EG UK Computer Graphics & Visual Computing (2018)
G. Tam and F. Vidal (Editors)

Spectrum: A C++ Header Library for Colour Map Management

Richard C. Roberts[1], Liam McNabb[1], Naif AlHarbi[1], Robert S. Laramee[1]

[1]Visual and Interactive Computer Group, Swansea University, Swansea, Wales

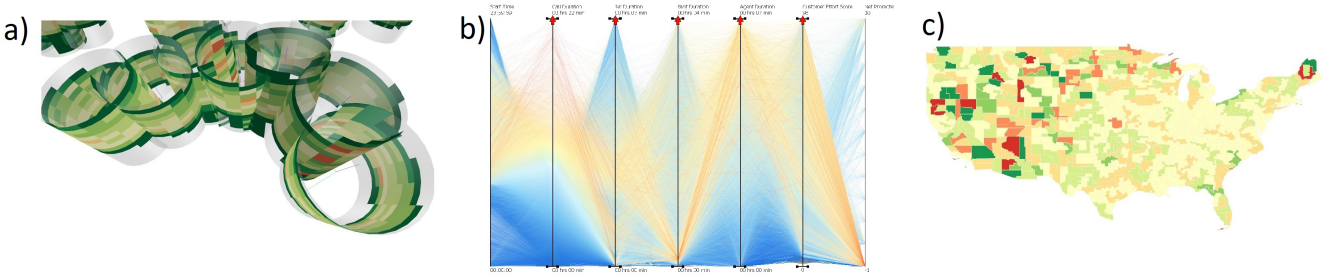

Figure 1: *Here are some example images taken from publications that have utilised the Spectrum library at various stages of its development. The library has been used in many fields of visualisation. Here we show a) scientific visualisation [NA18], b) information visualisation [RLS*18], and c) geospatial visualisation [MLF18].*

Abstract

The use of colour mapping is fundamental to visualisation research. It acts as an additional layer beyond rendering in the spatial dimensions and provides a link between values in any dataset. When designing and building visualisation research software, the process of creating and managing a colour mapping system can be time-consuming and complex. Existing alternatives offer niche features and require complex dependencies or installations. We present Spectrum; an open source colour map management library that is developer friendly with no installation required, and that offers a wide variety of features for the majority of use cases. We demonstrate the utility of the library through simple snippets of code and a number of examples which illustrate its ease of use and functionality, as well as a video demonstrating the installation and use of the library in under two minutes. It is a very valuable jump-start tool for developers and researchers who need to focus on other tasks.

CCS Concepts

•***Software and its engineering*** → ***Software libraries and repositories; Software design engineering;*** *Open source model;*

1. Introduction and Motivation

> *"Colour is a power which directly influences the soul."* - Kandinsky [Kan12]

Whilst the selection of a good colour map is a contested subject [ZH16], the implementation of these colour maps is rarely discussed. When developing both previous and novel visualisations, the beginning of each project forces the developer to either write a colour map manager from scratch or to recycle and modify old code written for the purpose of a different project. Our colour-map library has been developed over the past four years, serving as the colour manager for multiple visualisation projects and has been used by a number of Data Visualisation PhD students [RLS*18, NA18, MLF18, RTL*16]. See Figure 1. We implement a broad range of features, accommodating for the majority of use cases, and release the software in the form of an open source C++ header-only library available to the public.

Abstracting away from the implementation of a colour map manager enables the developers to focus on other important design decisions, such as the choice of colour map. We provide default colour maps bundled into the library but accommodate for the addition of new maps through a simple coding interface. The developer can switch freely between a selection of colour maps without modifying their own code by simply updating a static index value associated with the colour manager. We have implemented this library with simplicity in mind. Our goal is to create a library that can be

included in a project and used in a very short amount of time. We demonstrate the speed by which this library can be utilised in the accompanying video – showing the process of importing the library into the project, to extracting and utilising colour values in under two minutes. To view this tutorial, please see the attached video or go to:

https://vimeo.com/273322682

Our contribution is to provide an open source maturely developed header-only library that can be used by any C++ developer working with colours. Whilst the application has been developed for OpenGL, support for hex colour conversion is included and so can be used for an even wider range of projects.

We choose C++ Qt to develop our visualisation research applications because it enables us to utilise the powerful C++ language alongside the feature-rich framework of Qt. Qt is a fast, cross-platform, open source and well documented. It enables us to build analytics tools with the visual component of OpenGL quickly and easily. For these reasons, we choose to create a C++ library that can easily be used alongside our chosen framework and save us time in our future projects.

The next section briefly covers related work in the area. The following section describes the features of the library and demonstrates how it can be used for varying types of data. We provide pseudocode for the main colour interpolation algorithm - and code snippets demonstrating how the library works, examples of the colour map used in practice, and a supplementary tutorial video. The video demonstrates that the library can be incorporated and used in under two minutes.

2. Related Work

Colour Map Theory: Silva et al. present a survey of colour usage in visualisation [SSM11]. This paper presents an overview of the theory behind colour scales and the guidelines laid out for the efficient use of colour maps.

Zhou and Hansen also present a survey of colour maps in visualisation [ZH16], with the intention of it being a reference of colour map choices for readers based on the data and tasks they may face.

A body of research looks at the common mistakes made in colour mapping, [RT98, SMS07, BI07, PRS*15]. The most common mistake when choosing a colour map is the use of the rainbow map, which looks aesthetically pleasing, but fails to convey a correct linear scale that the user can accurately perceive. Some research focuses on how colour maps can be used efficiently [Mac99], by task [Rhe00], or by data [Hea96]. Even a number of visualisation books discuss colour map theory [Tel14, Kir16, Rhy16].

Wang et al. create an algorithm that selects appropriate colour maps [WGM*08] whilst overcoming the challenge of transparency issues in colour mapping. More colour selection guidelines have also been published since, [LFK*13, FWD*17, LSS12]

Meyer and Greenberg also explored the challenges associated with colour blind individuals [MG88], which was more recently explored by Kuhn et al. [KOF08]. Studies on the class intervals have also been completed [Tob73, BP02], which is largely used on choropleth maps.

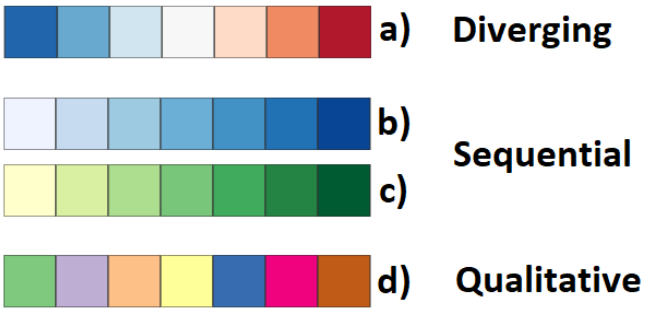

Figure 2: *This image exemplifies the different classifications of colour map. These colour maps are taken from ColorBrewer [HB03].*

The literature covers both univariate mapping [War88], and multivariate mapping [Mon99]. The scope of the colour map theory research is extensive – looking closely at many aspects of the colours within the maps. We abstract away from this and focus on the implementation of mapping colour in a range.

Colour Map Resources: Color Brewer is a popular online resource for selecting colour maps [BP02, Bre94], where colour map suggestions are made based on the type of data being presented. More recently, Colorgorical is an online tool that helps generate new maps which are highly customisable [GLS17].

Whilst most of this research identifies the theory behind the different types of colour maps, rarely does the research provide implementation details. Moreland provides some information on this, but only on the interpolation between colours [Mor09]. This library enables the user to focus on selecting the correct colour map without having to invest time into the implementation of a software specific colour map manager. Our library is generalisable for most use cases and can be downloaded (see section 6), and implemented in a matter of minutes.

3. Colour Mapping Background

In this section, we discuss the scope of this library and provide an overview of how different data types have varying requirements when mapping them to colour. We briefly discuss the theory behind colour maps and what differentiates them, then we demonstrate the broad spectrum of features implemented in our library.

3.1. Colour Map Classifications

The first important distinction to make is between the different types of colour maps. The most popular academic colour mapping resource, ColorBrewer divides maps into three different classifications [HB03, Bre94].

Diverging: The diverging colour scheme emphasises two ends of a spectrum with a middle divider [Bre94]. Two colours diverge from a centre lighter colour value. For example, this can be used when visualising a divergence from an average, where the upper colour represents higher than the average, and the lower colour represents worse than the average. See Figure 2 a).

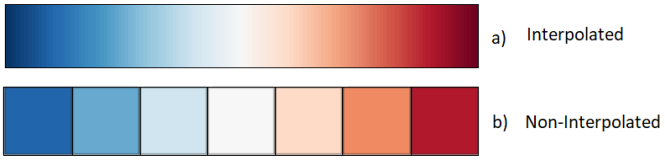

Figure 3: *This figure shows the difference between an interpolated and non-interpolated colour map. If the colour map returns an interpolated colour, then a new colour is generated representing a floating point along the spectrum of the colour map. A non-interpolated map returns the specific colour class from within the map.*

Sequential: This colour classification implies an order to the scale of data that range from low to high [Bre94]. Low range values are typically mapped to lighter colours and high range values are mapped to darker colours. Sequential maps can either be single hue – where one dominant colour is mapped from light to dark (See Figure 2 b)), or multi-hue – where a light colour transitions to a differing dark colour. See Figure 2 c).

Qualitative: This colour map is made up of categorical colours that do not imply order [Bre94]. The variance in colour hue enables nominal data to be represented without confusion. See Figure 2 d).

3.2. Colour Map Operating Modes

Colour maps have two main components; the colour class list and the range values. Each colour in the map list is known as a 'class'. In diverging and sequential colour maps, these classes are ordered using a gradient such that their neighbouring classes vary slightly in hue or shade. In qualitative colour maps class colours are unrelated and typically do not resemble any order. The upper and lower range values of a colour map represent the scale of the data being depicted. When retrieving a colour from a colour map, there are two operating modes;

Interpolated: When depicting a specific value within the data range of the colour map, a colour is generated at the precise point along the colour spectrum according to its value in the range. The correct value is interpolated between the class colours of the colour map. See Figure 3 a).

Non-Interpolated An alternative to the above method is to simply return the class colour that the value along the range falls into where no interpolation between class colours is necessary. See Figure 3 b).

3.3. Data Types

Data can be broken down into four basic categories; nominal, ordinal, interval, and ratio [Ste46]. Our library accommodates for each of these data types and their different requirements for colour mapping. Depending on what data is being rendered and the type of colour the developer is interested in accessing, different library functions should be used to return the values.

Nominal Data Nominal data exists in discrete groups which are independent of one another. These groups cannot generally be placed into a meaningful order and are typically identified by a name as opposed to a number. e.g. categories of food. This datatype is often dealt with according to frequency. We note that a count value of nominal data becomes ratio data.

Ordinal Data Ordinal data can be sorted, but the distance between groups is unspecified. e.g. the grading system (A-F).

Interval & Ratio Interval data can be ordered and the distance between the groups has meaning. Additionally a value of zero is not meaningful. For example, the measurements Celsius & Fahrenheit – 20 degrees Celsius is not twice as hot as 10 degrees Celsius because the base of zero degrees is not an absence of heat.

Ratio data can be ordered with a meaningful distance between values where a zero value has meaning and a fraction or ratio can be taken. For example, measurements of height and distance. One mile is half of two miles, and zero miles is a meaningful measure.

4. Spectrum Outline

Contained within the .h file ColourManager are four different classes; ColourManager, Colour, ColourMap, and CMList. We chose to build this open source library as a header-only file so that it can easily be used by anyone, without any dependencies and complicated build requirements. Whilst header files have to be compiled at build time, the code base of the library is lightweight enough for it to have virtually no effect on compilation time. In this section, we explain the roles of each class and provide some example code to demonstrate how they work.

4.1. Colour

The colour class is a simple container object for colour channel values. Each colour has a name value which can be used when mapping nominal data. The three class constructors instantiate the class through different methods. The first has three arguments, each being the RGB values of the colour, either a float 0.0-1.0 range or an int 0-255 range. The function automatically calculates which is being used. The second takes in a string hex value as the argument and the colour is derived from that. The third constructor is empty so that the developer can manually set the values later.

```
1  // Method 1
2  Colour c1(76, 196, 150);
3
4  //Method 2
5  Colour c2(''#4cc496'');
6
7  //Method 3
8  Colour c3();
9  c3.setR(76);
10 c3.setG(196);
11 c3.setB(150);
12
```

When retrieving the correct values from the colour object, the developer can either individually retrieve the red, green, and blue

colour channels through accessor functions, or they can return a string of the hex colour value.

```
//Returns channel values between 0-255
GLint iR = c.getIntR();
GLint iG = c.getIntG();
GLint iB = c.getIntB();

//Returns channel values between 0.0-1.0
float fR = c.getR();
float rG = c.getG();
float fB = c.get();

//Return hex colour value
std::string hexCol = getHexColour();


```

The OpenGL colour can now be set using;

```
glColour3i(iR,iG,iB);
//or
glColour3f(fR,fG,fB);

```

If required, the colour class also has an alpha value that can be accessed or set through similar methods.

4.2. Colour Map

The colour map class contains a vector of Colour objects which act as the colour classes of the map. In order to create a colour map, it is given a name, a classification, and a list of colours.

The CMClassification is an enum representing the type of colour map being created. These include Sequential, Diverging, and Qualitative.

```
//This is an example colour map creation
ColourMap redSeq("Red Sequential");
redSeq.setClassification(CMClassification::
  SEQUENTIAL);
float DEFAULT_OPACITY=1.0f
redSeq.addColour(255,255,204,DEFAULT_OPACITY);
redSeq.addColour(255,237,160,DEFAULT_OPACITY);
redSeq.addColour(254,217,118,DEFAULT_OPACITY);
redSeq.addColour(254,178,76,DEFAULT_OPACITY);
redSeq.addColour(253,141,60,DEFAULT_OPACITY);
redSeq.addColour(252,78,42,DEFAULT_OPACITY);
redSeq.addColour(227,26,28,DEFAULT_OPACITY);
redSeq.addColour(189,0,38,DEFAULT_OPACITY);
redSeq.addColour(128,0,38,DEFAULT_OPACITY);

```

The colours can also be given names when being added to the map.

```
redSeq.addColour(255,0,0,1.0f, "Red");

```

The colours can be accessed in the colour map through square bracket operators by both the index of the colour or the name of the colour.

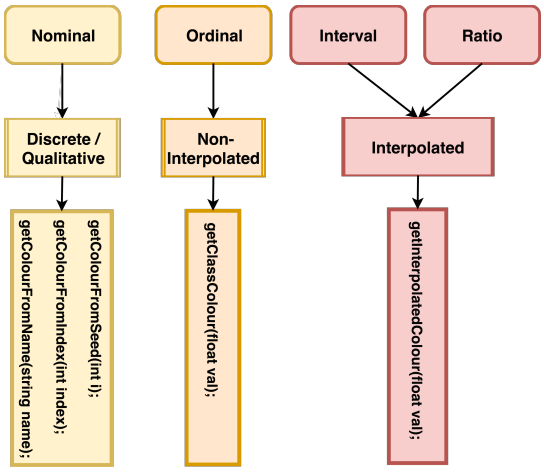

Figure 4: *This image shows the different data types and their relationship to each library feature the developer may want to use.*

```
//By index
redSeq[4];
//By name
redSeq["Red"];

```

4.3. CMList

The colour map list class is a simple container class that holds all the different colour maps loaded into the software. Adding a colour map to the list is done through the addColourMap() function.

```
CMList::addColourMap(redSeq);

```

Returning a subset of the colour maps is done using the CMClassification enum;

```
//Returns the complete list of colours
std::vector<Colour> list
= CMList::getMapList();

//Returns only SEQUENTIAL maps
std::vector<Colour> list
=getMapList(CMClassification::SEQUENTIAL);

```

Using the returned list, the developer can select their desired colour map and set it as the current map in the ColourManager class.

4.4. ColourManager

This is the main class of the Spectrum library. It contains the currently selected colour map and handles the responsibility surrounding the colour return values. Five functions return colour values for various use cases. Figure 4 describes the function purpose according to the data type being displayed. Each of these functions returns a Colour object. These functions are;

1. getClassColour(float val) – Returns the closest colour 'class' in the current colour list.
2. getInterpolatedColour(float val) – Returns the interpolated colour between the 'class' colour list.
3. getColourFromSeed(int seed) – Returns a random colour based off the seed value argument. This seed value ensures that the same colour can be recreated at any time in the future.
4. getColourFromIndex(int index) – Returns the class colour based on the index in the current colour map.
5. getColourFromName(std::string name) – Returns the colour in the current colour map with the same name as the function argument. If no match is found, a black colour is returned with the name 'NoColour'.

Flipping the bool value InvertColourMapFlag() will reverse the current colour map order, and the function returnRandomColourMap(int seed, int listSize) returns a brand new qualitative colour map that can then be added to the CMList. In this function, the seed value ensures the colour map can be replicated at any point in the future, and the listSize function argument determines the class size within the colour map.

5. Spectrum: Examples in Practice

In this section, we demonstrate how to use the Spectrum library using example code. Installation of the library is simple and requires no compilation. The developer simply adds the .h file to their project and includes the file in their rendering classes. The supplementary video demonstrates using the Spectrum library in only two minutes.

5.1. Basic Use

The first step is to initialise the colour manager in the main function – *ColourManager::Init_ColourManager();*. This sets up the colour lists and loads them into memory. It is important to remember to include the header in each file the library is used in.

```
#include "colourmanager.h"

int main(int argc, char *argv[])
{
ColourManager::Init_ColourManager();
QApplication a(argc, argv);
MainWindow w;
w.showMaximized();
return a.exec();
}

```

Once the setup is complete, the developer can instantiate a ColourManager object using an upper and lower range for the map. Throughout the lifetime of the object, the ranges can be adjusted using accessor functions.

```
float lowerRange = 0;
float upperRange = 100;
ColourManager manager(lowerRange,upperRange);

```

When returning a colour value, the developer can choose from a selection of functions, depending on their data and requirements. In this example, we choose the interpolated colour value.

```
float val = 27.0f;
Colour c = manager.getInterpolatedColour(val);

```

This Colour object 'c' now contains the interpolated colour between the upper and lower range of the colour map.

This demonstrates the complete pipeline of the Spectrum Library. In order to make this software valuable to developers, it needs to save time in development. Every aspect of the implementation ensures that the most work is done with the smallest amount of code, with the quickest setup. We demonstrate the speed at which the library can be used in under two minutes.

5.2. Changing Current Colour Map

The developer can select the desired colour map using the functions outlined in section 4.3. Once the correct colour map has been chosen, it can be selected as the current map in a number of different ways:

```

//Get the first diverging colour map.
ColourMap M = CMList::getMapList(
   CMClassification::DIVERGING)[0];

//Directly from a colour map
ColourManager::setCurrentColourMap(M);

//Or through changing the index value
ColourManager::setColourMapIndex(M.getIndex());

```

The map can be changed at any time – whereby every instance of a ColourManager class will use the same colour map. This universal map prevents confusion with duplicated instances of different colour schemes across the developer's code base.

5.3. Adding Colours

The Spectrum library is pre-bundled with a range of different colour maps. These can easily be removed or supplemented by modifying the *ColourManager::Init_ColourManager()* function. The function requires at least one colour map to be added, and then *setupIndexesInList()* to be called at the end of the function to assign index values to each map in the list.

```
Init_ColourManager(){
//Add colour
ColourMap redBlue;
redBlue.setMapName("Red to Blue");
redBlue.setClassification(DIVERGING);
redBlue.addColour(165,0,38,1.0f);
redBlue.addColour(215,48,39,1.0f);
redBlue.addColour(244,109,67,1.0f);
redBlue.addColour(253,174,97,1.0f);
redBlue.addColour(254,224,144,1.0f);
redBlue.addColour(255,255,191,1.0f);
```

```
12  redBlue.addColour(224,243,248,1.0f);
13  redBlue.addColour(171,217,233,1.0f);
14  redBlue.addColour(116,173,209,1.0f);
15  redBlue.addColour(69,117,180,1.0f);
16  redBlue.addColour(49,54,149,1.0f);
17  ColourMapList().addColourMap(redBlue);
18
19  //Add as many colours as required...
20
21  //Setup the colour list indexes.
22  ColourMapList().setupIndexesInList();
23  }
24
```

These colour maps are compiled and added to the CMList once *ColourManager::Init_ColourManager()* is called. This function is called as early as possible in the program to ensure the map list is set up before accessing the ColourManager.

In Figure 1, we show three published papers which utilise past implementations of the Spectrum Library. Image a) demonstrates a novel visualisation of lipid-protein interaction [NA18]. Image b) presents a parallel coordinates plot of call centre interaction data where the colour can be mapped to any of the axis values [RLS*18]. Image c) shows a choropleth map where colour depicts value accumulation of area amalgamation [MLF18].

6. Downloading Spectrum

The open source Spectrum library can be downloaded from github:

https://github.com/richardroberts1992/Spectrum

The use of Spectrum simply requires the developer to add the ColourManager.h file to their project. A test program is available within the project which exemplifies different use cases of the software. It should be noted that this library is implemented as a singleton, and should be used carefully to avoid memory leaks. The license allows free use of the library to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, and to permit persons to whom the software is furnished to do so (MIT License).

7. Conclusion and Future Work

We develop an open source library that can be used by any C++ developer to quickly build visual applications whilst abstracting away from the implementation of a colour map manager. We provide clear documentation for its use and a video demonstration to showcase the speed and ease by which the library can be used.

Whilst the code is not complex or unique, the design makes its application universal to most fields of visualisation and so can be used to get projects quickly up and running without additional man-hours wasted on implementing the same functionality over and over across different projects. The Spectrum library has been used by a number of PhD candidates in our visualisation research team, who have found the code to be a valuable resource in their research and software development. We have packaged and documented this library so that others may also benefit from its features and ease of use.

In the future we aim to improve the features of the Spectrum library, incorporating class intervals of differing sizes so that the developer is able to depict customised ranges of data – similar to colour gradient tools used in graphics applications such as Photoshop. We will continue work on ensuring the thread-safe nature of the singleton implementation of the library. We also plan to implement features that allow loading input files of colour values to create new colour maps and to get feedback from the library's users to incorporate their ideas for new features.

8. Acknowledgements

We would like to thank all who have contributed to this research. We thank QPC Ltd. for their financial support and their supply of the call centre data, and we thank KESSII who have been the primary funding body for this research. KESS is part-funded by the Welsh Government's European Social Fund (ESF) convergence programme for West Wales and the Valleys. In addition to this, would like to thank Dylan Rees and Liam McNabb for providing valuable feedback on this work.

References

[BI07] BORLAND D., II R. M. T.: Rainbow color map (still) considered harmful. *IEEE Computer Graphics and Applications 27*, 2 (March 2007), 14–17. doi:10.1109/MCG.2007.323435. 2

[BP02] BREWER C. A., PICKLE L.: Evaluation of methods for classifying epidemiological data on choropleth maps in series. *Annals of the Association of American Geographers 92*, 4 (2002), 662–681. 2

[Bre94] BREWER C. A.: Color use guidelines for mapping. *Visualization in modern cartography* (1994), 123–148. 2, 3

[FWD*17] FANG H., WALTON S., DELAHAYE E., HARRIS J., STORCHAK D. A., CHEN M.: Categorical colormap optimization with visualization case studies. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan 2017), 871–880. doi:10.1109/TVCG.2016.2599214. 2

[GLS17] GRAMAZIO C. C., LAIDLAW D. H., SCHLOSS K. B.: Colorgorical: Creating discriminable and preferable color palettes for information visualization. *IEEE transactions on visualization and computer graphics 23*, 1 (2017), 521–530. 2

[HB03] HARROWER M., BREWER C. A.: Colorbrewer. org: an online tool for selecting colour schemes for maps. *The Cartographic Journal 40*, 1 (2003), 27–37. 2

[Hea96] HEALEY C. G.: Choosing effective colours for data visualization. In *Proceedings of the 7th Conference on Visualization'96* (1996), IEEE Computer Society Press, pp. 263–ff. 2

[Kan12] KANDINSKY W.: *Concerning the spiritual in art*. Courier Corporation, 2012. 1

[Kir16] KIRK A.: *Data visualisation: a handbook for data driven design*. Sage, 2016. 2

[KOF08] KUHN G. R., OLIVEIRA M. M., FERNANDES L. A. F.: An efficient naturalness-preserving image-recoloring method for dichromats. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (Nov 2008), 1747–1754. doi:10.1109/TVCG.2008.112. 2

[LFK*13] LIN S., FORTUNA J., KULKARNI C., STONE M., HEER J.: Selecting semantically-resonant colors for data visualization. In *Proceedings of the 15th Eurographics Conference on Visualization* (Chichester, UK, 2013), EuroVis '13, The Eurographs Association & John Wiley & Sons, Ltd., pp. 401–410. URL: http://dx.doi.org/10.1111/cgf.12127, doi:10.1111/cgf.12127. 2

[LSS12] LEE S., SIPS M., SEIDEL H.-P.: Perceptually-driven visibility optimization for categorical data visualization. *IEEE Transactions on visualization and computer graphics* (2012), 1. 2

[Mac99] MACDONALD L. W.: Using color effectively in computer graphics. *IEEE Computer Graphics and Applications 19*, 4 (1999), 20–35. 2

[MG88] MEYER G. W., GREENBERG D. P.: Color-defective vision and computer graphics displays. *IEEE Computer Graphics and Applications 8*, 5 (1988), 28–40. 2

[MLF18] MCNABB L., LARAMEE R. S., FRY R.: Dynamic choropleth maps - using amalgamation to increase area perceivability. In *International Journal of Computer Vision and Image Processing* (2018). – forthcoming. 1, 6

[Mon99] MONTAG E. D.: The use of color in multidimensional graphical information display. In *Color and Imaging Conference* (1999), vol. 1999, Society for Imaging Science and Technology, pp. 222–226. 2

[Mor09] MORELAND K.: Diverging color maps for scientific visualization. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II* (Berlin, Heidelberg, 2009), ISVC '09, Springer-Verlag, pp. 92–103. URL: http://dx.doi.org/10.1007/978-3-642-10520-3_9, doi:10.1007/978-3-642-10520-3_9. 2

[NA18] NAIF ALHARBI MATTHIEU CHAVENT M. K. R. S. L.: Lpiv: A novel abstraction for time-dependent lipid-protein interaction. 1, 6

[PRS*15] PANDEY A. V., RALL K., SATTERTHWAITE M. L., NOV O., BERTINI E.: How deceptive are deceptive visualizations?: An empirical analysis of common distortion techniques. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (New York, NY, USA, 2015), CHI '15, ACM, pp. 1469–1478. URL: http://doi.acm.org/10.1145/2702123.2702608, doi:10.1145/2702123.2702608. 2

[Rhe00] RHEINGANS P. L.: Task-based color scale design. In *28th AIPR Workshop: 3D Visualization for Data Exploration and Decision Making* (2000), vol. 3905, International Society for Optics and Photonics, pp. 35–44. 2

[Rhy16] RHYNE T.-M.: Applying color theory to digital media and visualization. 2

[RLS*18] ROBERTS R., LARAMEE R. S., SMITH G. A., BROOKES P., D'CRUZE T.: Smart brushing for parallel coordinates. *IEEE Transactions on Visualization and Computer Graphics* (2018), 1–1. doi:10.1109/TVCG.2018.2808969. 1, 6

[RT98] ROGOWITZ B. E., TREINISH L. A.: Data visualization: the end of the rainbow. *IEEE spectrum 35*, 12 (1998), 52–59. 2

[RTL*16] ROBERTS R. C., TONG C., LARAMEE R. S., SMITH G. A., BROOKES P., D'CRUZE T.: Interactive analytical treemaps for visualisation of call centre data. In *Proceedings of the Conference on Smart Tools and Applications in Computer Graphics* (Goslar Germany, Germany, 2016), STAG '16, Eurographics Association, pp. 109–117. URL: https://doi.org/10.2312/stag.20161370, doi:10.2312/stag.20161370. 1

[SMS07] SILVA S., MADEIRA J., SANTOS B. S.: There is more to color scales than meets the eye: a review on the use of color in visualization. In *Information Visualization, 2007. IV'07. 11th International Conference* (2007), IEEE, pp. 943–950. 2

[SSM11] SILVA S., SANTOS B. S., MADEIRA J.: Using color in visualization: A survey. *Computers & Graphics 35*, 2 (2011), 320–333. 2

[Ste46] STEVENS S. S.: On the theory of scales of measurement. *Science 103*, 2684 (1946), 677–680. URL: http://science.sciencemag.org/content/103/2684/677, arXiv:http://science.sciencemag.org/content/103/2684/677.full.pdf, doi:10.1126/science.103.2684.677. 3

[Tel14] TELEA A. C.: *Data visualization: principles and practice*. CRC Press, 2014. 2

[Tob73] TOBLER W. R.: Choropleth maps without class intervals? *Geographical analysis 5*, 3 (1973), 262–265. 2

[War88] WARE C.: Color sequences for univariate maps: Theory, experiments and principles. *IEEE Computer Graphics and Applications 8*, 5 (1988), 41–49. 2

[WGM*08] WANG L., GIESEN J., MCDONNELL K. T., ZOLLIKER P., MUELLER K.: Color design for illustrative visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1739–1754. 2

[ZH16] ZHOU L., HANSEN C. D.: A survey of colormaps in visualization. *IEEE transactions on visualization and computer graphics 22*, 8 (2016), 2051–2069. 1, 2