Fast Design Space Rendering of Scatterplots

Simo Santala[1] and Antti Oulasvirta[1] and Tino Weinkauf[2]

[1] Department of Communications and Networking, Aalto University, Helsinki, Finland
[2] Department of Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

The design space of scatterplots consists of a number of parameters such as marker size and shape, image width and aspect ratio, and opacity. Different parameters yield different visual impressions of the scatterplot. Perceptual optimization of scatterplots means finding the best design parameters to support a given visualization task. This requires rendering thousands of design variations. We describe an image-based method for rendering scatterplots, which is tailored to this scenario: it enables quick updates of the design by re-using previously calculated intermediate results, and is independent of the data set size. Our approach outperforms the classic method of rendering scatterplots, i.e., drawing each marker individually onto an image, and can therefore dramatically speed up the perceptual optimization of scatterplots. We provide an open-source implementation and an online service for our method.

CCS Concepts

• ***Computing methodologies*** → ***Rendering;*** • ***Human-centered computing*** → ***Visualization design and evaluation methods;*** *Graph drawings; Visualization toolkits;*

1. Introduction

Scatterplots are a ubiquitous visualization technique. Used by visualization experts and novices alike, they can be found not only in academic papers and analysis tools, but also in mass and social media, and other communications to the general public. While designing them seems relatively straightforward, it still requires choosing a number of design parameters such as marker type and size, opacity, color, image width and aspect ratio. They all influence the final visual impression quite strongly [CDM82,STMT12,SMT13]. With the wrong set of parameters, insights into the data can be obscured. Choosing the "right" set of parameters is not trivial for novices and even visualization experts may find this difficult or at least cumbersome.

Scatterplots support different visualization tasks such as estimating the correlation between two variables, detecting outliers, identifying clusters, and much more [Mun14]. Each task requires a different design of the scatterplot: a different choice of its parameters to support that particular task best. For example, outliers are easier to detect when large and opaque markers are used.

Perceptual optimization of scatterplots aims at automating the choice of design parameters [MPOW17]. Given a visualization task and a data set, the system renders the data with varying parameters thousands of times and scores each rendered image based on different perceptual aspects. The set of parameters with the best score is suggested to the user for analyzing the data further. The set of all parameter combinations is called the *design space*.

We continue the work of Luana Micallef et al. [MPOW17], where the optimization of scatterplots is based on perceptual metrics of the visual attributes of the rendered image, including contrast, amounts of overplotting, and others. A difficulty with the approach is that rendering every possible design individually is computationally expensive and slows down with increasing data set size. For example, Micallef et al. [MPOW17] report a runtime of 22 minutes to optimize a scatterplot with 10000 points.

This paper proposes a different approach to rendering scatterplots which dramatically decreases the rendering times and makes perceptual optimization of scatterplots possible in under a minute. Our method is image-based and therefore independent of the data set size. We map all data points into a high-definition point density matrix, which serves as a precursor to the final image. This is similar to previous approaches using high-definition textures to render parallel coordinate plots [JLJC05] and a multi-resolution hierarchy to render 3D particles [FSW09]. Three operations are sequentially performed on this matrix to obtain a final rendering result: (i) scaling to the desired resolution and aspect ratio, (ii) imprinting the desired marker shape and size, and (iii) computing the final opacity for each pixel. Optimization methods require rendering similar designs, e.g., only the marker opacity changes, but all other parameters are the same. Our approach supports this by caching intermediate results of the algorithmic sequence. It is tailored for rendering entire design spaces.

We give the following contributions:

- We provide an algorithm to efficiently render a large number of scatterplot designs for the same data by using an image-based approach.
- We enable rapid interactive and automated exploration of all scatterplot design parameters even for very large data sets. In particular, this paves the way for the perceptual optimization of scatterplots to become a more practically feasible tool.
- We test our method on several data sets and provide an open source implementation, which we also run as an online service to lower the barrier for trying out scatterplot optimization.

DOI: 10.2312/evs.20201058

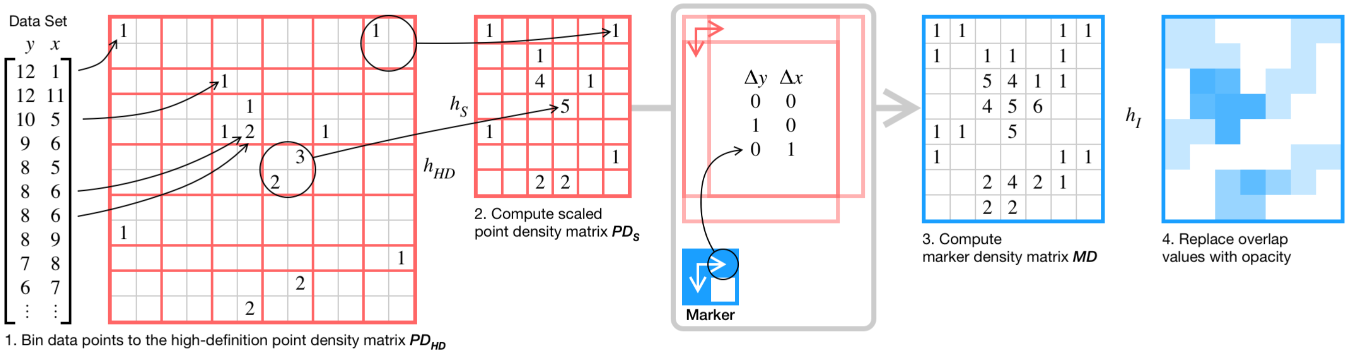
Data Set
y x
1. Bin data points to the high-definition point density matrix PD_{HD}
h_{HD}
h_S
2. Compute scaled point density matrix PD_S
Δy Δx
Marker
3. Compute marker density matrix MD
h_I
4. Replace overlap values with opacity

Figure 1: *Our algorithm divides rendering a scatterplot design into four steps. The results from each step can be cached for faster redrawing.*

2. Fast Design Space Rendering

We consider scatterplot designs in terms of plot dimensions (h_I, w_I), marker size, marker shape, and opacity. These dimensions form a multi-dimensional design space. Optimizing scatterplots involves finding the optimal design in this design space. This can be done by either evaluating the entire design space exhaustively, or by sampling it according to some heuristic. In both cases, several designs need to be rendered in rapid succession.

To enable fast rendering of scatterplots with the same data but different designs, we generate a high-resolution density matrix of the data points (Section 2.1). It allows all further operations to be independent of the original data set size.

We use three separate methods to efficiently update aspect ratio, marker size, and marker opacity. These methods build on each other. First, the point density matrix is scaled according to the desired aspect ratio and image resolution (Section 2.2). Second, the scaled point density matrix is added to itself multiple times at offsets defined by the marker size and shape (Section 2.3). Third, the resulting marker density matrix is converted to an alpha channel (Section 2.4). For optimal performance when rendering entire design spaces, the intermediate results from each step can be cached for faster redrawing of adjacent designs. Our approach enables us to iterate over a design space much faster than the approach of Micallef et al. [MPOW17], where each design was rendered from scratch.

2.1. High-definition Point Density Matrix

The *high-definition point density matrix* is generated by normalizing and scaling the data point coordinates to the matrix dimensions (h_{HD}, w_{HD}), rounding them to integers, and binning each to a cell with matching coordinates. The final value of each cell in the matrix denotes how many data points land in that cell (Algorithm 1).

2.2. Updating Aspect Ratio and Resolution: Point Density Scaling

The first step of rendering is downscaling the high-definition point density matrix to a resolution that fits in the final image (w_S, h_S), but is a bit smaller than the final resolution to account for marker size. We bin each cell in the high-definition matrix to the closest

Algorithm 1 High-definition Point Density Matrix

```
Input: Data points $\mathbf{Y}[], \mathbf{X}[]$, matrix dimensions $h_{HD}, w_{HD}$
 1: $\mathbf{R}[] = \texttt{round}(\texttt{normalize}(\mathbf{Y}) \cdot (h_{HD} - 1))$
 2: $\mathbf{C}[] = \texttt{round}(\texttt{normalize}(\mathbf{X}) \cdot (w_{HD} - 1))$
 3: $\mathbf{PD_{HD}}[[]] = \texttt{zeros}(h_{HD}, w_{HD})$
 4: for all $r, c \in [\mathbf{R}, \mathbf{C}]^\top$ do
 5:     $\mathbf{PD_{HD}}[r][c] \mathrel{+}= 1$
 6: return $\mathbf{PD_{HD}}$
Output: High-definition Point Density Matrix
```

relative cell in the downscaled matrix, and sum up the values. This preserves the total number of points in the density matrix as each cell in the original matrix is assigned to a single cell in the resulting matrix (Figure 1, Algorithm 2).

Algorithm 2 Downscale Point Density Matrix

```
Input: $\mathbf{PD_{HD}}[[]], h_{HD}, w_{HD}, h_S, w_S$
 1: $\mathbf{PD_S}[[]] = \texttt{zeros}(h_S, w_S)$
 2: for all $r_S \in [0, h_S)$ do
 3:     $r_{HD0} = \texttt{round}(r_S \frac{h_{HD}}{h_S})$
 4:     $r_{HD1} = \texttt{round}((r_S + 1) \frac{h_{HD}}{h_S})$
 5:     for all $c_S \in [0, w_S)$ do
 6:         $c_{HD0} = \texttt{round}(c_S \frac{w_{HD}}{w_S})$
 7:         $c_{HD1} = \texttt{round}((c_S + 1) \frac{w_{HD}}{w_S})$
 8:         for all $r_{HD} \in [r_{HD0}, r_{HD1})$ do
 9:             for all $c_{HD} \in [c_{HD0}, c_{HD1})$ do
10:                 $\mathbf{PD_S}[r_S][c_S] \mathrel{+}= \mathbf{PD_{HD}}[r_{HD}][c_{HD}]$
11: return $\mathbf{PD_S}$
Output: Downscaled Point Density Matrix
```

2.3. Updating Marker: Painting with the Density Matrix

So far, each data point is represented by a single pixel in the downscaled point density matrix. We will now expand each pixel to a proper marker with a particular size and shape. This yields the *marker density matrix*, whose resolution is equal to the final image resolution (w_I, h_I).

We compute this by repeatedly adding the entire downscaled point density matrix to itself, but at offsets according to the marker

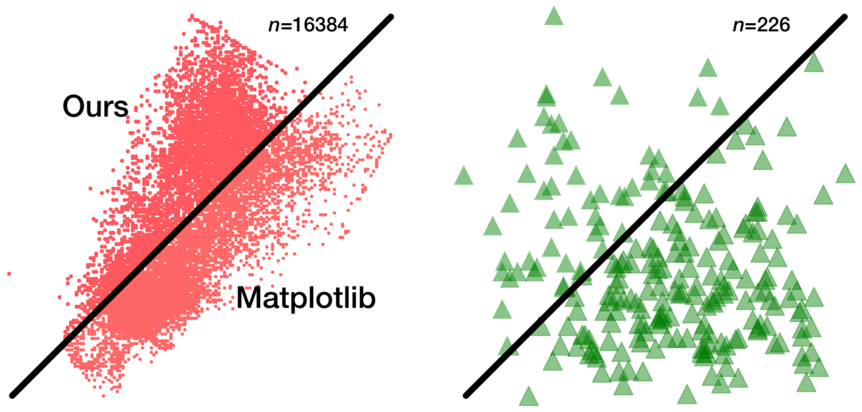

Figure 2: *The output of our algorithm matches that of traditional scatterplot renderers, but is able to update the design quickly, regardless of data set size.*

shape and size. The offsets correspond to the pixel coordinates of the opaque pixels in a marker. The result is a density matrix where each pixel value corresponds to the number of markers overlapping at that point (Figure 1). Rows 3–5 in Algorithm 3 represent matrix addition, which is highly parallelizable. In this approach, every marker is painted "simultaneously" pixel by pixel.

Algorithm 3 Marker Density Matrix

```
Input: $\mathbf{PD_S}[[]], h_S, w_S, \Delta\mathbf{Y}, \Delta\mathbf{X}$
1: $\mathbf{MD}[[]] = \texttt{zeros}(h_S + \texttt{max}(\Delta\mathbf{Y}), w_S + \texttt{max}(\Delta\mathbf{X}))$
2: for all $\Delta y, \Delta x \in [\Delta Y, \Delta X]^\top$ do
3:     for all $r_S \in [0, h_S)$ do
4:         for all $c_S \in [0, w_S)$ do
5:             $\mathbf{MD}[\Delta y + r_S][\Delta x + c_S] \mathrel{+}= \mathbf{PD_S}[r_S][c_S]$
6: return MD
Output: Marker Density Matrix
```

2.4. Updating Marker Opacity

Finally, we need to account for the marker opacity. We assume that all markers have the same opacity α, which allows us to compute the result of the alpha blending directly as a function of the number n_i of overlapping markers at pixel i as:

$$\alpha_i = 1 - (1-\alpha)^{n_i} \tag{1}$$

Our implementation uses lookup tables for this step, since this is faster than computing the exponential expression for each pixel, see Algorithm 4.

In essence, our method yields the same result as if we were to draw in a "classic" manner by painting a marker for each data point iteratively. See Figure 2 for an example. The only possible difference stems from potential aliasing artifacts introduced by the downscaling of the point density matrix. We address this in Section 3.3.

3. Evaluation

3.1. Time-Complexity and Measured Computation Times

Time-complexities for the described algorithms are as follows:

Algorithm 4 Opacity from a Lookup Table

```
Input: $\mathbf{MD}[[]], \mathbf{LUT}[], h_I, w_I$
1: $\mathbf{I}[[]] = \texttt{zeros}(h_I, w_I)$
2: for all $r \in [0, h_I)$ do
3:     for all $c \in [0, w_I)$ do
4:         $\mathbf{I}[r][c] = \mathbf{LUT}[\mathbf{MD}[r][c]]$
5: return I
Output: Alpha Channel
```

- Algorithm 1: $\mathcal{O}(n)$, where n equals the data set size.
- Algorithm 2: $\mathcal{O}(h_{HD}w_{HD})$, as each cell in $\mathbf{PD_{HD}}$ is visited only once.
- Algorithm 3: $\mathcal{O}(\delta h_S w_S)$, where δ equals the number of opaque pixels in a marker.
- Algorithm 4: $\mathcal{O}(h_I w_I)$

Therefore, after the algorithm 1 has been run once, the time complexity of rendering a new design is $\mathcal{O}(h_{HD}w_{HD} + \delta h_S w_S)$ if no other intermediate results have been cached. As we can see, this is independent of the data set size n, leading our algorithm to excel in rendering scatterplot design spaces of large data sets.

We implemented our algorithm with Python 3, using `OpenCV` Python bindings for matrix operations. All measurements of computation time were performed on the same computer: a 2019 16-inch MacBook Pro with a 2.3 GHz 8-core Intel Core i9 and 32 GB of RAM. All measurements of computation time were performed on a single thread. As seen in Table 1, the data set size has no practical effect on the computation time. This is in contrast to Micallef et al. [MPOW17] where each design is rendered from scratch.

3.2. Memory Requirements

Similarly to Abstract Rendering [CLW13], the original data set can be discarded from memory after the initial binning phase, or if the data set limits are known, the initial binning can be done in an out-of-core streaming fashion. The memory efficiency of the high-definition density matrix depends on its resolution (h_{HD}, w_{HD}), its bit depth b_{HD}, and the size n and bit depth b_{data} of the original

n	Our Method		Micallef et al. [MPOW17]	
	time (s)	designs/s	time (s)	designs/s
1 024	39.2	123.8	613.0	1.7
3 969	38.3	126.8	903.4	4.4
15 625	38.1	127.3	1998.6	7.8
62 500	39.0	124.5		
250 000	38.1	127.2		

Table 1: *We rendered 4851 different scatterplot designs as defined by the design space given in Micallef et al. [MPOW17]. Our high-definition point density matrix has a resolution of $h_{HD} = 4000$ and $w_{HD} = 6000$. Our method maintains a high rendering speed independent of the data set size n, whereas the method of Micallef et al. [MPOW17] becomes much slower for larger data sets.*

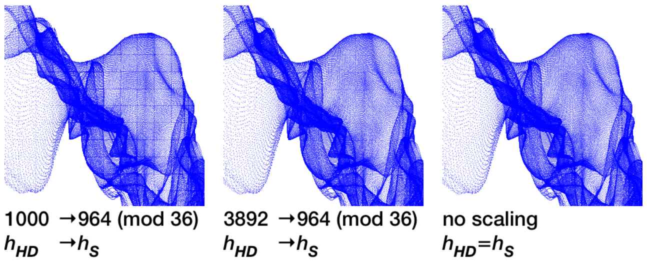

Figure 3: *Downscaling the high-definition point density matrix can introduce a grid-like aliasing effect. The number of line artifacts is equal to the remainder of the division $\frac{h_{HD}}{h_S}$. The effect may be mitigated by increasing the downscaling factor.*

data set. The density matrix will improve memory efficiency when $2nb_{data} > h_{HD}w_{HD}b_{HD}$. For example, a 4000×4000 pixel 16-bit density matrix would be equal in memory footprint to 32-bit data set with 4 million points: $2 \cdot 4000000 \cdot 32 = 4000^2 \cdot 16$.

In order to reduce the computational cost of updating a design, we cache the last used downscaled point density and marker density matrices, as well as the opacity lookup tables. To minimize the cache memory footprint, the design space is traversed in such an order that all designs with the same image dimensions and marker are evaluated consecutively. This ensures that when image dimensions or marker change, we can discard the previous scaled point density or marker density matrix. The memory footprint of the scaled point density and marker density matrices is similar to images with the same dimensions, meaning that keeping the latest intermediate results in memory is typically not too costly. An individual opacity lookup table requires $8 \cdot 2^{b_l}$ bits of memory, where b_l denotes the bit depth of the marker density matrix. If the number of different opacities is limited to a few hundred, keeping all lookup tables in memory should be feasible in most cases.

3.3. Accuracy

The main limitation of our approach is due to potential aliasing artifacts when downscaling the high-density matrix (Algorithm 2). If the size of the downscaled density matrix is not a factor of the size

n	E_α	E_r	I_μ	I_σ	$I_{\bar{\mu}}$	$I_{\bar{\sigma}}$	I_ℓ	I_p
1 024	0.6	0.5	0.0	0.0	0.0	0.1	0.1	0.0
3 969	0.4	0.3	0.0	0.0	0.0	0.1	0.0	0.0
15 625	0.2	0.2	0.0	0.0	0.0	0.1	0.0	0.0
62 500	0.3	0.7	0.0	0.1	0.0	0.1	0.0	0.0
250 000	0.3	0.6	0.0	0.1	0.0	0.1	0.0	0.0

Table 2: ***Mean Relative Error (%)*** *Using the quality metrics and design space defined in [MPOW17] for single-class scatterplots, we measured the error introduced by downscaling the point density matrix when $h_{HD} = 4000$ and $w_{HD} = 6000$. For each metric, the mean error was <1% of the difference between minimum and maximum values for that metric.*

n	E_α	E_r	I_μ	I_σ	$I_{\bar{\mu}}$	$I_{\bar{\sigma}}$	I_ℓ	I_p
1 024	4.3	2.3	0.0	0.1	0.0	0.1	0.1	0.0
3 969	2.4	1.9	0.0	0.0	0.0	0.1	0.0	0.0
15 625	1.4	1.1	0.0	0.1	0.0	0.1	0.0	0.0
62 500	0.5	0.9	0.0	0.1	0.0	0.1	0.0	0.0
250 000	0.4	0.7	0.0	0.1	0.0	0.2	0.0	0.0

Table 3: ***Standard Deviation of Relative Error (%)*** *In most cases, the error introduced by downscaling the point density matrix by a factor of 4 does not fluctuate. For small data sets, metrics that rely on point locations (denoted by E) are more sensitive to aliasing.*

of the high-density matrix, Moiré lines will appear. Their appearance is due to the underlying integer division during the scaling. The visibility of the Moiré lines depends on the ratio between the remainder and the quotient of that integer division: a smaller remainder and a larger quotient lead to less visible aliasing. In other words, the visibility of the Moiré lines is affected by the scaling factor: the larger the scaling factor, the less visible they are. See Figure 3 for an illustration.

As the primary purpose of our algorithm is rendering design spaces for perceptual optimization, it is important that these inaccuracies have minimal effect on the quality metrics used. Tables 2 and 3 show how the error is in most cases below 0.1%. If more accuracy is required, a larger high-definition point density matrix can be used.

4. Conclusion

We presented an image-based rendering method for scatterplots, which excels in rendering the same data many times with varying design parameters. Optimization algorithms for scatterplots such as Micallef et al. [MPOW17] benefit from this tremendously, since they have to generate and evaluate thousands of different designs.

We combined our rendering method with the perceptual optimization tool of Micallef et al. [MPOW17] and created an online tool for fast scatterplot optimization which showcases interactive ways to adjust scatterplot design parameters. For example, rather than adjusting design parameters directly, the user can select a meaningful analysis task, such as correlation estimation or outlier separation, and the design parameters are automatically adjusted for that task.

References

[CDM82] Cleveland W. S., Diaconis P., McGill R.: Variables on scatterplots look more highly correlated when the scales are increased. *Science 216*, 4550 (1982), 1138–1141. 1

[CLW13] Cottam J., Lumsdaine A., Wang P.: Overplotting: Unified solutions under abstract rendering. In *2013 IEEE International Conference on Big Data* (Oct 2013), pp. 9–16. 3

[FSW09] Fraedrich R., Schneider J., Westermann R.: Exploring the millennium run-scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1251–1258. 1

[JLJC05] Johansson J., Ljung P., Jern M., Cooper M.: Revealing structure within clustered parallel coordinates displays. In *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.* (2005), IEEE, pp. 125–132. 1

[MPOW17] Micallef L., Palmas G., Oulasvirta A., Weinkauf T.: Towards perceptual optimization of the visual design of scatterplots. *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE PacificVis) 23*, 6 (June 2017), 1588–1599. Received a Best Paper Honorable Mention. 1, 2, 3, 4

[Mun14] Munzner T.: *Visualization Analysis and Design*. CRC Press, Boca Raton, FL, USA, 2014. 1

[SMT13] Sedlmair M., Munzner T., Tory M.: Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE TVCG 19*, 12 (2013), 2634–2643. 1

[STMT12] Sedlmair M., Tatu A., Munzner T., Tory M.: A taxonomy of visual cluster separation factors. *Computer Graphics Forum 31*, 3pt4 (2012). 1