computes target visible rectangle in plot coordinates in accordance with the received gesture. Then it starts new *animation* which is an observable of visible rectangles (in plot coordinates). Navigation subscribes on the animation and updates the plot’s visible rectangle.

Each master plot has a single non-replaceable navigation object, which can use different animations. Dependent plots use its master’s navigation.

Animation is customizable and can be assigned externally to a plot’s navigation. D3 exposes following animations:

* D3.Navigation.panZoom
* D3.Navigation.ellipticalZoom
* D3.BingMapsAnimation

Each of these objects is a factory function with the following signature:

*Animate*: (*getVisible* : ()→(visibleRect x screenRect)) x *finalRect* : visibleRect → observable<visibleRect>

### API

Plot.navigation : Navigation // initialized at master plot in ctor

// API to change plot’s visible rect:

// - Each method stops previous animation

// - Calls inner method Plot.setVisible()

navigation.**setVisibleRect**(visibleRect, useAnimaton);

navigation.**stop**(); // stops animation

navigation.**gestureSource** { get; set; } // type is RX observable of gestures

// set undefined clears the current gesture source

// set new source subsribes to the new source

// NB.: add new source is: nav.gestureSource := nav.gestureSource.merge(newSource);

// get returns current source

// Plot’s routines

Plot.setVisible() –(eventually)--> Plot.updateLayout()

// Plot.setVisible turns off IsAutoFitEnabled

Plot.updateLayout(): may call Plot.Navigation.adjust(); // adjusts to current screen size and coordinate transform

Plot.fitToView(); // calls setVisibleRect(vr, false), thus stops previous animation

Plot.isAutoFitEnabled := true; // stops animation

### Multitouch

Multitouch is supported by D3 charts on iPad Safari and Windows 8 IE10.

D3.Gestures.getGesturesStream() returns a navigation gesture source depending on current browser’s capabilities. The method, if supported, enables pan and zoom multitouch gestures along with mouse movement, mouse buttons clicking and mouse wheel scrolling.

The chart by default calls the getGesturesStream() and sets the response to its Navigation.gestureSource. Also it adds same but filtered sources for vertical and horizontal axes; the filters enables panning and zooming only along an axis.

To make special handling of multitouch gestures in D3 plots, the default behavior for a web page should be overridden. For this, in iPad/Mac use D3.NavigationUtils.SuppressDefaultMultitouch(). For Windows 8, it is done automatically through CSS class .d3-plot-master’s property “-ms-touch-action: none;”.

## Legend

Legend is a <div> bound to a plot; it includes legend items for each of the plot’s dependent plots (incl. itself). Therefore there can be several legends, bound to same or different plots.

Legend reacts on the event D3.plot.appearanceChanged(plot, parameterName) and updates legend items corresponding to the plot.

Plot has getLegend() method which builds HTML of <div> representing its legend item (may return undefined).

Plot doesn’t know anything about its legends; instead legend is attached to a plot:

Var plot = D3.asPlot(‘plotdiv’);

…

Var legend = new D3.Legend(plot, ‘legendDiv’);

// legend subsribes on plot.childrenChanged

// legend calls plot.getLegend() foreach plot’s dependents and makes a list of these returned <div>

// default getLegend() returns undefined

// for each dependent plot (when we call getLegend()) we also subscribe plot.appearanceChanged.

Removing legend:

legend.isVisible = true|false; // set display: none

legend.remove(); // unbinds from events and removes the <div>

Chart adds a default legend which is shown within the chart.

Chart allows to disable a legend in HTML code: data-d3-style="isLegendVisible:false;"

Declarative definition:

<div data-d3-plot=”polyline” data-d3-legend=”legendId”>

…

<div id=”legendId”> // can be outside of the plot

</div>

</div>

Initialization of these declarative definition is performed by the plot which has the attribute “data-d3-legend” in initialization.

D3.css exposes d3-legend and d3-legend-item classes.

## Tooltips

## Implementation remarks and decisions

### Shared canvas vs Private canvas

… per plot. We have chosen a private canvas per plot so that it could be updated separately from others, if data or appearance of the plot changes.

Disadvantages: larger number of canvas elements on the page; several times we have to build context for them.

See Plot/Canvas-based plot section.

### Web Workers

Reference: <https://developer.mozilla.org/en-US/docs/DOM/Using_web_workers>

Web workers can be used in particular to make controls more responsive and improve performance. For plots, their following areas for potential use of workers:

* Preparing data for rendering.
* Rendering of data (heatmap, markers).

# Plots

Key concept: all draw() methods of plots (if presented) inherit missing properties of previous calls or initial d3 style value. E.g. if first draw() provides stroke for a polyline plot, and second doesn’t, the stroke remains from the first call unchanged.

## What can be inside of a plot element?

* A div element…
  + With attribute data-d3-plot is either dependent or a master plot.
  + With attribute data-d3-placement is an element to be placed in a given part of the figure.
  + With attributes data-d3-x, … is an element which is positioned within the plot’s data space.
* Any other element is just placed within the plot’s host element in accordance with the standard HTML rules.

(IFrame element is not properly clipped by Chrome. But it works in DIV.)

## PlotCanvas

It is a base object which should be used as a prototype for all plots which render on a canvas.

Depending on a browser type and a machine, maximum number of canvas is limited to about 50 – 100. Since we stated that each canvas plot should use its individual canvas, we limit maximum number of plots in that way.

The solution is to limit maximum number of individual canvases and then use shared canvases, minimizing number of plots per each canvas.

This would complicate tooltips building for polylines, since we cannot just check the pixel color under the mouse cursor to detect whether this point belongs to a line or not (but still this check can give correct negative answer).

## Polyline

Polyline is a canvas-based plot which draws a function using polyline. Polyline plot supports two types of data sources, array-based or functional.

In HTML, a polyline plot is indicated with data-d3-plot=”polyline”.

In JavaScript, use D3.Plot. polyline (name, data), or D3.PolylinePlot.draw(data).

The methods copy data so it can be re-used by the caller. In contrast, values of the data object are not copied because it can degrade the performance. So if a caller needs to re-use, for example, “x” array, it should be copied prior to calling the methods.

Array-based data source provides grid function to render: . Polyline draws segments .

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| y | array of doubles | Mandatory array representing values of the grid function.  May contain NaN indicating missing value. |
| x | absent | We take . |
|  | array of doubles, dates or strings | Defines an array of so that .  May contain NaN indicating missing value. |
| thickness | absent | Default thickness is 1 pixel. |
|  | double, positive | Thickness of the line in pixels. |
| stroke | absent | Default color. |
|  | string, color | E.g. ‘red’, ‘#f0bb20bb’. |
| lineCap |  |  |
| lineJoin |  |  |

Functional data source provides a function value(x) and domain of X as a range :

or as a set of values:

(the latter in particular enables use of strings for x-values).

### Declarative definition

A table element with data can be embedded in the plot div element:

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2px” >

x y

* 1. 20
  2. 10

</div>

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2px” data-d3-source=”D3.readTable”>

<table>

<tr><th>x</th><th>y</th></tr>

<tr><td>0.1</td><td>20</td></tr>

<tr><td>0.2</td><td>10</td></tr>

</table>

</div>

Columns of the table are data series with same header names as in the parameter of the plot() method.

To provide custom data source declaratively, use the attribute data-d3-datasource to provide the function which gets the host div as an argument and returns data source object (as defined above):

<div data-d3-plot=”polyline” data-d3-style=”color:blue;thickness:2” data-d3-datasource=”parseMyData”>

</div>

Default data source is “D3.readCsv” which allows to copy/paste data from Excel if only the table contains at least y column.

If style is defined in the declarative declaration, it is then used as a default for the consequent plot calls.

Polyline plot has properties enabling changing of appearance of the plot, without calling plot once again. Change of these properties causes re-rendering of this plot only, if the auto fit is disabled.

## Markers

In HTML, a marker plot is indicated with data-d3-plot=”markers”.

In JavaScript, use D3.Plot.markers(name, data), or D3.MarkerPlot.draw(data).

The methods copy data so it can be re-used by the caller. In contrast, values of the data object are not copied because it can degrade the performance. So if a caller needs to re-use, for example, “x” array, it should be copied prior to calling the methods.

Markers plot renders pairs as markers of different shape, size and color. It is also possible to provide a custom rendering function for a marker.

The argument data of the markers API is described in the following table.

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| y | array of doubles | Mandatory array representing values .  May contain NaN indicating missing value. |
| x | absent | We take . |
|  | array of doubles, dates or strings | Defines an array of so that .  May contain NaN indicating missing value. |
| shape | “box”, “circle”, “cross”, “diamond”, “triangle”, or D3.CustomMarker object | Determines the shape of each marker. |
| border | Absent | No border |
|  | Color | Renders a border around each marker |
| color | Color | All markers are filled with the color |
|  | Array of Color | Elements of the color array are colors for the corresponding markers. |
|  | Array of Numbers | The array is a data series and actual color of a marker is built using colorPalette. May contain NaN. |
| colorPalette | D3.ColorPalette | It is mandatory if color is an array of numbers. |
| size | Number | Size in pixels of all markers. |
|  | Array of Number | If sizePalette is defined, this array is a data serias and actual size of a marker is build using sizePalette. May contain NaN.  Otherwise, the array contains sizes in pixels of each marker. |
| sizePalette | D3.SizePalette | Builds size in pixels of a marker from size data series. |

If either x or y, or any required data series has value NaN, this marker is not rendered.

No border significantly increases performance!

## Heatmap

HeatmapPlot (data-d3-plot=”heatmap”) renders values defined on a rectangular grid using color palette.

In JavaScript, use D3.Plot.heatmap(name, data), or D3.HeatmapPlot.draw(data).

The methods copy data so it can be re-used by the caller. In contrast, values of the data object are not copied because it can degrade the performance. So if a caller needs to re-use, for example, “x” array, it should be copied prior to calling the methods.

### Data

Data for the heatmap is , where

and is either or .

In the former case heatmap renders in *a gradient mode*, when each cell of the grid has 4 colors defined in the corners of the cell and color within the cell is interpolated.

In the latter case heatmap renders in *a matrix, or bitmap, mode*, when a cell is built so that each point is in the middle of that cell and determines the color of the entire cell.

If or is missing, it is assumed that .

At last, is a color palette described in the section Color Palette.

### Declarative definition

Heatmap plot supports declarative definition. The data-d3-style can contain attributes:

* “*palette*” is a string representation of the color palette to be used in the heatmap;

Initial data can be read using data initializer (either default, which is D3.readCsv2d, or provided through data-d3-datasource). Default representation is chosen to enable simple copy/paste from Excel’s tables, e.g.

…

…

…

where is a regex pattern for the separators; delimits lines. Values must be float; can be .

### Implementation

Heatmap plot uses web workers to render the data. The rendering algorithm is as follows:

1. Build intersection of the visible rectangle and the bounding box of this plot’s grid. If intersection is empty, work is finished.
2. Project the intersection into the screen region (i.e. find the screen rectangle where the image should be rendered).
3. Render the placeholder where the actual heatmap image will appear.
4. If previously completed task was created for the region which intersects with the current visible rectangle and scale hasn’t changed (scaling might be too complex task to do it here), its result image is put on the screen.
5. If the resulting image of the previously completed task is less than the currently needed region (computed in steps 1 and 2), or it is missed, new task is created and is either posted to the worker (if it is free), or set as a pending task (probably, replaces previous pending task).

Worker takes a task and pixel by pixel fills the image data. When worker’s response received in the UI thread, just completed task is saved; if there is a pending task, it is passed to the worker; next frame is requested (it causes new rendering invocation).

Heatmap uses D3.SharedRenderWorker to enqueue tasks into a single queue so that only one web worker is used to handle tasks one by one. Each heatmap plot can have a single pending task which is replaced if new pending task is enqueued. If a browser doesn’t support for web workers, shared render worker renders the tasks in foreground.

Optimization tricks. Heatmap does not produce new render task during animation. This eliminates gaps while navigating the plot.

If a heatmap becomes invisible (i.e. it is too small or outside of the visible rectangle), it cancels its pending task.

Data transforms are functions and therefore cannot be passed from UI thread to the worker; there are several ways how to nevertheless pass them:

1. Common d3datatransform.js file for UI and workers; each data transform has string type attribute which is passed to the worker and d3datatransform.js has a factory function creating the transform instance from the type. Negative: restricted set of transforms; to use unknown transform, it should be added to the d3datatransform.js and to the factory. (It is implemented now.)
2. Serialize a data transform into a string and evaluate it on the worker side. Negative: performance drawbacks (estimate!), prone to errors (closures are lost).

Similar problem is for passing palettes. Solutions:

1. Discretize the palette as an array of colors. Negative: non-continuous palette require special handing.
2. Pass a state of the palette object. (It is implemented now).

Restrictions: both x and y grids should have at least 2 elements.

NaN in grid and value arrays are allowed.

### Data transforms

Data transforms are objects with functions. Functions cannot be passed to background workers, used by heatmap plots to render. Therefore, heatmap plot can use only data transforms defined in the d3transforms.js, and the type property should have name. This name is passed to the background worker to identify the transform.

To enable custom data transform, add it to the d3transforms.js and give it a name.

It is also possible a custom data transform into a string and then eval it in the background worker. (Not implemented yet).

### Transparency

Color palette of a heatmap enables transparent colors using the alpha channel.

Another option is to use HeatmapPlot’s property opacity, or data-d3-style’s attribute opacity, or provide opacity property to the draw method to set the opacity factor for the heatmap. Domain of the parameter is , where 0 means transparent, 1 means opaque heatmap.

## Color Palette

Color palette represents a mapping from a number to a color. It is presented in the API as an object

The function has an domain . If type is absolute, and are arbitrary numbers such that .

If type is relative, , and a user of the palette should normalize the values to that range.

Argument of the is normalized to the domain of the function:

### Palette API

Parsing a palette from a string:

D3.colorPalette.parse(“red,green,blue”);

Colors are identified by names (case insensitive), or using hex syntax #FFFFBE20 (case insensitive). Values “rgba(100,100,20,0.5)” are not supported.

A palette can be either non-normalized (absolute) or normalized (relative); this can be determined by the property D3.ColorPalette.isNormalized. Absolute palette exposes properties min and max, representing a segment which is mapped to colors; values outside of the range are replaces with nearest value of the segment. Relative palettes suppose that a using code normalizes given values to the segment (values outside of the segment are also trimmed).

Palettes defined using functions are not currently supported.

D3.colorPalette.getRgba() returns {r,g,b,a} structure, where r,g,b are integer numbers between 0 and 255, a is a float between 0 and 1. (Alpha 1 means opacity and 0 means transparency.)

D3.colorPalette.getHsla() returns {h,s,l,a} structure, where h (hue) is a float between 0 and 6; saturation (s), lightness (l) and alpha (a) are floats between 0 and 1.

Factory methods:

* D3.ColorPalette.parse(definition) parses the palette from a string.
* D3.ColorPalette.create (colors) creates a relative (normalized) uniform palette with given array of colors (each is either HSLA or RGBA). Note: to create non-uniform palette from code, use D3.ColorPalette constructor.
* D3.ColorPalette.absolute(min,max) converts a palette to an absolute with given range (points are changed proportionally).
* D3.ColorPalette.normalize() makes the palette relative.
* D3.ColorPalette.banded(Number|Array) makes the palette discrete with solid colors. If a number is given (the value must be an integer greater or equal to 1), result palette is divided into segments with solid colors. If an array is given (any length), it contains points within the range of the palette, representing boundaries of segments with solid color.

For the case when banded() gets a number (), the formulas for the banded palette’s colors are given below:

Let is a original palette with range . Total number of resulting bands is , bounds for the -th band are:

Let are colors of bands of the new palette, and

It can be easily proved that belongs to the -th band bounds for .

If banded() gets an array of points within , the bands bounds are known and colors are taken for the middles of the bands from the original palette.

Example:

D3.ColorPalette.create (new D3.rgbaColor(r1,g1,b1,a1), new D3.rgbaColor(r2,g2,b2,a2))

.absolute(-10, 10);

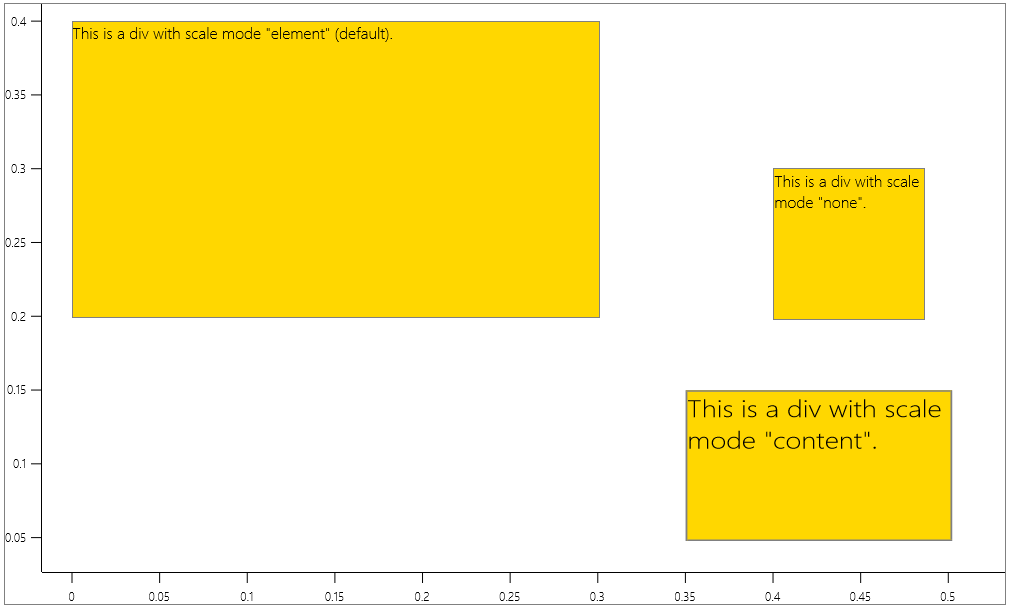
D3.ColorPalette.parse(“red,green”).absolute(-10,10);

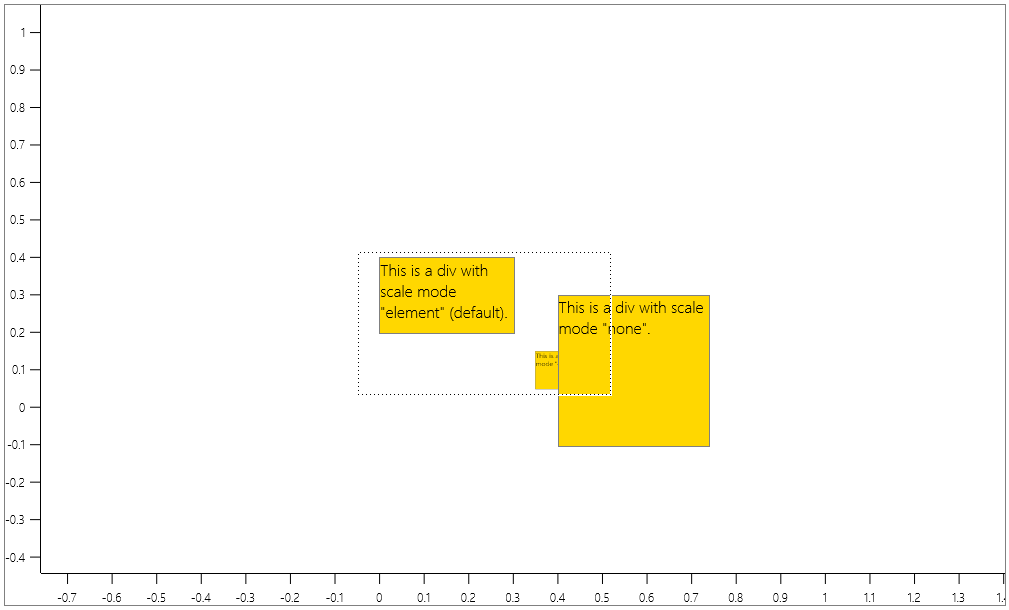
Standard palettes: D3.palettes.grayscale, D3.palettes.heatmap.

## DOM Markers

DOMPlot (data-d3-plot=”dom”) can contain one or more div elements, defined declaratively in HTML or added imperatively. For each DOM element, coordinates of the left-top corner in the parent plot’s data space should be provided; width and height of the element are computed in different ways depending on a scale mode.

There are three modes of scaling, i.e. computation of the element’s width and height in screen coordinates: element (default), content, none.





Scale mode “element”: width and height of the element in the data space are provided by a user (through HTML attributes data-d3-width, data-d3-height or through the Java Script API) and the element’s right-bottom corner is

therefore position and size of the element in the screen space depend on the zoom level and are:

Important that this scale mode changes the element’s size and position on the screen but layout of the element’s content is arrange by the browser itself. For example, this means that font size doesn’t depend on the zoom level.

Scale mode “content” also requires width and height in the data space to be provided by a user; but in this case entire element with its content is scaled to fit into the screen size computed as given above. In particular, this means that font size is also scaled.

Scale mode “none” doesn’t use width and height in the data space; instead, the element’s size is always as specified by a user via CSS style attributes width and height, though position of its left-top corner changes as the coordinate transform changes. This mode is especially useful to create pushpins bound to a certain point in the data space.

For declarative definition, add the DOM element inside the plot and specify both data-d3-x and data-d3-y attributes, which describe coordinates of the left-top corner of the element in the data space of the plot.

<div data-d3-plot=”plot”>

<div **data-d3-x=”0.1” data-d3-y=”0.2” data-d3-width=”0.5” data-d3-height=”0.2” data-d3-scale=”content”**>Custom content of the element</div>

</div>

For imperative adding/removing of the DOM element, use Plot.addDOM(element, *x,y*) and Plot.removeDOM(element). To change the position, use Plot.setPosition(element, x,y, *width,height*).

Imperative API of the DOMPlot exposes following methods:

* add(element, scale, x, y, width, height) : DIV
* remove(DIV)
* set(DIV, x, y, width, height)

## Bing Maps Plot

When using Bing Maps plot we must disable built-in D3 navigation and use standard Bing Maps navigation, because map.setView() method cannot set exact bounds as we provide (due to discrete zoom levels and aspect ratio of the map). Therefore we subscribe on the event viewchange of the map and set the D3’s visible rectangle in accordance with the map’s bounds. The problem here is when there are more than one period by longitude visible on the map. Also plot’s range must be -180..180 in order to have stable display of data on the map.

Since we use Bing Maps’ navigation, the plot should handle the mouse events. For this, either Bing Maps must be on top of other plots (but this is not good since it most cases it is a background layer), or be a parent of other plots rendered above the map.

# jQuery UI Controls

# WinJS Controls

# Release Repro Steps

1. In d3heatmap.js find reference to "d3heatmapworker.js" and rename it to "d3heatmapworker-1.0.0.js". In d3heatmapworker.js find reference to "d3transforms.js" and rename it to "d3transforms-1.0.0.js". Build D3.JS project, take local copies of d3.min.js, d3transforms.js and d3heatmapworker.js. Rename them to d3-1.0.0-min.js, d3transforms-1.0.0.js and d3heatmapworker-1.0.0.js.
2. Make a separate local folder for release
3. Copy folders "css" and "samples" there
4. Copy D3Samples.html there
5. Make "script" folder there and copy d3-1.0.0-min.js, d3transforms-1.0.0.js, d3heatmapworker-1.0.0.js, jquery-1.8.0.min.js, rx,js and rx.jQuery.js there.
6. Copy d3transforms-1.0.0.js and d3heatmapworker-1.0.0.js to "samples/script" folder
7. Ensure that all Sample pages refer to proper names of script files.

# Release 1.0.2 notes

1. Fixed heatmap image on bing maps (during navigation animation).
2. Fixed bug with resizing plot with bing maps and visualizations above it.
3. Improved bounding box computation. Now scenarios with horizontal and vertical data sources produces correct bounding boxes. Marker graph with single marker and line with single point are correctly rendered.
4. Fixed bug when navigation throwed an exception after setting undefined for gestureSource. Now navigation is correctly disabled after this.
5. Navigation works correctly in IE10 for Windows 7 now.
6. axis.update() without arguments now refreshes axis with its current range.
7. axis performance improvements (label cache improved).
8. \* DOMPlot now supports Jquery-UI "draggable" objects (objects can be made draggable and DOMPlot will handle their movements on the screen) **(not in D3, but in FC now).**
9. \* Tooltips for markers generalized and can be cutomized now **(in progress).**

# Unreleased notes

1. ColorPaletteViewer and SizePaletteViewer accept optional parameter to specify palette bar width and height and axis visibility.