Programmatically Drawing Graphics in the Web Browser

Survey Paper

Valerio Mariani, Sebastian Überreiter, and Christoph Söls

706.057 Information Visualisation 3VU SS 2022
Graz University of Technology

13 May 2022

Abstract

Programmatically drawing graphics in the web browser is an important method for creating charts or graphics, especially when users want to give their charts more individuality. Various online and offline tools offer automatic chart creation, but often have a lack of functionality or are expensive. Selecting the proper method can, however, also be a hard endeavour, considering the number of libraries and tools for programmatic chart creation. In this survey paper, the focus lies on three built-in graphics technologies: SVG-DOM, Canvas2D, and WebGL, and three drawing libraries: PixiJS, Two.js, and D3. By creating the same chart from the same dataset with each of the named technologies and tools and explaining the code, we want to give an insight how the different methods compare to each other, which one might be easier to use, and which one produces better output.

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

The goal of this survey was to compare six different technologies to programmaticaly draw graphics in the browser. To that end, the line chart shown in Figure 1.1 was remade with three built-in graphics technologies and three drawing libraries. In all cases, the results are compared against certain guidelines, and the outcome sheds light on which is the easiest and most convenient to use, and what obstacles users might face when using certain technologies or libraries.

1.1 Built-In Graphics Technologies

The three built-in graphics technologies selected for this survey are:

- *SVG-DOM*: Using JavaScript to dynamically inject SVG elements into the browser's DOM [Whitney 2013].
- *Canvas2D*: Using the "2d" rendering context of a canvas element and corresponding function calls [WHATWG 2022].
- *WebGL*: Using the "webgl" (or "webgl2") 3d rendering context of a canvas element and corresponding WebGL API calls [Khronos 2022].

1.2 Drawing Libraries

The three drawing libraries choosen for this survey are:

- *PixiJS*: JavaScript 2d drawing library, which can render using either WebGL or Canvas2D [PixiJS 2022b].
- *Two.js*: JavaScript 2d drawing library, which can render using any of WebGL, Canvas2D, or SVG-DOM [Brandel 2022].
- *D3*: JavaScript library which can manipulate HTML and SVG nodes in the browser's DOM and is often used to create charts and graphics with the SVG-DOM technique [Bostock 2021].

1.3 Dataset

To reproduce the same chart with every described method, we used the fictitious sales dataset from Keith Andrews' lecture notes [Andrews 2022]. As shown in Table 1.1, the data consists of Salesperson A and Salesperson B, each having integer values for sales for 12 months.

Figure 1.1 is also taken from Keith Andrews' lecture notes [Andrews 2022]. It shows a line chart of Salesperson A and B over the 12 months. We took this chart as a reference for drawing the graphics with the selected methods, based on the dataset.

Month	Salesperson A	Salesperson B
2012-01	28366	23274
2012-02	27050	21732
2012-03	29463	23845
2012-04	32561	28732
2012-05	28050	24023
2012-06	30100	26089
2012-07	22343	19026
2012-08	21506	17903
2012-09	24664	19387
2012-10	28842	23490
2012-11	30621	25873
2012-12	36254	28490

Table 1.1: Dataset with Salesperson A and Salesperson B.

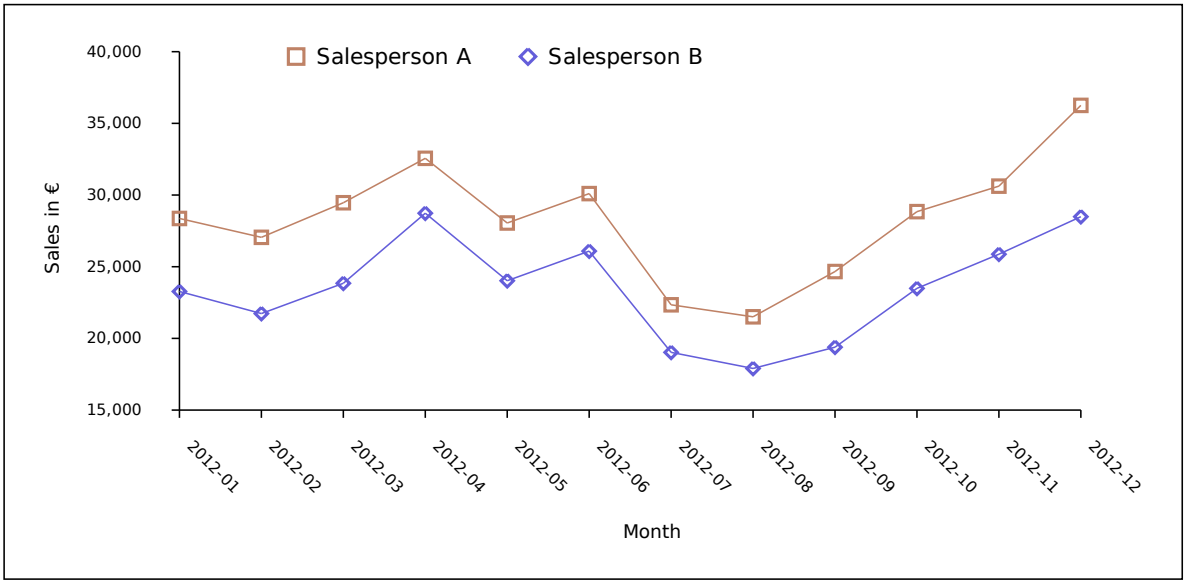

Figure 1.1: The original line chart retrieved from the lecture notes, which was drawn with Adobe Illustrator and saved as SVG. [Used with kind permission of Keith Andrews.]

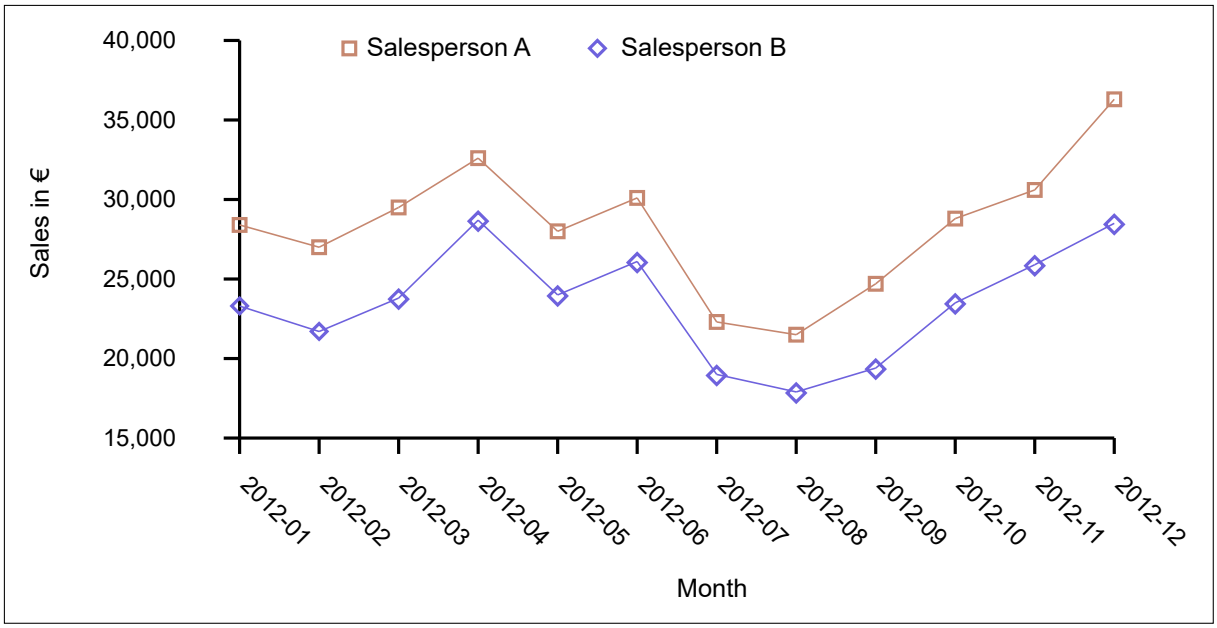

Figure 1.2: A hand-crafted SVG line chart created from the dataset. The full code can be found in Listing A.1.

1.4 Using Plain SVG

SVG, short form for Scalable Vector Graphics, is a markup language based on XML [W3C 2010a]. Most modern web browsers support SVG 1.1 [W3C 2011]. Creating charts or graphics using plain SVG is straightforward and does not require knowledge of a programming language like JavaScript. However, there are drawbacks. SVG allows scalability, but many elements in the same chart slow down the computation. Another drawback is, that with plain SVG, all the positioning of the elements must be done by hand, which means, that after changes in the dataset, every position has to be recalculated by hand, which can be really time consuming. The W3C SVG working group is currently developing a new SVG specification, SVG 2, which should be more feature-rich and at the same time get rid of deprecated features [W3C 2018]. Figure 1.2 shows the line chart hand-crafted in plain SVG. The full code can be found in Listing A.1, but will not be further commented, as the focus of this survey paper lies on the three built-in graphics technologies and the three drawing libraries.

Chapter 2

Built-In Graphics Technologies

There are three built-in ways to draw graphics programatically in the web browser, without any additional libraries. Here, they are refered to as SVG-DOM, Canvas2D, and WebGL.

2.1 SVG-DOM

After using plain SVG for one time, users might ask themselves if there is a direct way to draw an SVG with no libraries, but with being able to bypass the manual data calculation. In this section, the creation of SVG graphics by manipulating the DOM with JavaScript is presented. All the capabilities of JavaScript like variables and loops can be used to create and manipulate an SVG chart, which means that we can – if implemented correctly – update the dataset and the chart updates seamlessly. A tutorial to this method can be found at [Whitney 2013]. Figure 2.1 shows the chart created with the presented built in technology.

2.1.1 SVG-DOM Workflow

When creating charts with this method, only pure JavaScript has to be used. When adding SVG elements like lines or text to the DOM, the method createElementNS() is used, which specifies a namespace-URI. To set attributes, the function .setAttribute() is called. Since JavaScript allows the user to create elements with loops, most of the drawing in the following code snippets has been done by using them.

2.1.2 SVG-DOM Code

In this subsection, the code of SVG-DOM example is presented by explaining code snippets. The full code can be found in Listing A.2.

2.1.2.1 Data Preprocessing

Before starting with the implementation, we set the basic parameters. The chartData array contains the data. ViewBox and margin can be set to desired values. The size of the grid (size) is calculated by substracting the margin from the viewBox, the steps are calculated by dividing the size by the corresponding number of datapoints. To calculate the coordinates of the values for the salespeople, we normalize the values to a new range and substract them from the height of the grid, as the coordinate origin is top left. Listing 2.1 shows the code for this procedure.

2.1.2.2 Initialisation

To create a chart with SVG-DOM, first a HTML element is created in the body, and retrieved with document.getElementById(). The custom generateChart() function, which is presented in the code snippet below, creates the SVG element, sets the viewBox to the specified height and width and calls respective functions, as shown in Listing 2.2. The last point is mainly for code readability.

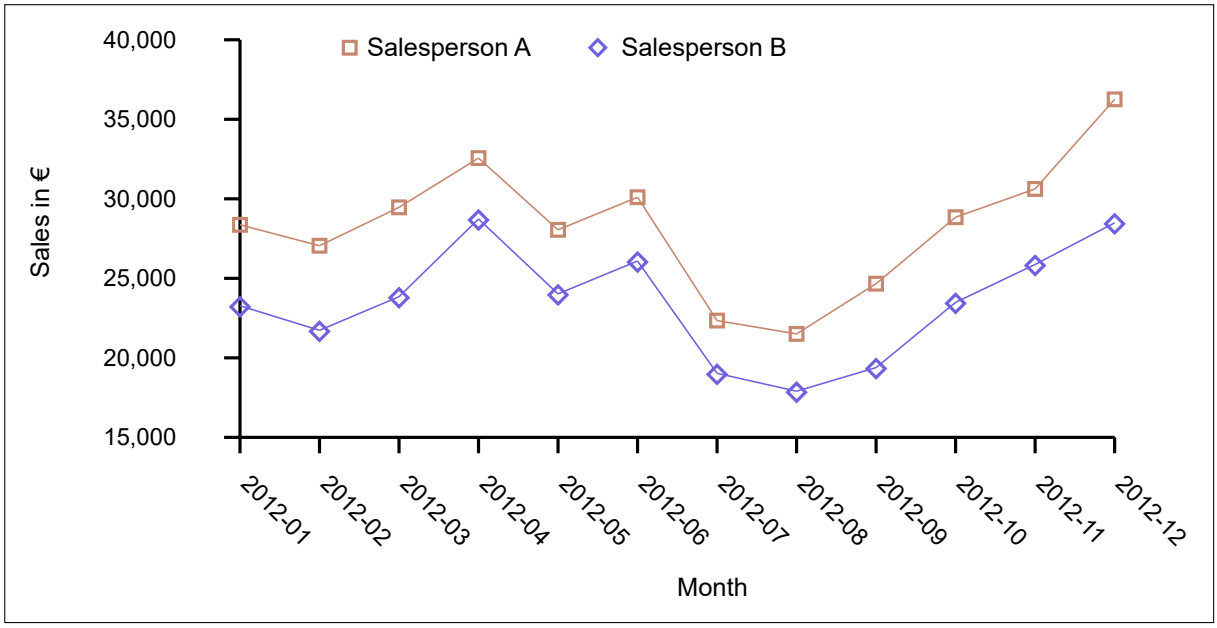

Figure 2.1: The chart created by the SVG-DOM code. The full code can be found in Listing A.2.

```
//Data array for computation
const chartData = [
  ["2012-01", 28366, 23274], ["2012-02", 27050, 21732],
  ["2012-03", 29463, 23845], ["2012-04", 32561, 28732],
  ["2012-05", 28050, 24023], ["2012-06", 30100, 26089],
  ["2012-07", 22343, 19026], ["2012-08", 21506, 17903],
  ["2012-09", 24664, 19387], ["2012-10", 28842, 23490],
  ["2012-11", 30621, 25873], ["2012-12", 36254, 28490]]

//Setting margin, size for later computation of coordinates
var view_box = { width: 950, height: 500 };
var margin = 100;
var size = { width: view_box.width - margin * 2,
  height: view_box.height - margin * 2 };

//Calculation of distance between points on x and y-axis
const x_step = size.width / (chartData.length - 1);
const y_step = size.height / 5;

//Function calculate the graph specific values for the money amounts
function calc_y_data(y) {
  return margin + (size.height - (((y - 15000) / 25000) * size.height));
}
```

Listing 2.1: SVG-DOM: Data preprocessing.

```
const generateChart = (data) => {
  //initialize the svg
  const svg = document.createElementNS('http://www.w3.org/2000/svg', 'svg');
  svg.setAttribute("viewBox", "0 0 " + view_box.width + " " + view_box.height + "");
  generateGrid(svg, data);
  generateLines(svg, data);
  generateRects(svg, data);
  generateAdditionalCaptions(svg);
  svg_chart.appendChild(svg);
}
```

Listing 2.2: SVG-DOM: initialisation.

```
const x_grid = document.createElementNS('http://www.w3.org/2000/svg', 'line');
x_grid.setAttribute("x1", margin);
x_grid.setAttribute("x2", size.width + margin);
x_grid.setAttribute("y1", size.height + margin);
x_grid.setAttribute("y2", size.height + margin);
x_grid.setAttribute("stroke", "black");
x_grid.setAttribute("stroke-width", "2");
chart.appendChild(x_grid);
```

Listing 2.3: SVG-DOM: Creation of the x-axis.

2.1.2.3 Axes

The creation of the x-axis is demonstrated. Since the SVG coordinate origin is top left, and every line has two coordinates, the coordinates of the "traditional" origin are (margin, size.height + margin), and the coordinates of the end of the x-axis are (size.width + margin, size.height + margin). After setting the attributes, the newly created SVG line is added to the SVG chart, as shown in Listing 2.3. The creation of the y-axis happens similarly, with different coordinates for the end of the y-axis (margin, margin).

2.1.2.4 Axis Markings and Captions

Axes markings and captions are created with the same loop, as their positioning is similar. Again, a line is created and set to the specified position. The same happens with the text, by creating an element with document.createElementNS() and setting text-specific attributes like font-family or font-size. The loop loops over an index for retrieving the captions and increases the x parameter by the pre-calculated value x_step. The code is shown in Listing 2.4. To rotate the text by 45 degrees, a transformation attribute is set like in plain SVG. The y-axis works similarly, but with fewer steps and by using y_step, which is also calculated in the preprocessing part.

2.1.2.5 Lines

To generate the two data lines, again a loop is used, which loops over the index for retrieving the data and increases the x-value by the precomputed x_step, as shown in Listing 2.5. To get the proper coordinates for the values from the dataset, calc_y_data() is used, which adjusts the values to the new scale.

```
var x = margin;
var index = 0;

while(index < data.length){
  const x_marking = document.createElementNS('http://www.w3.org/2000/svg', 'line');
  const x_caption = document.createElementNS('http://www.w3.org/2000/svg', 'text');

  x_marking.setAttribute("x1", x);
  x_marking.setAttribute("x2", x);
  x_marking.setAttribute("y1", size.height + margin);
  x_marking.setAttribute("y2", size.height + margin + 10);
  x_marking.setAttribute("stroke", "black");
  x_marking.setAttribute("stroke-width", "2");

  x_caption.setAttribute("fill", "black");
  x_caption.setAttribute("font-family", "Arial, sans-serif");
  x_caption.setAttribute("font-size", "15");
  x_caption.setAttribute("text-anchor", "start");
  x_caption.setAttribute("transform", "translate(" + x + "," +
    (size.height + margin + 30) + ") rotate(45)");
  x_caption.append(data[index][0]);

  chart.appendChild(x_caption);
  chart.appendChild(x_marking);
  x = x + x_step;
  index++;
}
```

Listing 2.4: SVG-DOM: Axis ticks and tick labels.

2.1.2.6 Markers

To draw rectangles as markers for the data points, rects are created with document.createElementNS(), an example can be found in Listing 2.6 The setting of the attributes is similar to the procedure for lines, with the difference that rects need a width and a height, and only one coordinate. For Salesman B, the rects are rotated by 45 degrees. The positions are again retrieved with calc_y_data() generated by using a loop over an index and x_step.

2.1.2.7 Axis Titles and Legend

The axis titles and the legend are also positioned using margin and x_step, but also fine-tuned by hand. This is a trial-and-error part of graph creation with SVG-DOM, but it has to be made sure that the elements are not accidentally drawn outside of the viewBox. The creation of the text and rectangles as well as the attribute setting is similar to the steps before, as shown in Listing 2.7. The code snippet shows, how the legend for Salesperson A with the corresponding rectangle is created and positioned.

```
var index = 0;
var x = margin;

while(index < data.length-1){
  const lineA = document.createElementNS('http://www.w3.org/2000/svg', 'line');
  const lineB = document.createElementNS('http://www.w3.org/2000/svg', 'line');

  lineA.setAttribute("stroke", "#c6876f");
  lineA.setAttribute("stroke-width", "1");
  lineA.setAttribute("x1", x);
  lineA.setAttribute("x2", x + x_step);
  lineA.setAttribute("y1", calc_y_data(data[index][1]));
  lineA.setAttribute("y2", calc_y_data(data[index+1][1]));

  lineB.setAttribute("stroke", "#6e62dd");
  lineB.setAttribute("stroke-width", "1");
  lineB.setAttribute("x1", x);
  lineB.setAttribute("x2", x + x_step);
  lineB.setAttribute("y1", calc_y_data(data[index][2]));
  lineB.setAttribute("y2", calc_y_data(data[index+1][2]));

  chart.appendChild(lineA);
  chart.appendChild(lineB);
  index++;
  x = x + x_step;
}
```

Listing 2.5: SVG-DOM: Lines representing the data.

```
var index = 0;
var x = margin;

while(index < data.length){
  const rectA = document.createElementNS('http://www.w3.org/2000/svg', 'rect');
  const rectB = document.createElementNS('http://www.w3.org/2000/svg', 'rect');

  rectA.setAttribute("stroke", "#c6876f");
  rectA.setAttribute("stroke-width", "2");
  rectA.setAttribute("fill", "none");
  rectA.setAttribute("x", x - 4);
  rectA.setAttribute("y", calc_y_data(data[index][1]) - 4);
  rectA.setAttribute("width", "8");
  rectA.setAttribute("height", "8");

  rectB.setAttribute("stroke", "#6e62dd");
  rectB.setAttribute("stroke-width", "2");
  rectB.setAttribute("fill", "none");
  rectB.setAttribute("transform", "translate(" + x + "," + (calc_y_data(data[index
      ][2]) - 5) + ") rotate(45)");
  rectB.setAttribute("width", "8");
  rectB.setAttribute("height", "8");

  chart.appendChild(rectA);
  chart.appendChild(rectB);
  index++;
  x = x + x_step;
}
```

Listing 2.6: SVG-DOM: Drawing rectangles as data markers.

```
const rectA = document.createElementNS('http://www.w3.org/2000/svg', 'rect');
const spA = document.createElementNS('http://www.w3.org/2000/svg', 'text');

rectA.setAttribute("stroke", "#c6876f");
rectA.setAttribute("stroke-width", "2");
rectA.setAttribute("fill", "none");
rectA.setAttribute("x", margin + x_step * 2);
rectA.setAttribute("y", margin - 8);
rectA.setAttribute("width", "8");
rectA.setAttribute("height", "8");

spA.setAttribute("fill", "black");
spA.setAttribute("font-family", "Arial, sans-serif");
spA.setAttribute("font-size", "16");
spA.setAttribute("x", margin + x_step * 2 + 20);
spA.setAttribute("y", margin);
spA.append("Salesperson A");

chart.appendChild(rectA);
chart.appendChild(spA);
```

Listing 2.7: SVG-DOM: Creating axis titles and the legend.

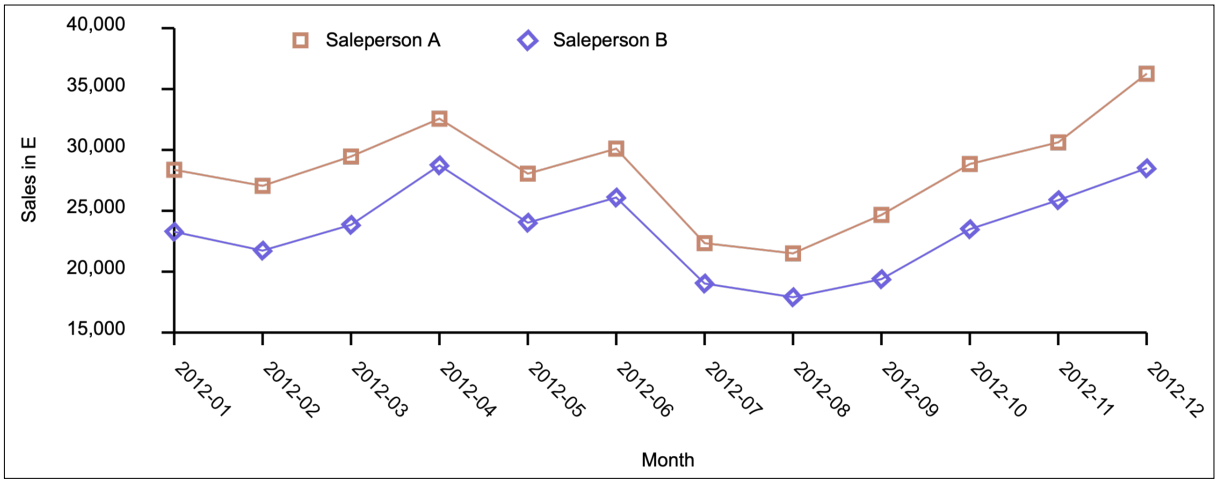

Figure 2.2: The line chart created by the Canvas2D code. The full code can be found in Listing A.3.

2.2 Canvas2D

The HTML <canvas> element was introduced in HTML5 as a tool to draw raster graphics in the web browser. It comes with JavaScript APIs to draw graphics using a resolution-depend bitmap, and it can be used in various "context modes" [WHATWG 2022]. In this section, the word "Canvas2D" is used to refer to the practice of drawing graphics through the "2d" context of a canvas element, as shown in Listings 2.8 and 2.9. The chart created by the Canvas2D code is shown in Figure 2.2. This built-in graphics technology is particularly suitable to situations in which:

- some graphics need to be rendered on-the-fly, maybe based on input or interaction by the user, or in general in such situations in which the graphic element cannot be computed a-priori and rendered on the screen as an image;
- the graphics contain a large number of different elements to be rendered, and the computation happens one time only and there is no need to maintain models of the graphic elements and use them to make them scale with the size of the viewport or with the zoom level;
- it is acceptable for the graphics not to be vector-based (so automatically scalable).

2.2.1 Canvas2D Workflow

To get started, a canvas element needs to be created, either by plain HTML code or through JavaScript, then its "2d" context needs to be extracted in order to use the interface it exposes. This interface offers methods to draw lines like with a pen, fill shapes and text, define color gradients, apply transformations to the graphic elements, and much more.

2.2.2 Canvas2D Code

In this subsection, the code of Canvas2D is presented by explaining code snippets. The full code can be found in Listing A.3.

2.2.2.1 Initialisation

One very fundamental concept to use the Canvas2D scripting environment is the way dimensions of the graphics are defined: when rendering with Canvas2D, we are working with a "resolution-dependent bitmap" [W3C 2010b] , so when instantiating a Canvas2D element, the possibility is given to set two pairs of width and height properties: an "inner" pair given as HTML properties, and an "outer" pair through

```
1 <canvas id="chart" width="2700" height="1300"></canvas>
```

Listing 2.8: Canvas2D: Creation of the canvas element.

```
1 var canvas = document.getElementById("chart")
2 var ctx = canvas.getContext("2d");
```

Listing 2.9: Canvas2D: Retrieving the "2d" context of the canvas element.

the style properties (CSS). The "inner" pair gives the resolution of the bitmap in CSS pixels [W3C 2010b] and also defines a set of coordinates in a similar manner to what is done with the SVG viewBox, in the sense that graphic objects will be positioned by using coordinates of the form (x,y), where $x \in [0, widht]$, $y \in [0, height]$. The "outer" pair tells the browser which actual dimensions should be used to render the bitmap, these can be e.g. relative to the sizes of other elements in the document and do not affect the functioning of the scripting APIs in any ways so they will not be discussed in the following Listings.

The Canvas2D element is created as shown in Listing 2.8. The graphics are then rendered by using the methods offered by the JavaScript interface of a "2d" context object, instantiated like specified in Listing 2.9, and positions are specified by means of the set of coordinates defined by the "inner" dimensions specified when creating the canvas element.

In the default configuration, the (0,0) point is positioned in top-left corner of the bitmap, and the vertical coordinates span from top to bottom. In the context of this work, the coordinates system is re-parameterised by translating the (0,0) point in the lower-left corner of the bitmap, and flipping the growing direction of vertical coordinates in such a way that a point with coordinates (x,y) is rendered like if it was positioned by means of a Cartesian coordinate system. The code for this procedure is shown in Listing 2.10 and a diagram of the procedure is shown in Figure 2.3. This way, the coordinates can be referred to in a more natural way.

2.2.2.2 Data Preprocessing

Before starting with the implementation, we set the basic parameters. The chartData array contains the data. ViewBox and margin can be set to desired values. The size of the grid (size) is calculated by substracting the margin from the viewBox, the steps are calculated by dividing the size by the corresponding number of datapoints. This time, to calculate the coordinates of the values for the salespeople, we just need to normalize the values to a new range (line 19). The procedure is easier because of coordinate reparameterisation (Listing 2.10). Listing 2.11 reports the code for this procedure.

2.2.2.3 Axes

Canvas2D does not provide any primitive mechanism to render the axes of a chart, so they must be rendered manually using standard drawing procedures. The code for the creation of the axes is shown in Listing 2.12, where functions defined in Section 2.16 are used.

```
1 ctx.translate(margin, margin+size.height);
2 ctx.scale(1, -1);
```

Listing 2.10: Canvas2D: Re-parameterisation of the coordinate system.

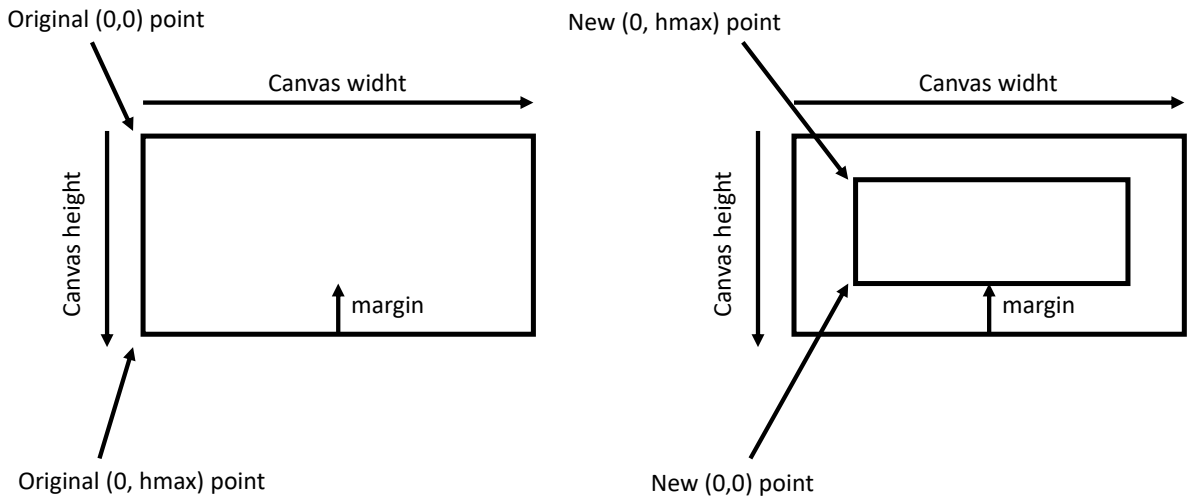

Figure 2.3: Canvas2D: Reparameterisation of the coordinate system.

2.2.2.4 Axis Ticks, Tick Labels, and Titles

Canvas2D does not provide any primitive mechanism to render axis ticks, tick labels, and axis titles associated with a chart, so they must be rendered manually using standard drawing procedures. The code for the creation of axis ticks, tick labels, and axis titles is shown in Listing 2.13.

2.2.2.5 Text

The standard way to create text in Canvas2D is to use the text primitive, as shown in Listing 2.14. Here the discussion is articulated by presenting a function which prints rotated text in Canvas2D. The code for the function is shown in Listing 2.15. This function rotates the text around its center by temporarily translating the origin of the coordinates at the center of the text, if the temporary translation is not done, Canvas2D rotates the text around the previously configured (0,0) point.

2.2.2.6 Lines and Points

The standard procedure to draw lines in Canvas2D is presented in Listing 2.16. The function presented in Listing 2.16 can be used to render the actual plots like described in Listing 2.17, which also makes use of the function in Listing 2.18 to plot markers for data points.

2.2.2.7 Legend

Canvas2D does not provide any primitive mechanism to render legends associated with a chart, so they must be rendered manually using standard drawing procedures. Code for the creation of the legend is shown in Listing 2.19.

```
  //data array
  const chartData = [
    ["2012-01", 28366, 23274], ["2012-02", 27050, 21732],
    ["2012-03", 29463, 23845], ["2012-04", 32561, 28732],
    ["2012-05", 28050, 24023], ["2012-06", 30100, 26089],
    ["2012-07", 22343, 19026], ["2012-08", 21506, 17903],
    ["2012-09", 24664, 19387], ["2012-10", 28842, 23490],
    ["2012-11", 30621, 25873], ["2012-12", 36254, 28490]]

  //Setting margin, size for later computation of corrdinates for drawing
  const view_box = { width: canvas.width, height: canvas.height };
  const margin = 350;
  const size = { width: view_box.width - margin * 2,
    height: view_box.height - margin * 2 };

  //Calculation of distance between points on x and y-axis
  const x_step = size.width / (chartData.length - 1);
  const y_step = size.height / 5;

  //Function to gradually change the distance between the points on the y axis.
  //Differently from the other files, here we translate the 0,0 to fit the margin
  // so there is no need to include it in the calculations
  function calc_y_data(y) {
    return ((y-15000) / 25000) * size.height ;
  }
```

Listing 2.11: Canvas2D: Data preprocessing.

```
// y Axis
drawLine(0, 0, 0, size.height);
// x Axis
drawLine(0, 0, size.width, 0);
```

Listing 2.12: Canvas2D: drawing axes in Canvas2D. Code from section 2.2.2.6 is used.

```
// ticks on y-axis
for (let i = 0; i < y_ax_marks.length; i ++) {
  drawLine(0, i*y_step, -0.01 * size.width, i*y_step);
}
// ticks on x-axis
for (let i = 0; i < chartData.length; i ++) {
  drawLine(i*x_step, 0, i*x_step, -0.01 * size.width)
}

// tick labels on y-axis
ctx.font = 0.015 *size.width + "px Arial";
ctx.scale(1, -1);
for (let i = 0; i < y_ax_marks.length; i ++) {
  draw_rot_text((-0.085 *size.width), -i*y_step + 5, 0 , y_ax_marks[i]);
}
// tick lables on x-axis
for (let i = 0; i < chartData.length; i ++) {
  draw_rot_text(i*x_step, 0.1 *size.height, 45 * Math.PI / 180, chartData[i][0]);
}

// axis titles
draw_rot_text((-0.11 *size.width), (-0.5 *size.height),
  -90 * Math.PI / 180, "Sales in €");
draw_rot_text(0.5 *size.width, (0.4 *size.height), 0 * Math.PI / 180, "Month");
ctx.scale(1, -1);
```

Listing 2.13: Canvas2D: Drawing axis ticks, tick lables, and titles. Code from Sections 2.2.2.6 and 2.2.2.5 is used.

```
ctx.fillText(text, x, y);
```

Listing 2.14: Canvas2D: fillText function.

```
/** draw a rotated piece of text
* x, y: coordinates where text is positioned
* a:    angle of rotation
* text
*/
function draw_rot_text(x, y, a, text) {
  ctx.translate(x, y);
  ctx.rotate(a);
  ctx.fillText(text, 0, 0);
  ctx.rotate(-a);
  ctx.translate(-x, -y);
}
```

Listing 2.15: Canvas2D: Drawing rotated text.

```
/**
* draw a line from (x0,y0) to (x1,y1)
*/
function drawLine(x0, y0, x1, y1) {
  ctx.beginPath();
  ctx.moveTo(x0, y0);
  ctx.lineTo(x1, y1);
  ctx.stroke();
}
```

Listing 2.16: Canvas2D: Drawing lines.

```
/** plot a line given array of datapoints
* arr:   array of datapoints
* sq_s:  square size (markers)
* a:     angle of rotation for the squares (markers)
*/
function plot(j, sq_s, a) {
  ctx.beginPath();
  ctx.moveTo(0, calc_y_data( chartData[0][j]));
  for (let i = 0; i < chartData.length; i ++) {
    ctx.lineTo(i*x_step,  calc_y_data( chartData[i][j]));
  }
  ctx.stroke();
  for (let i = 0; i < chartData.length; i ++) {
    draw_rot_square(i*x_step, calc_y_data( chartData[i][j]), sq_s, a);
  }
}
```

Listing 2.17: Canvas2D: Plotting a line with markers.

```
/** draw a rotated square
* x, y center point
* s:   size of the square
* a:   angle of rotation
*/
function draw_rot_square(x, y, s, a) {
  ctx.beginPath();
  ctx.translate(x, y);
  ctx.rotate(a);
  ctx.rect(0 - (s / 2), 0 - (s / 2), s, s);
  ctx.rotate(-a);
  ctx.translate(-x, -y);
  ctx.stroke();
}
```

Listing 2.18: Canvas2D: Drawing rotated squares.

```
draw_rot_square(
  0.1 *size.width, size.height, 0.03 *size.height, 0
);
ctx.fillText("Saleperson A", 0.12 *size.width, -0.98*size.height);

draw_rot_square(
  0.28 *size.width, size.height, 0.03 *size.height, 40
);
ctx.fillText("Saleperson B", 0.3 *size.width, -0.98*size.height);
```

Listing 2.19: Canvas2D: Drawing the legend.

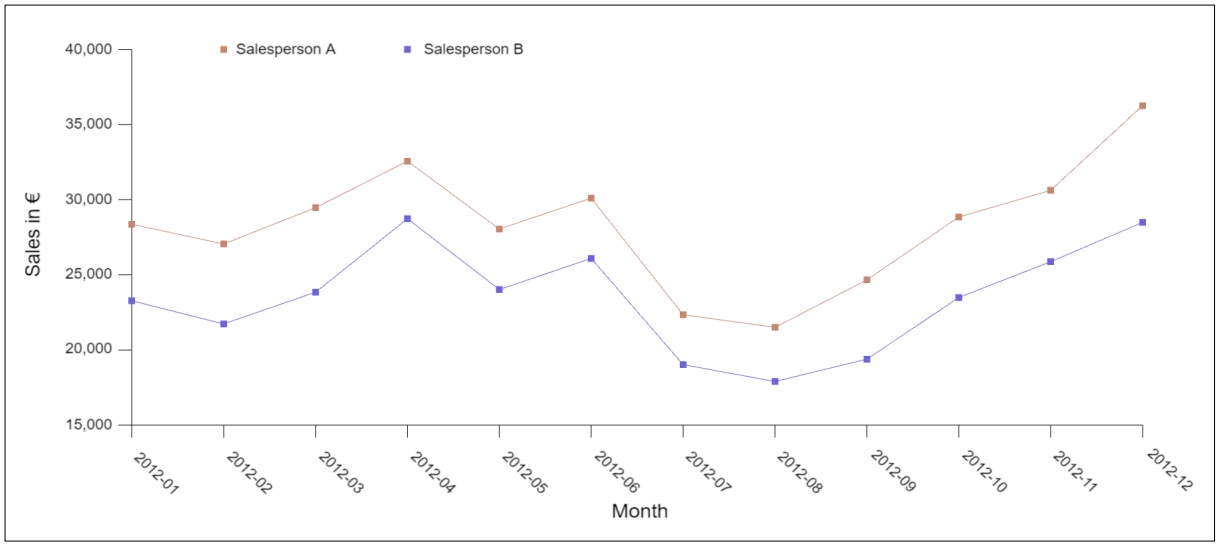

Figure 2.4: WebGL: Line chart. The full code can be found in Listing A.4.

2.3 WebGL

WebGL is a very well-known JavaScript API for rendering 2D and 3D graphics in the web browser [Khronos 2022]. To check whether a browser supports WebGL, one can simply look at [WebGL 2022]. WebGL itself is based upon OpenGL with the inclusion of shaders written in the GLSL language [OpenGL 2022]. Shaders can either be written inline (which later can be seen in the example) or created as external shader files. The most important feature included by WebGL is its hardware acceleration. It is much faster compared to other libraries in this area.

2.3.1 WebGL Text API Problem

With all the benefits and hardware acceleration one can achieve from using WebGL, one major drawback has to be taken into consideration. WebGL itself does not have its own text API. Meaning, one can not simply create text in WebGL graphics. Although workarounds already exist, most of them are either very complicated or plain not worth using them. These workarounds include:

- Bitmap: Create a bitmap for every letter in the alphabet. The problem here is, that if one wants to use different font sizes, every letter has to be created in different sizes in the bitmap, leading to a shear amount of rendered letters in the bitmap.
- Vector Font: Another solution would be to draw letters by rendering lines, more commonly known as a “vector font”.
- Text Canvas: The most common solution, also used in this coding example is by using a second canvas. The second canvas has the same size as the WebGL canvas and is overlayed on top of the WebGL canvas. In the Text canvas, the user can use the generic HTML Canvas2D text API to create text and render onto the screen.

An important step to make the second overlay canvas work with the WebGL canvas is to set the width and height of both to the same amount, as shown in Listing 2.20.

2.3.2 WebGL Workflow

Before starting with WebGL, it is highly recommended to read some tutorials on the internet to get a basic thought about what WebGL is all about and how it works. It’s initial steps are highly different to the other two basic languages especially in respect to the shaders and the buffer. A good way to start would be with

```
<canvas id="chart" width="1500" height="800"></canvas>
<canvas id="text" width="1500" height="800"></canvas>
```

Listing 2.20: WebGL: WebGL and overlay text canvases must be the same size.

Greggman [2022]. For creating more advanced charts or objects in WebGL, a better understanding of GLSL and OpenGL is definitely needed. Once the initial setup for WebGL itself has been done, the user can draw everything using Float32Arrays and the drawArrays() function supported by WebGL to render different objects in the browser.

2.3.3 WebGL Code

In this subsection, the WebGL code for the example line chart is presented by explaining code snippets. The full code can be found in Listing A.4.

2.3.3.1 Data Preprocessing

Before starting with the implementation, we set the basic parameters. The chartData array contains the data. ViewBox and margin can be set to desired values. The size of the grid (size) is calculated by substracting the margin from the viewBox, the steps are calculated by dividing the size by the corresponding number of datapoints. To calculate the coordinates of the values for the salespeople, we normalize the values to a new range and add them to the margin, as the coordinate origin is the bottom left. All of the above can bee seen in Listing 2.21.

2.3.3.2 Initialisation

At the beginning, some basic steps have to be done, before one can simply start drawing with WebGL. The basics first steps include setting the WebGL context to a <canvas> element and clearing the canvas, as can be seen in Listing 2.22.

After the canvas has been setup, the fragment shader and vertex shader have to be initialised and created inside the WebGL context, as shown in Listing 2.23. The shaders can either be written as external files or be written inline as in this work. The vertex shader is assigned to an object, its GLSL source is uploaded and then compiled. The code inside the vertex shader contains a variable position as input. The position is then assigned to the "gl_Position" to set the position of the point and "gl_PointSize" assigns the size of the point.

The same steps have to be done for the fragment shader, the only difference lies inside the GLSL fragment shader code. Here the input value consists of a color, which is assigned to the "gl_FragColor" to set the color of the point seen in Listing 2.24.

After both shaders have been created and compiled, a program is created where both shaders get linked to. This program is then added to the WebGL context, as shown in Listing 2.25. Afterwards, WebGL is instructed to use the linked program containing both shaders and the universal color for the program is set to black, as shown in Listing 2.26.

One important function used in all methods is the createBindBuffer() function. It creates a vertexBuffer, sets the buffer as the current one to be worked on in gl.bindBuffer(), and adds the data from the pointarray into the buffer using gl.bufferData(). Afterwards, WebGL needs to know how to get data out of the buffer and give it to the vertex shader's attributes. To assign the location, WebGL is asked for the location of

```
//Data array for computation
const chartData = [["2012-01",28366,23274],["2012-02",27050,21732],
  ["2012-03",29463,23845],["2012-04",32561,28732],
  ["2012-05",28050,24023],["2012-06",30100,26089],
  ["2012-07",22343,19026],["2012-08",21506,17903],
  ["2012-09",24664,19387],["2012-10",28842,23490],
  ["2012-11",30621,25873],["2012-12",36254,28490]]

//Setting margin, size, and ratio for later computation of coordinates
let view_box = {width: 950, height: 500};
var margin = 100;
var size = {width: view_box.width - margin * 2,
  height: view_box.height - margin * 2};

//Calculation of distance between points on x and y-axis
let x_step = size.width / (chartData.length - 1);
let y_step = size.height/5;

//Function calculate the graph specific values for the money amounts
function calc_y_data(y) {
  return margin + (((y-15000) / 25000) * size.height);
}
```

Listing 2.21: WebGL: Data preprocessing.

```
let gl = canvas.getContext('webgl');
context.clearRect(0, 0, context.canvas.width, context.canvas.height)
gl.clearColor(1,1,1,1);
gl.clear(gl.COLOR_BUFFER_BIT);
```

Listing 2.22: WebGL: Setting the basic context.

```
//Vertex Shader initialization
let vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, [
  'attribute vec2 position;',
  'void main() {',
  'gl_Position = vec4(position, 0.0, 1.0);',
  'gl_PointSize = 10.0;',
  '}'
].join('\n'));
gl.compileShader(vertexShader);
```

Listing 2.23: WebGL: The vertex shader.

```
//Fragment Shader initialization
let fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, [
  'precision highp float;',
  'uniform vec4 color;',
  'void main() {',
  'gl_FragColor = color;',
  '}'
].join('\n'));
gl.compileShader(fragmentShader);
```

Listing 2.24: WebGL: The fragment shader.

```
//Creation of program and attachment of shader
let program = gl.createProgram();
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
gl.linkProgram(program);
```

Listing 2.25: WebGL: Shader and program creation.

```
//Setting universal used colour to black for the program
gl.useProgram(program)
program.color = gl.getUniformLocation(program, 'color');
gl.uniform4fv(program.color, [0.0,0,0,1.0]);
```

Listing 2.26: WebGL: Setting the universal colour.

```
function createBindBuffer(array){
  let vertexBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, array, gl.STATIC_DRAW);

  program.position = gl.getAttribLocation(program, 'position');
  gl.enableVertexAttribArray(program.position);
  gl.vertexAttribPointer(program.position, 2, gl.FLOAT, false, 0, 0);
}
```

Listing 2.27: WebGL: Create and binding a buffer.

```
gl.drawArrays(gl.POINTS, 0, 1);
gl.drawArrays(gl.LINES, 0, n);
gl.drawArrays(gl.LINE_STRIP, 0, n);
```

Listing 2.28: WebGL: drawArrays function with the modes used for this example.

```
function normalize(value){
  return value = ((value/view_box.width) * 2) - 1;
}
function normalizeheight(value){
  return value = ((value/view_box.height) * 2) - 1;
}
```

Listing 2.29: WebGL: Normalising coordinates.

the attributes using getAttribLocation(). Finally, WebGL is told that the user supplies data from a buffer, and a pointer is set to these vertex attributes. The full function can bee seen in Listing 2.27.

The second important function which can be seen in almost every WebGL code example for chart creation is the drawArrays(). It is used to render the points inside the given arrays. It takes three arguments:

- mode:
  - gl.Points: To draw a single point.
  - gl.LINE_STRIP: To draw straight line to the next point.
  - gl.LINES: To draw a line between a pair of points.
- first: The starting index in the array of given points.
- count: How many points should be rendered.

Mode supports more possible input arguments, but only the ones used in this work are listed in Listing 2.28.

Since the WebGL clip space is always between -1 and 1, all the points used for the WebGL graph are normalised using two functions: one for the width and one for the height, as shown in Listing 2.29.

2.3.3.3 Axes

Axes are created by using two points for each axes and connecting them via a line. All points for the axes and axes tick marks are saved in a Float32Array and passed on to the createBindBuffer() function. The variable n always denotes how many points are stored in the array. Creation of the axes can be seen in Listing 2.30.

```
let vertices = new Float32Array([
  //x-axis
  normalize(margin), normalizeheight(margin),
  normalize(margin + size.width), normalizeheight(margin),

  //y-axis
  normalize(margin), normalizeheight(margin),
  normalize(margin), normalizeheight(margin + size.height),
  ....
]);
let n=68;
createBindBuffer(vertices);
return n;
```

Listing 2.30: WebGL: Creating the axes.

```
//Pointmarks x-axis
normalize(margin), normalizeheight(margin),
normalize(margin), normalizeheight(margin - 10),
....
normalize(margin + x_step*11), normalizeheight(margin),
normalize(margin + x_step*11), normalizeheight(margin - 10),

//Pointmarks y-axis
normalize(margin), normalizeheight(margin),
normalize(margin - 10), normalizeheight(margin),
....
normalize(margin), normalizeheight(margin + y_step*5),
normalize(margin - 10), normalizeheight(margin + y_step*5)
```

Listing 2.31: WebGL: Creating axis tick marks.

2.3.3.4 Axis Tick Marks

Tick marks on the axes are created in the same array as the axes itself, as shown in Listing 2.31. Each point is duplicated with a slight offset of -10 to create the tick marks on the x and y axes. All tick marks have been individually created by points which makes the array quite large.

2.3.3.5 Axes Tick Labels

Tick labels are created on the overlayed text canvas using the generic fillText() function from Canvas2D. Each String has to be created on its own at a certain coordinate on the canvas. Since we use a different canvas here, the values do not have be normalised since it is not in the WebGL context. Markings on the x-axis have to be rotated and translated using the context.rotate() function provided by Canvas2D seen in Listing 2.32 and Listing 2.33.

```
context.rotate(45 * Math.PI/180);
context.font = "18px sans-serif";
context.fillText("2012-01", 584, 358);
....
context.fillText("2012-12", 1420, -485);
```

Listing 2.32: WebGL: Creating x-axis tick labels.

```
context.font = "18px sans-serif";
context.fillText("15,000", 80, 630);
....
context.fillText("40,000", 80, 162);
```

Listing 2.33: WebGL: Creating y-axis tick labels.

2.3.3.6 Lines

Lines are created using an array which contains all data points used in the chart for each individual salesperson. The lines for Salesperson A are shown in Listing 2.34, and for Salesperson B in Listing 2.35. They are created from their individual array using the drawArrays() function with the argument gl.LINE_STRIP argument. Once again, all points are saved in a Float32Array to be later used in the createBindBuffer() function.

2.3.3.7 Markers for Data Points

Markers for the individual data points are created using the same array as for the lines, but changing the mode of the drawArrays() function to gl.POINTS, as shown in Listing 2.36.

2.3.3.8 Axis Titles

The axis titles "Month" and "Sales in €" are again created on the overlayed text canvas using the generic fillText() capability of Canvas2D. The code can be seen in Listing 2.37. The y-axis title "Sales in €" has to be rotated and translated into position. Additionally, the canvas context is saved before and restored later, so no later points are affected by the rotation and translation.

2.3.3.9 Legend

The legend for the chart is created in two different sections. The plotted markers next to the labels are part of the WebGL canvas, the text of the labels is part of the 2d canvas. Points are plotted using the drawArrays() function with the gl.POINTS mode and text is created using the fillText() function from Canvas2D. Marker creation can be seen in Listing 2.38 and the corresponding text labels in Listing 2.39.

```
function initSalesAPoints(){
  let vertices = new Float32Array([
    normalize(margin), normalizeheight(calc_y_data(chartData[0][1])),
    ....
    normalize(margin + x_step*11), normalizeheight(calc_y_data(chartData[11][1]))
  ]);
  let n = 12;
  createBindBuffer(vertices);
  return n;
}

function drawConnectedLinesSalesA(){
  let n = initSalesAPoints();
  gl.drawArrays(gl.LINE_STRIP, 0, n);
}
```

Listing 2.34: WebGL: Drawing the line for Salesperson A.

```
function initSalesBPoints(){
  let vertices = new Float32Array([
    normalize(margin), normalizeheight(calc_y_data(chartData[0][2])),
    ....
    normalize(margin + x_step*11), normalizeheight(calc_y_data(chartData[11][2]))
  ]);
  let n = 12;
  createBindBuffer(vertices);
  return n;
}

function drawConnectedLinesSalesB(){
  let n = initSalesBPoints();
  gl.drawArrays(gl.LINE_STRIP, 0, n);
}
```

Listing 2.35: WebGL: Drawing the line for Salesperson B.

```
function drawPointSalesA(){
  let n = initSalesAPoints();
  gl.drawArrays(gl.POINTS, 0, n);
}

function drawPointSalesB(){
  let n = initSalesBPoints();
  gl.drawArrays(gl.POINTS, 0, n);
}
```

Listing 2.36: WebGL: Drawing markers for data points.

```
// x-axis title
context.font = "24px sans-serif";
context.fillText("Month", 720, 740);

// y-axis title
context.save()
context.rotate(-Math.PI/2);
context.translate(-750,-400)
context.font = "24px sans-serif";
context.fillText("Sales in €", 310, 450);
context.restore();
```

Listing 2.37: WebGL: Drawing the axis titles.

```
function drawPointMarkerA(){
  let vertices = new Float32Array([
    normalize(margin+x_step), 0.6
  ]);
}

function drawPointMarkerB(){
  let vertices = new Float32Array([
    normalize(margin+x_step*3), 0.6
  ]);
}
```

Listing 2.38: WebGL: Drawing markers for the legend.

```
1 context.fillText("Salesperson A", 280, 162);
2 context.fillText("Salesperson B", 500, 162);
```

Listing 2.39: WebGL: Drawing text labels for the legend.

Chapter 3

Drawing Libraries

There are a number of JavaScript libraries intended to make drawing of graphics easier. This chapter looks at three of them: PixiJS, Two.js, and D3.

3.1 PixiJS

PixiJS is a 2D JavaScript rendering library for the web browser, which supports hardware acceleration where available [PixiJS 2022b]. For usage, the PixiJS library file has to be downloaded and referenced in JavaScript. Core features of PixiJS include:

- Very easy to understand and work with.
- Based on WebGL but falls back to Canvas2D when needed.
- No prior knowledge of WebGL needed.

The Text API problem of WebGL, which was mentioned in Chapter 2.3.1 is solved by using the Canvas API to draw text [PixiJS 2022a]. In our example, the graphics drawn by PixiJS unfortunately show a lot of aliasing. Trial-and-error, trying to remove this effect also did not offer the anticipated result of clean lines. Figure 3.1 shows the graph created with the presented drawing library.

3.1.1 PixiJS Workflow

A major advantage of PixiJS is its simple workflow: it is like drawing with a pencil. With graph.moveTo(), the user moves to a position and can draw a line to the next position with graph.lineTo(). When another color is needed, simply graph.lineStyle() is called. Text and objects like rectangles are positioned directly.

3.1.2 PixiJS Code

In this subsection, the code of PixiJS is presented by explaining code snippets. The full code can be found in Listing A.5.

3.1.2.1 Data Preprocessing

Before starting with the implementation, we set the basic parameters, as shown in Listing 3.1. The chartData array contains the data. ViewBox and margin can be set to desired values. The size of the grid (size) is calculated by substracting the margin from the viewBox, the steps are calculated by dividing the size by the corresponding number of datapoints. To calculate the coordinates of the values for the salespeople, we normalize the values to a new range and substract them from the height of the grid, as the coordinate origin is top left.

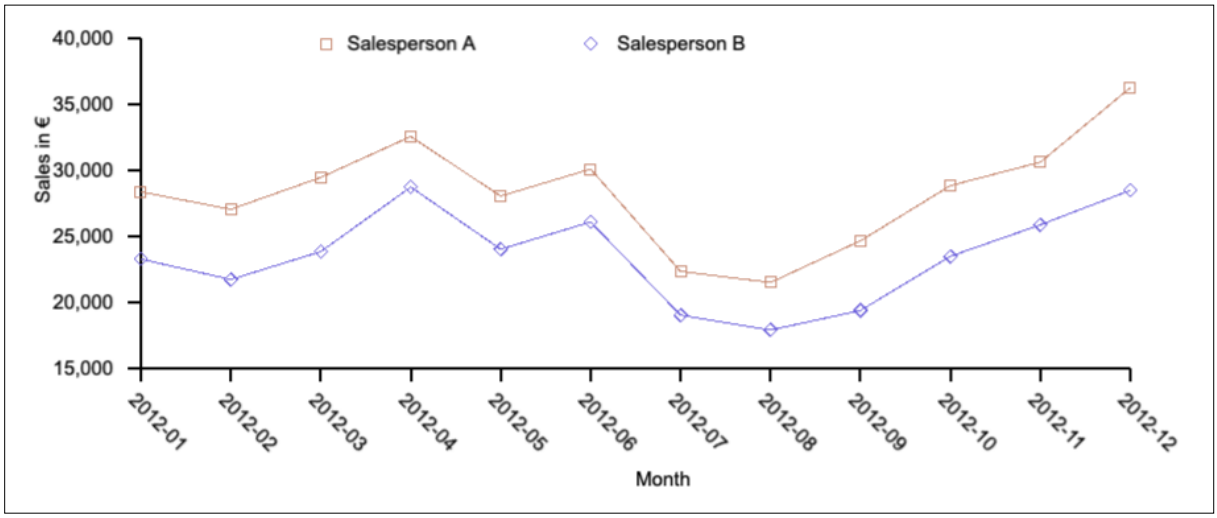

Figure 3.1: The chart created by the PixiJS code. The full code can be found in Listing A.5.

```
1  //Data array for computation
2  const chartData = [
3    ["2012-01",28366,23274],["2012-02",27050,21732],
4    ["2012-03",29463,23845],["2012-04",32561,28732],
5    ["2012-05",28050,24023],["2012-06",30100,26089],
6    ["2012-07",22343,19026],["2012-08",21506,17903],
7    ["2012-09",24664,19387],["2012-10",28842,23490],
8    ["2012-11",30621,25873],["2012-12",36254,28490]]
9
10 //Setting margin, size for later computation of corrdinates for drawing
11 var view_box = { width: 950, height: 500 };
12 var margin = 100;
13 var size = { width: view_box.width - margin * 2,
14   height: view_box.height - margin * 2 };
15
16 //Calculation of distance between points on x and y-axis
17 const x_step = size.width / (chartData.length - 1);
18 const y_step = size.height / 5;
19
20 //Function calculate the graph specific values for the money amounts
21 function calc_y_data(y) {
22   return margin + (size.height - (((y - 15000) / 25000) * size.height));
23 }
```

Listing 3.1: PixiJS: Data preprocessing.

```
let chart = new PIXI.Application({
  width: view_box.width,
  height: view_box.height,
  backgroundColor: 0xffffff
});
document.body.appendChild(chart.view);
let graph = new PIXI.Graphics();
```

Listing 3.2: PixiJS: Initialisation.

```
  graph.endFill();
  chart.stage.addChild(graph);
```

Listing 3.3: PixiJS: Wrapping up.

3.1.2.2 Initialisation

The initialisatiton of PixiJS is straightforward, well documented and – when sticking to the documentation – a short process. Three lines are enough to set everything up, as shown in Listing 3.2, and one can proceed to drawing. After drawing has finished, filling is stopped and the drawings are added to the stage, as shown in Listing 3.3.

After the drawing has finished, filling is stopped and the drawings are added to the stage, as shown in Listing 3.3.

3.1.2.3 Axes

A line thickness of 2 and a line colour black is selected. After that, we move to the top left corner and draw a line for the y-axis down to the bottom left corner of the grid. Then a horizontal line is drawn for the x-axis, as shown in Listing 3.4.

3.1.2.4 Axis Tick Marks and Tick Labels

We start in the bottom left corner of the grid, and loop over an index until the number of data points (minus one) is reached. A short line is drawn vertically, and a new text is initialised, containing the dates of the number values. Every generated text element is added to the stage with chart.stage.addChild(). After setting text attributes, the x value is increased by the predefined step, the position is updated and the loop can continue to run, as shown in Listing 3.5. The same procedure is used for calculating the y-axis tick marks and tick labels, with different text, different loop length, and different y increments.

3.1.2.5 Lines and Points

The line for a dataset and the point markers are drawn in the same loop. First, a thinner line style and a different colour is set. To get the y positions of the datp oints, the already mentioned calc_y_data function is used. To end up with the correct number of lines and rectangles, one line and rectangle have to be

```
graph.lineStyle(2, 0x000000);
graph.moveTo(margin, margin);
graph.lineTo(margin, margin + size.height);
graph.lineTo(margin + size.width, margin + size.height);
```

Listing 3.4: PixiJS: Drawing the axes.

```
graph.moveTo(margin, margin + size.height);
var x = margin;
var index = 0;
while (index < chartData.length) {
  graph.lineTo(x, 360);
  var text = new PIXI.Text(chartData[index][0],
    { fontFamily: 'Arial, sans-serif', fontSize: 15,
      fill: 0x000000, align: 'start' });
  text.x = x;
  text.y = margin + size.height + 15;
  text.angle = 45;
  chart.stage.addChild(text);
  x = x + x_step;
  graph.moveTo(x, margin + size.height);

  index++;
}
```

Listing 3.5: PixiJS: Creating axis tick marks and labels.

```
graph.lineStyle(1, 0xc6876f);
graph.moveTo(margin, calc_y_data(chartData[0][1]))
graph.drawRect(margin - 4, calc_y_data(chartData[0][1]) - 4, 8, 8)
var x = margin + x_step;
var index = 1;
while (index < chartData.length) {
  graph.lineTo(x, calc_y_data(chartData[index][1]));
  graph.drawRect(x - 4, calc_y_data(chartData[index][1]) - 4, 8, 8)
  x = x + x_step;
  index++;
}
```

Listing 3.6: PixiJS: Drawing the data line and point markers for Salesperson A.

```
graph.lineStyle(1, 0x6e62dd);
graph.moveTo(margin, calc_y_data(chartData[0][2]))
graph.drawRegularPolygon(margin, calc_y_data(chartData[0][2]), 5, 4, 0);
var x = margin + x_step;
var index = 1;
while (index < chartData.length) {
  graph.lineTo(x, calc_y_data(chartData[index][2]));
  graph.drawRegularPolygon(x, calc_y_data(chartData[index][2]), 5, 4, 0);
  x = x + x_step;
  index++;
}
```

Listing 3.7: PixiJS: Drawing the data line and point markers for Salesperson B.

drawn before the loop, as shown in Listing 3.6.

For the second line, representing Salesperson B, rectangles with a 45 degree rotation are used as point markers. This is not included in basic PixiJS, but an additional library, "graphics-extras", is able to draw these slightly more sophisticated graphics. Apart from this rotation, the different colour, and the different datapoints from the chartData array, everything stays very similar, as shown in Listing 3.7.

3.1.2.6 Axis Titles

The text is generated like the axis tick labels, but positioned by hand until it fits sufficiently, as seen in Listing 3.8.

3.1.2.7 Legend

The text is generated as before, and the rectangle and text positioning are made by hand until fitting, as shown in Listing 3.9.

3.1.2.8 Resizing

For proper resizing, a custom resize function was made. It takes the ratio of the current viewBox and resizes it to the current window size, as shown in Listing 3.10. This function was found on the online Q and A platform stackoverflow [Bitwise Creative 2022].

```
var text = new PIXI.Text("Month", {
  fontFamily: 'Arial, sans-serif',
  fontSize: 15,
  fill: 0x000000,
  align: 'start'
});
text.x = view_box.width / 2;
text.y = view_box.height - margin / 4;
chart.stage.addChild(text);

var text = new PIXI.Text("Sales in €", {
  fontFamily: 'Arial, sans-serif',
  fontSize: 15,
  fill: 0x000000,
  align: 'start'
});
text.x = margin / 6;
text.y = view_box.height / 2;
text.angle = 270;
chart.stage.addChild(text);
```

Listing 3.8: PixiJS: Creating axis titles.

```
graph.lineStyle(1, 0xc6876f);
graph.drawRect(margin + x_step * 2, margin, 8, 8)

var text = new PIXI.Text("Salesperson A",
  { fontFamily: 'Arial, sans-serif', fontSize: 15,
    fill: 0x000000, align: 'start' });
text.x = margin + x_step * 2 + 20;
text.y = margin - 5;
chart.stage.addChild(text);
```

Listing 3.9: PixiJS: Creating the legend.

```
function resize() {
  if (window.innerWidth / window.innerHeight >= ratio) {
    var w = window.innerHeight * ratio;
    var h = window.innerHeight;
  } else {
    var w = window.innerWidth;
    var h = window.innerWidth / ratio;
  }
  chart.view.style.width = w + 'px';
  chart.view.style.height = h + 'px';
}

window.onresize = resize;
```

Listing 3.10: PixiJS: Resizing the window.

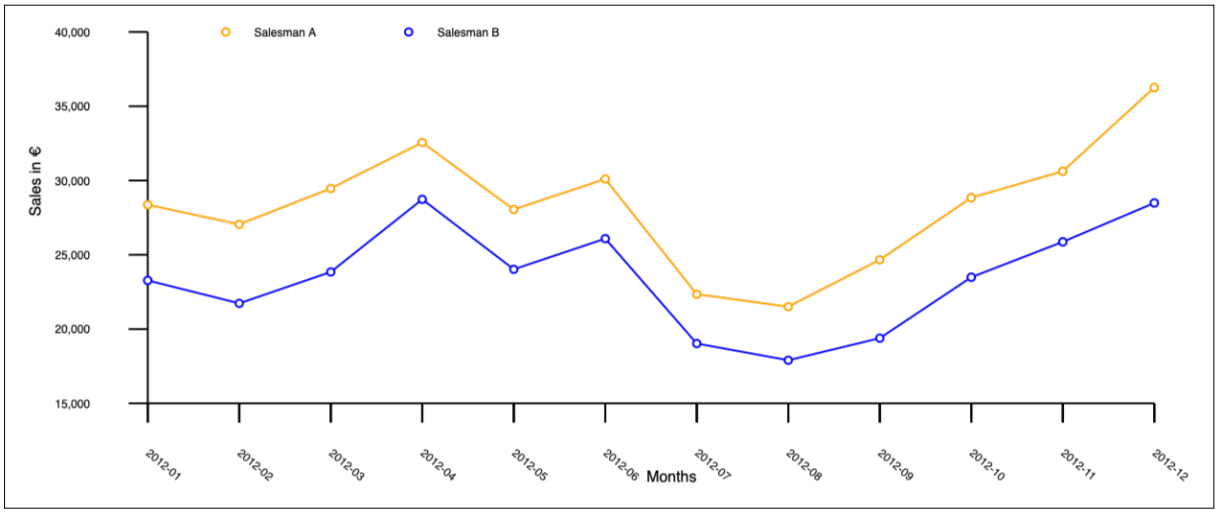

Figure 3.2: The chart created by the Two.js code. The full code can be found in Listing A.6.

3.2 Two.js

In this section, the code of the Two.js example is presented by explaining code snippets.

Two.js is a two-dimensional drawing API with many capabilities [Brandel 2022], including:

- Easily create and animate flat 2D shapes.
- Easy object storage for later manipulation, translation or transforming.
- Two.js has its own animation loop and can also be paired with any animation libraries.
- Externally created SVG elements can be brought into the Two.js scene.

One important fact about Two.js is that it is “render agnostic”. This means that it supports multiple underlying rendering technologies: SVG, Canvas2D, and WebGL. The sample chart created by Two.js can be seen in Listing 3.2.

3.2.1 Two.js Workflow

Two.js is an easy library to get into. Everything works on the basis of coordinates and objects can be easily drawn with different commands. Some examples would be: two.makeLine(), two.makeRectangle(), two.makePoints()... and so forth. An important part of Two.js is the easy integration of text into the chart. Two.makeText() lets the user easily add text to the chart at various coordinates and these texts can easily be transformed, translated or manipulated with the built-in parameters. The underlying technique for text is based on "Shapes", which are created in the form of the text input.

3.2.2 Two.js Code

The code of the Two.js example is presented by explaining code snippets. The full code can be found in Listing A.6.

3.2.2.1 Data Preprocessing

Before starting with the implementation, we set the basic parameters. The chartData array contains the data. ViewBox is set beforehand to initiate the Two.js object and can be as just as the margin set to desired values. The ratio is calculated for resizing purposes later on. The size of the grid (size) is calculated by substracting the margin from the viewBox, the steps are calculated by dividing the size by

```
1  //Data array for computation
2  const chartData = [
3    ["2012-01",28366,23274],["2012-02",27050,21732],
4    ["2012-03",29463,23845],["2012-04",32561,28732],
5    ["2012-05",28050,24023],["2012-06",30100,26089],
6    ["2012-07",22343,19026],["2012-08",21506,17903],
7    ["2012-09",24664,19387],["2012-10",28842,23490],
8    ["2012-11",30621,25873],["2012-12",36254,28490]]
9
10 //Setting margin, size and ratio for later computation of coordinates
11 var margin = 200;
12 var size = {width: view_box.width - margin * 2,
13   height: view_box.height - margin * 2};
14 let ratio = view_box.width / view_box.height;
15
16 //Calculation of distance between points on x and y-axis
17 let x_step = size.width / (chartData.length - 1);
18 let y_step = size.height / 5;
19
20 //Function calculate the graph spcific values for the money amounts
21 function calc_y_data(y) {
22   return margin + (size.height - (((y-15000) / 25000) * size.height));
23 }
```

Listing 3.11: Two.js: Data preprocessing.

the corresponding number of datapoints. To calculate the coordinates of the values for the salespeople, we normalize the values to a new range and substract them from the height of the grid, as the coordinate origin is top left. The code for this can be seen in Listing 3.11.

3.2.2.2 Initialisation

To get started with Two.js, one has to initialise the Two.js object. The said object can easily be created with the built in new Two() constructor declaring a new object. Certain parameters can be given to the new object, including:

- fullscreen: makes the Two.js object automatically adapt to the width and height of the parent document.
- width: sets the width of the Two.js object.
- height: sets the height of the Two.js object.
- autostart: automatically starts an AnimationFrame.
- type: Here the user can choose the type of renderer for the drawing. These include SVG, Canvas2D, and WebGL.

After setting all the parameters for the object, the last thing to do is add the Two.js object to the canvas. These initialisation steps can be seen in Listing 3.12.

```
let canvas = document.getElementById('chart');
let two = new Two({
  fullscreen: true,
  width: view_box.width,
  height:view_box.height,
  autostart: true,
  type: Two.Types.canvas
}).appendTo(canvas);
```

Listing 3.12: Two.js: Initialisation.

3.2.2.3 Resizing

Two.js unlike the other drawing libraries is not capable of doing resizing by itself. The developer has to write their own resizing function to make the chart or graphic adapt to a change in window size. In the resizing function itself, the view_box.width and view_box.height are updated to the new window size. The ratio for the calculation of the changing sizes gets updated and the margin gets gradually smaller. The new viewbox gets applied and all functions the user has already drawn before have to be called again. Meaning, the complete graph or object has to be rerendered in the browsers once the window size changes. The function is applied to the window.onresize option to be called anytime a resize happens. The full resizing function can be seen in Listing 3.13.

3.2.2.4 Axes

Drawing axes in Two.js is straightforward, specifiying the two endpoints for each axis line. For this, the function Two.makeLine() is used, as shown in Listing 3.14.

3.2.2.5 Axes Tick Marks

Axes tick marks are created by repeatedly calling the Two.makeLine() function in a loop with two coordinates only varying slightly on the x or y axis, as shown in Listing 3.15.

3.2.2.6 Axes Tick Labels

Tick labels on the axes are created using the built-in Two.js text function two.makeText(). This creates text as a "Shape" and places it onto the canvas at the given coordinates. The text is further stored in an object which can then be translated, rotated, or aligned to fit. The tick labels for the x-axis needs an extra line of code to rotate them appropriately, as can be seen in Listing 3.16.

3.2.2.7 Lines

Lines for the two datasets are created using the Two.js makeLine() function, as shown in Listing 3.17. It takes two points and connects them with each other. Each line segment is stored in an object and the colour is changed using the stroke parameter.

3.2.2.8 Point Markers

Markers for the data points are created by using the Two.js makePoints() function. The data for the points is retrieved from the previously created "ChartData" array. MakePoints() gets a point as input and creates an point object at that location. The object is stored and afterwards resized and coloured with the given parameters, as shown in Listing 3.18.

```
//Function to resize the graph when window gets resized
function resize() {
  if (window.innerWidth / window.innerHeight >= ratio) {
    var w = window.innerHeight * ratio;
    var h = window.innerHeight;
  } else {
    var w = window.innerWidth;
    var h = window.innerWidth / ratio;
  }

  view_box.width = w;
  view_box.height = h;
  size = {width: view_box.width - margin * 2,
    height: view_box.height - margin * 2};
  ratio = view_box.width / view_box.height;
  margin = view_box.width/8;
  canvas.setAttribute("viewBox", "0 0 " + view_box.width +
    " " + view_box.height + "");
  x_step = size.width / (chartData.length - 1);
  y_step = size.height / 5;
  two.clear();
  drawline();
  drawPointsXSalesA();
  drawPointsXSalesB();
  drawMarkingsX();
  drawMarkingsY();
  drawTopTextSalesA();
  drawTopTextSalesB();
  drawXText();
  drawYText();
  two.update();
}

window.onresize = resize;
```

Listing 3.13: Two.js resizing function.

```
//x-axis
two.makeLine(margin, margin+size.height, margin + size.width, margin+size.height);
//y-axis
two.makeLine(margin, margin, margin, margin+size.height);
```

Listing 3.14: Two.js: Creating the axes.

```
for(let i = margin; i <= margin+size.width; i += x_step){
  two.makeLine(i, margin+size.height, i, margin+size.height+10);

for(let i = margin; i <= margin+size.height; i += y_step){
  two.makeLine(margin, i, margin-10, i);
}
```

Listing 3.15: Two.js: Creating axis tick marks.

```
let text1 = two.makeText("2012-01", margin, margin + size.height + 25);
text1.alignment = "left";
text1.rotation = 0.7;
text1.size = size.height / 30;

let text1 = two.makeText("40,000", margin-30, margin);
text1.alignment = 'right';
text1.size = size.height/30;
```

Listing 3.16: Two.js: Creating axis tick labels.

```
let line = two.makeLine(i, calc_y_data(chartData[count][1]),
  i+x_step, calc_y_data(chartData[(count+1)][1]));
line.stroke = "#c6876f";
```

Listing 3.17: Two.js: Creating a line for a dataset.

```
let point = two.makePoints(i, calc_y_data(chartData[count][1]));
point.size = size.height/50;
point.stroke = "#c6876f";
```

Listing 3.18: Two.js: Creating point markers.

```
let xtitle = two.makeText("Month", margin + (x_step*6),
  margin+ size.height + size.height*0.2);
xtitle.alignment = 'right';
xtitle.size = size.height/25;

let ytitle = two.makeText("Sales in €",
  margin-size.height*0.3, margin+y_step*3-20);
ytitle.rotation = -1.57;
ytitle.alignment = 'left';
ytitle.size = size.height / 25;
```

Listing 3.19: Two.js: Creating the axis titles.

```
let point = two.makePoints((x_step*3)-15, margin);
point.size = size.height/50;
point.stroke = "#c6876f";

let salesA = two.makeText("Salesman A", (x_step*3), margin);
salesA.alignment = "left";
salesA.size = size.height/30;
```

Listing 3.20: Two.js: Creating the legend.

3.2.2.9 Axis Titles

The titles for the two axes are created using the Two.js makeText() function. The text is created and then stored in an object which can be translated or rotated as desired. The code to create the axis titles is shown in Listing 3.19.

3.2.2.10 Legend

The legend for both Salesperson A and Salesperson B is created using a combination of the makePoints() and makeText() functions. Both get a dedicated coordinate for creation and the makeText() function gets a string containing the text to be written. Both objects get stored and translated using the given parameters. The code for the part of the legend for Salesperson A is shown in Listing 3.20. The only difference for Salesperson B is the text string and the colour.

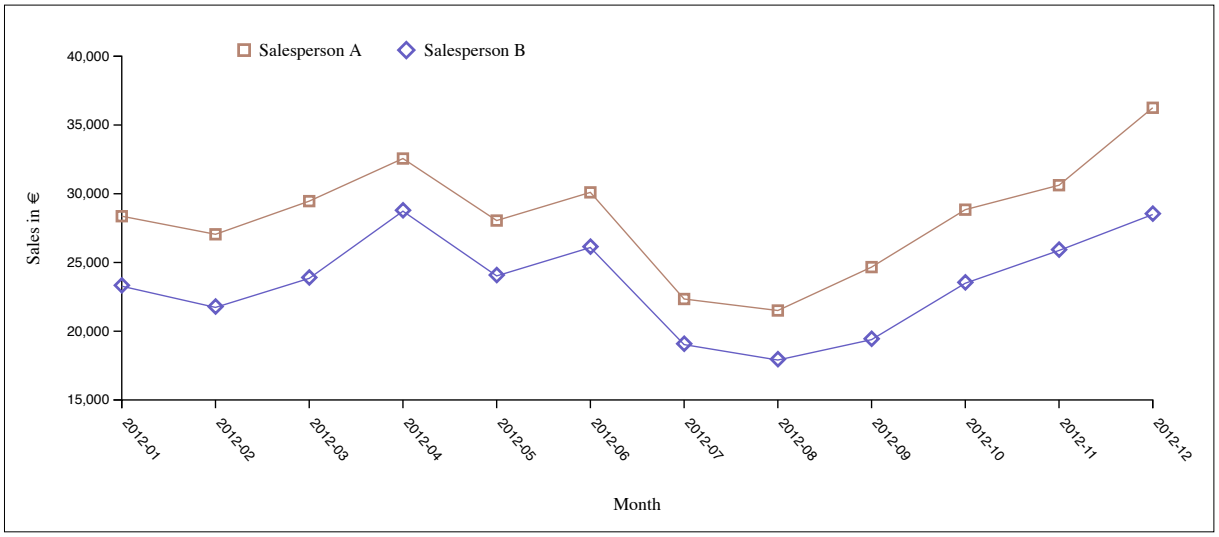

Figure 3.3: The chart created by the D3 code. The full code can be found in Listing A.7.

3.3 D3

D3 (Data-Driven Documents) [Bostock 2021] is the most widespread JavaScript library for document manipulation and for rendering information visualisations inside web browsers. Intrested reader is referred to Wattenberger [2019] and Bostock et al. [2011] for further reading. D3 is used to manipulate HTML and SVG elements inside a web document and to bind the elements that compose them to data. The possibility to bind sets of elements to data allows for efficient creation of dynamic information visualizations based on the underlying data. If the data changes, the visualisation will adapt accordingly. The sample chart created with D3 is shown in Figure 3.3.

3.3.1 Built-In Graphics Technologies and D3

Since D3 is made for manipulating HTML and SVG elements inside a web document, it is most naturally suitable for creating SVG-DOM visualisations. It is possible, though difficult, to combine D3 with one of the other two built-in rendering technologies (Canvas2D or WebGL) using one of two workarounds.

The first workaround is to create an SVG visualisation with D3, then use an instance of the Two.js SVG interpreter to encode it into Two.js internal representation, and finally use Two.js to render the newly encoded visualisation with one of the other built-in graphics technologies.

The second workaround involves representing the visualisation in a custom, XML-based document, then writing a script to compile a document-like representation of a visualisation into, say, Canvas2D code. This method is definitely more involved and the details will not be presented here, but the interested reader is referred to [Ros 2014] for further reading.

3.3.2 D3 Workflow

To get started, an SVG element must be instantiated either manually in the HTML code or by Javascript. Then this element must be selected with D3 and worked on through the D3 interface. D3 is used to append elements to the SVG document, and assign attributes to these newly created SVG elements. Before starting to draw, some functions must be defined using D3's functions to specify the projection of points from their domain to the final coordinate system.

```
/******** prepare the data *********/
const months = ["2012-01", "2012-02", "2012-03", "2012-04",
  "2012-05", "2012-06", "2012-07", "2012-08", "2012-09",
  "2012-10", "2012-11", "2012-12"];
const sales_a = [28366, 27050, 29463, 32561, 28050, 30100, 22343, 21506,
  24664, 28842, 30621, 36254];
const sales_b = [23274, 21732, 23845, 28732, 24023, 26089, 19026, 17903,
  19387, 23490, 25873, 28490];

/** d3 needs the data in the form
* [
*  d1 = {x: x1, y: y1},
*  d2 = {x: x2, y: y2},
*  d3 = {x: x3, y: y3},
*  d4 = {x: x4, y: y4},
*  d5 = {x: x5, y: y5} ]
*/

const points_a = sales_a.map(
  function (el, ind) { return { x: months[ind], y: el } });
const points_b = sales_b.map(
  function (el, ind) { return { x: months[ind], y: el } });
/**********************************/
```

Listing 3.21: D3 data preprocessing.

3.3.3 D3 Code

In this section, the code of D3 sample chart is presented by explaining code snippets. The full code can be found in Listing A.7.

3.3.3.1 Data Preprocessing

In order to make a chart with D3, data must be represented in a suitable format. Listing 3.21 shows how this can be done starting from an array of datapoints. The margin and viewBox are also prepared.

3.3.3.2 Initialisation

To initialise the D3 visualisation, the empty SVG element is retrieved, then initial SVG code is inserted into it, as shown in Listing 3.22.

D3 functions are used to project data points from their original domain to their position within the coordinate system of the SVG viewBox, as shown in Listing 3.23 for the y-axis.

3.3.3.3 Axes

D3 provides primitives to display axes on SVG visualisations based on scaling functions like those in Listing 3.23. The procedure to display the y-axis in D3 is shown in Listing 3.24.

3.3.3.4 Axis Tick Marks and Tick Labels

D3 provides primitives to display axes tick marks. In the case of a classical cardinal axis, tick marks and labels are automatically drawn when executing Listing 3.24. For more specialised needs, for example drawing an ordinal axis with months as tick labels, the procedure is more complicated. Listing 3.25

```
// Append initial SVG code to our SVG object
const svg = d3.select("#chart")
  .append("svg")
  .attr("viewBox", "0 0 " + view_box.width + " " + view_box.height)
  .append("g")
  // move the origin to realize the margin
  .attr("transform", "translate(" + margin + "," + margin + ")");
```

Listing 3.22: D3: Initialisation.

```
const yscl = d3.scaleLinear()
  .domain([15000, 40000])
  .range([size.height, 0]);
```

Listing 3.23: D3: Defining a scaling function for the y-axis.

shows a procedure to draw an ordinal axis with custom markings. Lines 24 to 28 are used to rotate the tick labels.

3.3.3.5 Lines and Points

To draw lines with D3, first a function must be created to specify how data should be interpreted by D3, like in Listing 3.26. Then, that function can be used to render the actual plots, as shown in Listing 3.27. Note that, here, a previously defined SVG marker element is used to mark each data point.

3.3.3.6 Axis Titles

D3 does not provide any primitive mechanism to render axis titles, so these must be rendered manually by standard drawing procedures. The code to create the x-axis title is shown in Listing 3.28.

3.3.3.7 Legend

D3 does not provide any primitive mechanism to render legends associated with a chart, so they must be rendered manually by standard drawing procedures.

```
svg.append("g")
  .call(d3.axisLeft(yscl)
  .ticks(6));
```

Listing 3.24: D3: Rendering the y-axis.

```
/******** X Axis.  *********/
// Making an ordinal axis in d3 with text rather than numbers is not very simple:

// firt the axis needs a function that linearily projects x values from
// their domain to the domain [0, size.width]
const xscl = d3.scaleLinear()
  // The axis can display ticks that go from 0 to months.length - 1
  .domain([0, months.length - 1])
  // The axis goes from point 0 to point size.width
  .range([0, size.width]);

// given our function, we ask the ticks to display months rather
// than actual numbers in [0,12] = [0, months.length - 1]
var xAxis = d3.axisBottom(xscl)
  .tickFormat(function (d) { return months[d]; })
// if tick was 1 we render it as months[1]...

// display the axis
svg.append("g")
  .attr("transform", "translate(0," + size.height + ")")
  // translate it up
  // so that it starts where y axis ends
  .call(xAxis)                    // render the axis
  .selectAll("text")              // adjust the ticks
  .style("text-anchor", "end")
  .attr("dx", "5em")
  .attr("dy", ".15em")
  .attr("transform", "rotate(50)")
```

Listing 3.25: D3: Rendering an ordinal axis with custom tick labels (months).

```
/* this creates a function that tells d3 how to interpret the data:
* this will be used like line(points)
* where each point in our case is an (x_value, y_value) couple.
* here we specify that given the list of couples, the line is created by
* just applying the x scale to x_values and y scale to the y_values
*/
var line = d3.line()
  .x(function (d) { return xscl(months.indexOf(d.x)); }) // apply the
  // x scale to the x data
  .y(function (d) { return yscl(d.y); }) // apply the y scale to the y data
```

Listing 3.26: D3: Preparing the plot function.

```
svg.append("path")
  .attr("class", "line")
  .attr("d", line(points_a))
  .style("fill", "none")
  .style("stroke", "#b58471")
  .style("stroke-width", 1)
  .attr('marker-start', 'url(#marker_a)')
  .attr('marker-mid', 'url(#marker_a)')
  .attr('marker-end', 'url(#marker_a)');
```

Listing 3.27: D3: Creating lines and markers.

```
// x axis title
svg.append("text")
  .attr("x", size.width / 2)
  .attr("y", size.height + 80)
  .attr("text-anchor", "middle")
  .attr("font-size", 13)
  .text('Month');
```

Listing 3.28: D3: Creating the x-axis title.

Chapter 4

Concluding Remarks

This work surveyed three built-in technologies for drawing graphics and charts in the web browser (SVG-DOM, Canvas2D, and WebGL), and three drawing libraries that work on top of those built-in technologies to provide a higher-level approach to drawing graphics in the web browser (PixiJS, Two.js, and D3).

4.1 Comparison of Built in Graphics Technologies

Table 4.1 shows an overview of the three built-in graphics technologies for programmatically creating charts and graphics in the web browser. All three use JavaScript, the programming language built in to modern web browsers. Immediately apparent is that WebGL takes advantage of hardware acceleration wherever the device supports it, but WebGL does not support a text API. The developer must decide whether faster perfomance or the convenience of a built-in text API is more important.

If we compare the number of lines of code needed to create the sample chart in each technology, Canvas2D needed around 200 lines, SVG-DOM around 250, and WebGL around 330.

4.2 Comparison of Drawing Libraries

Table 4.2 shows an overview of the three drawing libraries surveyed in this work. PixiJS renders with WebGL, but falls back to Canvas2D where WebGL is not available. With Two.js, the developer can choose to render with any one of SVG-DOM, Canvas2D, or WebGL. D3 effortlessly renders only with SVG-DOM, but as explained in Section 3.3.1, it is technically possible to render with other built-in technologies via workarounds. D3 is the only drawing library with support for data binding and automatic scaling of data values, possibly one reason for its widespread adoption.

In terms of lines of code to create the sample chart, PixiJS is the clear winner with only 165 lines of code, D3 having around 240 lines, and Two.js around 270 lines.

4.3 Recommendation

Of the built-in technologies, only SVG-DOM renders vector graphics, while Canvas2D and WebGL can only produce raster graphics. This means that, as long as not too many graphics objects are involved, the developer can stick with SVG-DOM to produce automatically scalable charts or visualisations. When the visualisation starts to become computationally intensive, WebGL might be a better choice, due to its hardware acceleration capabilities. If, instead, one needs to achieve a computationally expensive graphic result while keeping bundle size as low as possible (not importing any external code) and considering ease of use (e.g. avoiding complicated shaders), then the best way is to use the Canvas2D API, which allows for out-of-the-box graphics without the need for external packages, but still offers a very intuitive and complete programming experience.

	SVG-DOM	Canvas2D	WebGL
Rendering Technique:	Vector graphics	Raster graphics	Raster graphics
Language:	JavaScript	JavaScript	JavaScript
Text API:	yes	yes	no
Hardware Acceleration:	no	no	yes
Lines of Code:	200	250	330

Table 4.1: Comparison of built-in graphics technologies.

	PixiJS	Two.js	D3
Rendering Technology:	WebGL, Canvas2D	SVG-DOM Canvas2D, WebGL	SVG-DOM
Language:	JavaScript	JavaScript	JavaScript
Text API:	yes	yes	yes
Data Binding:	no	no	yes
Auto-Scaling:	no	no	yes
Lines of Code:	165	270	240

Table 4.2: Comparison of drawing libraries.

Regarding the drawing libraries, D3 provides a large number of drawing primitives and utility functions. Even though D3 is the most widely used it does not provide out-of-the box support for hardware acceleration. If one is not concerned about bundle size, a combination of D3 and Two.js can be exploited to effortlessly transform D3-based SVG visualisations into Canvas2D or WebGL ones. PixiJS is probably most suitable for interactive visualisations, because it has been written with a particular attention to its game development capabilities, so it provides easy ways to implement detailed interactivity. Two.js comes with a very useful SVG interpreter to import external SVG graphics. This allows to blueprint the visualisation with external libraries and then programmatically manipulate it with a high-level, WebGL-capable drawing library.

Appendix A

Listings

This appendix contains full source code listings of the various examples:

- Hand-crafted SVG line chart in Listing A.1.
- SVG-DOM example in Listing A.2.
- Canvas2D example in Listing A.3.
- WebGL example in Listing A.4.
- PixiJS example in Listing A.5.
- Two.js example in Listing A.6.
- D3 example in Listing A.7.

```
<svg viewBox="0 0 950 500"
  aria-labelledby="title desc"
  role="img"
  version="1.1"
  baseProfile="full"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink">

<title id="title">Monthly Sales</title>
<desc id="desc">Survey Chart as drawn in Lecture notes Chapter 1.1</desc>

<!--
Copyright © 2022 by Christoph Söls.
This work is placed under a Creative Commons Attribution 4.0
International (CC BY 4.0) licence.
https://creativecommons.org/licenses/by/4.0/
-->

<defs>
<!-- bounding box, for "inner" use double stroke-width -->
<path id="box" d="M 0 0 L 0 500 L 950 500 L 950 0 z" fill="none"
  stroke="black" stroke-width="6" />
<clipPath id="clip">
  <path d = "M 0 0 L 0 500 L 950 500 L 950 0 z"/>
</clipPath>
</defs>

<g class="grid">
  <g class="grid x-grid" stroke="black" stroke-width="2">
    <line x1="200" x2="750" y1="350" y2="350"></line>
    <line x1="200" x2="200" y1="350" y2="360"></line>
    <line x1="250" x2="250" y1="350" y2="360"></line>
    <line x1="300" x2="300" y1="350" y2="360"></line>
    <line x1="350" x2="350" y1="350" y2="360"></line>
    <line x1="400" x2="400" y1="350" y2="360"></line>
    <line x1="450" x2="450" y1="350" y2="360"></line>
    <line x1="500" x2="500" y1="350" y2="360"></line>
    <line x1="550" x2="550" y1="350" y2="360"></line>
    <line x1="600" x2="600" y1="350" y2="360"></line>
    <line x1="650" x2="650" y1="350" y2="360"></line>
    <line x1="700" x2="700" y1="350" y2="360"></line>
    <line x1="750" x2="750" y1="350" y2="360"></line>
  </g>
```

Listing A.1: Full source code of the hand-crafted SVG line chart shown in Figure 1.2.

```
44
45  <g class="grid x-grid caption" fill="black" font-family="Arial, sans-serif"
46    font-size="15" text-anchor="start">
47    <text transform="translate(200,380) rotate(45)">2012-01</text>
48    <text transform="translate(250,380) rotate(45)">2012-02</text>
49    <text transform="translate(300,380) rotate(45)">2012-03</text>
50    <text transform="translate(350,380) rotate(45)">2012-04</text>
51    <text transform="translate(400,380) rotate(45)">2012-05</text>
52    <text transform="translate(450,380) rotate(45)">2012-06</text>
53    <text transform="translate(500,380) rotate(45)">2012-07</text>
54    <text transform="translate(550,380) rotate(45)">2012-08</text>
55    <text transform="translate(600,380) rotate(45)">2012-09</text>
56    <text transform="translate(650,380) rotate(45)">2012-10</text>
57    <text transform="translate(700,380) rotate(45)">2012-11</text>
58    <text transform="translate(750,380) rotate(45)">2012-12</text>
59  </g>
60
61  <g class="grid y-grid caption" fill="black" font-family="Arial, sans-serif"
62    font-size="15" text-anchor="end">
63    <text x="160" y="105">40,000</text>
64    <text x="160" y="155">35,000</text>
65    <text x="160" y="205">30,000</text>
66    <text x="160" y="255">25,000</text>
67    <text x="160" y="305">20,000</text>
68    <text x="160" y="355">15,000</text>
69  </g>
70
71  <g class="grid y-grid" stroke="black" stroke-width="2">
72    <line x1="200" x2="200" y1="100" y2="350"></line>
73    <line x1="190" x2="200" y1="100" y2="100"></line>
74    <line x1="190" x2="200" y1="150" y2="150"></line>
75    <line x1="190" x2="200" y1="200" y2="200"></line>
76    <line x1="190" x2="200" y1="250" y2="250"></line>
77    <line x1="190" x2="200" y1="300" y2="300"></line>
78    <line x1="190" x2="200" y1="350" y2="350"></line>
79  </g>
80
81  <g class="axis captions" fill="black" font-family="Arial, sans-serif"
82    font-size="16">
83    <text transform="translate(80,250) rotate(270)">Sales in €</text>
84    <text x="475" y="450">Month</text>
85  </g>
86
87  <g class="line A" stroke="#c6876f" stroke-width="1">
88    <line x1="200" x2="250" y1="216" y2="230"></line>
89    <line x1="250" x2="300" y1="230" y2="205"></line>
90    <line x1="300" x2="350" y1="205" y2="174"></line>
91    <line x1="350" x2="400" y1="174" y2="220"></line>
92    <line x1="400" x2="450" y1="220" y2="199"></line>
93    <line x1="450" x2="500" y1="199" y2="277"></line>
94    <line x1="500" x2="550" y1="277" y2="285"></line>
95    <line x1="550" x2="600" y1="285" y2="253"></line>
96    <line x1="600" x2="650" y1="253" y2="212"></line>
97    <line x1="650" x2="700" y1="212" y2="194"></line>
98    <line x1="700" x2="750" y1="194" y2="137"></line>
99  </g>
```

Listing A.1 (cont.)**:** Full source code of the hand-crafted SVG line chart.

```
100
101     <g class="line B" stroke="#6e62dd" stroke-width="1">
102     <line x1="200" x2="250" y1="267" y2="283"></line>
103     <line x1="250" x2="300" y1="283" y2="262"></line>
104     <line x1="300" x2="350" y1="262" y2="213"></line>
105     <line x1="350" x2="400" y1="213" y2="260"></line>
106     <line x1="400" x2="450" y1="260" y2="239"></line>
107     <line x1="450" x2="500" y1="239" y2="310"></line>
108     <line x1="500" x2="550" y1="310" y2="321"></line>
109     <line x1="550" x2="600" y1="321" y2="306"></line>
110     <line x1="600" x2="650" y1="306" y2="265"></line>
111     <line x1="650" x2="700" y1="265" y2="241"></line>
112     <line x1="700" x2="750" y1="241" y2="215"></line>
113   </g>
114
115   <g class="rects line A" stroke="#c6876f" stroke-width="2" fill="none">
116     <rect x="196" y="212" width="8" height="8"></rect>
117     <rect x="246" y="226" width="8" height="8"></rect>
118     <rect x="296" y="201" width="8" height="8"></rect>
119     <rect x="346" y="170" width="8" height="8"></rect>
120     <rect x="396" y="216" width="8" height="8"></rect>
121     <rect x="446" y="195" width="8" height="8"></rect>
122     <rect x="496" y="273" width="8" height="8"></rect>
123     <rect x="546" y="281" width="8" height="8"></rect>
124     <rect x="596" y="249" width="8" height="8"></rect>
125     <rect x="646" y="208" width="8" height="8"></rect>
126     <rect x="696" y="190" width="8" height="8"></rect>
127     <rect x="746" y="133" width="8" height="8"></rect>
128   </g>
129
130   <g class="rects line B" stroke="#6e62dd" stroke-width="2" fill="none">
131     <rect transform="translate(200,262) rotate(45)" width="7" height="7"></rect>
132     <rect transform="translate(250,278) rotate(45)" width="7" height="7"></rect>
133     <rect transform="translate(300,257) rotate(45)" width="8" height="8"></rect>
134     <rect transform="translate(350,208) rotate(45)" width="8" height="8"></rect>
135     <rect transform="translate(400,255) rotate(45)" width="8" height="8"></rect>
136     <rect transform="translate(450,234) rotate(45)" width="8" height="8"></rect>
137     <rect transform="translate(500,305) rotate(45)" width="8" height="8"></rect>
138     <rect transform="translate(550,316) rotate(45)" width="8" height="8"></rect>
139     <rect transform="translate(600,301) rotate(45)" width="8" height="8"></rect>
140     <rect transform="translate(650,260) rotate(45)" width="8" height="8"></rect>
141     <rect transform="translate(700,236) rotate(45)" width="8" height="8"></rect>
142     <rect transform="translate(750,210) rotate(45)" width="8" height="8"></rect>
143   </g>
144
145   <g class="legend">
146     <rect x="265" y="101" width="8" height="8" stroke="#c6876f"
147       stroke-width="2" fill="none"></rect>
148     <text x="280" y="110" fill="black" font-family="Arial, sans-serif"
149       font-size="16">Salesperson A</text>
150     <rect transform="translate(425,100) rotate(45)" width="7" height="7"
151       stroke="#6e62dd" stroke-width="2" fill="none"></rect>
152     <text x="440" y="110" fill="black" font-family="Arial, sans-serif"
153       font-size="16">Salesperson B</text>
154   </g>
155
156 </g>
157 </svg>
```

Listing A.1 (cont.)**:** Full source code of the hand-crafted SVG line chart.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
  <meta charset="UTF-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <title>SVG DOM</title>
</head>

<style>
  div {
    width: 95vw;
    height: 95vh;
  }
</style>

<body>

<div id="chart"></div>

<script>
  var svg_chart = document.getElementById("chart");

  //data array
  const chartData = [
    ["2012-01",28366,23274],["2012-02",27050,21732],["2012-03",29463,23845],
    ["2012-04",32561,28732],["2012-05",28050,24023],["2012-06",30100,26089],
    ["2012-07",22343,19026],["2012-08",21506,17903],["2012-09",24664,19387],
    ["2012-10",28842,23490],["2012-11",30621,25873],["2012-12",36254,28490]]

  //Setting margin, size for later computation of corrdinates for drawing
  const view_box = {width: 950, height: 450};
  const margin = 100;
  const size = {width: view_box.width - margin * 2,
    height: view_box.height - margin * 2};

  //Calculation of distance between points on x and y-axis
  const x_step = size.width / (chartData.length - 1);
  const y_step = size.height / 5;

  //Function to gradually change the distance between the points on the y axis
  function calc_y_data(y) {
    return margin + (size.height - (((y-15000) / 25000) * size.height));
  }

  //generating the axes with 2 loops
  const generateGrid = (chart, data) => {
    const x_grid = document.createElementNS('http://www.w3.org/2000/svg', 'line');
    x_grid.setAttribute("x1", margin);
    x_grid.setAttribute("x2", size.width + margin);
    x_grid.setAttribute("y1", size.height + margin);
    x_grid.setAttribute("y2", size.height + margin);
    x_grid.setAttribute("stroke", "black");
    x_grid.setAttribute("stroke-width", "2");
    chart.appendChild(x_grid);
```

Listing A.2: Full source code of the SVG-DOM example shown in Figure 2.1.

```
56
57      var x = margin;
58      var index = 0;
59
60      while(index < data.length){
61        const x_marking = document.createElementNS('http://www.w3.org/2000/svg',
62          'line');
63        const x_caption = document.createElementNS('http://www.w3.org/2000/svg',
64          'text');
65
66        x_marking.setAttribute("x1", x);
67        x_marking.setAttribute("x2", x);
68        x_marking.setAttribute("y1", size.height + margin);
69        x_marking.setAttribute("y2", size.height + margin + 10);
70        x_marking.setAttribute("stroke", "black");
71        x_marking.setAttribute("stroke-width", "2");
72
73        x_caption.setAttribute("fill", "black");
74        x_caption.setAttribute("font-family", "Arial, sans-serif");
75        x_caption.setAttribute("font-size", "15");
76        x_caption.setAttribute("text-anchor", "start");
77        x_caption.setAttribute("transform", "translate(" + x + "," +
78          (size.height + margin + 30) + ") rotate(45)");
79        x_caption.append(data[index][0]);
80
81        chart.appendChild(x_caption);
82        chart.appendChild(x_marking);
83        x = x + x_step;
84        index++;
85      }
86
87      const y_grid = document.createElementNS('http://www.w3.org/2000/svg',
88        'line');
89      y_grid.setAttribute("x1", margin);
90      y_grid.setAttribute("x2", margin);
91      y_grid.setAttribute("y1", margin);
92      y_grid.setAttribute("y2", margin + size.height);
93      y_grid.setAttribute("stroke", "black");
94      y_grid.setAttribute("stroke-width", "2");
95      chart.appendChild(y_grid);
96
97      var y = margin;
98      var sales_val = new Array('40,000', '35,000', '30,000',
99        '25,000', '20,000', '15,000');
100     index = 0;
101
102     while(index < 6){
103       const y_marking = document.createElementNS('http://www.w3.org/2000/svg',
104         'line');
105       const y_caption = document.createElementNS('http://www.w3.org/2000/svg',
106         'text');
```

Listing A.2 (cont.)**:** Full source code of the SVG-DOM example.

```
107
108       y_marking.setAttribute("x1", margin - 10);
109       y_marking.setAttribute("x2", margin);
110       y_marking.setAttribute("y1", y);
111       y_marking.setAttribute("y2", y);
112       y_marking.setAttribute("stroke", "black");
113       y_marking.setAttribute("stroke-width", "2");
114
115       y_caption.setAttribute("fill", "black");
116       y_caption.setAttribute("font-family", "Arial, sans-serif");
117       y_caption.setAttribute("font-size", "15");
118       y_caption.setAttribute("text-anchor", "end");
119       y_caption.setAttribute("x", margin - 20);
120       y_caption.setAttribute("y", y + 5);
121       y_caption.append(sales_val[index]);
122
123       chart.appendChild(y_caption);
124       chart.appendChild(y_marking);
125       y = y + y_step;
126       index++;
127     }
128
129   }
130
131   // generating the lines and rects by looping through the data array
132   // and adjusting the values
133   const generateLines = (chart, data) => {
134
135     var index = 0;
136     var x = margin;
137
138     while(index < data.length-1){
139       const lineA = document.createElementNS('http://www.w3.org/2000/svg',
140         'line');
141       const lineB = document.createElementNS('http://www.w3.org/2000/svg',
142         'line');
143
144       lineA.setAttribute("stroke", "#c6876f");
145       lineA.setAttribute("stroke-width", "1");
146       lineA.setAttribute("x1", x);
147       lineA.setAttribute("x2", x + x_step);
148       lineA.setAttribute("y1", calc_y_data(data[index][1]));
149       lineA.setAttribute("y2", calc_y_data(data[index+1][1]));
150
151       lineB.setAttribute("stroke", "#6e62dd");
152       lineB.setAttribute("stroke-width", "1");
153       lineB.setAttribute("x1", x);
154       lineB.setAttribute("x2", x + x_step);
155       lineB.setAttribute("y1", calc_y_data(data[index][2]));
156       lineB.setAttribute("y2", calc_y_data(data[index+1][2]));
157
158       chart.appendChild(lineA);
159       chart.appendChild(lineB);
160       index++;
161       x = x + x_step;
162     }
163   }
```

Listing A.2 (cont.)**:** Full source code of the SVG-DOM example.

```
164
165   generateRects = (chart, data) => {
166     var index = 0;
167     var x = margin;
168
169     while(index < data.length){
170       const rectA = document.createElementNS('http://www.w3.org/2000/svg',
171         'rect');
172       const rectB = document.createElementNS('http://www.w3.org/2000/svg',
173         'rect');
174
175       rectA.setAttribute("stroke", "#c6876f");
176       rectA.setAttribute("stroke-width", "2");
177       rectA.setAttribute("fill", "none");
178       rectA.setAttribute("x", x - 4);
179       rectA.setAttribute("y", calc_y_data(data[index][1]) - 4);
180       rectA.setAttribute("width", "8");
181       rectA.setAttribute("height", "8");
182
183       rectB.setAttribute("stroke", "#6e62dd");
184       rectB.setAttribute("stroke-width", "2");
185       rectB.setAttribute("fill", "none");
186       rectB.setAttribute("transform", "translate(" + x + "," +
187         (calc_y_data(data[index][2]) - 5) + ") rotate(45)");
188       rectB.setAttribute("width", "8");
189       rectB.setAttribute("height", "8");
190
191       chart.appendChild(rectA);
192       chart.appendChild(rectB);
193       index++;
194       x = x + x_step;
195     }
196   }
197
198   generateAdditionalCaptions = (chart) => {
199     const rectA = document.createElementNS('http://www.w3.org/2000/svg',
200       'rect');
201     const rectB = document.createElementNS('http://www.w3.org/2000/svg',
202       'rect');
203     const spA = document.createElementNS('http://www.w3.org/2000/svg',
204       'text');
205     const spB = document.createElementNS('http://www.w3.org/2000/svg',
206       'text');
207     const x_caption = document.createElementNS('http://www.w3.org/2000/svg',
208       'text');
209     const y_caption = document.createElementNS('http://www.w3.org/2000/svg',
210       'text');
```

Listing A.2 (cont.)**:** Full source code of the SVG-DOM example.

```
211
212     rectA.setAttribute("stroke", "#c6876f");
213     rectA.setAttribute("stroke-width", "2");
214     rectA.setAttribute("fill", "none");
215     rectA.setAttribute("x", margin + x_step * 2);
216     rectA.setAttribute("y", margin - 8);
217     rectA.setAttribute("width", "8");
218     rectA.setAttribute("height", "8");
219
220     rectB.setAttribute("stroke", "#6e62dd");
221     rectB.setAttribute("stroke-width", "2");
222     rectB.setAttribute("fill", "none");
223     rectB.setAttribute("transform", "translate(" + (margin + x_step * 5) +
224       "," + (margin - 8) + ") rotate(45)");
225     rectB.setAttribute("width", "7");
226     rectB.setAttribute("height", "7");
227
228     spA.setAttribute("fill", "black");
229     spA.setAttribute("font-family", "Arial, sans-serif");
230     spA.setAttribute("font-size", "16");
231     spA.setAttribute("x", margin + x_step * 2 + 20);
232     spA.setAttribute("y", margin);
233     spA.append("Salesperson A");
234
235     spB.setAttribute("fill", "black");
236     spB.setAttribute("font-family", "Arial, sans-serif");
237     spB.setAttribute("font-size", "16");
238     spB.setAttribute("x", (margin + x_step * 5) + 20);
239     spB.setAttribute("y", margin);
240     spB.append("Salesperson B");
241
242     x_caption.setAttribute("fill", "black");
243     x_caption.setAttribute("font-family", "Arial, sans-serif");
244     x_caption.setAttribute("font-size", "16");
245     x_caption.setAttribute("x", view_box.width / 2);
246     x_caption.setAttribute("y", view_box.height - margin / 8);
247     x_caption.append("Month");
248
249     y_caption.setAttribute("fill", "black");
250     y_caption.setAttribute("font-family", "Arial, sans-serif");
251     y_caption.setAttribute("font-size", "16");
252     y_caption.setAttribute("transform", "translate(" + margin / 4 +
253       "," + view_box.height / 2 + ") rotate(270)");
254     y_caption.append("Sales in €");
255
256     chart.appendChild(x_caption);
257     chart.appendChild(y_caption);
258     chart.appendChild(spA);
259     chart.appendChild(spB);
260     chart.appendChild(rectA);
261     chart.appendChild(rectB);
262   }
```

Listing A.2 (cont.)**:** Full source code of the SVG-DOM example.

```
263
264   const generateChart = (data) => {
265     //initialize the svg
266     const svg = document.createElementNS('http://www.w3.org/2000/svg', 'svg');
267     svg.setAttribute("viewBox", "0 0 " + view_box.width + " " +
268       view_box.height + "");
269     generateGrid(svg, data);
270     generateLines(svg, data);
271     generateRects(svg, data);
272     generateAdditionalCaptions(svg);
273     svg_chart.appendChild(svg);
274   }
275
276   generateChart(chartData);
277 </script>
278 </body>
279 </html>
```

Listing A.2 (cont.)**:** Full source code of the SVG-DOM example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Canvas2D</title>

  <style>
    body {
      margin: 0;
    }

    #chart {
      /* define proportions of the graph */
      width: 100vw;
      height: 50vw;

      /* prevent appearance of the scrollbar by not
        allowing the chart height to grow more than window size
       */
      max-height: 100vh;

      /* this is used only in the canvas2d case: it does not allow
         the chart to loose the original aspect ratio when it is
         resized only vertically
       */
      min-height: 50vw;
    }
  </style>
</head>


<body>
  <canvas id="chart" width="2700" height="1300"></canvas>
</body>

<script>
  var canvas = document.getElementById("chart")
  var ctx = canvas.getContext("2d");

  //data array
  const chartData = [["2012-01", 28366, 23274], ["2012-02", 27050, 21732],
    ["2012-03", 29463, 23845], ["2012-04", 32561, 28732],
    ["2012-05", 28050, 24023], ["2012-06", 30100, 26089],
    ["2012-07", 22343, 19026], ["2012-08", 21506, 17903],
    ["2012-09", 24664, 19387], ["2012-10", 28842, 23490],
    ["2012-11", 30621, 25873], ["2012-12", 36254, 28490]]

  //Setting margin, size for later computation of corrdinates for drawing
  const view_box = { width: canvas.width, height: canvas.height };
  const margin = 350;
  const size = { width: view_box.width - margin * 2,
    height: view_box.height - margin * 2 };
```

Listing A.3: Full source code of the Canvas2D example shown in Figure 2.2.

```
55
56  //Calculation of distance between points on x and y-axis
57  const x_step = size.width / (chartData.length - 1);
58  const y_step = size.height / 5;
59
60  //Function to gradually change the distance between the points on the y axis.
61  //Differently from the other files, here we translate the 0,0 to fit the margin
62  // so there is no need to include it in the calculations
63  function calc_y_data(y) {
64    return ((y-15000) / 25000) * size.height ;
65  }
66
67  //Following is a reparameterisaztion of the coordinates to translate the 0,0
68  // point in the low-left corner and work in a more natural way
69  // carefull: if this trick is used then when printing text temporary rescaling
70  // in needed
71  ctx.translate(margin, margin+size.height);
72  ctx.scale(1, -1); // this also puts 0,height in the top-left corner
73
74  /**
75   * draw a line from (x0,y0) to (x1,y1)
76  */
77  function drawLine(x0, y0, x1, y1) {
78    ctx.beginPath();
79    ctx.moveTo(x0, y0);
80    ctx.lineTo(x1, y1);
81    ctx.stroke();
82  }
83
84  /** draw a rotated square
85   * x, y center point
86   * s:   size of the square
87   * a:   angle of rotation
88  */
89  function draw_rot_square(x, y, s, a) {
90    ctx.lineWidth = 0.003 *size.width;
91    ctx.beginPath();
92    ctx.translate(x, y);
93    ctx.rotate(a);
94    ctx.rect(0 - (s / 2), 0 - (s / 2), s, s);
95    ctx.rotate(-a);
96    ctx.translate(-x, -y);
97    ctx.stroke();
98    ctx.lineWidth = 0.0015 *size.width;
99  }
```

Listing A.3 (cont.)**:** Full source code of the Canvas2D example.

```
100
101   /** draw a rotated piece of text
102    * x, y starting point
103    * a:   angle of rotation
104    * text
105   */
106   function draw_rot_text(x, y, a, text) {
107     ctx.translate(x, y);
108     ctx.rotate(a);
109     ctx.fillText(text, 0, 0);
110     ctx.rotate(-a);
111     ctx.translate(-x, -y);
112   }
113
114   /** plot a line given array of datapoints
115    * arr:   array of datapoints
116    * sq_s:  square size (markers)
117    * a:     angle of rotation for the squares (markers)
118   */
119   function plot(j, sq_s, a) {
120     ctx.beginPath();
121     ctx.moveTo(0, calc_y_data( chartData[0][j]));
122     for (let i = 0; i < chartData.length; i ++) {
123       console.log(i);
124       ctx.lineTo(i*x_step,  calc_y_data( chartData[i][j]));
125     }
126     ctx.stroke();
127     for (let i = 0; i < chartData.length; i ++) {
128       draw_rot_square(i*x_step, calc_y_data( chartData[i][j]), sq_s, a);
129     }
130   }
131
132   /******** draw axes *********/
133   // y axis
134   y_ax_marks = ["15,000", "20,000", "25,000", "30,000", "35,000", "40,000"]
135   ctx.lineWidth = 0.002 * size.width;
136   drawLine(0, 0, 0, size.height);
137   // ticks on y axis
138   for (let i = 0; i < y_ax_marks.length; i ++) {
139     drawLine(0, i*y_step, -0.01 * size.width, i*y_step);
140   }
141   // x axis
142   drawLine(0, 0, size.width, 0);
143   // ticks on x axis
144   for (let i = 0; i < chartData.length; i ++) {
145     drawLine(i*x_step, 0, i*x_step, -0.01 * size.width)
146   }
```

Listing A.3 (cont.)**:** Full source code of the Canvas2D example.

```
147
148   /******** axes labels (text) *********/
149   ctx.font = 0.015 *size.width + "px Arial";
150   ctx.scale(1, -1);
151   for (let i = 0; i < y_ax_marks.length; i ++) {
152     draw_rot_text((-0.085 *size.width), -i*y_step + 5, 0 , y_ax_marks[i]);
153   }
154   for (let i = 0; i < chartData.length; i ++) {
155     draw_rot_text(i*x_step, 0.1 *size.height,
156       45 * Math.PI / 180, chartData[i][0]);
157   }
158   draw_rot_text((-0.11 *size.width), (-0.5 *size.height),
159     -90 * Math.PI / 180, "Sales in €");
160   draw_rot_text(0.5 *size.width, (0.4 *size.height),
161     0 * Math.PI / 180, "Month");
162   ctx.scale(1, -1);
163
164   /******** plot the lines *********/
165   //ctx.lineWidth = 0.001 *size.width;
166
167   // orange line, 7pts squares, no rotation
168   ctx.strokeStyle = "#c6876f";
169   // for the legend
170   draw_rot_square(0.1 *size.width, size.height, 0.03 *size.height, 0);
171   plot(1, 0.03 *size.height);
172
173   // blue line, 6pts squares, 40deg rotation
174   ctx.strokeStyle = "#6e62dd";
175   // for the legend
176   draw_rot_square(0.28 *size.width, size.height, 0.03 *size.height, 40);
177   plot(2, 0.03 *size.height, 40);
178
179   ctx.scale(1, -1);
180   ctx.fillText("Saleperson A", 0.12 *size.width, -0.98*size.height);
181   ctx.fillText("Saleperson B", 0.3 *size.width, -0.98*size.height);
182   ctx.scale(1, -1);
183 </script>
184
185 </html>
```

Listing A.3 (cont.)**:** Full source code of the Canvas2D example.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <style>
      /*  */
      html, body {
          margin: 0;
      }
      /* make the canvas fill its container */
      #container{
          position: relative;
      }
      #chart {
          /*aspect-ratio: 16/10;*/
          width: 100vw;
          height: 50vw;
          left: 0px;
          top: 0px;
          display: block;
          z-index: 8;
          max-height: 100vh;
      }
      #text{
          background-color: transparent;
          color: #fff;
          width: 100vw;
          height: 50vw;
          left: 0px;
          pointer-events:none;
          position: absolute;
          top: 0px;
          z-index: 10;
      }
  </style>
</head>

<body>
<div id="container">
  <canvas id="chart" width="1500" height="800"></canvas>
  <canvas id="text" width="1500" height="800"></canvas>
</div>

<script>
  let canvas = document.getElementById('chart');
  let text = document.querySelector('#text');

  let gl = canvas.getContext('webgl');
  let context = text.getContext('2d');


  context.clearRect(0, 0, context.canvas.width, context.canvas.height);
```

Listing A.4: Full source code of the WebGL example shown in Figure 2.4.

```
54
55  //Data array for computation
56  const chartData = [["2012-01",28366,23274],["2012-02",27050,21732],
57    ["2012-03",29463,23845],["2012-04",32561,28732],
58    ["2012-05",28050,24023],["2012-06",30100,26089],
59    ["2012-07",22343,19026],["2012-08",21506,17903],
60    ["2012-09",24664,19387],["2012-10",28842,23490],
61    ["2012-11",30621,25873],["2012-12",36254,28490]]
62
63  //Setting margin, size and ratio for later computation of corrdinates for drawing
64  let view_box = {width: 950, height: 500};
65  var margin = 100;
66  var size = {width: view_box.width - margin * 2,
67    height: view_box.height - margin * 2};
68
69  //Calculation of distance between points on x and y-axis
70  let x_step = size.width / (chartData.length - 1);
71  let y_step = size.height/5;
72
73  //Function calculate the graph specific values for the money amounts
74  function calc_y_data(y) {
75      return margin + (((y-15000) / 25000) * size.height);
76  }
77
78  canvas.setAttribute("viewBox", "0 0 " + view_box.width +
79    " " + view_box.height + "");
80  text.setAttribute("viewBox", "0 0 " + view_box.width +
81    " " + view_box.height + "");
82
83  //fixing blurry text
84  view_box.devicePixelRatio = 1;
85  let scale = window.devicePixelRatio;
86
87  //Adding Text to Graph
88  addText(context);
89
90  gl.clearColor(1,1,1,1);
91  gl.clear(gl.COLOR_BUFFER_BIT);
92
93
94  //Vertex Shader initialization
95  let vertexShader = gl.createShader(gl.VERTEX_SHADER);
96  gl.shaderSource(vertexShader, [
97    'attribute vec2 position;',
98    'void main() {',
99    'gl_Position = vec4(position, 0.0, 1.0);',
100   'gl_PointSize = 10.0;',
101   '}'
102 ].join('\n'));
103 gl.compileShader(vertexShader);
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
104
105   //Fragment Shader initialization
106   let fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
107   gl.shaderSource(fragmentShader, [
108     'precision highp float;',
109     'uniform vec4 color;',
110     'void main() {',
111     'gl_FragColor = color;',
112     '}'
113   ].join('\n'));
114   gl.compileShader(fragmentShader);
115
116   //Creation of program and attachment of shader
117   let program = gl.createProgram();
118   gl.attachShader(program, vertexShader);
119   gl.attachShader(program, fragmentShader);
120   gl.linkProgram(program);
121
122   //Setting universal used color to black for the program
123   gl.useProgram(program)
124   program.color = gl.getUniformLocation(program, 'color');
125   gl.uniform4fv(program.color, [0.0,0,0,1.0]);
126
127   canvas.width = window.innerWidth;
128   canvas.height = window.innerHeight;
129   gl.viewport(0,0, gl.canvas.width, gl.canvas.height);
130   gl.lineWidth(1);
131
132   drawLines();
133   //Changing color to orange for sales person A
134   gl.uniform4fv(program.color, [0.776,0.529,0.435, 1.0]);
135   drawPointSalesA();
136   drawConnectedLinesSalesA();
137   drawPointMarkerA();
138
139   //Changing color to blue for sales person B
140   gl.uniform4fv(program.color, [0.431,0.384,0.867, 1.0]);
141   drawPointSalesB();
142   drawConnectedLinesSalesB();
143   drawPointMarkerB();
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
145
146   function addText(context){
147     //Text for x and y axis
148     context.font = "24px sans-serif";
149     context.fillText("Month", 720, 740);
150
151     context.font = "18px sans-serif";
152     context.fillText("15,000", 80, 630);
153     context.fillText("20,000", 80, 535);
154     context.fillText("25,000", 80, 442);
155     context.fillText("30,000", 80, 349);
156     context.fillText("35,000", 80, 255);
157     context.fillText("40,000", 80, 162);
158     context.fillText("Salesperson A", 280, 162);
159     context.fillText("Salesperson B", 500, 162);
160
161     context.save()
162     context.rotate(-Math.PI/2);
163     context.translate(-750,-400)
164     context.font = "24px sans-serif";
165     context.fillText("Sales in €", 310, 450);
166     context.restore();
167
168     context.rotate(45 * Math.PI/180);
169     context.font = "18px sans-serif";
170     context.fillText("2012-01", 584, 358);
171     context.fillText("2012-02", 660, 280);
172     context.fillText("2012-03", 736, 204);
173     context.fillText("2012-04", 812, 128);
174     context.fillText("2012-05", 888, 52);
175     context.fillText("2012-06", 964, -25);
176     context.fillText("2012-07", 1040, -100);
177     context.fillText("2012-08", 1116, -176);
178     context.fillText("2012-09", 1192, -255);
179     context.fillText("2012-10", 1268, -330);
180     context.fillText("2012-11", 1344, -405);
181     context.fillText("2012-12", 1420, -485);
182   }
183
184   function drawPointMarkerA(){
185     let vertices = new Float32Array([
186       normalize(margin+x_step), 0.6
187     ]);
188
189     createBindBuffer(vertices);
190     gl.drawArrays(gl.POINTS, 0, 1);
191   }
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
192
193   function drawPointMarkerB(){
194     let vertices = new Float32Array([
195       normalize(margin+x_step*3), 0.6
196     ]);
197
198     createBindBuffer(vertices);
199     gl.drawArrays(gl.POINTS, 0, 1);
200   }
201
202   function drawLines(){
203     let n = initLinebuffer();
204     gl.drawArrays(gl.LINES, 0, n);
205   }
206
207   function initSalesAPoints(){
208     let vertices = new Float32Array([
209       normalize(margin), normalizeheight(calc_y_data(chartData[0][1])),
210       normalize(margin + x_step), normalizeheight(calc_y_data(chartData[1][1])),
211       normalize(margin + x_step*2), normalizeheight(calc_y_data(chartData[2][1])),
212       normalize(margin + x_step*3), normalizeheight(calc_y_data(chartData[3][1])),
213       normalize(margin + x_step*4), normalizeheight(calc_y_data(chartData[4][1])),
214       normalize(margin + x_step*5), normalizeheight(calc_y_data(chartData[5][1])),
215       normalize(margin + x_step*6), normalizeheight(calc_y_data(chartData[6][1])),
216       normalize(margin + x_step*7), normalizeheight(calc_y_data(chartData[7][1])),
217       normalize(margin + x_step*8), normalizeheight(calc_y_data(chartData[8][1])),
218       normalize(margin + x_step*9), normalizeheight(calc_y_data(chartData[9][1])),
219       normalize(margin + x_step*10), normalizeheight(calc_y_data(chartData[10][1])),
220       normalize(margin + x_step*11), normalizeheight(calc_y_data(chartData[11][1]))
221     ]);
222     let n = 12;
223     createBindBuffer(vertices);
224     return n;
225   }
226
227   function drawPointSalesA(){
228       let n = initSalesAPoints();
229       gl.drawArrays(gl.POINTS, 0, n);
230   }
231
232   function drawConnectedLinesSalesA(){
233       let n = initSalesAPoints();
234       gl.drawArrays(gl.LINE_STRIP, 0, n);
235   }
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
236
237   function initSalesBPoints(){
238     let vertices = new Float32Array([
239       normalize(margin), normalizeheight(calc_y_data(chartData[0][2])),
240       normalize(margin + x_step), normalizeheight(calc_y_data(chartData[1][2])),
241       normalize(margin + x_step*2), normalizeheight(calc_y_data(chartData[2][2])),
242       normalize(margin + x_step*3), normalizeheight(calc_y_data(chartData[3][2])),
243       normalize(margin + x_step*4), normalizeheight(calc_y_data(chartData[4][2])),
244       normalize(margin + x_step*5), normalizeheight(calc_y_data(chartData[5][2])),
245       normalize(margin + x_step*6), normalizeheight(calc_y_data(chartData[6][2])),
246       normalize(margin + x_step*7), normalizeheight(calc_y_data(chartData[7][2])),
247       normalize(margin + x_step*8), normalizeheight(calc_y_data(chartData[8][2])),
248       normalize(margin + x_step*9), normalizeheight(calc_y_data(chartData[9][2])),
249       normalize(margin + x_step*10), normalizeheight(calc_y_data(chartData[10][2])),
250       normalize(margin + x_step*11), normalizeheight(calc_y_data(chartData[11][2]))
251     ]);
252     let n = 12;
253     createBindBuffer(vertices);
254     return n;
255   }
256
257   function drawPointSalesB(){
258     let n = initSalesBPoints();
259     gl.drawArrays(gl.POINTS, 0, n);
260   }
261
262   function drawConnectedLinesSalesB(){
263     let n = initSalesBPoints();
264     gl.drawArrays(gl.LINE_STRIP, 0, n);
265   }
266
267
268   //New Calculation of x and y-axis with margin and steps
269   function initLinebuffer(){
270     let vertices = new Float32Array([
271       //x-axis
272       normalize(margin), normalizeheight(margin),
273       normalize(margin + size.width), normalizeheight(margin),
274
275       //y-axis
276       normalize(margin), normalizeheight(margin),
277       normalize(margin), normalizeheight(margin + size.height),
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
278
279       //Pointmarks x-axis
280       normalize(margin), normalizeheight(margin),
281       normalize(margin), normalizeheight(margin - 10),
282       normalize(margin + x_step), normalizeheight(margin),
283       normalize(margin + x_step), normalizeheight(margin - 10),
284       normalize(margin + x_step*2), normalizeheight(margin),
285       normalize(margin + x_step*2), normalizeheight(margin - 10),
286       normalize(margin + x_step*3), normalizeheight(margin),
287       normalize(margin + x_step*3), normalizeheight(margin - 10),
288       normalize(margin + x_step*4), normalizeheight(margin),
289       normalize(margin + x_step*4), normalizeheight(margin - 10),
290       normalize(margin + x_step*5), normalizeheight(margin),
291       normalize(margin + x_step*5), normalizeheight(margin - 10),
292       normalize(margin + x_step*6), normalizeheight(margin),
293       normalize(margin + x_step*6), normalizeheight(margin - 10),
294       normalize(margin + x_step*7), normalizeheight(margin),
295       normalize(margin + x_step*7), normalizeheight(margin - 10),
296       normalize(margin + x_step*8), normalizeheight(margin),
297       normalize(margin + x_step*8), normalizeheight(margin - 10),
298       normalize(margin + x_step*9), normalizeheight(margin),
299       normalize(margin + x_step*9), normalizeheight(margin - 10),
300       normalize(margin + x_step*10), normalizeheight(margin),
301       normalize(margin + x_step*10), normalizeheight(margin - 10),
302       normalize(margin + x_step*11), normalizeheight(margin),
303       normalize(margin + x_step*11), normalizeheight(margin - 10),
304       //44
305       //Pointmarks y-axis
306       normalize(margin), normalizeheight(margin),
307       normalize(margin - 10), normalizeheight(margin),
308       normalize(margin), normalizeheight(margin + y_step),
309       normalize(margin - 10), normalizeheight(margin + y_step),
310       normalize(margin), normalizeheight(margin + y_step*2),
311       normalize(margin - 10), normalizeheight(margin + y_step*2),
312       normalize(margin), normalizeheight(margin + y_step*3),
313       normalize(margin - 10), normalizeheight(margin + y_step*3),
314       normalize(margin), normalizeheight(margin + y_step*4),
315       normalize(margin - 10), normalizeheight(margin + y_step*4),
316       normalize(margin), normalizeheight(margin + y_step*5),
317       normalize(margin - 10), normalizeheight(margin + y_step*5)
318     ]);
319     let n = 68 ;
320     createBindBuffer(vertices);
321     return n;
322   }
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
323
324   function createBindBuffer(array){
325     let vertexBuffer = gl.createBuffer();
326
327     gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
328     gl.bufferData(gl.ARRAY_BUFFER, array, gl.STATIC_DRAW);
329
330     program.position = gl.getAttribLocation(program, 'position');
331     gl.enableVertexAttribArray(program.position);
332     gl.vertexAttribPointer(program.position, 2, gl.FLOAT, false, 0, 0);
333   }
334
335   function normalize(value){
336     return value = ((value/view_box.width) * 2) - 1;
337   }
338
339   function normalizeheight(value){
340     return value = ((value/view_box.height) * 2) - 1;
341   }
342 </script>
343 </body>
344 </html>
```

Listing A.4 (cont.)**:** Full source code of the WebGL example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <script src="pixi.min.js"></script>
  <script src="graphics-extras.js"></script>
  <title>PixiJS</title>
</head>

<body>

<script>
  //Data array for computation
  const chartData = [
    ["2012-01", 28366, 23274], ["2012-02", 27050, 21732],
    ["2012-03", 29463, 23845], ["2012-04", 32561, 28732],
    ["2012-05", 28050, 24023], ["2012-06", 30100, 26089],
    ["2012-07", 22343, 19026], ["2012-08", 21506, 17903],
    ["2012-09", 24664, 19387], ["2012-10", 28842, 23490],
    ["2012-11", 30621, 25873], ["2012-12", 36254, 28490]]

  //Setting margin, size for later computation of corrdinates for drawing
  var view_box = { width: 950, height: 450 };
  var margin = 100;
  var size = { width: view_box.width - margin * 2,
    height: view_box.height - margin * 2 };

  //Calculation of distance between points on x and y-axis
  const x_step = size.width / (chartData.length - 1);
  const y_step = size.height / 5;

  //Function to gradually change the distance between the points on the y axis
  function calc_y_data(y) {
    return margin + (size.height - (((y - 15000) / 25000) * size.height));
  }

  PIXI.settings.SCALE_MODE = 1;
  var ratio = view_box.width / view_box.height;
  let chart = new PIXI.Application({ width: view_box.width,
    height: view_box.height, backgroundColor: 0xffffff });
  document.body.appendChild(chart.view);

  let graph = new PIXI.Graphics();

  //draw x and y lines
  graph.lineStyle(2, 0x000000);
  graph.moveTo(margin, margin);
  graph.lineTo(margin, margin + size.height);
  graph.lineTo(margin + size.width, margin + size.height);
```

Listing A.5: Full source code of the PixiJS example shown in Figure 3.1.

```
52
53    //draw line markings x
54    graph.moveTo(margin, margin + size.height);
55    var x = margin;
56    var index = 0;
57    while (index < chartData.length) {
58      graph.lineTo(x, 360);
59      var text = new PIXI.Text(chartData[index][0],
60        { fontFamily: 'Arial, sans-serif', fontSize: 15,
61          fill: 0x000000, align: 'start' });
62      text.x = x;
63      text.y = margin + size.height + 15;
64      text.angle = 45;
65      chart.stage.addChild(text);
66      x = x + x_step;
67      graph.moveTo(x, margin + size.height);
68
69      index++;
70    }
71
72    var sales_val = new Array('15,000', '20,000', '25,000',
73      '30,000', '35,000', '40,000');
74
75    //draw line markings y
76    graph.moveTo(margin, margin + size.height);
77    var y = margin + size.height;
78    var index = 0;
79    while (index < 6) {
80      graph.lineTo(margin - 10, y);
81      var text = new PIXI.Text(sales_val[index],
82        { fontFamily: 'Arial, sans-serif', fontSize: 15,
83          fill: 0x000000, align: 'end' });
84      text.x = margin / 3;
85      text.y = y - 9;
86      chart.stage.addChild(text);
87      y = y - y_step;
88      graph.moveTo(margin, y);
89      index++;
90    }
91
92
93    //drawing line A with rects
94    graph.lineStyle(1, 0xc6876f);
95    graph.moveTo(margin, calc_y_data(chartData[0][1]))
96    graph.drawRect(margin - 4, calc_y_data(chartData[0][1]) - 4, 8, 8)
97    var x = margin + x_step;
98    var index = 1;
99    while (index < chartData.length) {
100     graph.lineTo(x, calc_y_data(chartData[index][1]));
101     graph.drawRect(x - 4, calc_y_data(chartData[index][1]) - 4, 8, 8)
102     x = x + x_step;
103     index++;
104   }
```

Listing A.5 (cont.)**:** Full source code of the PixiJS example.

```
105
106   //drawing line B with rects
107   graph.lineStyle(1, 0x6e62dd);
108   graph.moveTo(margin, calc_y_data(chartData[0][2]))
109   graph.drawRegularPolygon(margin, calc_y_data(chartData[0][2]), 5, 4, 0);
110   var x = margin + x_step;
111   var index = 1;
112   while (index < chartData.length) {
113     graph.lineTo(x, calc_y_data(chartData[index][2]));
114     graph.drawRegularPolygon(x, calc_y_data(chartData[index][2]), 5, 4, 0);
115     x = x + x_step;
116     index++;
117   }
118
119   //additional captions
120
121   var text = new PIXI.Text("Month",
122     { fontFamily: 'Arial, sans-serif', fontSize: 15,
123       fill: 0x000000, align: 'start' });
124   text.x = view_box.width / 2;
125   text.y = view_box.height - margin / 4;
126   chart.stage.addChild(text);
127
128   var text = new PIXI.Text("Sales in €",
129     { fontFamily: 'Arial, sans-serif', fontSize: 15,
130       fill: 0x000000, align: 'start' });
131   text.x = margin / 6;
132   text.y = view_box.height / 2;
133   text.angle = 270;
134   chart.stage.addChild(text);
135
136   graph.lineStyle(1, 0xc6876f);
137   graph.drawRect(margin + x_step * 2, margin, 8, 8)
138
139   var text = new PIXI.Text("Salesperson A",
140     { fontFamily: 'Arial, sans-serif', fontSize: 15,
141       fill: 0x000000, align: 'start' });
142   text.x = margin + x_step * 2 + 20;
143   text.y = margin - 5;
144   chart.stage.addChild(text);
145
146   graph.lineStyle(1, 0x6e62dd);
147   graph.drawRegularPolygon(margin + x_step * 5, margin + 4, 5, 4, 0);
148
149   var text = new PIXI.Text("Salesperson B",
150     { fontFamily: 'Arial, sans-serif', fontSize: 15,
151       fill: 0x000000, align: 'start' });
152   text.x = (margin + x_step * 5) + 20;
153   text.y = margin - 5;
154   chart.stage.addChild(text);
```

Listing A.5 (cont.)**:** Full source code of the PixiJS example.

```
155
156   resize();
157   graph.endFill();
158   chart.stage.addChild(graph);
159
160   // resize function, found on https://stackoverflow.com/questions/30554533/
161   function resize() {
162     if (window.innerWidth / window.innerHeight >= ratio) {
163       var w = window.innerHeight * ratio;
164       var h = window.innerHeight;
165     } else {
166       var w = window.innerWidth;
167       var h = window.innerWidth / ratio;
168     }
169     chart.view.style.width = w + 'px';
170     chart.view.style.height = h + 'px';
171   }
172
173   window.onresize = resize;
174 </script>
175
176 </body>
177 </html>
```

Listing A.5 (cont.)**:** Full source code of the PixiJS example.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script src="two.js"></script>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<style>
  body{
    margin : 0;
  }
  #chart {
    /* define proportions of the graph */
    width: 100vw;
    height: 50vw;

    /* prevent appearance of the scrollbar by not
      allowing the chart height to grow more than window size
     */
    max-height: 100vh;
  }
</style>

<div id="chart"></div>

<body>
<script>
  //Initialize Two to start drawing
  let view_box = {width: 950, height: 500};
  let canvas = document.getElementById('chart');
  let two = new Two({
    fullscreen: true,
    //fitted: true,
    width: view_box.width,
    height:view_box.height,
    autostart: true,
    type: Two.Types.canvas
  }).appendTo(canvas);

  //Data array for computation
  const chartData = [
    ["2012-01",28366,23274],["2012-02",27050,21732],
    ["2012-03",29463,23845],["2012-04",32561,28732],
    ["2012-05",28050,24023],["2012-06",30100,26089],
    ["2012-07",22343,19026],["2012-08",21506,17903],
    ["2012-09",24664,19387],["2012-10",28842,23490],
    ["2012-11",30621,25873],["2012-12",36254,28490]]

  //Setting margin, size, and ratio for later computation of coordinates
  var margin = 200;
  var size = {width: view_box.width - margin * 2,
    height: view_box.height - margin * 2};
  let ratio = view_box.width / view_box.height;
```

Listing A.6: Full source code of the Two.js example shown in Figure 3.2.

```
55
56  //Calculation of distance between points on x and y-axis
57  let x_step = size.width / (chartData.length - 1);
58  let y_step = size.height / 5;
59
60  //Function calculate the graph spcific values for the money amounts
61  function calc_y_data(y) {
62    return margin + (size.height - (((y-15000) / 25000) * size.height));
63  }
64
65  canvas.setAttribute("viewBox", "0 0 " + view_box.width +
66    " " + view_box.height + "");
67
68  function drawline(){
69    //x-axis
70    two.makeLine(margin, margin+size.height, margin + size.width,
71      margin+size.height);
72    //y-axis
73    two.makeLine(margin, margin, margin, margin+size.height);
74  }
75
76  function drawMarkingsX(){
77    for(let i = margin; i <= margin+size.width; i += x_step)
78    {
79      two.makeLine(i, margin+size.height, i, margin+size.height+10);
80    }
81  }
82
83  function drawMarkingsY(){
84    for(let i = margin; i <= margin+size.height; i += y_step)
85    {
86      two.makeLine(margin, i, margin-10, i);
87    }
88  }
89
90  function drawPointsXSalesA(){
91    let count = 0;
92    for(let i = margin; i <= margin+size.width; i += x_step)
93    {
94      if(i < margin+size.width - x_step)
95      {
96        let line = two.makeLine(i, calc_y_data(chartData[count][1]),
97          i+x_step, calc_y_data(chartData[(count+1)][1]));
98        line.stroke = "#c6876f";
99      }
100     let point = two.makePoints(i, calc_y_data(chartData[count][1]));
101     point.size = size.height/50;
102     point.stroke = "#c6876f";
103     two.makeLine(i, margin+size.height, i, margin+size.height+10);
104     count++;
105   }
106 }
```

Listing A.6 (cont.)**:** Full source code of the Two.js example.

```
107
108   function drawPointsXSalesB(){
109     let count = 0;
110     for(let i = margin; i <= margin+size.width; i += x_step)
111     {
112       if(i < margin+size.width - x_step)
113       {
114         let line = two.makeLine(i, calc_y_data(chartData[count][2]),
115           i+x_step, calc_y_data(chartData[(count+1)][2]));
116         line.stroke = "#6e62dd";
117       }
118       let point = two.makePoints(i, calc_y_data(chartData[count][2]));
119       point.size = size.height/50;
120       point.stroke = "#6e62dd";
121       two.makeLine(i, margin+size.height, i, margin+size.height+10);
122       count++;
123     }
124   }
125
126   function drawTopTextSalesA(){
127     let point = two.makePoints((x_step*3)-15, margin);
128     point.size = size.height/50;
129     point.stroke = "#c6876f";
130     let salesA = two.makeText("Salesman A", (x_step*3), margin);
131     salesA.alignment = "left";
132     salesA.size = size.height/30;
133   }
134
135   function drawTopTextSalesB(){
136     let point = two.makePoints((x_step*5)-15, margin);
137     point.size = size.height/50;
138     point.stroke = "#6e62dd";
139     let salesA = two.makeText("Salesman B", (x_step*5), margin);
140     salesA.alignment = "left";
141     salesA.size = size.height/30;
142   }
143
144   //Drawing the text on the x-axis
145   function drawXText() {
146     let text1 = two.makeText("2012-01", margin, margin + size.height + 25);
147     text1.alignment = "left";
148     text1.rotation = 0.7;
149     text1.size = size.height / 30;
150     let text2 = two.makeText("2012-02", margin + (x_step),
151       margin + size.height + 25);
152     text2.alignment = "left";
153     text2.rotation = 0.7;
154     text2.size = size.height / 30;
155     let text3 = two.makeText("2012-03", margin + (x_step * 2),
156       margin + size.height + 25);
157     text3.alignment = "left";
158     text3.rotation = 0.7;
159     text3.size = size.height / 30;
```

Listing A.6 (cont.)**:** Full source code of the Two.js example.

```
160     let text4 = two.makeText("2012-04", margin + (x_step * 3),
161       margin + size.height + 25);
162     text4.alignment = "left";
163     text4.rotation = 0.7;
164     text4.size = size.height / 30;
165     let text5 = two.makeText("2012-05", margin + (x_step * 4),
166       margin + size.height + 25);
167     text5.alignment = "left";
168     text5.rotation = 0.7;
169     text5.size = size.height / 30;
170     let text6 = two.makeText("2012-06", margin + (x_step * 5),
171       margin + size.height + 25);
172     text6.alignment = "left";
173     text6.rotation = 0.7;
174     text6.size = size.height / 30;
175     let text7 = two.makeText("2012-07", margin + (x_step * 6),
176       margin + size.height + 25);
177     text7.alignment = "left";
178     text7.rotation = 0.7;
179     text7.size = size.height / 30;
180     let text8 = two.makeText("2012-08", margin + (x_step * 7),
181       margin + size.height + 25);
182     text8.alignment = "left";
183     text8.rotation = 0.7;
184     text8.size = size.height / 30;
185     let text9 = two.makeText("2012-09", margin + (x_step * 8),
186       margin + size.height + 25);
187     text9.alignment = "left";
188     text9.rotation = 0.7;
189     text9.size = size.height / 30;
190     let text10 = two.makeText("2012-10", margin + (x_step * 9),
191       margin + size.height + 25);
192     text10.alignment = "left";
193     text10.rotation = 0.7;
194     text10.size = size.height / 30;
195     let text11 = two.makeText("2012-11", margin + (x_step * 10),
196       margin + size.height + 25);
197     text11.alignment = "left";
198     text11.rotation = 0.7;
199     text11.size = size.height / 30;
200     let text12 = two.makeText("2012-12", margin + (x_step * 11),
201       margin + size.height + 25);
202     text12.alignment = "left";
203     text12.rotation = 0.7;
204     text12.size = size.height / 30;
205
206     let sidetext = two.makeText("Month", margin + (x_step*6),
207       margin+ size.height + size.height*0.2);
208     sidetext.alignment = 'right';
209     sidetext.size = size.height/25;
210   }
```

Listing A.6 (cont.)**:** Full source code of the Two.js example.

```
211
212   //Drawing the text on the y-axis
213   function drawYText(){
214     let text1 = two.makeText("40,000", margin-30, margin);
215     text1.alignment = 'right';
216     text1.size = size.height/30;
217     let text2 = two.makeText("35,000", margin-30, margin+y_step);
218     text2.alignment = 'right';
219     text2.size = size.height/30;
220     let text3 = two.makeText("30,000", margin-30, margin+y_step*2);
221     text3.alignment = 'right';
222     text3.size = size.height / 30;
223     let text4 = two.makeText("25,000", margin-30, margin+y_step*3);
224     text4.alignment = 'right';
225     text4.size = size.height / 30;
226     let text5 = two.makeText("20,000", margin-30, margin+y_step*4);
227     text5.alignment = 'right';
228     text5.size = size.height / 30;
229     let text6 = two.makeText("15,000", margin-30, margin+y_step*5);
230     text6.alignment = 'right';
231     text6.size = size.height / 30;
232
233     let sidetext = two.makeText("Sales in €", margin-size.height*0.3,
234       margin+y_step*3-20);
235     sidetext.rotation = -1.57;
236     sidetext.alignment = 'left';
237     sidetext.size = size.height / 25;
238   }
239
240   drawTopTextSalesB();
241   drawTopTextSalesA();
242   drawPointsXSalesB();
243   drawPointsXSalesA();
244   drawMarkingsY();
245   drawMarkingsX();
246   drawline();
247   drawXText();
248   drawYText();
```

Listing A.6 (cont.)**:** Full source code of the Two.js example.

```
249
250   resize();
251
252   //Function to resize the graph when window gets resized
253   function resize() {
254     if (window.innerWidth / window.innerHeight >= ratio) {
255       var w = window.innerHeight * ratio;
256       var h = window.innerHeight;
257     } else {
258       var w = window.innerWidth;
259       var h = window.innerWidth / ratio;
260     }
261
262     view_box.width = w;
263     view_box.height = h;
264     size = {width: view_box.width - margin * 2,
265       height: view_box.height - margin * 2};
266     ratio = view_box.width / view_box.height;
267     margin = view_box.width/8;
268     canvas.setAttribute("viewBox", "0 0 " + view_box.width +
269       " " + view_box.height + "");
270     x_step = size.width / (chartData.length - 1);
271     y_step = size.height / 5;
272     two.clear();
273     drawline();
274     drawPointsXSalesA();
275     drawPointsXSalesB();
276     drawMarkingsX();
277     drawMarkingsY();
278     drawTopTextSalesA();
279     drawTopTextSalesB();
280     drawXText();
281     drawYText();
282     two.update();
283   }
284
285   window.onresize = resize;
286
287 </script>
288
289 </body>
290 </html>
```

Listing A.6 (cont.)**:** Full source code of the Two.js example.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<script src="d3_v7.js"></script>

<style>
  body {
    margin: 0;
  }

  #chart {
    /* define proportions of the graph */
    width: 100vw;
    height: 50vw;

    /* prevent appearance of the scrollbar by not
      allowing the chart height to grow more than window size
     */
    max-height: 100vh;
  }
</style>

<body>
  <svg id="chart"></svg>

  <!-- the max height atrr. prevents a scroll-bar to happen but may distort
    the graph if rendered in too narrow of a window -->

<script>
  function min(a, b) {
    return (a < b) ? a : b;
  }

  /******** init the document *********/
  var el = document.getElementById('p');
  //Setting margin, size for later computation of coordinates
  const view_box = { width: 950, height: 450 };
  const margin = 100;
  const size = { width: view_box.width - margin * 2,
    height: view_box.height - margin * 2 };

  // Append initial svg code to our svg object
  const svg = d3.select("#chart")
    .append("svg")
    .attr("viewBox", "0 0 " + view_box.width + " " + view_box.height)
    .append("g")
    // move the origin to realize the margin
    .attr("transform", "translate(" + margin + "," + margin + ")");
```

Listing A.7: Full source code of the D3 example shown in Figure 3.3.

```
49
50
51  /******** prepare the data *********/
52  const months = ["2012-01", "2012-02", "2012-03", "2012-04",
53    "2012-05", "2012-06", "2012-07", "2012-08", "2012-09",
54    "2012-10", "2012-11", "2012-12"];
55  const sales_a = [28366, 27050, 29463, 32561, 28050, 30100, 22343, 21506,
56    24664, 28842, 30621, 36254];
57  const sales_b = [23274, 21732, 23845, 28732, 24023, 26089, 19026, 17903,
58    19387, 23490, 25873, 28490];
59
60  /** d3 needs the data in the form
61  * [
62  *  d1 = {x: x1, y: y1},
63  *  d2 = {x: x2, y: y2},
64  *  d3 = {x: x3, y: y3},
65  *  d4 = {x: x4, y: y4},
66  *  d5 = {x: x5, y: y5} ]
67  */
68  const points_a = sales_a.map(
69    function (el, ind) { return { x: months[ind], y: el } });
70  const points_b = sales_b.map(
71    function (el, ind) { return { x: months[ind], y: el } });
72
73
74  /******** X Axis.  *********/
75  // Making an ordinal axis in d3 with text rather than number is not very simple:
76
77  // firt the axis needs a function that linearily projects x values from
78  // their domain to the domain [0, size.width]
79  const xscl = d3.scaleLinear()
80    // The axis can display ticks that go from 0 to months.length - 1
81    .domain([0, months.length - 1])
82    // The axis goes from point 0 to point size.width
83    .range([0, size.width]);
84
85  // given our function, we ask the ticks to display months rather than
86  // actual numbers in [0,12] = [0, months.length - 1]
87  var xAxis = d3.axisBottom(xscl)
88    .tickFormat(function (d) { return months[d]; })
89  // if tick was 1 we render it as months[1]...
90
91  svg.append("g") // render the axis
92    .attr("transform", "translate(0," + size.height + ")") // translate it up
93    // so that it starts where y axis ends
94    .call(xAxis)                                    // render the axis
95    .selectAll("text")                              // adjust the ticks
96    .style("text-anchor", "end")
97    .attr("dx", "5em")
98    .attr("dy", ".15em")
99    .attr("transform", "rotate(50)")
```

Listing A.7 (cont.)**:** Full source code of the D3 example.

```
100
101
102   /******** Y Axis *********/
103   const yscl = d3.scaleLinear()
104     .domain([15000, 40000])
105     .range([size.height, 0]);
106
107   svg.append("g")
108     .call(d3.axisLeft(yscl)
109       .ticks(6));
110
111
112   /* this creates a function that tells d3 how to interpret the data:
113   * this will be used like line(points)
114   * where each point in our case is an (x_value, y_value) couple
115   * here we specify that given the list of couples, the line is created by
116   * just applying the x scale to x_values and y scale to the y_values
117   */
118   var line = d3.line()
119     .x(function (d) { return xscl(months.indexOf(d.x)); }) // apply the
120     // x scale to the x data
121     .y(function (d) { return yscl(d.y); }) // apply the y scale to the y data
122
123   /*********** definition of the markers ***********/
124   //so that they can later be used when plotting
125   svg.append('defs')
126     .append('marker')
127     .attr('id', 'marker_a')
128     .attr('viewBox', [0, 0, 15, 15])
129     .attr('refX', 7)
130     .attr('refY', 7)
131     .attr('markerWidth', 15)
132     .attr('markerHeight', 15)
133     .append('rect')
134     .attr('x', 3.5)
135     .attr('y', 3.5)
136     .attr('width', 7)
137     .attr('height', 7)
138     .attr('stroke', "#b58471")
139     .attr('stroke-width', "2")
140     .style('fill', 'transparent');
141
142   svg.append('defs')
143     .append('marker')
144     .attr('id', 'marker_b')
145     .attr('viewBox', [0, 0, 15, 15])
146     .attr('refX', 7)
147     .attr('refY', 7)
148     .attr('markerWidth', 15)
149     .attr('markerHeight', 15)
150     .append('rect')
151     .attr('x', 6)
152     .attr('y', -4)
153     .attr('width', 7)
154     .attr('height', 7)
155     .attr('stroke', "#665ec4")
156     .attr('stroke-width', "2")
157     .attr('transform', "rotate(45)")
158     .style('fill', 'transparent');
```

Listing A.7 (cont.)**:** Full source code of the D3 example.

```
159
160   /********** create the actual plots **********/
161   svg.append("path")
162     .attr("class", "line")
163     .attr("d", line(points_a))
164     .style("fill", "none")
165     .style("stroke", "#b58471")
166     .style("stroke-width", 1)
167     .attr('marker-start', 'url(#marker_a)')
168     .attr('marker-mid', 'url(#marker_a)')
169     .attr('marker-end', 'url(#marker_a)');
170
171   svg.append("path")
172     .attr("class", "line")
173     .attr("d", line(points_b))
174     .style("fill", "none")
175     .style("stroke", "#665ec4")
176     .style("stroke-width", 1)
177     .attr('marker-start', 'url(#marker_b)')
178     .attr('marker-mid', 'url(#marker_b)')
179     .attr('marker-end', 'url(#marker_b)');
180
181   // x axis label
182   svg.append("text")
183     .attr("x", -size.height / 2)
184     .attr("y", -60)
185     .attr("text-anchor", "middle")
186     .attr("font-size", 13)
187     .text('Sales in €')
188     .attr('transform', "rotate(-90)");
189
190   // y axis label
191   svg.append("text")
192     .attr("x", size.width / 2)
193     .attr("y", size.height + 80)
194     .attr("text-anchor", "middle")
195     .attr("font-size", 13)
196     .text('Month');
```

Listing A.7 (cont.)**:** Full source code of the D3 example.

```
197
198   /***************** legend *****************/
199   svg.append('rect')
200     .attr('x', 85)
201     .attr('y', -8.5)
202     .attr('width', 8)
203     .attr('height', 8)
204     .attr('stroke', "#b58471")
205     .attr('stroke-width', "2")
206     .style('fill', 'transparent');
207
208   svg.append("text")
209     .attr("x", 100)
210     .attr("y", 0)
211     .attr("font-size", 13)
212     .text('Salesperson A');
213
214   svg.append('rect')
215     .attr('x', 0)
216     .attr('y', 0)
217     .attr('width', 8)
218     .attr('height', 8)
219     .attr('stroke', "#665ec4")
220     .attr('stroke-width', "2")
221     .style('fill', 'transparent')
222     .attr('transform', "translate(203 -8.5) rotate(45 4 4)  ");
223
224   svg.append("text")
225     .attr("x", 220)
226     .attr("y", 0)
227     .attr("font-size", 13)
228     .text('Salesperson B');
229
230 </script>
231
232 </body>
233 </html>
```

Listing A.7 (cont.)**:** Full source code of the D3 example.

Bibliography

Andrews, Keith [2022]. *Information Visualisation: Lecture Notes*. 16 Mar 2022. https://courses.isds.tugraz.at/ivis/ivis.pdf (cited on page 1).

Bitwise Creative [2022]. *Dynamically resize the pixi stage*. 31 May 2022. https://stackoverflow.com/questions/30554533/dynamically-resize-the-pixi-stage-and-its-contents-on-window-resize-and-window (cited on page 33).

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *D³ Data-Driven Documents*. IEEE Transactions on Visualization and Computer Graphics 17.12 (2011), pages 2301–2309. doi:10.1109/TVCG.2011.185 (cited on page 42).

Bostock, Mike [2021]. *D3.js - Data Driven Documents*. 15 May 2021. https://d3js.org/ (cited on pages 1, 42).

Brandel, Jonathan [2022]. *Two.js*. 15 May 2022. https://two.js.org/ (cited on pages 1, 36).

Greggman [2022]. *WebGL Fundamentals*. 22 May 2022. https://webglfundamentals.org/ (cited on page 20).

Khronos [2022]. *WebGL*. 15 May 2022. https://webgl.org/ (cited on pages 1, 19).

OpenGL [2022]. *OpenGL*. 15 May 2022. https://khronos.org/opengl/wiki/ (cited on page 19).

PixiJS [2022a]. *PIXI.Text*. 14 May 2022. https://pixijs.download/dev/docs/PIXI.Text.html (cited on page 29).

PixiJS [2022b]. *PixiJS API Documentation*. 14 May 2022. https://pixijs.download/release/docs/index.html (cited on pages 1, 29).

Ros, Irene [2014]. *Working with D3.js and Canvas: When and How*. 17 Jul 2014. https://bocoup.com/blog/d3js-and-canvas/ (cited on page 42).

W3C [2010a]. *Scalable Vector Graphics (SVG)*. 2010. https://w3.org/Graphics/SVG/ (cited on page 3).

W3C [2010b]. *W3 Wiki - Elements/Canvas*. 30 Nov 2010. https://w3.org/html/wiki/Elements/canvas/ (cited on pages 12–13).

W3C [2011]. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. 16 Aug 2011. https://w3.org/TR/SVG11/ (cited on page 3).

W3C [2018]. *Scalable Vector Graphics (SVG) 2*. W3C Candidate Recommendation. 04 Oct 2018. https://w3.org/TR/SVG2/ (cited on page 3).

Wattenberger, Amelia [2019]. *Fullstack D3 and Data Visualization*. Fullstack.io, 2019. ISBN 0991344650 (cited on page 42).

WebGL [2022]. *WebGL Support*. 2022. https://get.webgl.org/ (cited on page 19).

WHATWG [2022]. *HTML Living Standard - The Canvas Element*. 10 May 2022. https://html.spec.whatwg.org/multipage/canvas.html#the-canvas-element/ (cited on pages 1, 12).

Whitney, Justin [2013]. *Surviving the Zombie Apocalypse: Manipulating SVG with JavaScript*. 24 Jun 2013. https://sitepoint.com/surviving-the-zombie-apocalypse-manipulating-svg-with-javascript/ (cited on pages 1, 5).