Exploring the Expediency of Multiclass Density Maps

Tobias Lahmann*

Universität Ulm

Abstract

Visualizing data with scatterplots fails to scale as the complexity and amount of information increases. As a consequence, there exist many design options modifying or expanding the traditional scatterplot design to meet these greater scales. Multiclass maps are scatterplots, multidimensional projections, or thematic geographic maps where data points have two quantitative attributes in addition to one or more categorical attributes. The latter is frequently rendered by drawing shapes or using colors. These properties unfortunately do not scale well when the data or labels in a data visualization overlap. In this case visualization pipelines and frameworks have to resort to data aggregation to remain comprehensible. For this purpose Jo et al. introduce the Class Buffer model, which uses multiple 2D histograms, computed for each class of the data, to render *multiclass density maps*. This article describes mechanics of this model, what obstacles are to tackle and how the Class Buffer model can be used to create descriptive visualizations of complex data.

Index Terms: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Evaluation/methodology

1 Introduction

A scatter plot is a two-dimensional data visualization that uses dots to represent the values present for two different variables - one plotted along the x-axis and the other plotted along the y-axis. This two dimensional space can either be a geographical space or two coherent dimensions of data. The dots used in scatterplots are commonly representing one single occurrence of a value the data in the background represents.

When these maps scale to larger amounts of data or when the data shown gets an additional dimension a dot distribution map, comes to hand. This is a type of thematic map that most often uses dots but also other symbols or glyphs on the two spatial or variable dimensions to show the values of one or more numeric data fields, or class. With this, quantitative data can be shown in the two dimensional space (x, y) using the domains color, size or opacity to maximize the amount of information shown in maps. Each dot on these dot-density map represents some amount of data where the dots can be encoded using the domains mentioned to show differences. By combining multiple distinctive colors, different scaling of glyphs or masking, these maps can show even more data, such as different classes or properties of fields, it thus gets called a multiclass dot-density map and by using data aggregation in combination with masking or mixing, to show the classes, these are called multiclass density map.

In a density map, areas with many dots, bigger glyphs or specific colors indicate high concentrations of values for a tile of data and fewer dots, smaller glyphs or a different colors indicate lower concentrations.

*e-mail: tobias.lahmann@uni-ulm.de

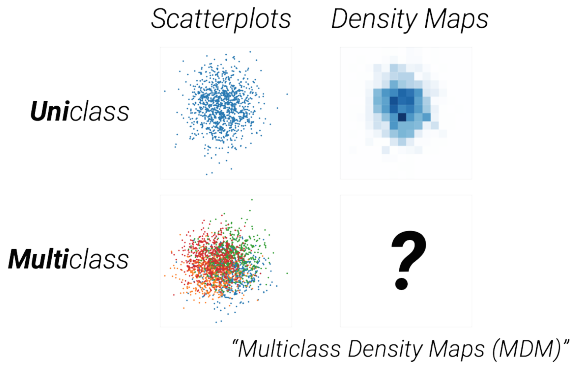

Figure 1: Scatterplots show two dimensional data with no additional property. These are called uniclass scatterplots (top left). When the scatterplot is extended with multiple classes it becomes a multiclass scatterplot (bottom left). Instead of single points to represent data points, these values can be aggregated and displayed in a way that indicates higher concentrations with i.e. darker colors (top right). An important task remains to show multiple classes with these density maps. These kind of maps are multiclass density maps.
© Jaemin Jo, [Online; accessed June 04., 2019] `https://raw.githubusercontent.com/e-/Multiclass-Density-Maps/master/motivation.png` - BSD-2-Clause

2 RelatedWork

It is a nontrivial task to visualize multiclass data on density maps and various designs have been used. The Class Buffer model unifies those various designs into a single model [12].

With more data to show, the separation of classes, perception of clusters and even just the basic comprehension becomes an ever-growing problem [12]. Scatterplots as well as density maps additionally have a fundamental drawback: they're terrible for retrieving rates or numbers from the map. For example, few people will have the time or interest in counting hundreds (or thousands) or dots in order to know the precise number represented. They'll likely know that some places have "more" of what is shown than others, but they won't know necessarily by how much. The same principle applies to density maps where it is not easy for viewers to tell what color in a region represents how much data. When dealing with multiclass density maps different obstacles come to hand that limit the readability and comprehensibility of the data that is shown in the plot. As soon as the number of data points increases, multiclass maps must resort to dimension reduction (DR) or visual encoding (VE) techniques to remain readable. Multiclass Density Maps use multiple overlaying density maps, derived from scatterplots, to display data in a way that the comprehension of the data can be improved [12].

Scatterplots have been used for almost two centuries as a way to visually represent data [9]. Much of their popularity has been due to their ability to allow correlations to be easily perceived by a human viewer [5, 10]. Multiclass scatterplots contain more complex data and hence have to be designed more carefully. When it comes to density maps the data can be masked or mixed, among other visual enhancements. This process might hide or obfuscate data from being perceived fast. The viewer consequently might has to engage

in greater cognitive effort to understand what is depicted.

The problem of scaling the visualization, either by adding or by filtering information via aggregation or binning, has been addressed in various articles [6, 7, 16, 30]. From this work three facets can be identified for scalability:

- **data size** related to the number of data points and categories,
- **perceptual processing** related to the ability to perform some tasks efficiently given a data size, and
- **computation speed** related to the time to compute an image from a visualization technique given a data size.

visualization techniques should be versatile, support large data sizes and be easy to understand, all while performing important perceptual tasks quickly. Interactive exploration can be facilitated by fast reaction and computation times. The type of interaction influences the time a user may wait for the plot to be refreshed [18, 22], ranging from 25 milliseconds up to 10 seconds. Managing large data sizes while allowing effective perceptual processing is already a challenge, and the goal of the Class Buffer model is to design a model that can improve all the three facets of scalability [12].

3 Taxonomy of Separation and Perception

With the previous work of creating effective and efficient algorithms for DR a lack of guidance for the user of certain models or algorithms has occurred [21, 9]. The question how and to what extend a user can be taken by the hand and show what DR and VE techniques to use is a non-trivial task [21].

The DimStiller system uses a workflow structure to guide users through the process of choosing DE and VE techniques [11], but automatic algorithms to provide such guidance have not yet emerged. To service this goal Sedlmair et al. used visual cluster separation measures [23, 24], originally developed for selecting good views within scatterplot matricies (SPLOM), to provide guidance for DR and VE technique choices. Two particular measures have been identified as the most effective [25]: the **centroid** and the **grid** measure. These names where chosen by Sedlmair et al. for readability and have been named other by different researchers [21].

They further showed that these measures fail to produce reliable results, meaning it resulted in a mismatch between the algorithms results and the quality judgement made by a person [21]. The two cases of false positives and false negatives occurred; either the algorithm showed that a distinct measure was sufficient for a convincing separation to happen, but the human judged the visual separation as poor, or the algorithm failed to provide such separation but humans were indeed able to distinguish clusters in scatterplots.

These measures can be used in multiclass density maps to select the tiling, as described later, and thus enhance the visual impression.

3.1 Visual Cluster Separation Factors

The general idea behind all existing separation measures is to evaluate how "pure" the neighborhoods of a scatterplot's data points are. If neighborhoods include points from many different classes they are intermixed, if only one class then they are pure [1]. The general concept to differentiate between clusters remain generally the same and differ only based on the definitions of a neighborhood around points. Based on the work of Aupetit et al. the two definitions chosen are [1]:

- measures with *hard-neighborhoods* looks at a specific subset of the data points close to the one under focus
- measures with a *soft-neighborhood* use the weighting of all the data points with respect to a focus point.

Local class-purity values are averaged over all possible focus points. This gives a higher purity when the local neighborhood class-purity is high for a large set of focal points, resulting in a good separation of clusters.

The separation of classes in scatterplots and density maps is dependent on different features of the plot. Participants of a study conducted by van Onzenoodt et al. indicate that spread and density of scatterplots have influence on the perception of clusters [27]. When viewing scatterplots or density maps these factors are not calculated based on the mentioned approaches but rather the visual perception of clusters. This is based on each persons subjective perception of correlation and clusters [19, 2, 25]. This process can be supported by design. When the designers of maps construct the plot numerous approaches can be taken to enhance the desired effects. Some points of reference, amongst others, include count, size or density as can be seen in Figure 2. Design choices result in the improved comprehension of scatterplots and density maps.

Because multiclass scatterplots and density maps have a class property with items that need to be distinguishable, Figure 2 focuses on the factors that enhance this nature and neglects within-class factors. Between-Class factors encapsulate the variance and combination of Within-Class factors across multiple classes. These factors are not described here because of the multiclass viewpoint in multiclass density maps. For more information, please consult Sedlmair et al. [21]. In the Scale section of Figure 2, the *count* factor is the ratio between the number of classes and the number of points of the dataset. Sedlmair et al. have found that fewer classes and many points was easier to perceive than the case of many classes, few points [21].

In the Point Distance section the first factor *variance of density* is the mutual product of *variance of size* and *variance of count*. In plots with a big sparse class overlapping a smaller more dense class the smaller one might be identified more easily because of the law of proximity from common Gestalt psychology [13].

The variance of shape factor ranges from similar shapes for all clusters to very different shapes across each cluster. When applied to density maps one shape might be applied to one class of data points. This will result in a mixture of shapes across the plot where a fine distinction needs to be done for the shape to be easily distinguishable to all other shapes used in the plot [3].

The inner-outer position in the position category describes a positional relationship between classes where a inner class can be surrounded by outer classes. Sedlmair et al. distinguish between *existent* and *non-existent* relations [21]. They further showed that inner classes were more difficult to identify, especially those in the synthetic-gaussian data family. This shows that density maps must countermeasure this behavior with correct design adaptations.

The overarching factor is class separation. In data plots with two well separated, round, and contiguous clusters, data separation can be strongly influenced by nearly every other factor [21]. For density maps the separation of classes could be the most viable factor to countermeasure the effects mentioned, mixing and masking, where data gets overlaid by other properties. In these kinds of maps it might not always be possible to apply sufficient class separation without sophisticating the two dimensional properties so designers must reside to different measures like axis scaling or using endorsed color schemes.

All of these factors can more or less be applied to multiclass density maps and the Class Buffer model, where the most significant difference is the work done before plotting the data.

3.2 Perception of Classes an Outliers

How well different points of data are perceived in the scatterplot and density maps is a relevant task for any group of data, or even individual points. Micallef et al. focused on classes and outliers when investigating the perception within plots [17]. They employ

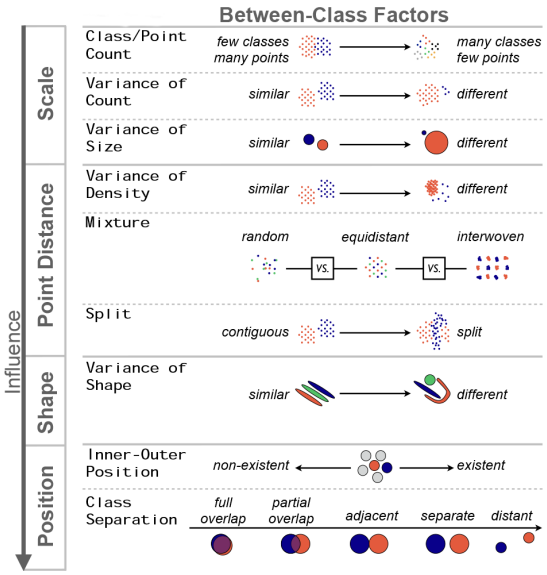

Figure 2: A taxonomy of data characteristics with respect to class separation in scatterplots. Some factors are organized as axes (arrows) while others are binned. Factors at the top can strongly influence factors below them. Class Separation is therefore dependent on all other factors.

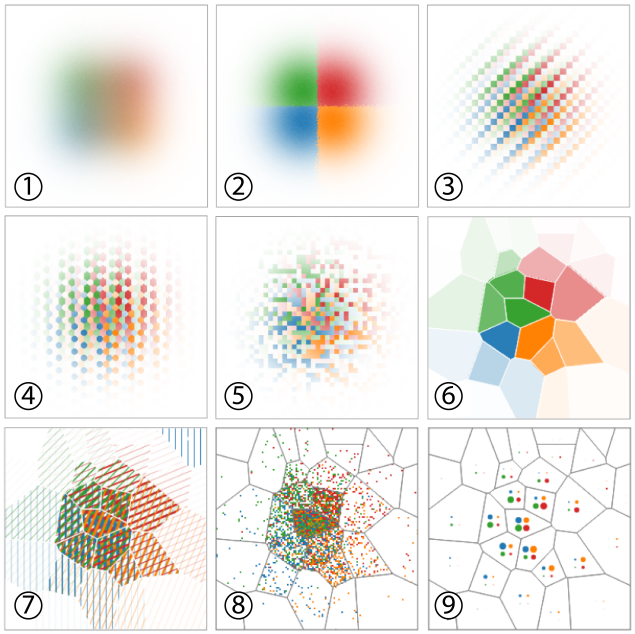

Figure 3: A few results generated from the Class Buffer model. 1. Blending of the density maps. 2. winner-takes-all approach. The class with the highest count is chosen. 3. - 5. Three different weaving patterns. 3. and 4. are regular patterns, 5. is an irregular weaving pattern. 6. - 9. Uses rebinning (binning and aggregation over the density maps) with tiles produced by a random Voronoi pattern. 6. Shows the highest count in the tile, 7. Shows hatching, 8. shows a dot density plot, 9. Shows a punch card.
© Extracted from *A Declarative Rendering Model for Multiclass Density Maps* and digitally edited, Jo et al., [12]

structural similarity [29] to measure the perceivability of a group of points. Structural similarity is a highly reliable image quality assessment model often applied to measure the similarity between two images [17]. Details on this algorithm can be taken from Wang et al. [29]. The mean structural similarity as implemented in scikit-image [26] can be used an applied to parts of the data with non-zero opacity. In the case of two scatterplots a and b the mean structural similarity $SSIM(a;b)$ returns values between 0 and 1, where 1 denotes that images a and b are identical. In a case where the data of two scatterplots are almost identical but one map has a group of data points that the other one does not have, structural similarity can be taken to measure the perceived similarity between these two plots. If they are rather similar, the group of points is difficult to perceive. If the two scatterplots are rather different, then the group of points is easy to perceive [17].

Overall, once users are familiar with the environment, they find it less difficult to perceive clusters [27]. Users could thus be trained to perceive classes more easily when applying similar design patterns to plots. Further, when users view multiple scatterplots or density maps they might get better at understanding these.

4 The Class Buffer Model

As a result of the previously mentioned challenges the designers of maps face various tasks to convey the correct idea to support their data. In many approached the designers have to think of different data aggregation, dimension reduction, visual encoding or rendering methods and know the advantages of many. The challenge of visualizing different approaches beforehand to have a mental image when creating density maps lead to the implementation of the Class Buffer model. This model takes the designer through a interactive approach to visualize data in multiclass density maps.

The Class Buffer model is implemented to use the JSON data-interchange format. It defines an expressive visualization grammar for multiclass density maps, the specification of this grammar can be seen in Listing 1. The Class Buffer model is based on the Class Buffer idiom [4] as the primary building block. The implementation of the Class Buffer model from Jo et al. consists of six stages, that are apportioned to either the back-end that is processing much of the data or the front-end of the system displaying the data. This approach comes from the advantages of the specification, where tasks can easily be implemented on both sides of modern web applications. The back-end is responsible and capable of doing larger calculations and the front-end is dedicated to filtering, styling and displaying the data. The six stages used in the model can be seen in Figure 4 and are described as follows:

1. **Binning.** Data that is given into the system is divided into data buffers where the number of data buffers corresponds to the number of classes in the data. A data buffer can be understood as a 2D histogram that counts the occurrence of a data case and groups those belonging together. This binning is implemented in the backend of the system because it consists of only semantic calculations and no styling. The backend will transfer the data buffers to the front-end where the second and following steps will take place.
   The main goal of doing this step on the back-end side is to enable the rendering to happen in interactive time [12] and because this step has to be done generally only once for the lifecycle of a map.

2. **Preprocessing.** When the data buffers get passed to the front-end of the application a preprocessing operation is realized. This preprocessing is most often a gaussian smoothing over each buffer individually. According to Hadley Wickham "Smoothing is an important step because it allows us to resolve problems with excessive variability in the summaries" [30].
   Data can likewise be filtered by different properties and criteria that should be left out of the rendering process. Com-

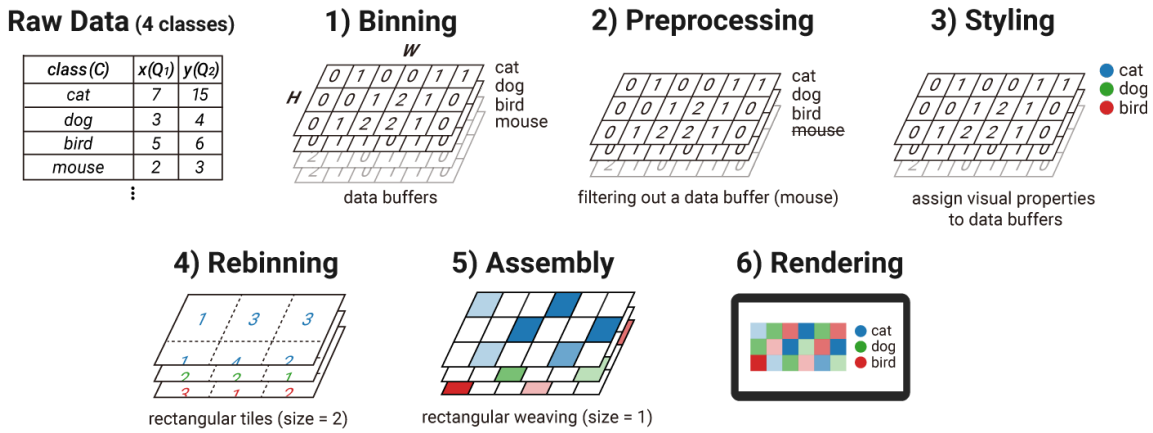

Figure 4: The six stages of the Class Buffer model. First binning, the data is split into as many data buffers as tables.⋆ Second the preprocessing, like gaussian smoothing.• Third styling where the data buffers are assigned i.e. colors.• Fourth rebinning where the classes get partitioned into tiles, aggregated and normalized.• Fifth the assembly where tiles and normalized counts are turned into a single density map.• Sixth the rendering of the data, here legends axis and landmarks etc. are added for better understanding of the visualization.•
⋆: Done in the back-end.•: Done in the front-end. © Extracted from *A Declarative Rendering Model for Multiclass Density Maps*, Jo et al., [12]

binations of different data buffers can moreover be created. Because of this filtering and combining data into mixed data buffers the preprocessing step is done in the front-end to eliminate expensive round trips when filters change.

3. **Styling.** The first styling step performed is a transformation from data buffers to class buffers. Class buffers represent the same separated data as data buffers but with additional information such as class specific color, hatching angle or scale. These visual properties are specified on the front-end using the grammar of the Class Buffer model, as described by Jo et al. and can be seen in Listing 1.

4. **Rebinning.** The front-end then partitions the class buffers, that is divided into grids, into tiles. These can be equidistant partitions of the two dimensional domain that the data resides in, but also a more irregular divided separation of space like a Voronoy separation. An important property is that the tiling is defined so that there is no overlapping within them. Additionally an aggregated count is computed for each class and stored in a data vector. This aggregation is all pixels from class i that belong to the tile t. It is afterwards normalized to a value between 0 and 1 using a linear, log, square root or equi-depth histogram scale to be used in the legend.

5. **Assembly.** Tiles and normalized data vectors are turned into a single density map image. This step adds more styling to the process, like an alpha channel, and different approaches to unify the maps are possible, namely, masking, mixing, hatching, and generating glyphs based on the aggregated count described below.
Masking is the process where a mask is assigned to each class buffer. This mask is a grid of the same size as the original grid and is defined such that the sum of every mask grid equals 1 for each pixel. Then each tile is rendered for each class buffer with a uniform color previously assigned in the styling stage and the corresponding opacity value of the normalized count stored in the data vector. The resulting image is then alpha-blended using the opacity values stored in the mask. This approach is used for the weaving pattern that can be seen in Figure 3 in number 3 and number 4.
The **Mixing** operation on the other hand combine tiles by blending them. The main goal is to receive a resulting image that is similar to the one generated by masking but with colors that are blended across the entire tile instead of being masked. Different mixing methods can be used, in example *additive mixing* sums each RGB channel of the colors, thus generating brighter colors as *multiplicative mixing*, which generates darker colors for high density regions. Additionally, it is possible to take an *winner-takes-all* or *loser-takes-all* approach where either the class with the highest or lowest count is chosen.
Tiles can furthermore be rendered by using the **hatching** operation, which fills each tile with evenly spaced lines. Typically the line thickness encodes the normalized count, while the line orientation encodes the class. The hatches can be combined by stacking them side by side within each or by superimposing them, like in Figure 3, number 7.
A more conventional pipeline is **glyph generation** to visualize the previously processed data. Glyphs are miniature visualizations that are used to encode the normalized counts of the data and are typically rendered in the center of the tile, like in Figure 3, in number 8 and 9. The position can be slightly moved to make space for glyphs from other classes. This can result in visualizations like a punch card or the mixture of bar chart and density map where each tile in the visualization holds a bar chart indicating the normalized value of the data.
Jo et al. compute the position of the glyph as the largest rectangle in polygon for each tile and place the glyph in the middle of this rectangle [12]. This assembly step might be the most comprehensive step of the Class Buffer model but it is optional as well. When no assembly operation is specified, each tile is rendered using a uniform translucent color, and the process outputs multiple density map images instead of a single one, leaving the conflict resolution to the final rendering stage.

6. **Rendering.** The final step shows the single density map on the medium the user desires, the model supports stationary as well handheld devices because of the separation between back- and front-end calculations. The background color of the plot defines the lower end of the color scale used in the visualization. For example, a white background will result in a color scale where white is indicating the lowest, or zero density. Black backgrounds will produce the same effect but might have other (dis-)advantages. The background does not have to be of uniform color but can also show cartographic or

other information.
In this stage a variety of helping decorations and annotations can be added, i.e. landmarks and axes, to make the visualization easier to understand by the viewer and a legend can be added to complete the visualization process.

The implementation form Jo et al. is available, with example datasets, at `https://github.com/e-/Multiclass-Density-Maps` and examples can additionally be explored at `https://jaeminjo.github.io/Multiclass-Density-Maps/`.

The Class Buffer model is implemented in TypeScript, a strongly typed language that can be transpiled into JavaScript, which runs in every modern browser as basis of the dynamic web. Additionally, the implementation relies on the D3 libraryfor contours and cartographic projections. Interpreting a specification takes between a few hundred milliseconds to one second depending on the complexity of the operations to perform [12], not counting the time to transfer the data, including data buffers and the TopoJSON file if needed. Changing a specification and reinterpreting data is generally a lot quicker as information isn't changed. Intermediate operations, such as rebinning, can be cached if the tiling is not changed, which is common for geographical maps [12]. Tile glyphs are rendered using Vega-Lite [20].

```
{"description"?: <string>,
"background"?: <Color>,
"data": {"url": <url> | "dataSpec": <DataSpec>},
"smooth"?: {"radius": <number>},
"reencoding"?: {
  "label"?: <LabelSpec>, "color"?: <ColorSpec>,
  "hatching"?: <HatchingSpec>
},
"rescale"?: {
  "type": "linear"|"log"|"pow"|"sqrt"|"cbrt"|"equidepth",
  "rebin"?: {
      "type": "none"|"square"|"rect"|"topojson"|"voronoi",
      "aggregation": "mean"|"max"|"sum"|"min"|"density",
      "width"?: <number>, "height"?: <number>,
      "size"?: <number>, "topojson"?: <TopoJSONSpec>,
      "url"?: <string>, "feature"?: <string>,
      "points"?: <Point[]>, "stroke"?: <Color>
  },
  "compose"?: {
    "mix": "none"|"invmin"|"mean"|"max"| "blend"|
          "weavingrandom" | "weavingsquare"|"weavinghex"|
          "weavingtri"|"propline"|"hatching"|"separate"|
          "glyph"|"dotdensity"|"time",
    "mixing"?: "additive"|"subtractive"|"multiplicative",
    "size"?: <number>, "widthprop"?: <string|number>,
    "colprop"?:<boolean>, "order"?: <number[]>,
    "glyphSpec"?: <GlyphSpec>, "interval"?: <number>
  },
  "levels"?: <number>
},
"contour"?: {
  "stroke": <number>, "lineWidth"?: <number>,
  "values"?: <number[]>, "blur"?: <number>
},
"legend"?: <LegendSpec>, "stroke"?: <StrokeSpec>,
"axis"?: <AxisSpec>}
```

Listing 1: Syntax of Class Buffer specifications. Different TypeScript notations show the required JSON fields for the model to interpret the data in the minimal way [12]. Fields denoted by questionmarks can be omitted.

4.1 Benefits of the Class Buffer model

The Class Buffer model is more scalable over the three facets, data size, perceptual processing, and computation speed when compared with other frameworks for visualizing large maps [12]. This is largely due to the separation of computational tasks between the back-end and the front-end in the implementation. The back-end side computes density maps, which will result in proportional computation time in relation to the number of points. Once this expensive computation is done, the visualizations offered by the model can be used to explore the data at 'interactive' speed, regardless of the size of the original data. Jo et al. argue that it can be about millions, billions, or any higher scale of data points. "The sheer amount is irrelevant to our model, and users can explore data using multiple visualizations with different trade-offs regarding the visual tasks supported." [12]

Other visualization libraries compute the aggregation and the visualization together, needing a time proportional to the number of points to generate a new visualization and require expensive round-trips from the back-end to the browser. Abstract Rendering [6, 7] performs the binning operations in complex buffers that are configured early before their contents are computed. Once computed, most of the rendering has to be performed in the back-end as well because the composite structure at each bin is too complex to be sent transparently to a front-end. Each modification of the pipeline requires an expensive recomputation starting at the binning stage, requiring the handling of the whole dataset.

By contrast, the Class Buffer model receives the raw unsmoothed density map and can apply a smoothing kernel to it at its first stage. According to Wickham [30], applying the smoothing to the binned data produces very similar results than applying it before binning, with a substantial performance improvement [28]. Compared with Wickham's BSS model [30], the Class Buffer model provides a richer set of visualization options, but less statistical operations at the prebinning stage. This is because the BSS model does not manage multiclass data.

4.2 Interactive Data Exploration

The Class Buffer model uses 2D histograms, calculated from data buffers, as data sources, and this provides the model with an ability to abstract underlying computation of large-scale data [12]. When the dataset is filtered by a new dimension, the data buffers should be invalidated and recomputed. In interactive exploration with multidimensional data, this would be a frequent case, and modern data structures [14, 15] have been proposed to speed up the recomputation. The model naturally lends itself to working with those optimized data structures through transfer of abstract data buffers. In addition, the ProgressiVis toolkit [8] already transfers data buffers of progressively aggregated data to its front-end, which can be used with the Class Buffer model.

4.3 Limitations

A wide variety of visualizations, suited to complex tasks, can be created using the Class Buffer model. On the downside it needs additional data to be fully comprehensible, such as landmarks, points of interest, and outliers; they should be combined at the rendering stage. Landmarks for maps include important locations and names, sometimes additional shapes to add context, such as rivers or points of interest. Landmarks for scatterplots and multidimensional projections include location of interesting points. For example, in publication data, highly cited publications or authors can be used as landmarks. The Class Buffer model can replicate several techniques used to visualize multiclass density maps, but it does not offer guidance on their best use but providing this usable implementation will enable the visualization community to start researching these maps, opening up a new area of scalable visualization [12].

5 Discussion

The Class Buffer model is more scalable over the three facets compared to other frameworks for viewing large scale data: data size,

perceptual processing, and computation speed. The enhanced perceptual processing arises from the broad spectrum of visualization methods that can be generated for density maps of multiple classes. The model on the other hand does not guide users in a very good way. The challenges of understanding and using the different visualization methods have to be further investigated. Multiclass density maps often have the disadvantage of obfuscating information when mixing or masking is introduced but they can also display more data classes that can be easily shown when using dot density maps. Different classes intersect, overlap and interfere with each other and by this make it hard to understand these kind of maps. The Class Buffer model implemented by Jo et al. uses different approaches to display data but just few of them result in classical density maps [12], some also arouse the impression of dot density maps with different tiling or aggregation. A density map can be very complicated when used in the wrong context or when implemented wrong or without an appropriate understanding of the field. It is thus essential to improve the presented method further. This could be done by enhancing the visual separation with additional options or by implementing an expressive guiding toolbox or some other kind of guideline. By providing the user with the basic tools to generate multiclass density maps the first step has been taken.

6 Conclusion

Multiclass density maps are an expressive way of displaying data with two quantitative and an additional qualitative attribute. They combine the readability of scatterplots with the information entropy of density maps. So far, little has been done regarding the design space and scalability of multiclass maps. This is where the Class Buffer model can be of use.
Jo et al. started from examples of cartography to abstract multiclass density map idioms into a unified Class Buffer model. With this model the design space is widened because of the separation of computation between front-end and back-end. Changing visualization requirements do not have to be binned and aggregated to be shown on a screen. And by using a declarative visualization grammar users can explore the design space of multiclass density maps, finding the best design for achieving their goal.

References

[1] M. Aupetit and M. Sedlmair. Sepme: 2002 new visual separation measures. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8. IEEE, 2016.

[2] R. Beyth-Marom. Perception of correlation reexamined. *Memory & Cognition*, 10(6):511–519, 1982.

[3] R. Borgo, J. Kehrer, D. H. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. Ward, and M. Chen. Glyph-based visualization: Foundations, design guidelines, techniques and applications. In *Eurographics (STARs)*, pages 39–63, 2013.

[4] H. Chen, S. Engle, A. Joshi, E. D. Ragan, B. F. Yuksel, and L. Harrison. Using animation to alleviate overdraw in multiclass scatterplot matrices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 417. ACM, 2018.

[5] W. S. Cleveland. *Visualizing data*. Hobart Press, 1993.

[6] J. Cottam, A. Lumsdaine, and P. Wang. Overplotting: Unified solutions under abstract rendering. In *2013 IEEE International Conference on Big Data*, pages 9–16. IEEE, 2013.

[7] J. A. Cottam, A. Lumsdaine, and P. Wang. Abstract rendering: out-of-core rendering for information visualization. In *Visualization and Data Analysis 2014*, volume 9017, page 90170K. International Society for Optics and Photonics, 2014.

[8] J.-D. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv preprint arXiv:1607.05162*, 2016.

[9] M. Friendly and D. Denis. The early origins and development of the scatterplot. *Journal of the History of the Behavioral Sciences*, 41(2):103–130, 2005.

[10] R. L. Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, 2000.

[11] S. Ingram, T. Munzner, V. Irvine, M. Tory, S. Bergner, and T. Möller. Dimstiller: Workflows for dimensional analysis and reduction. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pages 3–10. IEEE, 2010.

[12] J. Jo, F. Vernier, P. Dragicevic, and J.-D. Fekete. A declarative rendering model for multiclass density maps. *IEEE transactions on visualization and computer graphics*, 25(1):470–480, 2019.

[13] S. Kim, K.-J. Yoon, and I. S. Kweon. Object recognition using a generalized robust invariant feature and gestalt's law of proximity and similarity. *Pattern Recognition*, 41(2):726–741, 2008.

[14] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[15] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[16] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE transactions on visualization and computer graphics*, 19(9):1526–1538, 2013.

[17] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE transactions on visualization and computer graphics*, 23(6):1588–1599, 2017.

[18] R. B. Miller. Response time in man-computer conversational transactions. In *AFIPS Fall Joint Computing Conference (1)*, pages 267–277, 1968.

[19] R. A. Rensink and G. Baldridge. The perception of correlation in scatterplots. In *Computer Graphics Forum*, volume 29, pages 1203–1210. Wiley Online Library, 2010.

[20] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016.

[21] M. Sedlmair, A. Tatu, T. Munzner, and M. Tory. A taxonomy of visual cluster separation factors. In *Computer Graphics Forum*, volume 31, pages 1335–1344. Wiley Online Library, 2012.

[22] B. Shneiderman. Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, 16(3):265–285, 1984.

[23] M. Sips, B. Neubert, J. P. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. In *Computer Graphics Forum*, volume 28, pages 831–838. Wiley Online Library, 2009.

[24] A. Tatu, G. Albuquerque, M. Eisemann, J. Schneidewind, H. Theisel, M. Magnork, and D. Keim. Combining automated analysis and visualization techniques for effective exploration of high-dimensional data. In *2009 IEEE Symposium on Visual Analytics Science and Technology*, pages 59–66. IEEE, 2009.

[25] A. Tatu, P. Bak, E. Bertini, D. Keim, and J. Schneidewind. Visual quality metrics and human perception: an initial study on 2d projections of large multidimensional data. In *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 49–56. ACM, 2010.

[26] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[27] C. van Onzenoodt, J. Kreiser, D. Heer, and T. Ropinski. Evaluating the influence of stereoscopy on cluster perception in scatterplots. In *Proceedings of SIGRAD 2017, August 17-18, 2017 Norrköping, Sweden*, number 143, pages 25–31. Linköping University Electronic Press, 2017.

[28] M. Wand. Fast computation of multivariate kernel estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994.

[29] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[30] H. Wickham. Bin-summarise-smooth: a framework for visualising large data. *had. co. nz, Tech. Rep*, 2013.