A Framework for Multiclass Contour Visualization

Sihang Li, Jiacheng Yu, Mingxuan Li, Le Liu, Xiaolong (Luke) Zhang, and Xiaoru Yuan

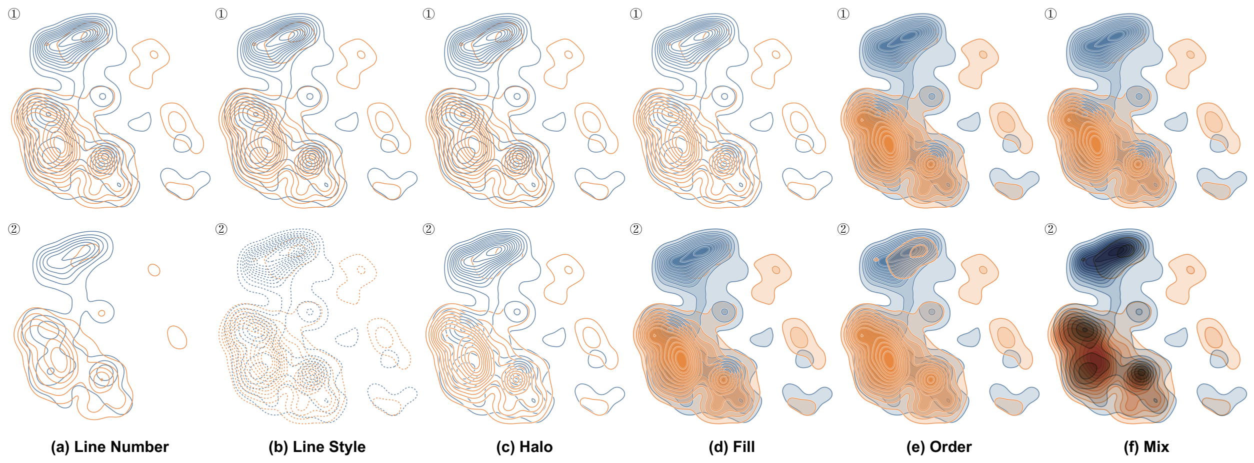


Fig. 1. Examples of some design parameter attribute under our proposed multiclass contour visualization framework. (a) **Line Number**: the number of contour lines in each class, 12 (1) vs. 6 (2). (b) **Line Style**: solid (1) vs. dashed (2). (c) **Halo**: contour lines without halos (1) vs. with halos (2). (d) **Fill**: the contours without fillings (1) vs. with fillings (2). (e) **Order**: multiple contours plotted by level (1) vs. by class (2). (f) **Mix**: multiple contours stacked with direct overlay (1) vs. with color blending (2).

Abstract— Multiclass contour visualization is often used to interpret complex data attributes in such fields as weather forecasting, computational fluid dynamics, and artificial intelligence. However, effective and accurate representations of underlying data patterns and correlations can be challenging in multiclass contour visualization, primarily due to the inevitable visual cluttering and occlusions when the number of classes is significant. To address this issue, visualization design must carefully choose design parameters to make visualization more comprehensible. With this goal in mind, we proposed a framework for multiclass contour visualization. The framework has two components: a set of four visualization design parameters, which are developed based on an extensive review of literature on contour visualization, and a declarative domain-specific language (DSL) for creating multiclass contour rendering, which enables a fast exploration of those design parameters. A task-oriented user study was conducted to assess how those design parameters affect users' interpretations of real-world data. The study results offered some suggestions on the value choices of design parameters in multiclass contour visualization.

Index Terms—Contour, multiclass visualization, visualization framework, domain-specific language, visualization design

1 Introduction

Contour plots are widely used to visualize scalar quantities and field data in many fields, such as weather analysis [2], computational fluid dynamics [1], and artificial intelligence [25]. A contour plot draws a set of contour lines by setting different values and connecting points with the same value of a given attribute. We call the lines representing higher values the higher-level contour lines. The contour with contour lines of different levels shows the data attribute's value levels and helps users see its distribution and identify some essential characteristics such as extreme points.

Nowadays, field data has become more complex. Visualizing and analyzing the contours of multiple variables and attributes associated with the same spatial region has become a common practice. Such contours are referred to as multiclass contours. A typical example is meteorological data, which contains attributes like air temperature, pressure, humidity, etc., and different data in different years. The multiclass contours of meteorological data are essential for meteorologists to obtain a comprehensive overview of climate change over a specific region. In addition, multiclass contours have recently been extended to other fields [15, 19], where they can help compare the multiple distributions.

Multiple contours can be rendered as separate displays. However, Early studies have reported that using separate displays creates more perceptual and cognitive work for viewers and significantly prevents them from discovering and analyzing patterns of multiple variables [9]. Another method is to overlay the contours into a single view. However, directly overlaying these contours will inevitably lead to visual clutters. Researchers have proposed various design approaches to address this issue, including reducing the number of contour lines of individual classes and using different visual channels for portraying contours [21, 37]. However, little research is done to consider how these approaches should be integrated holistically. On the one hand, there is a lack of

- *Sihang Li, Jiacheng Yu, Mingxuan Li and Xiaoru Yuan are with Key Laboratory of Machine Perception (Ministry of Education), School of AI, Peking University. E-mail: {lisihang, jiachengyu, catherine9946, xiaoru.yuan}@pku.edu.cn.*
- *Sihang Li and Jiacheng Yu are also with Center for Data Science, Academy for Advanced Interdisciplinary Studies, Peking University. Xiaoru Yuan is also with National Engineering Laboratory for Big Data Analysis and Application, Peking University. Xiaoru Yuan is the corresponding author.*
- *Le Liu is with School of Computer Science, Northwestern Polytechnical University. E-mail: lel@nwpu.edu.cn.*
- *Xiaolong (Luke) Zhang is with College of Information Sciences and Technology, Pennsylvania State University. E-mail: lzhang@ist.psu.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

summary of the existing approaches, and what design parameters can be chosen in the visualization designs of the multiclass contour is unclear. Designers are hard to explore the design parameters and possible values systematically. On the other hand, in practice, it is often difficult for users to choose appropriate multiclass contour designs for different tasks. Even for the contours from experienced visual designers, such as those in visualization research papers, the rationales behind designs are often missing. We argued that the design parameters of multiclass contour visualization must be carefully chosen to make the display more comprehensible and enable users to perform specific tasks better. The guidance of the design parameters selection strategy is essential.

With this in mind, we proposed a framework for multiclass contour visualization. This paper reports our work in constructing two components of the framework: identifying some essential design parameters and developing a declarative language to facilitate the description and design of multiclass contour visualization. To identify the design parameters, we conducted an extensive review of literature in the field of multiclass contour visualization and summarized four design parameters: **Line**, **Fill**, **Order**, and **Mix**. Each of these parameters has some attributes that can be considered in the design. Then, to streamline the design by using these parameters, we developed a declarative domain-specific language (DSL) so that users can quickly generate multiclass contour visualizations with a simple text description of samples.

We also set up a task-oriented lab study on the impacts of some design parameters on user interpretations of multiclass contours. The results of the user study, which examined how different value choices and combinations of some design parameters influence the understanding of multiclass contours, offer suggestions on the design choices of multiclass contour visualization.

In general, our research made the following contributions:

- Developed a framework for multiclass contour visualization, including a set of essential design parameters based on literature review and a declarative domain-specific language for quick generation of multiclass contour visualization.

- Provided design suggestions based on the results of a task-oriented user study on the impacts of design parameters on the understanding of multiclass contours.

2 Related Work

In this section, we reviewed previous work on two areas: contour visualizations and declarative languages for visualization generation.

2.1 Contour Visualization

Contour is a very common visualization method to visualize scalar fields. By setting different value thresholds, multiple closed contour lines can be obtained to represent the distribution of the values. In the visualization community, research in this direction includes the use of contour lines to represent the area with high density and outlier points [21], the combination of multiple contours to express uncertainty [35], the application of line width to encode extreme values [37], and the incorporation of contour-like strokes into the nodes of scatterplot to show global data structure [17].

Another usage of the contour is to visually enclosure objects to indicate classes explicitly. Research efforts have been made to improve contour visualizations by incorporating other visualization parameters, such as the textures and alpha blending to show intersecting sets [31], Bubble Sets to show sets and their relationships [6], and Rectangular Euler Diagrams to improve the readability of the set intersections [23].

In addition to research on new methods to enhance contour visualization, some efforts were made to adapt traditional contour visualization methods to different data types, including density maps [12], texts [3, 15, 16], graphs [4, 38], scatterplots [32], spatial data [18], high-dimensional data [19], and medical data [29].

These novel approaches lay the foundation for our framework and suggest what design parameters should be considered.

2.2 Declarative Language for Visualization Design

A declarative language allows developers to describe the results of computation. Compared to traditional programming languages requiring developers to describe how the results should be generated, declarative languages can simplify development processes [10]. Some declarative languages exist for visualization design. For example, Vega [27] provided a set of fundamental abstractions for constructing visualizations, and Vega-Lite [28] offered a high-level grammar that enables rapid specification of interactive data visualizations. However, these general declarative languages did not cover all types of visualization designs and could not easily accommodate the requirements in some specific domains.

In addition, some research has focused on declarative languages for specific domains. Such declarative domain-specific language (DSL) can support different types of data or visualization, such as hierarchical data [14], volume data [5, 30], scalable scatterplot [32], graph [11], unit visualization [22], and multiclass density map [12].

In this paper, we proposed a declarative DSL focusing on multiclass contour generation based on the design parameters and their attributes. We followed the approach of Vega, which takes JSON as the format for the DSL.

3 Analysis of Design Parameters in Multiclass Contour

Multiclass contour is widely used, including some research works that have already applied multiclass contours in different application scenarios and under different research problems. However, these contours are various in design. The possible design parameters are unclear and lack guidance on making better choices for different use. We want first to propose a generalized framework for multiclass contour, which includes available design parameters. The framework development started with a survey of the literature in the visualization community that has used multiclass contours.

The papers we reviewed meet the following two criteria: 1) the contours used in a paper need to be multi-categorical so that research on a single contour can be excluded; and 2) the contours must be displayed explicitly and statically to avoid research on dynamic methods like animation. As a result, we obtained 9 papers. We then classified them according to different design parameters, and the taxonomy based on their analysis is shown in Table 1.

Works	Line Number	Filling	Mix Method
Collins et al. [6]	1	Yes	Alpha Blending
Malkai et al. [19]	1	Yes	Alpha Blending
Simonetto et al. [31]	1	Yes	Alpha Blending
Mayorga et al. [21]	1	Yes	Color Blending
Riche and Dwyer [23]	1	Yes	Overlay
Yuan et al. [36]	1	No	N/A
Jo et al. [12]	>1	No	N/A
Lu et al. [17]	>1	No	N/A
Scheepens et al. [29]	>1	No	N/A

Table 1. Taxonomy of existing papers related to multiclass contour drawing. We classified them based on three categories: **Line Number**, **Filling**, and **Mix Method**.

Here, we classified these works on three visual parameters: **Line Number**, **Filling**, and **Mix Method**. **Line Number** refers to whether the contour in each class contains only one line or multiple lines, **Filling** concerns whether each contour line has filling inside or not, and **Mix Method** is about how contours in different classes are mixed into one view. For those contours without fillings, we excluded them from the analysis of mix methods because the mix between the lines becomes challenging to distinguish. We identified three mix methods: *Overlay*, *Alpha Blending* and *Color Blending*. For *Color Blending*, Splatterplots [21] used an approach that pushed the overlapping areas toward black.

In addition to the above design parameters, we also found that some researchers considered other encoding methods. Lu et al. [17] added contour-like strokes to the scatterplots, which can be regarded as discontinuous contour lines. Zhao et al. [37] used line width to encode distance to the location of the maximum value. Scheepens et al. [29] distinguished contours with line opacity and halos. Simonetto et al. [31] used textures, instead of solid colors, in the fillings. These design considerations can all be incorporated into our framework.

4 Our Framework of Multiclass Contour Visualization

Based on the review of works and references of some basic visual channels, we developed a framework for multiclass contour visualization by first summarizing four design parameters and then developing a declarative domain-specific language (DSL) to facilitate the generation of multiclass contours. In this section, we described these two components.

4.1 Design Parameters

Here, we discussed these parameters and related attributes. In the framework, we want the design parameters to be generalized by avoiding specific visual encoding methods and letting designers choose what encoding methods to use. Here we just explained some commonly used encoding approaches.

4.1.1 Line

The lines are the most visible visual elements in contour visualization and can strongly influence contour representation. This parameter contains several attributes that can be manipulated in design.

Line Number is one attribute that influences what a contour looks like (Fig. 1(a)). As mentioned earlier, existing works used various line numbers in their designs.

Line Style is inspired by Winglets [17], which used different visual representations (e.g., points with wings) to represent equipotential surfaces of point density. Here we suggested that contours can be visualized with different line styles, such as solid or dashed lines (Fig. 1(b)).

Line Color is very commonly used to encode information in contour visualizations. In most papers we have surveyed, it is used to distinguish different classes by assigning each class a specific color. Even within the same contour, different lines can have different colors to show the level information.

Line Width is a basic visual channel of contour and can be used to encode additional information, such as that in PhoenixMap [37] that encodes the distance from the position of maximum value. For the contour with multiple lines, we can take a similar approach that relates the level with line width: the higher the line level, the wider it is. This approach can make the areas with higher values more visible.

Line Opacity is another basic visual channel. Scheepens et al. [29] used different line opacities to highlight different contours. We can also relate opacity with the level, just like **Line Width**.

Halo is also used by Scheepens et al. [29] to distinguish different contours with a good result. Here we follow its approach by adding halos to the lines (Fig. 1(c)). Similar to the **Line** parameter, **Halo** can have attributes like **Halo Color**, **Halo Width**, and **Halo Opacity**.

4.1.2 Fill

A contour line depicts not only the line itself but also the area it covers. As shown in Table 1, some research had fillings inside contour lines. So we added **Fill** to our framework. It is an optional parameter: a contour can have fillings or no filling (Fig. 1(d)). Several attributes can be considered in the design if a filling is applied.

Fill Style is similar to **Line Style**. Except for common standard colors, textures [31] and other fill patterns can also be applied to the fillings.

Fill Color is usually used for distinguishing different classes, just like **Line Color**. Generally, contour lines in the same class will use the same color. However, a color scheme can also be used to distinguish different contour levels in a specific contour.

Fill Opacity is similar to **Line Opacity**, which can be used to indicate level information of contour lines. It can also be used to achieve *Alpha Blending*, which can be regarded as a stack of multiple contours with different fill opacities.

4.1.3 Order

When drawing multiclass contours, the **Order** in which individual contours are rendered is essential. The most common approach is to draw contours by class. However, this method can cause the classes drawn first to be covered underneath and become invisible. In practice, we found that users tend to be more interested in the areas with high values, where the high-level contour lines are located. Thus, we adopted another drawing order: drawing contours by level (Fig. 1(e)), in which the lines of the lower levels of all classes are drawn first, and then the lines of the higher levels are rendered in turn. This method will ensure that the higher-level lines will always override the lower-level lines. As a result, the areas with higher values will be more visible.

4.1.4 Mix

As we have shown in Table 1, there are some different mixing methods, and they influence the final presentation. Thus, we added **Mix** to the framework as a design parameter to deal with areas with overlapping contours (Fig. 1(f)), where *Overlay*, *Alpha Blending*, and *Color Blending* are all common choices.

4.2 DSL for Multiclass Contour Generation

The first component of our framework only covers the design parameters and their attributes that can be considered and includes no specific encoding approaches in each parameter. To help the design processes of multiclass contour visualization that involves these parameters, we further developed a declarative domain-specific language (DSL) which provides some common encoding methods to facilitate the quick generation of multiclass contours. A declarative language lets users directly specify visualization contents and attributes without worrying about technical implementation details [10].

Our DSL uses the JSON format to specify the visualization of multiclass contours. Its grammar is shown in Fig. 2. A specification first refers to an array of 2D points with class labels as the data input and then provides details on individual parameters. The required parameters include **Line**, **Order**, and **Mix**. **Fill** is optional, and so is **Halo** for line rendering.

We implemented a process with JavaScript to parse a given specification and then render multiclass contours with SVG elements. With the given data points which contain position information, our process first blurs these points into scalar fields by class according to the point density using Kernel Density Estimation (KDE) with a uniform kernel function. Then a Marching Square algorithm, a commonly used method for contour generation [20], is applied to create contours for each class. The algorithm receives a list of value thresholds and outputs the contour lines corresponding to each threshold. In this process, different canvas sizes, cell sizes, and blur radii can affect the generation of the scalar field and then further affect the contours generated by the algorithm. Here we fix the drawing on a canvas of 256*256, with the cell size of 2 and the blur radius of 8. These parameters will be allowed to be specified in the grammar in future implementations.

The line number receives an integer as input. It is transformed into a sequence of equivariant thresholds related to the maximum value of all classes. For example, suppose the line number is k. In that case, our process gets the maximum value w within all classes, divides the interval $[0,w]$ into $k+1$ equal sub-intervals, and uses k interval values as the thresholds for contour generation. It should be noted that the same threshold list is used in all classes, so those classes that do not reach the maximum value will possibly have fewer contour lines than the given number.

The process gives two ways of encoding for the attributes with numerical values (**Line Width**, **Line Opacity**, **Halo Width**, **Halo Opacity**, and **Fill Opacity**). If the input is a single number, the corresponding attribute of all contour lines will have the same value. If the input is an interval, then the corresponding attribute of the lines in the same contour will take the values in the range defined by the interval. In addition, the attribute of the highest-level line will be set to the maximum value of the interval to highlight the areas with higher values, and the opposite is for the lowest-level line. The values of the other lines are obtained by interpolation. Currently, our process only

```
{
   "data": <Point[]>,
   "line": {
      "number": <Number>,
      "style": "solid" | "dashed",
      "color": <Color[]> | <Color[][]>,
      "width": <Number> | <Number[2]>,
      "opacity": <Number> | <Number[2]>,
      "halo"?: {
         "color": <Color>,
         "width": <Number> | <Number[2]>,
         "opacity": <Number> | <Number[2]>
      }
   },
   "fill"?: {
      "style": "solid",
      "color": <Color[]> | <Color[][]>,
      "opacity": <Number> | <Number[2]>
   },
   "order": "class" | "level",
   "mix": {
      "level": "normal" | "multiply" | "screen" | "overlay" |
               "darken" | "lighten" | "color-dodge" | "color-burn" |
               "hard-light" | "soft-light" | "difference" | "exclusion" |
               "hue" | "saturation" | "color" | "luminosity",
      "class": "normal" | "multiply" | "screen" | "overlay" |
               "darken" | "lighten" | "color-dodge" | "color-burn" |
               "hard-light" | "soft-light" | "difference" | "exclusion" |
               "hue" | "saturation" | "color" | "luminosity"
   }
}
```

Fig. 2. Grammar of our DSL for multiclass contour generation. We added input specifications to the parameter attributes in the framework. **Fill** and **Halo** are two optional parameters.

supports linear interpolation, and other methods can be implemented as needed.

Two attributes related to color (**Line Color**, **Fill Color**) are mainly used to distinguish different classes, so we need to specify the colors of different classes. Just like the numeric attributes, we want to specify both the same color and different colors for the lines in the same contour. However, color interpolation is challenging, so our DSL grammar only supports the color specification for individual lines. It should be noted that our grammar does not support different halo colors because introducing more colors would make visual clutter worse. Thus, a universal color is set for the halo, while white is one of the most commonly used.

For the mix methods, as the contours are drawn on SVG, the process uses the *mix-blend-mode* CSS property to do the mixing. Our grammar supports all values of this CSS property. Two commonly used values are *normal* and *multiply*: *normal* is for overlay and alpha blending, while *multiply* is similar to what Splatterplots [21] did. Also, as two different drawing orders are available, our process can distinguish two values: the mix between *level*s and *class*es.

We have mentioned that **Fill Style** has other choices like textures except for solid colors. However, in practice, we found that textures are indistinguishable from contour lines. Thus, we only support solid colors for the filling style in our DSL.

Fig. 3 shows an example of specifying multiclass contour visualization with our DSL and the visualization result generated by our process. The implementation was done upon d3-contour [1].

[1]https://github.com/d3/d3-contour

```
{
   "data": "CIFAR10",
   "line": {
      "number": 10,
      "style": "solid",
      "color": [#4e79a7, #f28e2b, #e15759, #76b7b2,
               #59a14f, #edc948, #b07aa1, #ff9da7,
               #9c755f, #bab0ac],
      "width": [0.56, 1.27],
      "opacity": [0.59, 0.92],
      "halo": {
         "color": #ffffff,
         "width": [0.83, 1.07],
         "opacity": [0.69, 0.77]
      }
   },
   "fill": {
      "style": "solid",
      "color": [#4e79a7, #f28e2b, #e15759, #76b7b2,
               #59a14f, #edc948, #b07aa1, #ff9da7,
               #9c755f, #bab0ac],
      "opacity": [0.05, 0.21]
   },
   "order": "level",
   "mix": {
      "level": "normal",
      "class": "normal"
   }
```

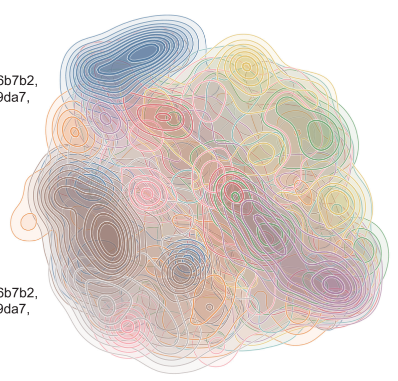


Fig. 3. An example of the declarative DSL and the corresponding contour generated by it. Parameter attributes here are set to the default values got from the preliminary study.

5 User Study on Different Designs

With the proposed framework, we next want to know whether different choices of these parameters and their attributes can affect the user's cognitive ability, which can be summarized as guidelines for designing multiclass contours. Using the DSL, we conducted a task-oriented user study to evaluate the effectiveness of different designs by varying the values of some design parameters.

5.1 Tasks

We first need to choose representative tasks for the user study, where the validity of different multiclass contour designs can be measured. Massive literature is available to identify relevant tasks used in the visualization-based user study. After reviewing various tasks seen in the literature and examining our needs, we chose four tasks listed below:

- **T1**: Finding the place with the highest value of one specific class.
- **T2**: Finding multiple value peaks of one specific class.
- **T3**: Comparing values of different locations of one specific class.
- **T4**: Comparing values of one specific location of different classes.

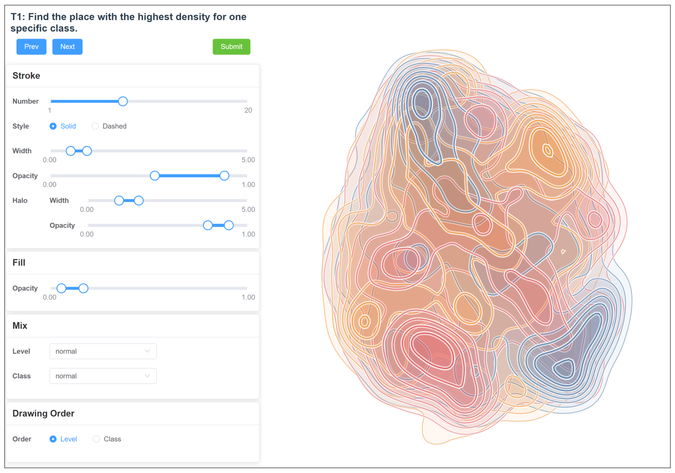
In nature, these four tasks have two goals that are articulated by Roth [24]: *Identify* and *Compare*. **T1** and **T2** fall into the category of *Identify*, which is about to find objects, while **T3** and **T4** are in the category of *Compare* to conduct comparisons. These tasks can also be classified according to their target objects. **T1**, **T2** and **T3** are for a specific class, and **T4** is for the relationships between different classes. In addition, **T2** is inspired by the task of finding clusters among different distribution areas in scatterplots by Sarikaya and Gleicher [26], which can be analogous to scalar fields.

5.2 Preliminary Study

The design parameters in the framework for the multiclass contour are too many to be tested for all possible combinations of attribute values, so we had to narrow down the scope of parameters and attributes by choosing a subset that is ”important” to users. We conducted a preliminary study to learn which design parameters and attributes are ”important” enough to be used in the formal study and what default values should be used for others.

We implemented a system for the preliminary study. The system allows participants to freely adjust attribute values and generate the resulting contours in real-time with our DSL. The system's user interface is shown in Fig. 4. It should be noted that the system does not support color adjustment. Colors of lines and fillings are set to Tableau 10 palettes, with the same color in the same contour. The system also provides the two most common mix methods: *normal* and *multiply*.

The procedure of the preliminary study includes the following steps. We first provided participants with the four tasks mentioned above and let them adjust the attribute values freely, with a goal that the generated contours would be perceived as the best for the provided tasks. Then they need to submit the attribute settings. A participant can submit multiple settings for one task. After completing the adjustments, participants were given a questionnaire on which parameters and attributes they thought would influence the tasks. We used a Likert scale (1 to 5) for each parameter attribute. The questionnaire also includes open-end questions for comments.


T1: Find the place with the highest density for one specific class.

Stroke
- Number: 1 – 20
- Style: [✓ Solid] Dashed
- Width: 0.00 – 5.00
- Opacity: 0.00 – 1.00
- Halo Width: 0.00 – 5.00
- Halo Opacity: 0.00 – 1.00

Fill
- Opacity: 0.00 – 1.00

Mix
- Level: normal
- Class: normal

Drawing Order
- Order: [✓ Level] Class

Fig. 4. The user interface of the system used in the preliminary study. A task is presented on the top, the interactive panel below is used to adjust the values of parameter attributes, and the contour on the right is generated based on selected attribute values. Clicking the submit button to finalize a choice on the setting. Multiple settings for one task are also allowed.

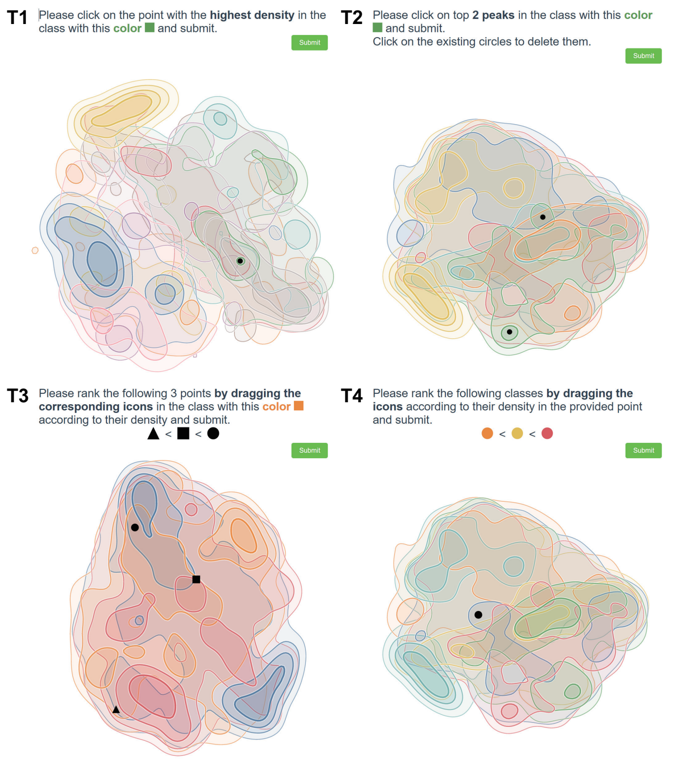
We recruited 11 participants and obtained 50 combinations of parameter attribute settings from them. We finally selected 3 parameters and attributes for the user study: **Line Number**, **Fill**, and **Halo**. **Line Number** and **Fill** got the highest scores in the preliminary study (more than 4) and were regarded as the most "important". We were also interested in **Halo** because some participants indicated in their comments that halos could help them better distinguish different classes, especially in cases where visual clutter is severe. Thus, we also chose it to test the effectiveness of halos. Some other parameters also got high scores, such as **Mix** and **Order**. However, most participants preferred to choose the same value (*normal* for **Mix** and *level* for **Order**), showing that other value choices would not help users on these tasks, so we did not select them for the user study.

The default values for other attributes were generated as follows. For numerical intervals, we handled left and right endpoints separately. We first used KDE to smooth the values and then selected the value with the highest density. For other categorical attributes, we used the value that appeared most. Fig. 3 is the contour drawn based on the obtained default values of all attributes.

5.3 User Study

The study aims to examine the effectiveness of three selected parameters and attributes on four proposed tasks. First, we gave three hypotheses:

- **H1**: Variations of individual parameters affect the user's ability to complete the tasks. We need to state whether the changes in individual parameters affected the user's ability on both accuracy and completion time.


T1 Please click on the point with the **highest density** in the class with this **color** 🟩 and submit.

T2 Please click on top **2 peaks** in the class with this **color** 🟩 and submit.
Click on the existing circles to delete them.

T3 Please rank the following 3 points **by dragging the corresponding icons** in the class with this **color** 🟧 according to their density and submit.
▲ < ■ < ●

T4 Please rank the following classes **by dragging the icons** according to their density in the provided point and submit.
● < ● < ●

Fig. 5. The user interface of the system used in the user study. Participants received a task promptly, completed the task by clicking on the contour or dragging the icon above, and then clicked the submit button for task completion. The system then moves to the next task automatically.

- **H2**: There exist interactions between pairs of parameters. This hypothesis is based on our previous finding that filled contours tend to have only one contour line. We want to verify whether this phenomenon occurs due to a coupling between any pairs of parameters.

- **H3**: The number of classes in the data can influence users to complete the tasks. As well as the number of classes itself, we also want to verify if it interacts with other design parameters.

5.3.1 Data

We used the CIFAR-10 data set [13] in the user study. The CIFAR-10 data set consists of 60,000 32x32 color images in 10 classes. We chose this data set because after being projected to a 2D space, each image class has several distribution centers, and there are more crossovers between classes. These features are ideal for our tasks.

Additionally, as we were interested in the impact of the number of classes on understanding multiclass contour, we further constructed two more data sets by reducing the classes of CIFAR-10 to 3 and 6 by merging some classes. Together with the original one, we had three data sets in total.

5.3.2 Experimental Design

The study is a within-subjects design with three independent parameters: **Line Number**, **Fill**, and **Halo**. The dependent variables are task accuracy and completion time on the four tasks mentioned early. For **Line Number**, early research showed a great variety of choices on its value (Table 1). In this study, we selected four line numbers: 1, 4, 8, and 12. Both **Fill** and **Halo** are binary. In our study, contours were rendered with or without fillings and halos in different treatments. For other attributes, we gave them a fixed value as we got in the preliminary study.

In total, we had 16 treatment combinations (4 x 2 x 2). All participants were asked to complete 4 tasks under these 16 treatments, so each participant should do 64 trials. Although we used 3 data sets in our study, we did not fully combine them with treatments because a complete combination of data sets and treatments would result in 192 trials, which were burdensome to participants. Instead, we chose a data set for each trial in a random manner. This approach ensured that all data sets and treatment levels could be balanced.

5.3.3 Participants and Apparatus

We recruited 17 participants. They all had knowledge of visualization but less knowledge of contour. None of them were involved in the preliminary study.

The study used a web-based system shown in Fig. 5. The resolution of the screen in the study was 3,840 x 2,160 for all participants, and all contours were scaled to fit the screen size. To reduce the operational burden of participants, they only needed to complete simple operations such as clicking and dragging in the system. The system then calculated the scores according to their operations and the task completion time.

5.3.4 Procedure

First, participants were briefed about the study and provided basic information about contour, common features of the contours used in the subsequent study, and details of the four tasks. Then, they were given eight warm-up tasks for exercise, including two rounds of four tasks. In the first round, we gave hints that helped users understand how to complete the four tasks. In the second round, we let the users operate independently and got familiar with the whole process. The results of these warm-up assignments were not recorded.

After the exercise, participants completed the tasks based on the prompt on the screen. The procedures of the four tasks are given below:

- **T1**: Participants were given a class and asked to click on the location with the highest value of that class. The score is the value at the clicked point compared to the highest value of the given class.

- **T2**: Participants were given a class and the number of peaks in that class. They were asked to click on the corresponding positions of the peaks. The score is the percentage of peaks that have clicked points inside. It has to be mentioned that the peaks were gotten from the contours, and different numbers of lines may cause different peaks.

- **T3**: Participants were given a class and three random positions on the contour and asked to drag the corresponding icons between the contour and the question to sort them according to the values on these three positions of the given class. Scores are calculated by the percentage of correct alignments.

- **T4**: Participants were given at most three classes and one position and asked to drag the corresponding icons to sort according to the values of the given classes on the given point. The classes were selected randomly. To avoid a large gap between the values of the three classes, we randomly generated 10 locations and selected the position with the highest minimum value. The score calculation method is the same as T3.

In addition, we randomly assigned colors to different classes in each trial to reduce the carryover effect.

6 Results

We analyzed the user study results, verified the validity of the proposed hypotheses, and summarized the design guidelines for contours based on the analysis results. **Classes** were analyzed together with the other three design parameters.

6.1 Analysis of Individual Parameters

We analyzed how the changes of one single parameter affect users' ability on different tasks. Fig. 6 shows the results with the distribution of scores and time on different parameters and tasks. As shown in the figure, user performances varied with different values of the parameters.

We used one-way ANOVA to verify whether there exists significant differences in user performances in **Line Number** and **Classes**, which have more than two choices, and t-test for **Fill** and **Halo**.

6.1.1 Line Number

Fig. 6 shows that different choices of **Line Number** affect task accuracy scores and completion time. For accuracy, ANOVA showed a significant difference among four choices on T1 ($F(3,268) = 19.6$, $p < .001$) and T2 ($F(3,268) = 2.75$, $p = .0434$). Post-hoc analysis (Tukey's HSD) indicated that for T1, significant differences exist between the value of 1 and 4 (95% CI = [.0764, .198]), 8 (95% CI = [.0995, .221]), and 12 (95% CI = [.0800, .203]). All these values are with $p < .001$.

For T1, as shown in Fig. 6, the accuracy under the value of 1 is significantly lower than those under 4, 8, and 12. This result suggests that using more lines helps identify the highest data values. However, even though there are no significant differences between using 4, 8, or 12 lines, an interesting observation is that the accuracy scores of using 8 lines are slightly higher than those of using 3 and 12 lines. A reasonable interpretation of this is that using 8 lines presents more details of the data distribution than using 4 lines, just as what using 12 lines is able to achieve, but induces less visual clutters than using 12 lines. It also implied that an optimal choice of line number should be able to both convey sufficient information and reduce visual complexity, while 8 lines may be a compromise option.

For T2, we can see that task accuracy drop rapidly as the line number increases when it is fewer than 8, but the accuracy under the value of 12 is nearly the same as that under 8. However, Tukey's HSD showed that the difference between them is marginally between parameter values in T2 but gets an edge value between 1 and 8 (95% CI = [-.00174, .195], $p = .0563$). It indicated that the sensitivity of this task to line numbers is low but might also lead to a conclusion that presenting more information by more lines prevents users from accurately identifying multiple peaks of the data distribution to a certain extent.

Line number has obvious influence on the completion time in T2 ($F(3,268) = 5.25$, $p = .00156$) and T3 ($F(3,268) = 3.23$, $p = .0229$). In T2, we found that the average time under the value of 8 is much higher than 1 (95% CI = [1.58, 11.7], $p = .00439$) and 4 (95% CI = [1.63, 11.7], $p = .00401$). It is unclear how this could happen.

For T3, we found that the time cost increases as the number of lines increases. The results from Tukey's HSD also showed that the difference between the values of 1 and 12 is significant (95% CI = [.809, 11.6], $p = 0.0169$). It confirmed that more lines increase the burden on the users to make the comparison for a particular class.

6.1.2 Fill

As shown in Fig. 6, the accuracy scores with fillings are lower than those without in T2 and T3. Results from t-test confirmed this finding ($t(270) = 2.33$, $p = .0207$ and $t(270) = 2.03$, $p = .0434$, respectively). It can be explained that fillings make it more difficult for users to distinguish a particular class as the colors of different contours are mixed. For the completion time, the results from the quantitative analysis did not show a significant difference.

6.1.3 Halo

To our surprise, **Halo** does not significantly influence the four tasks. Both the average scores and time are almost the same. It implied that the presence or absence of a halo does not affect these tasks.

6.1.4 Classes

For task accuracy, we can generally see that in T2 and T3, user performances with the 10-classes data set are worse than in the other two data sets. Results of ANOVA confirmed this observation, with $F(2,269) = 5.90$, $p = .00311$ for T2 and $F(2,269) = 3.07$, $p = .0482$ for T3. Post-hoc analysis indicated that in T2, accuracy performance

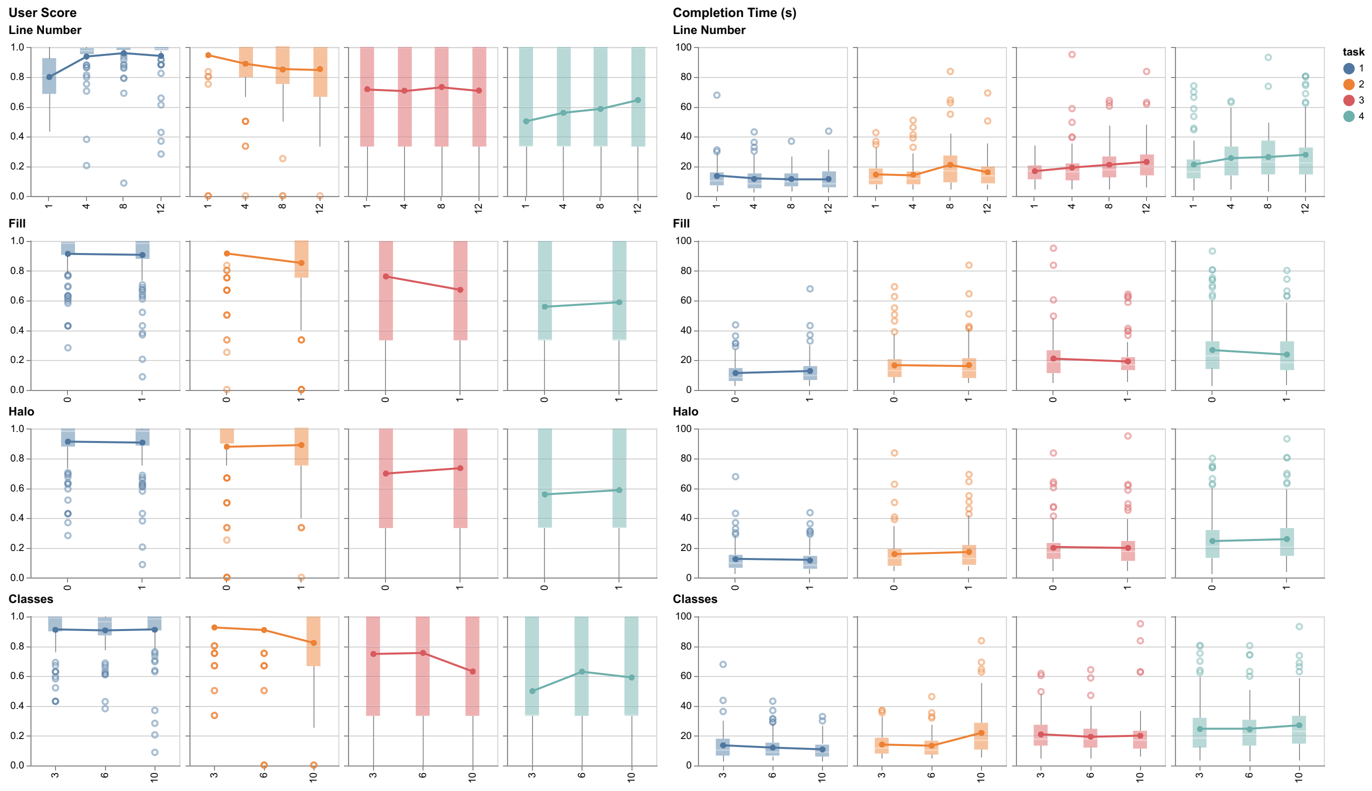


Fig. 6. Score and time distributions of different parameters on different tasks, from statistics of the user study.

with the 10-class data set is significantly worse than that with the 3-class data set (95% CI = [.0271, .181], $p = .00454$), as well as that with the 6-class data set (95% CI = [.00974, 0.164], $p = .0228$). For T3, however, the post-hoc analysis did not show any pairwise difference, indicating the low sensitivity of this task to the number of classes.

As for the time users spent on the tasks, we found in T2 that the average completion time on the data set with the data of 10 classes is much higher than those with the other two data sets. The post-hoc analysis confirmed the difference is significant, with 95% CI = [4.04, 11.7] for 3 classes and [4.80, 12.5] for 6 classes. All p-values are less than .001.

The above findings stated that more classes in the data set could negatively influence user performances for searching value peaks of a particular class. The more classes there are, the more difficult it is for users to distinguish one specific class. Especially when there are more than 10 classes, user ability will significantly decrease. Thus, designers need to be more careful using contour or try other visualization methods when dealing with data with many classes.

In general, **H1** was confirmed to some extent. Except for the **Halo** parameter, different choices of **Line Number** and **Fill** can influence user performances. The same is true for data with a different number of classes, which means that **H3** was also confirmed.

6.2 Interaction Analysis of Pairs of Parameters

We further analyzed the interactions of pairs of parameters on user performances by using two-way ANOVA. However, in the user study, we did not make a complete combination of classes in data sets and other parameters for one user, which made the results unbalanced. It is not suitable to directly apply two-way ANOVA. To address this issue, we used Type III ANOVA to analyze unbalanced data without the order of specifications. Unfortunately, we did not find any significant interaction between **Line Number** and **Fill**, which is not what we expected. However, the interactions between **Line Number** and **Classes** are significant in scores on T1 and T4, and time on T2. The results are shown in Fig. 7.

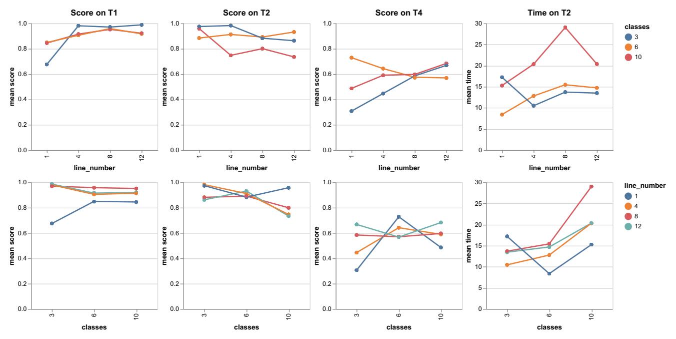


Fig. 7. Results by using contours with different line number choices on different data.

For T1, **Line Number** and **Classes** have significant interactions on score ($F(6,260) = 5.26$, $p < .001$). Fig. 7 shows that the average score of the combination of the values of 1 line and 3 classes is low, a rather strange phenomenon that cannot be reasonably explained. We then checked the look of the contour with this parameter setting. We can see that the distribution of each class in this data is sparse, as shown in Fig. 8(a). It made the completion of T1, finding the highest value, more complex, especially when using 1 line. So we assumed that the use of fewer lines in the case of a more dispersed distribution would negatively influence the search for the maximum value.

A similar situation is seen in the completion time on T2 ($F(6,260) = 3.37$, $p = .00323$). As shown in Fig. 7, the combination of 3 classes and 1 line again showed an exceptional value. We then found that one of the classes have two peaks so close in this case (Fig. 8(a)). It made participants treat them as one peak and thus spend more time looking for another peak that does not exist. Some participants also had the same feedback during the experiment. Moreover, this is not the case with 4 lines and more. So we attributed this to the characteristics of the data itself rather than a general nature.

For the accuracy score on T2 ($F(6,260) = 2.18$, $p = .0452$), Fig. 7 shows that when the number of lines increases from 1 to 4, there is a significant decrease in the score of the 10-class data. This phenomenon suggested that finding peaks is more sensitive to the number of lines in data with more classes. It can be explained that more data classes and more line numbers result in more visual clutter.

For the score on T4 ($F(6,260) = 2.18$, $p = .0450$), We can see in Fig. 7 that, unlike the other data, the accuracy rate decreases as the number of lines rises in the data of 6 classes, which is also a strange phenomenon. After checking the data, we found that in the data of 6 classes, there is no heavy crossover between different classes (Fig. 8(b)), which means that a few contour lines are enough to make the comparison between classes. Moreover, with the increase of lines, it will instead lead to visual clutters and affect the user's judgment. It is the opposite when the crossover between classes is heavy, just like the data with 3 (Fig. 8(a)) and 10 classes (Fig. 8(c)). This result told us that if the crossover between classes is not very serious, a few contour lines are more appropriate for comparing different classes.

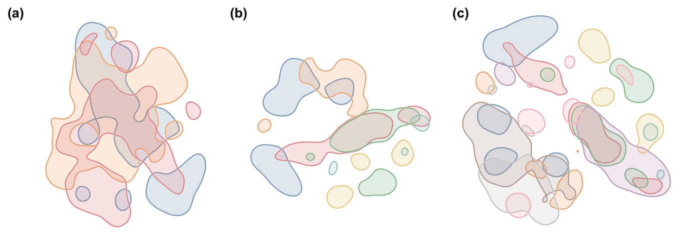


Fig. 8. Contours with 1 line on the data with 3 classes (a), 6 classes (b), and 10 classes (c). For (a), the distributions are sparse, together with two closely-connected peaks for the blue one, making the accuracy score of both T1 and T2 relatively low. For (b), the crossover between classes is not severe, which makes a single line already sufficient for comparison between classes.

As the results showed, **H2** was rejected, while **H3** can be somewhat confirmed. There exist interactions between **Line Number** and **Classes** on the score of T2. Besides, we also found some valuable findings for further guidance.

6.3 Insights

Finally, according to the previous result analysis, we gave the following general conclusions:

- Fewer contour lines do better in finding value distribution peaks, especially for data sets with more classes. They are also suitable for comparing values within one class and between different classes when the intersection is not severe. However, too few contour lines are bad for finding the highest values, especially when the value distribution is sparse, while the choice of 8 lines perhaps has the best performance. They are also inappropriate for multiclass comparison when heavy crossover exists between classes.

- Fillings can negatively influence finding peaks and comparing within one class because of the overlapping of colors.

- More classes in data can negatively affect the task of finding peaks, and the limitation may be around 10.

These findings can serve as guidance for further use. For example, suppose the goal of using multiclass contour is to show the overall distributions and roughly show some peaks. In that case, it is better to draw fewer lines, while in contrast, finding extreme values is better to use more contour lines without fillings. For comparison-related tasks, if comparisons are within a class, few contour lines with no fillings work better. If comparisons are between classes and the crossover is heavy, adding more contour lines can help. If the data has more than 10 classes, the multiclass contour may not be a suitable visualization method.

7 Application

In this section, we used the guidance to illustrate how it helps generate better contours for analyzing a real-life data set. The data set we used here contains NSF (Nation Science Foundation) projects. NSF supports fundamental research and education in science and engineering, and the projects funded by it are representative of advanced research. As time goes by, the directions of advanced research can change, which allows us to analyze the research directions changes.

Here we used contour to show the changes. We used projects in 2000, 2010, and 2020, and then explored how the research directions vary in these years. Each project has a title to describe its research. We first used BERT [8] to embed each title into a 768-dimensional vector to represent the semantics. In this high-dimensional space, the semantics of the titles represented by the two close vectors are similar, implying that the two projects are close in the research direction. We then projected these vectors onto a 2D space with t-SNE [33] and used our DSL to show the distributions.

Specific analysis goals are also needed to set appropriate attribute values. Through the contours, we want to get: 1) peaks of the distributions in 3 years, which may show the possible research directions; and 2) areas where extreme value points are possibly located, which may show the most popular direction. According to our insights from the user study, a small line number performs well on finding peaks. However, too few lines are bad for finding extreme values. To weigh these two factors, we then set the line number to 4. For fillings that are bad at finding peaks, we decided not to use them. For the halo, as we found no evident impact of it on the tasks, we chose to use it as a preference. As a result, we obtained the multiclass contour, as shown in Fig. 9.

In the figure, we can easily see the peaks, representing the research directions, move towards the right as time passed. This plot provides a very intuitive representation of the general development of research over the past 20 years. More detailed, we boxed the areas where the extreme values most probably appear in 3 years, representing the most popular directions. For 2020, we drew two boxes, as we thought these two areas might have similar research popularity.

By adding more semantics information to the contour visualization, we can know precisely the contents of the research in these areas. We placed keywords from the titles on the contours according to the location of the relevant projects. The keywords in the boxed areas may give some interpretations to the semantics. For 2000, the box contains keywords like *machine*, *system*, and *structure*, which may relate to research on traditional engineering. For 2010, the box contains keywords like *plant*, *molecular*, and *experiment*, indicating that research in life science may be the most popular around 2010. For 2020, each of the two boxes represents one possible hottest direction. One of the boxes has keywords like *network*, *system*, and *data* that are more likely to be relevant to computer science. The keyword *covid-19* appears in another box, demonstrating that the research on the COVID-19 pandemic was also popular.

In the above example, we showed how to apply our guidelines to the design of a multiclass contour for specific tasks, which let the analysts get findings more easily.

8 Discussion

This section discussed this work's scalability and limitations and provided some further research directions.

First, the proposed framework was developed by summarizing existing techniques in the field of multiclass contour visualization and was further investigated via a user study. The proposed framework and design guidance will help designers give better designs for specific tasks to a certain extent, especially for more and more inexperienced

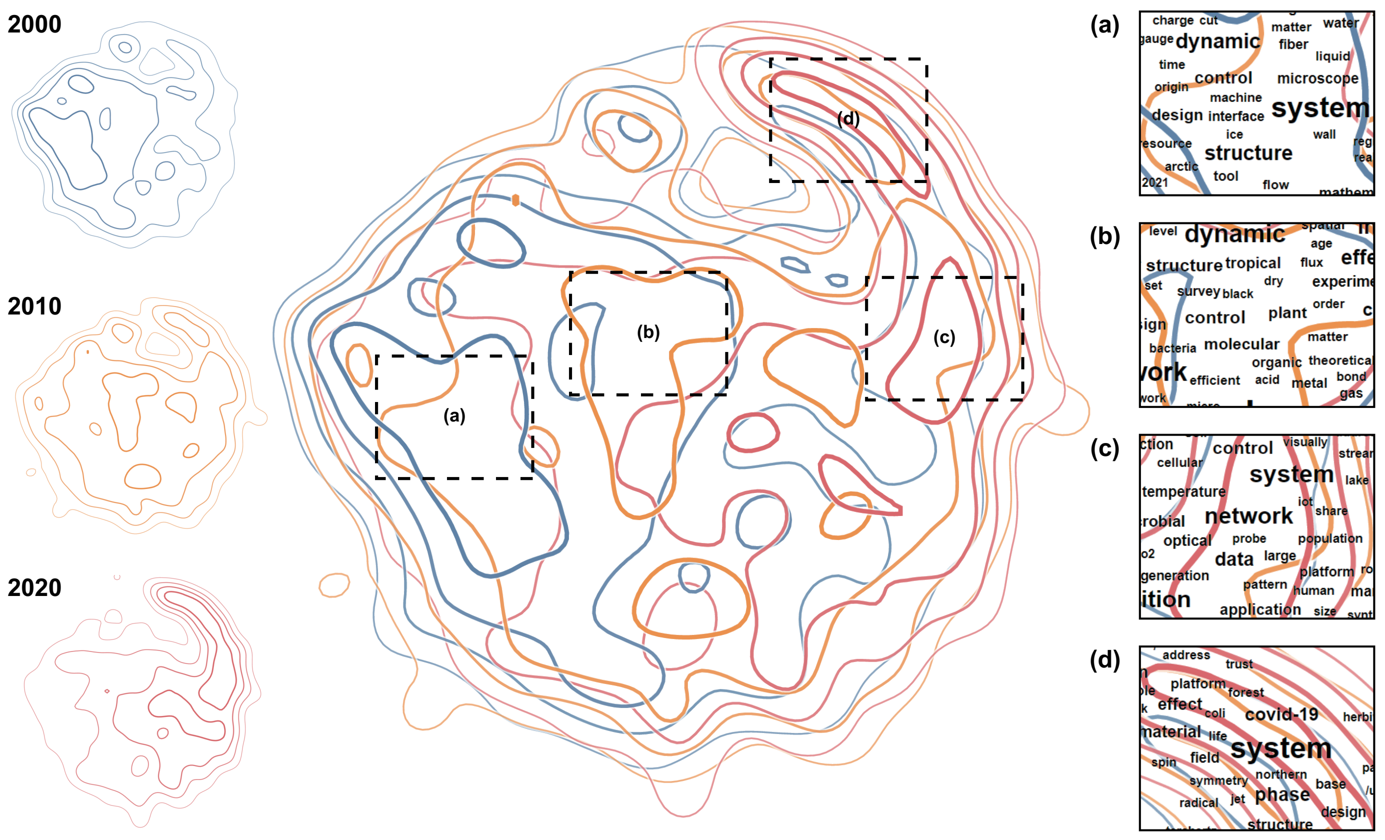


Fig. 9. An example of applying multiclass contour generated by our DSL and guidance into real-life data set. According to the semantics, we projected NSF projects in 2000, 2010, and 2020 into a 2D space. Contours in three years are shown on the left, each with an independent value threshold, while we used the same threshold in the middle multiclass contour. Additional keywords can provide more information on detailed semantics.

users, as the threshold of visualization creation is decreasing. These guidelines will also bring help to the possible future automated generation of multiclass contours. Our proposed framework and related research methods can be further applied to other methods in the scalar field, such as heatmaps [7] and dots [34], or even other common visualization methods that are not very relevant. However, applying it to scenarios involving more complex tasks or data may face some challenges. Thus, more efforts are needed to enhance the framework, such as enriching the design parameters or testing it with data from more domains. At the same time, the proposed declarative DSL also needs to be further extended by supporting more visual coding channels and user interaction types.

Second, in this work, our user study only examined three design parameters and attributes, and more research is needed to evaluate user performances under other parameters. Contour visualization can be used in different domains, and each domain may have its traditions and styles in visualization designs. In addition, user tasks related to multiclass contour can be diverse, way beyond those we used in our experiment, and the sensitivities of different tasks to these parameters may vary. Thus, it is necessary to investigate the effectiveness of all essential parameters on diverse tasks and their interaction effects in design. More user studies are needed, and the details of the experiment also need more thought.

Last but not least, from the application in a real-world data set, we have observed design conflicts when picking attribute values. A specific rendering configuration that enhances one task may lead to negative effects on completing another one. Such conflicts would be more evident and severe when the complexity of the tasks increased. It suggests that even a carefully constructed framework may have some limitations in guiding practical applications; there is no panacea for addressing every type of task. Thus, a fine-grained design summary oriented by task type in practice is worth thinking about.

9 Conclusion and Future Work

This paper aimed to provide task-oriented guidance for the design of multiclass contour visualization. To this end, we developed a framework for multiclass contour visualization by first reviewing and summarizing existing technologies and then developing a declarative DSL for fast implementation of multiclass contour visualization. Based on the framework, a task-oriented user study was conducted to demonstrate how different choices of parameters and their attributes in the framework affect user performances in different analysis tasks. The results provided valuable guidance for choosing attribute values for constructing multiclass contour visualization. Finally, we showed how the guidance could enhance real-world analysis tasks by giving an example of NSF project data.

We will extend the framework for future work by considering more design parameters and user interaction activities and testing it with data from broader application domains. The DSL can also be further improved to support the extension of the design parameters. More experimental studies are also needed to validate other design parameters and deepen the understanding of the interaction among various design parameters. Finally, we will also make efforts to develop more comprehensive design guidelines based on the framework, including more detailed user studies and summaries.

Acknowledgments

The authors thank the participants for attending the user study and the anonymous reviewers for their valuable comments. This work was supported by NSFC No. 61872013.

REFERENCES

[1] J. D. Anderson and J. Wendt. *Computational fluid dynamics*, vol. 206. Springer, 1995.

[2] P. Bergthörsson and B. R. Döös. Numerical weather map analysis. *Tellus*, 7(3):329–340, 1955.

[3] N. Cao, J. Sun, Y. R. Lin, D. Gotz, S. Liu, and H. Qu. FacetAtlas: Multifaceted visualization for rich text corpora. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1172–1181, 2010. doi: 10.1109/TVCG.2010.154

[4] W. Chen, F. Guo, D. Han, J. Pan, X. Nie, J. Xia, and X. Zhang. Structure-based suggestive exploration: A new approach for effective exploration of large networks. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):555–565, 2019. doi: 10.1109/TVCG.2018.2865139

[5] H. Choi, W. Choi, T. M. Quan, D. G. Hildebrand, H. Pfister, and W. K. Jeong. Vivaldi: A domain-specific language for volume processing and visualization on distributed heterogeneous systems. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2407–2416, 2014. doi: 10.1109/TVCG.2014.2346322

[6] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):929–936, 2009. doi: 10.1109/TVCG.2009.122

[7] A. Coninx, G. Bonneau, J. Droulez, and G. Thibault. Visualization of uncertain scalar data fields using color scales and perceptually adapted noise. In *Proceedings of the 8th Symposium on Applied Perception in Graphics and Visualization, APGV 2011, Toulouse, France, August 27-28, 2011*, pp. 59–66. ACM, 2011. doi: 10.1145/2077451.2077462

[8] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

[9] C. Healey and J. Enns. Attention and visual memory in visualization and computer graphics. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1170–1188, 2012. doi: 10.1109/TVCG.2011.127

[10] J. Heer and M. Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, 2010. doi: 10.1109/TVCG.2010.144

[11] J. Hoffswell, A. Borning, and J. Heer. SetCoLa: High-level constraints for graph layout. *Computer Graphics Forum*, 37(3):537–548, 2018. doi: 10.1111/cgf.13440

[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. doi: 10.1109/TVCG.2018.2865141

[13] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[14] G. Li, M. Tian, Q. Xu, M. J. McGuffin, and X. Yuan. GoTree: A grammar of tree visualizations. In *Proceedings of the International Conference on Human Factors in Computing Systems*, pp. 1–13, 2020. doi: 10.1145/3313831.3376297

[15] Z. Li, C. Zhang, S. Jia, and J. Zhang. Galex: Exploring the evolution and intersection of disciplines. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1182–1192, 2020. doi: 10.1109/TVCG.2019.2934667

[16] Y. R. Lin, J. Sun, N. Cao, and S. Liu. ContexTour: Contextual contour visual analysis on dynamic multi-relational clustering. In *Proceedings of the 10th SIAM International Conference on Data Mining, SDM 2010*, pp. 418–429, 2010. doi: 10.1137/1.9781611972801.37

[17] M. Lu, S. Wang, J. Lanir, N. Fish, Y. Yue, D. Cohen-Or, and H. Huang. Winglets: Visualizing association with uncertainty in multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):770–779, 2020. doi: 10.1109/TVCG.2019.2934811

[18] R. MacIejewski, S. Rudolph, R. Hafen, A. Abusalah, M. Yakout, M. Ouzzani, W. S. Cleveland, S. J. Grannis, and D. S. Ebert. A visual analytics approach to understanding spatiotemporal hotspots. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):205–220, 2010. doi: 10.1109/TVCG.2009.100

[19] O. Malkai, M. Lu, and D. Cohen-Or. Clusterplot: High-dimensional cluster visualization. *CoRR*, abs/2103.02992, 2021.

[20] C. Maple. Geometric design and space planning using the marching squares and marching cube algorithms. In *Proceedings of the 2003 International Conference on Geometric Modeling and Graphics, GMAG 2003*, pp. 90–95, 2003. doi: 10.1109/GMAG.2003.1219671

[21] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, 2013. doi: 10.1109/TVCG.2013.65

[22] D. Park, S. M. Drucker, R. Fernandez, and N. Elmqvist. Atom: A grammar for unit visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3032–3043, 2018. doi: 10.1109/TVCG.2017.2785807

[23] N. H. Riche and T. Dwyer. Untangling euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010. doi: 10.1109/TVCG.2010.210

[24] R. E. Roth. An empirically-derived taxonomy of interaction primitives for interactive cartography and geovisualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2356–2365, 2013. doi: 10.1109/TVCG.2013.130

[25] S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[26] A. Sarikaya, S. Member, and M. Gleicher. Scatterplots: Tasks, data, and designs. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):402–412, 2017. doi: 10.1109/TVCG.2017.2744184

[27] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, 2014. doi: 10.1111/cgf.12391

[28] 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, 2017. doi: 10.1109/TVCG.2016.2599030

[29] R. Scheepens, H. van de Wetering, and J. J. van Wijk. Contour based visualization of vessel movement predictions. *International Journal of Geographical Information Science*, 28(5):891–909, 2014. doi: 10.1080/13658816.2013.868466

[30] M. Shih, C. Rozhon, and K. L. Ma. A declarative grammar of flexible volume visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1050–1059, 2019. doi: 10.1109/TVCG.2018.2864841

[31] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28(3):967–974, 2009. doi: 10.1111/j.1467-8659.2009.01452.x

[32] W. Tao, X. Hou, A. Sah, L. Battle, R. Chang, and M. Stonebraker. Kyrix-S: Authoring scalable scatterplot visualizations of big data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):401–411, 2021. doi: 10.1109/TVCG.2020.3030372

[33] L. Van Der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11):2579–2605, 2008.

[34] C. Ware. Quantitative texton sequences for legible bivariate maps. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1523–1530, 2009. doi: 10.1109/TVCG.2009.175

[35] R. T. Whitaker, M. Mirzargar, and R. M. Kirby. Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2713–2722, 2013. doi: 10.1109/TVCG.2013.143

[36] J. Yuan, S. Xiang, J. Xia, L. Yu, and S. Liu. Evaluation of sampling methods for scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1720–1730, 2021. doi: 10.1109/TVCG.2020.3030432

[37] J. Zhao, X. Liu, C. Guo, Z. C. Qian, and Y. V. Chen. Phoenixmap: An abstract approach to visualize 2d spatial distributions. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2000–2014, 2021. doi: 10.1109/TVCG.2019.2945960

[38] M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt. Interactive level-of-detail rendering of large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2486–2495, 2012. doi: 10.1109/TVCG.2012.238