

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Scatterplot visualization of hierarchically clustered data

Daniel Gruncl

Supervisor: Ing. Čmolík Ladislav, PhD.
May 2023

I. Personal and study details

Student's name: **Gruncel Daniel** Personal ID number: **483765**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Graphics and Interaction**
Study program: **Open Informatics**
Specialisation: **Computer Graphics**

II. Master's thesis details

Master's thesis title in English:

Scatterplot visualization of hierarchically clustered data

Master's thesis title in Czech:

Vizualizace hierarchicky seskupených dat pomocí bodového diagramu

Guidelines:

Get familiar with dimension reduction techniques (e.g., star coordinates, t-SNE) and with clustering methods, including hierarchical clustering, for n-dimensional data. Further, get familiar with visualization techniques suitable for visualization of n-dimensional data whose dimension was reduced to 2D. Focus, especially on visualization of the individual clusters, even when the clusters are overlapping. Based on the analysis, design navigation in the 2D space with semantic zoom, which will allow transition from overview of all clusters through visualization of their subclusters to the visualization of individual data points. Further, design visualization of hierarchy of the clusters and link both visualizations with linked/connected views. Implement the designed visualization application using Java language and OpenGL. Verify functionality of the resulting application on data sets provided by the supervisor and with qualitative test with at least six participants.

Bibliography / sources:

- [1] T. Munzner. Visualization Analysis and Design. A K Peters Visualization Series, CRC Press, 2014.
- [2] A. C. Telea. Data Visualization: Principles and Practice (2nd edition). CRC Press, 2014.
- [3] A. Sarikaya and M. Gleicher, Scatterplots: Tasks, Data, and Designs, IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 1, pp. 402-412, 2018.
- [4] N. Waldin, M. Le Muzic, M. Waldner, E. Gröller, D. Goodsell, A. Ludovic, and I. Viola, Chameleon: dynamic color mapping for multi-scale structural biology models. In Proceedings of the Eurographics Workshop on Visual Computing for Biology and Medicine (VCBM '16). Eurographics Association, 2016.

Name and workplace of master's thesis supervisor:

Ing. Ladislav molík, Ph.D. Department of Computer Graphics and Interaction

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **16.02.2023** Deadline for master's thesis submission: _____

Assignment valid until: **22.09.2024**

Ing. Ladislav molík, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to thank my supervisor, Ing. Ladislav Čmolík, PhD., for his support and guidance during the work. I would also like to thank my father Ing. Michal Gruncl for always being there when I needed a piece of advice.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Abstract

The focus of this work is interactive visualization of scatterplots with hundreds of thousands of data points where the points are organized into hierarchy of clusters. The visualization automatically selects color palette for the clusters of selected level. The selected level of the cluster hierarchy and the color palette are dynamically adjusted by zooming the scatterplot. Furthermore, the visualization improves visibility of the displayed clusters by reducing occlusion of the overlapping clusters. The visualization technique is demonstrated using two real medical datasets containing 2D coordinates of hundreds of thousands points.

Keywords: hierarchical data, cluster, visualization, scatterplot, dynamic color palette

Supervisor: Ing. Čmolík Ladislav, PhD.
E-418,
Karlovo náměstí 13,
12000 Praha 2

Abstrakt

Cílem této práce je interaktivní vizualizace bodových diagramů s statisíci hierarchicky seskupenými datovými body. Vizualizace automaticky vybírá barevnou paletu pro shluky na vybrané úrovni hierarchie. Úroveň hierarchie i barevná paleta se automaticky přizpůsobují podle úrovně přiblížení bodového diagramu. Dále, tato vizualizace zlepšuje viditelnost zobrazených shluků snížením jejich vzájemného zákryvu. Tato vizualizační technika je demonstrována na dvou reálných datových sadách obsahujících 2D souřadnice statisíců bodů.

Klíčová slova: hierarchická data, shluk, vizualizace, bodový diagram, dynamická barevná paleta

Překlad názvu: Vizualizace hierarchicky seskupených dat pomocí bodového diagramu

Contents

1 Introduction	1
2 Design	4
2.1 Dynamic color palette	5
2.2 Color model	7
2.3 Color assignment order	8
2.4 Cluster selection	8
2.5 Point cloud visualization	9
2.5.1 Opaque rendering	11
2.5.2 Transparent rendering	12
2.5.3 Subsampling	13
2.6 Area visualization	14
2.6.1 Contouring	15
2.6.2 Relief shading	16
2.7 Label placement	18
2.8 Application control	19
3 Implementation	22
3.1 Project structure	22
3.1.1 GUI	23
3.1.2 Data	26
4 Results	33
5 User evaluation	37
6 Conclusion	42
6.1 Further developments	43
Bibliography	44
Attachments	47

Figures

2.1 Zoom to levels of hierarchy	5
2.2 Hierarchical subdivision of color	6
2.3 Comparison of HCL implementations	7
2.4 Coloring with hue and luminance	8
2.5 Color assignment order	9
2.6 Cluster highlighting	10
2.7 Hierarchy visualization	11
2.8 Order independent rendering	12
2.9 Two-pass transparency	13
2.10 Contouring of the density function	15
2.11 Absolute and normalized relief shading	17
2.12 Depth peeling	18
2.13 Point cloud versus density function	18
2.14 Too many clusters overlapping.	19
2.15 Confusing label placement	20
2.16 Application window	21
3.1 Schema of indexing	31
4.1 Discernability and density mode	34
4.2 Hierarchical coloring	35
4.3 Cluster selection	35
4.4 Contours at different heights	35
4.5 Absolute and normalized relief shading	36
4.6 Overlaps in relief mode	36
5.1 Image for the tasks 1. and 2.	37
5.2 Image for the task 6.	39

Tables

5.1 Scatterplot analysis tasks	38
5.2 Participant answers	39
5.3 Task difficulty rating	41



Chapter 1

Introduction


Data analyses of large amounts of observations provide novel insights into the observations, allowing for a better understanding of various artificial or natural processes, for example in the medical field [1] [17]. Such an analyses usually consists of hierarchical clustering of data points representing individual observations based on their mutual proximity and reduction of the dimension of the data points to 2D to allow for their visualization.

Hierarchical clustering as a method of cluster analysis that aims to build a hierarchy of similar clusters to identify informative natural clusters of observations adds additional complexity to the visualization. The visualization aims to present the properties of such datasets, that is the hierarchical structure, the density of the clusters, and their mutual overlaps as well.

The clustering algorithms can be divided into hierarchical and partitioning. Hierarchical clustering algorithms can be subdivided into agglomerative (bottom-up clustering) and divisive (top-down), which perform recursive partitioning. Since the boundaries of the clusters cannot be objectively defined, hundreds of clustering algorithms have been proposed, each with different priorities. Thus, it cannot be said which algorithms are better or worse, as the algorithm's performance is often dependent on the characteristics of the information demanded as well as on the dataset itself.

To present the data, a reduction from n -dimensional to 2D or 3D space is needed. Based on the deformation of space caused by the dimension reduction, the dimension reduction algorithms can be divided into linear, nonlinear, and those that have been implemented in both linear and nonlinear variants. PCA is a commonly used linear method that flattens the data along axes of minimal variance. NMF is a matrix factorization method, similarly to PCA, but it creates factorization with both produced matrices having only non-negative values [12]. t-SNE, a nonlinear method, aims to separate clusters in the data and avoid overlap of clusters of different categories, which may distort the data as the clusters might have overlapped in the original data. MDS has been implemented both as linear and nonlinear. This method reduces dimension while minimizing distortion of mutual distances between data points. An overview of the clustering and dimension reduction algorithms is provided by Wenskovitch et al. [19].

There are visualization techniques that conduct dimension reduction im-

plicitly. These are the star coordinates and the parallel coordinates. The star coordinates replace the axes of the high dimensional space with linearly dependent axes in the 2D space. The axes are displayed as a number of concentric line segments pointing in user-given direction, allowing the user to change the parameters of the dimension reduction. The parallel coordinates technique allows to visualize data in original n-dimensional space by representing the clusters as a polyline connecting points on a number of parallel lines, each of the lines representing a single dimension. The thickness and density of the line communicates the distribution of the cluster's points in given direction. Hierarchical structure can be also communicated by subdividing the line into a number of thinner lines, representing the subclusters .

This work deals with the design of a scatterplot visualization of a large number of n-dimensional data points organized into a hierarchy of clusters. Hierarchy is given as a tree, whose leaves contain the data points projected to 2D space. The hierarchy subdivides data points into clusters based on their mutual proximity. At the same time, some of the nodes in the hierarchy maintain the assignment of all their child nodes to a certain population, representing the category of measurements. These nodes are disjunctive and provide complete coverage, meaning that every point in the data set is associated with exactly one population. Therefore, while the nodes of the hierarchy signify spatial proximity and clustering in the data points, some of the nodes also categorize the clusters into populations. Overlaps of clusters of different populations signify interactions of the populations.

The challenges of visualizing such observations are the scale of the data, namely the high count of points, which causes heavy overdraw, and the high number of categories, which is only amplified by the hierarchical structure of clusters. The next challenge is the overlapping of points of different categories, which makes the separation of points difficult. The characteristics of the data also rule out the utilization of position as a visual channel, as the data are represented as a set of hundreds of thousands of points, whose original n-dimensional position is projected into 2D using dimension reduction techniques (e.g., PCA, NMF, t-SNE). This leaves us to separate clusters only by color since high point count limits the utilization of visual channels such as the size and shape of points as well.

The goals of the created visualization are real-time visualization rendering, allowing interactive close-up examination of visualized data, color separability of clusters and their identification in the hierarchy and finally, the identification of cluster density and their overlaps.

Throughout the work, we will analyze the techniques that tackle the same or similar issues and select the ones suitable for our purposes and design a visualization that will employ these techniques to fulfill our goals. We will go through color separability, identification of cluster density and overlaps in scatterplot visualization of a point cloud, reduction of overdraw in the scatterplot, and then evaluate the options for placing labels in the visualization. Our next focus will be a visualization of data aggregated by point density estimation. Finally, we will present the solution and document

its implementation, and then we will conduct a user evaluation, conclude results and discuss possible improvements.

Chapter 2

Design

There is number a of techniques developed for the visualization of various types of data on a scatterplot. The techniques address issues such as overdraw in the case of large datasets or visualization of high-dimensional data. The overdrawing issue can be reduced by data abstraction techniques, such as binning [9], contouring the density function (that is, converting dense parts of the clusters into areas while outliers are displayed as points), or subsampling of the data [6]. The need for discernability of a large number of categories is, however, unaddressed.

The options for visualization of high-dimensional data on a scatterplot are dimension reduction or pairwise visualization of the dimensions in a set of connected views. Each of the views displays different pair of dimensions, therefore the number of views is given by the number of pairs in a set of n elements, where the elements are individual dimensions. The number of pairs is quadratic (precisely $(n - 1)(\frac{n}{2})$), which makes this technique suitable only for a low number of dimensions. The datasets we visualize contain 20-dimensional and 26-dimensional points, which would require 190 and 325 views respectively. Therefore, the visualization of points with reduced dimension is the remaining choice.

The visualization now available is an R script that displays clusters by plotting all their points, places labels on individual clusters, and displays their hierarchical grouping as a tree, all using data visualization features of the language. The data contains the order of hundreds of thousands of points, and since the R language is designed primarily for statistical analysis and data processing, and its visualization functions are only auxiliary, it is not designed for the large volume of data we are working with, and the data takes several seconds to render, making interactive viewing by highlighting areas of interest, zooming, or panning impossible.

Also, the color palette used in the current visualization offers an insufficient amount of distinguishable shades. At least twenty distinguishable shades are needed for our application, which is due to the nature of the data, where the number of groups ranges from twenty-two to fifteen, and at the same time, each group is assigned a letter of the alphabet, which suggests that the number of groups will typically be smaller than the range of the alphabet, which is 26 characters.

2.1 Dynamic color palette

The color separability of many categories can be improved by using a dynamic color palette, that takes advantage of situations where only some categories have significant representation on the screen. Such a technique was developed by Waldin et al. [16]. The technique, named Chameleon, also provides a hierarchical subdivision of the color space, which too is desired for our purposes.

Chameleon is designed for coloring the internal structures of viruses and cells. Here, at the highest level of the hierarchy are the structures of the virus, such as the lipid envelope and the capsid, where the individual proteins can be distinguished after zooming in, as well as the domains and the atomic structure of the proteins. When gradually zooming in, only one or a few structures from higher levels of the hierarchy can fit on the screen, so there is no need to divide the color space among all of them but only among the visible ones. Thus, for each structure on a higher level, a larger part of the color space can be allocated, which can be further divided to be allocated to lower structures, as can be seen in Figure 2.2.

We based the method of coloring data points on Chameleon, but with regard to the different characteristics of the data, several modifications have been made. The first modification is given by the fact that the data are individual points, not forming distinguishable continuous objects such as proteins. Therefore, it is impossible for the color spaces of individual hierarchy nodes to overlap, thereby reducing the space available to nodes at lower levels of the hierarchy. Furthermore, the number of hierarchy levels can be greater than it is in the case of visualizing a virus or cells. Therefore, the dynamic palette method [16] is applied at all levels of the hierarchy, except for the last one, where we do not have to consider the needs of any subclusters, and thus maintaining color discernability remains the only priority. The color space is thus – in correspondence to the hierarchy – recursively divided into smaller and smaller sections (Figure 2.2).

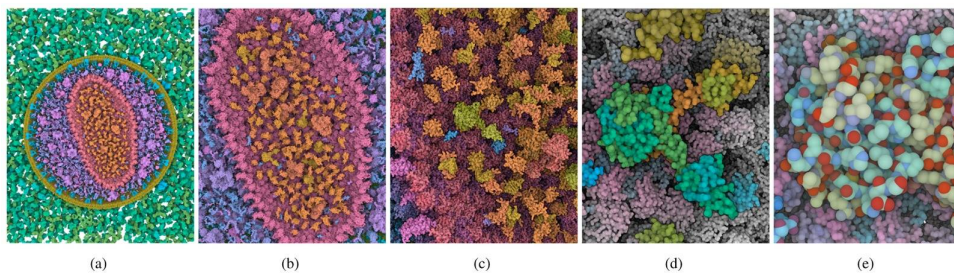


Figure 2.1: Zoom to levels of hierarchy. Image courtesy of Waldin et al. [16]

To prevent one dominant cluster from taking up all the color space, Waldin et al. [16] have imposed a limit on the maximal size of the color wedge allocated to a single cluster. This way, the maximal size of one single color wedge at the k -th level of the hierarchy is a^k , where a , ranging from 0 to

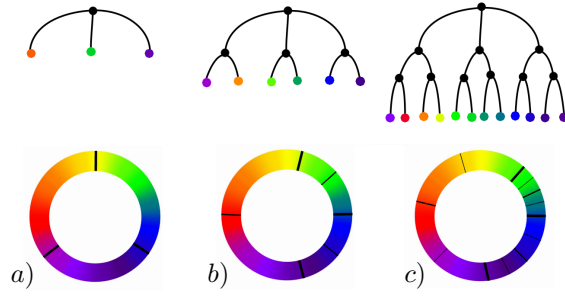


Figure 2.2: Hierarchical subdivision of color when zooming in on certain cluster. a) Color wedges are allocated only for the top level of hierarchy. b), c) With increased zoom, lower levels of hierarchy are visualized with distinct colors. The size of the color wedges corresponds to the hypothetical relative representation of individual clusters and subclusters on the screen.

1, is the size limit. To maintain the distinguishability of colors at lower levels of the hierarchy, the size limit a was set to a value higher than recommended by Waldin et al., namely 0.75 instead of 0.5. Also, because the number of subclusters may vary significantly across clusters, the size limit may be too high for some clusters with only a few subclusters. Thus, we have imposed an additional size limit at the top level of the hierarchy. The limit is the number of descendants n_i of a i -th cluster at the top level of the hierarchy multiplied by an appropriately chosen constant of 0.08. In conclusion, the maximal size for a cluster at k -th level of the hierarchy being a descendant of i -th top-level cluster is

$$\min(0.08n_i, 0.75) \cdot 0.75^{k-1}. \quad (2.1)$$

The original method sets the size of the color wedge allocated to individual clusters proportionally to the number of subclusters visible on the screen. In our case, even at a high level of zoom, most subclusters are present on the screen due to the wide scattering of the clusters. This results in almost no color space being released when zooming. Therefore, we have decided to size the color wedge proportionally to the number of points in the given cluster located on the screen. Allocating by the number of points is very aggressive, which achieves a good distinction of points according to their belonging in the hierarchy. On the other hand, in certain situations, the assigned colors may change significantly when the view changes, possibly confusing the user. The limits imposed on the maximal size of color wedges reduced this artifact. Other applicable methods are normalizing the number of points in individual clusters, weighting by the square root of the number of plotted points, or blending uniform and weighted color allocation.

The position and mutual distances of color wedges are balanced by a force method – every wedge is attracted to its initial position and at the same time repelled from other wedges. The attracting force ensures that the wedges will not stray away from their initial position and the clusters will maintain their original color. The repelling force is given by size and distance of the wedges, that is, their overlap. The repelling force is linearly dependent on the size of

the overlap and it only drops to zero after the distance between borders of neighboring wedges exceeds some minimal margin. The margin reduces the possibility of overlap or too small distances between the wedges in a balanced state of the forces. In conclusion, the resulting force for i -th cluster is

$$f_i = sigdist(g_i, I_i) + \sum_{i=1}^{n, i \neq j} dir(g_i, g_j) max(r_i + r_j - dist(g_i, g_j), 0), \quad (2.2)$$

where g is the current position of a wedge, I is the initial position a r is the radius of a wedge plus margin. The force attracting the color wedge to its initial position is weighted by a factor of 0.1 against the force repelling the color ranges from each other. This reduces the overlap in the case of several large wedges located next to each other in the color space. To prevent oscillation and jitter, a damping factor of 0.25 is then applied to both forces.

2.2 Color model

HCL is a family of color models having channels hue, chroma, and luminance. HCL models are isoluminant and perceptually uniform and thus often used in computer visualizations [16] [6] [13]. We have used the implementation of the HCL color model developed by Zeileis et al. [20], which contains an HCL-to-RGB conversion chain. Part of the chain is XYZ-to-RGB conversion, where we have used Catalano’s implementation [5] instead since in our experience it better satisfies the property of isoluminance (see Figure 2.3). The HCL color model is implemented as CIELAB with axes A^* and B^* transformed to polar coordinates. The hue channel is used to differentiate the clusters, as it is the only one that does not visually imply sorting.

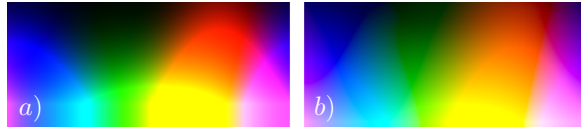


Figure 2.3: a) HCL implementation by Zeileis et al. [20]. HCL implementation from the Catalano framework [5] satisfies the isoluminant property better, notably in the dark green part of the spectra.

But, since the first level of the hierarchy of visualized data contains over twenty nodes, comparing to the data, for which the dynamic color palette developed by Waldin et al. [16] was designed, where the first level of the hierarchy only contains up to ten nodes, the hue channel alone proved to be insufficient to reliably distinguish clusters of data points (see Figure 2.4). This is also due to the used model CIELAB not being perfectly hue uniform [11] [14] and in the area of the blue hues, the distance between distinguishable hues is greater than it is for the other color hues.

Therefore, two channels, hue, and luminance, are used in the coloring of the first level of the hierarchy. The chroma channel is not utilized for its interference with hue. Hue is used as the primary channel, where each group

uses a different hue. Luminance is used as a complementary channel, although it provides more levels of resolution than hue, to ensure that coloration does not visually imply alignment and to avoid blending in with the background. Because the technique of color optimization designed by Waldin et al. [16] is only designed for optimizing the color palette in one dimension, a low number (five) of fixed luminance levels are used, assigned to successive nodes in the cluster hierarchy. The order of the levels is permuted so that adjacent sections in the hue channel are not assigned adjacent luminance levels, thereby increasing the tonal distance of the colors assigned to successive clusters. All nodes in the hierarchy are colored using the same luminance level as their parent. The use of two channels improved the separability of clusters, as can be seen in Figure 2.4.

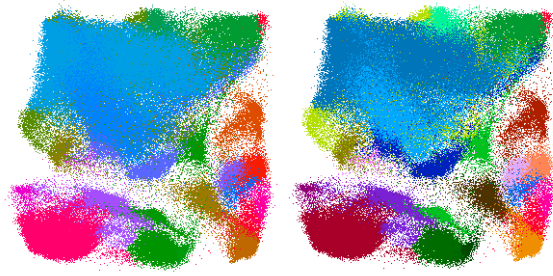


Figure 2.4: a) coloring using hue only. b) coloring using hue and luminance.

2.3 Color assignment order

Looking at Figure 2.5, we notice that some very close or overlapping clusters of points are colored in similar colors, thus merging and creating the impression of a single group, for example, the two clusters a) and b), labeled in the figure, colored in a similar shade of green. At the same time, an easy distinction between those two clusters could be achieved by permuting the order of colors so that the co-located clusters are colored by colors that are distant in the color space, as proposed by Lu et al. [13]. But this may create a problem of two distant clusters dyed with a similar color creating the impression of one disjointed cluster, as shown in Figure 2.5 by clusters c) and d). Similar shades of color are easier to recognize if placed close to each other. Thus, we decided not to permute the color spaces.

2.4 Cluster selection

Waldin et al. mention improving the discernability of user-selected area of interest by desaturation of the surroundings. In our case, the cluster selection aims to give a full overview of clusters of the hierarchy. However, since the mere desaturation of the surroundings turned out to be an insufficiently discriminating channel (see Figure 2.6b), the luminance channel was additionally

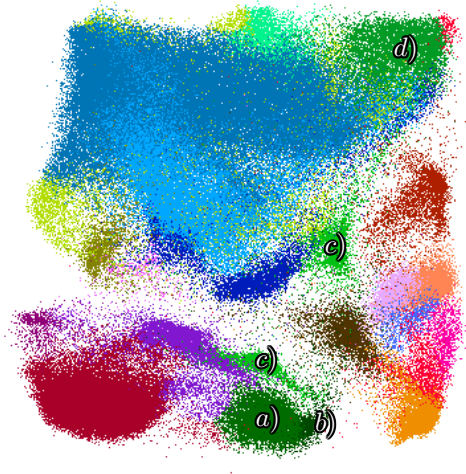


Figure 2.5: a),b) Co-located groups colored with a similar shade. c) Disjointed cluster. c) and d) might create the impression of a single disjointed cluster.

used to brighten the surroundings. Desaturation needs to be used together with brightening, as brightening alone makes colors appear more saturated. Saturated background is undesirable, as it draws users' attention away from the selected context. Desaturation and brightening are already sufficiently distinctive, but there remains the problem of covering the highlighted cluster with a denser cluster (2.6c), which can be solved by moving the highlighted cluster to the foreground, but this results in the suppression of the context, and we lose information about the background of the marked cluster (2.6d). The user is given the option to switch between these two methods since it cannot be conclusively said which one is better. Comparing to 2.6a, we can now see that the cluster selection gives us an overview of a cluster that would otherwise remain unknown.

The context can be selected by clicking on the cluster of interest, or by clicking on a node in the hierarchy tree (Figure 2.7), that was added to the visualization for this purpose. While clicking directly on a cluster selects a leaf cluster in the hierarchy, clicking into the tree allows selecting a node at any level of the hierarchy. Multi-select is also possible, both in the cluster and the tree visualizations. The tree visualization also communicates the number of points in individual clusters by the size of the tree nodes, where the node edge length is proportionate to the number of points in the corresponding cluster. The nodes of the top-level clusters are also labeled with the population of the corresponding clusters. The nodes are also tinted by cluster color.

2.5 Point cloud visualization

The point cloud visualization is based on drawing individual data points. This visualization is augmented with methods aimed at reducing overplotting, improving the identification of clusters, and improving the perception of

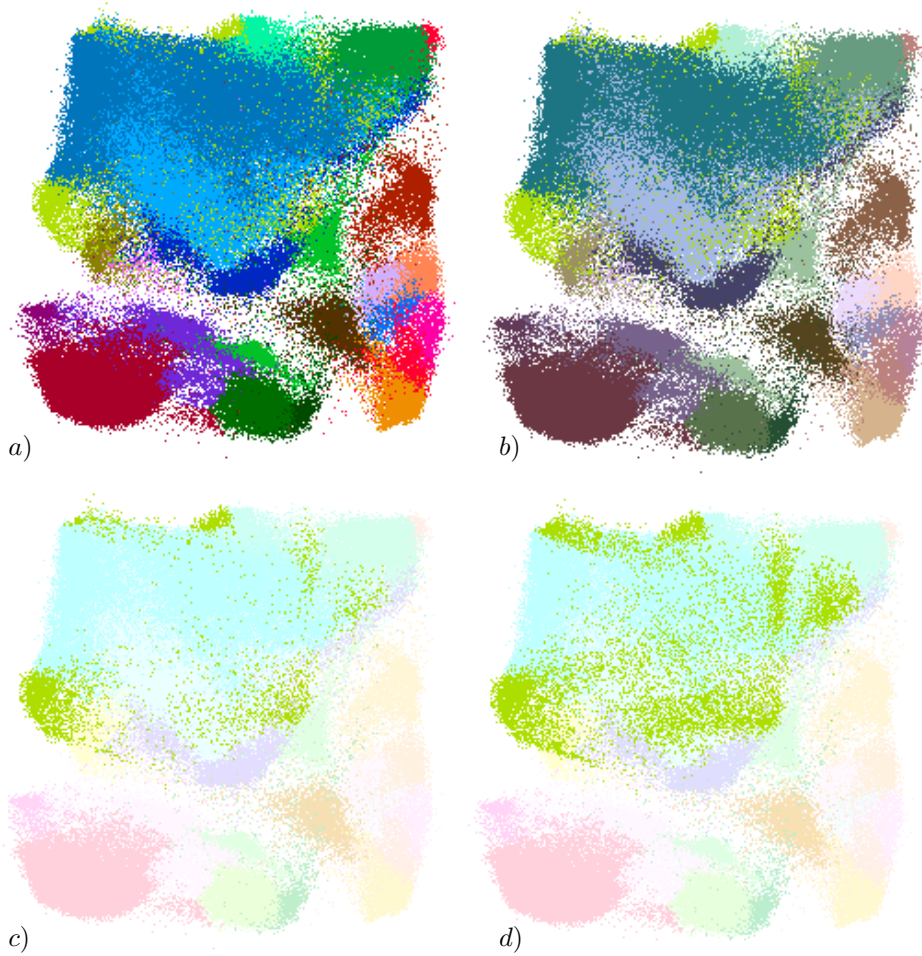


Figure 2.6: A comparison of context highlighting methods: a) No context highlighting. b) Highlighting with chroma. c), d) Highlighting with chroma and luminance. d) Highlighted cluster is brought to front, thus not overlapped by any cluster.

density and overlaps.

Perception of density can be improved by rendering the data points transparent, but, as explained in [2.1](#), the luminance channel is also used for coloring the clusters, and this excludes the rendering of points using transparency. This is because in the case of a white¹ (achromatic in general) background, luminance interferes with transparency, that is, the difference in the brightness of the colors of individual clusters is not discernible; darker clusters only look sparser. Also, when transparency is used, colors will be mixed, which will worsen the distinctness of individual clusters, and new colors may even appear.

Opaque rendering, that is, without transparency, does not suffer from this

¹Visualization uses black background by default, as it makes clusters stand out better, white background mode was added for print.

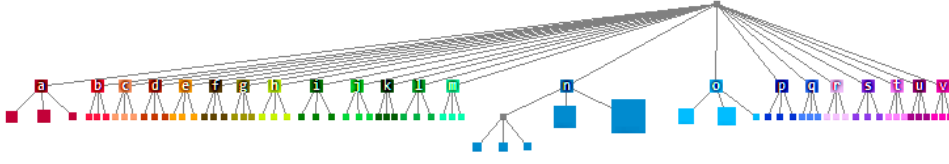


Figure 2.7: Hierarchy visualization

issue, but two other problems arise: the dependence of the visualization results on the rendering order (Figure 2.8a, b), which distorts the information about clusters overlap, and the loss of the data density information.

Although some of these problems can be solved, transparent rendering overallly allows for a better perception of density, and opaque rendering for better identification of clusters. Therefore, both approaches were implemented. The visualization is also provided with the option of subsampling the data [6].

2.5.1 Opaque rendering

The rendering order issue was resolved by turning on the depth test and assigning a random depth to each point. In this way, the overlap of the groups depends on their density; a denser group has a statistically greater chance that out of the number of points falling within one pixel on the screen, its point will have the smallest depth of all. This technique also achieves proper visualization of cluster overlaps without color mixing. Figure 2.8 compares the differences on two overlapping clusters, the blue one linearly falling off in the down direction and the brown one linearly falling off in the left direction.

We will now demonstrate that the probability of a cluster being on top in a given pixel corresponds to its relative representation at the pixel. The CDF function of the depth of cluster X contributing to pixel p with l samples is a minimum of l samples of an uniform random variable ranging from 0 to 1. Probability of one such sample being lower than $x \in \langle 0; 1 \rangle$ is x . Therefore, probability of l samples being lower than x is x^l , which is the CDF function of cluster depth at the pixel. Thus, when cluster X is contributing to a pixel with l samples, and all other clusters (Y) combined contribute with k samples, probability of X being on the top is calculated as the difference between two random variables with CDF functions

$$F_X(x) = x^l, \quad (2.3)$$

$$F_Y(y) = y^k. \quad (2.4)$$

Sum of two random variable is calculated as a convolution. The formula for sum can be modified to difference as such:

$$P(X + Y \leq z) = \int_{-\infty}^{\infty} f_X(x)F_Y(z - x)dx, \quad (2.5)$$

$$P(X - Y \leq z) = \int_{-\infty}^{\infty} F_X^l(x)F_Y(z + x)dx, \quad (2.6)$$

although in case of difference, the formula is not commutative anymore. The formula can be further modified to

$$P(X \leq Y) = \int_0^1 lx^{l-1}x^k dx. \quad (2.7)$$

The result of the integral is

$$P(X \leq Y) = \frac{l}{l+k}. \quad (2.8)$$

As we can see now, when $k+l$ is normalized to 100, then l is the percentage share of points of cluster X at pixel p and at the same time the percentage probability of X being on the top in the pixel.

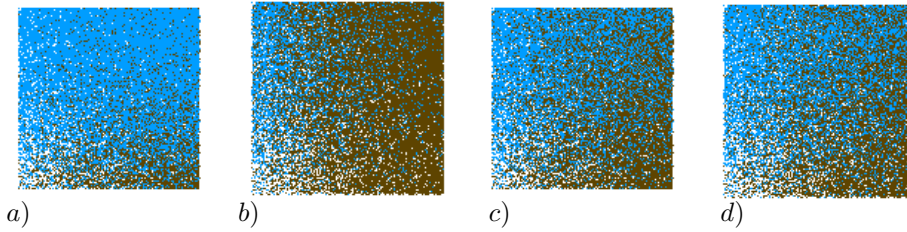


Figure 2.8: a),c) Blue over brown. b),d) Brown over blue. a),b) No depth test. c),d) Depth test with randomized point depth.

2.5.2 Transparent rendering

As for visualizing the data density, the transparency settings of the plotted points have been made available to the user so that s/he can view the density if necessary, even though this decreases the cluster discernability. At the same time, it is also necessary to disable the depth test so that no points are neglected, and to disable the use of the luminance channel, as it would distort the perceptual transparency.

When visualizing data density using transparency, we find that for high transparency settings, very sparse areas where there are only a few points, surrounded by a black background, are hard to see, and at low transparency settings, the difference in density between the rims and the centers of the clusters cannot be discerned. We cannot increase transparency, as due to alpha blending, clusters drawn last would cover earlier drawn clusters, as can be seen in Figure 2.9c, top right, where the blue cluster incorrectly covers the brown cluster. Therefore, a non-linear dependence between transparency and density is needed. We implemented this as rendering with transparency in two stages.

At first, all points are rendered with a transparency lower than the user set, and on each pixel of the screen, only the first point that fell into the pixel is rendered. Then all the data are re-rendered over the previously drawn data using the transparency level set by the user. In both passes, the points are

drawn in the same order, so that the distortion given by color mixing is the same in both passes.

The size of the points increases with the zoom level to maintain color discernability at a high level of zoom, where the clusters become sparser.

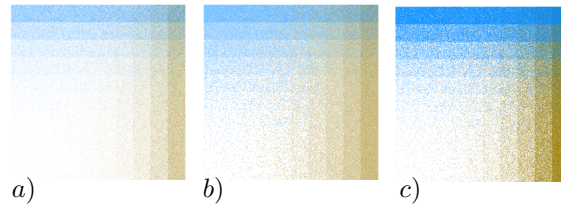


Figure 2.9: a) Single-pass rendering. b) Two-pass rendering. c) Single-pass with lower transparency causes order-dependent result. Unlike in Figure 2.8 an exponential falloff is used here.

2.5.3 Subsampling

To reduce overdraw of the scatterplot, Heimerl et al. [9] designed a technique of binning data points into a honeycomb, with each cell containing a bar graph or pie chart representing the distribution of individual categories in the cells. The background of the cell can also be saturated to communicate the density of the data points. However, the often complex structure of clusters in our data and rapid changes in cluster density would require a high number of cells to keep the visualization representative. Breaking continuous areas of single color into a number of small icons, coupled with the much higher number of categories than Heimerl et al. planned for, would lead to visual clutter and decrease the discernability of categories significantly.

Chen et al. [6] in their article demonstrate reducing the number of points by sampling a density function in the space of original data, reducing overdraw while preserving density information and relative representation of data categories compared to original data. Reducing the number of points will also allow drawing points larger, which aids with recognizing colors.

Chen et al. have utilized the multi-class blue noise sampling technique designed by Wei [18]. The blue noise sampling provides an alias-free spatial distribution of the samples of individual classes while preserving densities, that is, the spatial distribution of the class points in the original data. Blue noise sampling is simply implemented as choosing each of the random samples from the most underfilled class. This also ensures consistent fill rates of the classes and prevents the creation of new points, that were not present in the original data, which would happen with sampling in a continuous domain. The fill rate of a class is calculated as the number of samples of its total samples over the number of accepted samples. During the sampling, a chosen sample can be only accepted if there are no other already accepted samples within a given radius r . However, Wei has shown, that to maintain alias-free distribution, removing existing samples is necessary. Samples in conflict with incoming sample can be removed, when:

- It is impossible to add sample to the class c_I of the incoming sample,
- Every conflicting sample belongs to class with lower r than c_I ,
- Every conflicting sample belongs to class at least as filled as c_I .

If all samples conflicting with the incoming sample can be removed, they are removed and the incoming sample is accepted. In our case, the second condition does not have an effect, as we use equal r for all classes. The possibility to add a sample is checked approximately by simple timeout mechanism – if adding a sample fails t times, where t is the timeout, adding the sample is considered impossible.

Chen et al. have extended this technique with hierarchical subsampling. The subsampling is precomputed for several consecutive zoom levels, each being double of the previous zoom level. At first, subsampling is computed for the first, most coarse, level with some initial radius r . The r is halved for the next level of zoom and the samples accepted in the previous level are used as initial state. Removing samples accepted in previous levels is not allowed, as this would cause some points to disappear when zooming in.

The number of casted samples n is, however, not specified. We estimated the number of samples in the first level as

$$n_0 = \frac{S}{r_0^2}, \quad (2.9)$$

where S is the area of the bounding box of the dataset points and r_0 is the minimal distance between the data points at the first level.

The number of samples for the next levels, n_i , is calculated as

$$n_i = \left(\frac{r_{i-1}}{r_i}\right)^2 m_{i-1}, \quad (2.10)$$

where m_i is number of samples accepted at the i -th level. The value of m is usually much smaller than n , leading to improvement of the algorithm performance. Since the radius is halved after each zoom level, this formula can be simplified to

$$n_i = 4m_{i-1}. \quad (2.11)$$

■ 2.6 Area visualization

Chen et al. [6] also provide visualization of aggregated data by thresholding the density function of the data. This is a technique where the density function is compared to a certain threshold and areas of values larger than the threshold are used to represent the associated cluster. It is a common approach when handling large datasets. The areas are drawn semitransparent, which allows seeing overlaps of clusters, but overlapping of clusters will cause color mixing, which will reduce the discernability of colors, on the other hand, continuous clearly bounded areas will provide both better color and cluster

discernability than point clusters. Points in the below-threshold areas are subsampled and drawn as points rather than to be aggregated into an area.

The density function can be also visualized using contouring, where only the boundary would be drawn and the inner areas would be fully transparent. Relief shading of the areas can be used to give the impression of the density of the clusters. Altogether, the performance of such visualization might still be comparable to or even better than the visualization of individual points.

The density function is created using kernel density estimation (KDE). The value of the density function f at position x is the sum of distances to every point in the dataset, weighted by the kernel function:

$$f = \sum_{i=1}^{i < n} K(x - x_i). \quad (2.12)$$

This is implemented as drawing every point with a Gauss kernel. There are several methods of visualizing the density function, such as direct visualization (where density is mapped to transparency or luminance), visualization by contouring or by relief shading. The relief shading and contouring are techniques that are used for communicating terrain relief in topographical maps.

2.6.1 Contouring

Contour is a closed loop line produced by cutting the heightmap surface by a horizontal plane. Usually, contouring is done by placing a set of contours equidistant in the height dimension. In our case, we contour clusters separately with contour lines colored correspondingly to the associated cluster. This means that there will be many contour lines per one level of density contoured and that the contour lines will cross each other, creating visual clutter and overplotting, which would be strongly amplified by multiplying the total number of contour lines several times. Thus, we have decided to contour only one density level at a time. The contoured density value can be user-edited.



Figure 2.10: Contouring of the density function

2.6.2 Relief shading

The relief shading assumes interpreting density function as a 2D heightmap by mapping the density to height. Having one heightmap per each cluster, the clusters form a set of overlapping 2D surfaces in 3D space. This technique simulates the lighting of the heightmap surface by a directional light source, giving an impression of the surface shape. This allows us to communicate both the height and the slope of the surface. The lighting calculation requires light direction and surface normal direction. The light direction is chosen to be vertical down, as any other direction would distort the perception of the slope based on the direction of the slope face. The surface normal is estimated by a cross-product of surface tangents. The surface tangents are a pair of vectors t_x, t_y , whose direction in the horizontal plane is given by two-dimensional canonical vectors e_x and e_y . At position p , the tangents are constructed as

$$t_x = \begin{pmatrix} e_{x,x} & s \cdot (f(p + e_x) - f(p)) & e_{x,y} \end{pmatrix}^T, \quad (2.13)$$

$$t_y = \begin{pmatrix} e_{y,x} & s \cdot (f(p + e_y) - f(p)) & e_{y,y} \end{pmatrix}^T, \quad (2.14)$$

where f is the density function and s is the vertical scaling factor. This parameter regulates the steepness of the slopes.

With this construction of tangents, slope steepness depends on the absolute difference of densities, not the relative one, which makes slopes in low-density areas unnoticeable. This can be tackled, although at the expense of perception of absolute density, by normalizing the samples taken at each fragment by dividing them by $f(x) + \epsilon$, where ϵ prevents division by zero. The approaches are compared in Figure 2.11. The normalized relief shading (2.11a) gives more detailed information about shape, but the distribution of density inside a cluster appears to be more even than with absolute (not normalized) relief shading (2.11b). The knowledge of the density distribution in a cluster allows us, for example, to distinguish between the dense center of the cluster and an area of outliers.

Relief shading together with transparency allows for visualizing density and overlaps of reliefs of different clusters at the same time. The height of the reliefs must be considered when using transparent rendering because the composition of transparency is order dependent and therefore we need to draw reliefs sorted by height. Since the height of the reliefs varies per fragment the sorting also needs to be done per fragment. This is achieved with the technique of depth peeling (also known as order-independent transparency), which was designed specifically for the correct composition of transparency [7]. With this technique, geometry is sorted by depth into several layers, as depicted in Figure 2.12. Usually, four layers give enough precise results, even if there are areas on the screen with more geometry overlapping. The first layer contains fragments closest to the camera, the second layer fragments second closest to the camera, etc. Two depth buffers are needed to achieve this, the first one to discard fragments closer to the camera than fragments rendered to the previous layer (black thick lines in Figure 2.12), the second

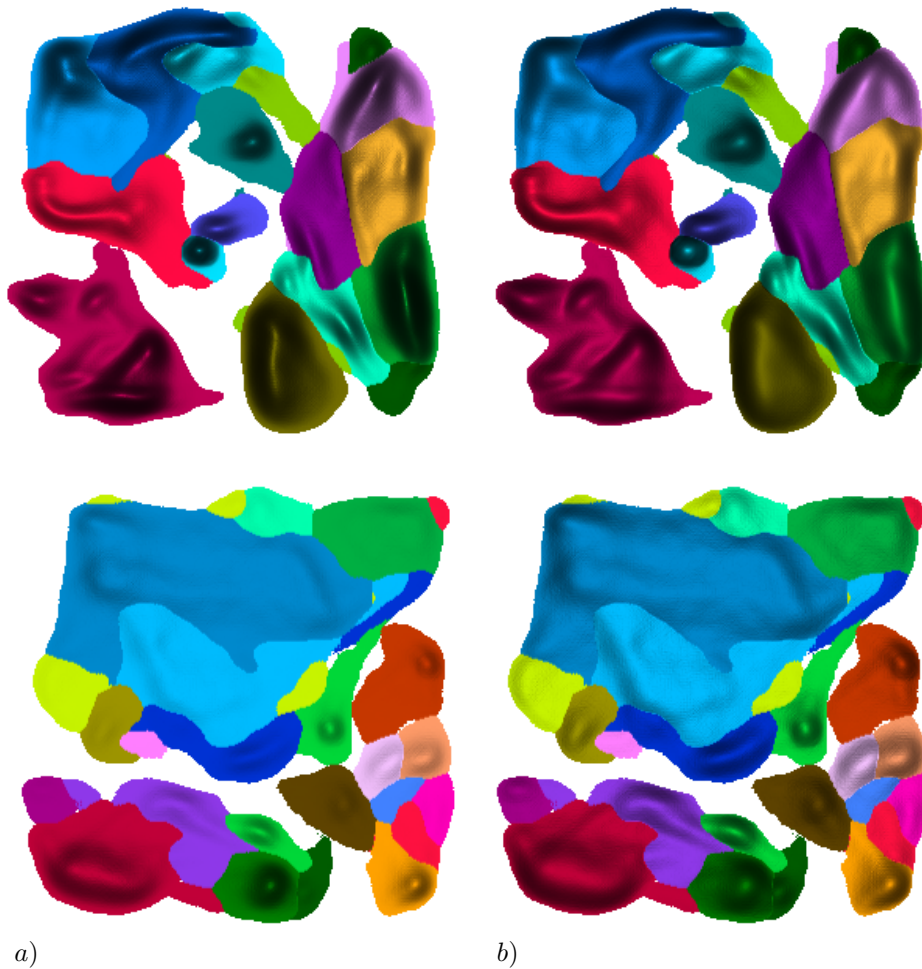


Figure 2.11: a) Absolute relief shading. b) Normalized (relative) relief shading.

one to discard fragments further from the camera than fragments rendered to the current layer (gray thick lines in figure 2.12). Since OpenGL only supports one depth buffer per framebuffer, one of the depth buffers must be implemented in the fragment shader. The fragment shader will sample a depth texture and perform the depth test on the sample and the fragment depth. Texture bound to a shader cannot be written into, because sampling texture that is concurrently written into would produce undefined results due to caching and thread races. This means that the depth buffer sampled from the fragment shader cannot change during the rendering of a layer. Luckily, the depth buffer that contains the depth of the previous layer does not change during the rendering of the current layer, therefore this is the one that has to be sampled from the fragment shader.

As can be seen in Figure 2.13, the overlaps of clusters are now visible much better than with point cloud visualization. We can see too, that the discernability of clusters is maintained, as long as there are not too many clusters overlapping each other. Practically, such visualization is clear for two clusters overlapping each other. For three mutually overlapping clusters

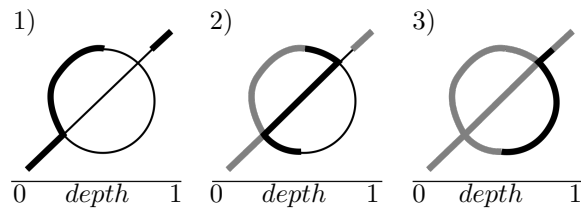


Figure 2.12: Depth peeling. Adapted from "Order independent rendering" [7].

the clarity varies per situation, for more than three, the visualization is not clear anymore, as shown in Figure 2.14

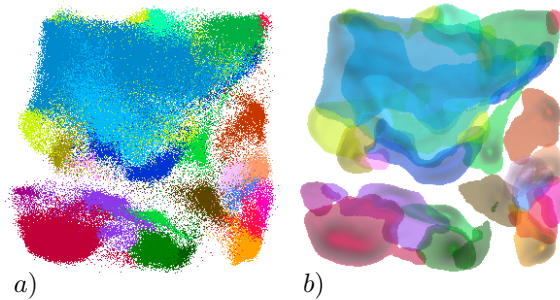


Figure 2.13: a) Point cloud visualization. b) Density function with transparency.

2.7 Label placement

There are two basic approaches to label placement – external and internal labeling. In internal labeling, the label is placed over its target, while in external labeling, the labels are placed outside the visualized object and are connected to the target by a leader line. Placing labels outside clusters makes finding the label of a cluster more difficult, but placing the label over the cluster may lead to obstruction of the area of interest or even to ambiguous labeling in case of cluster overlap. This can be noted in the picture from the original application 2.15, where label h , marked red, is located at the area of dominant occurrence of population n ; a more suitable placement of the label is marked by the green cross. Obstruction of the area of interest is not a problem in the case of labeling just by letter of the alphabet instead of labeling by the whole name of the population, which can also be seen in the figure.

It is also possible to combine these two approaches and use internal labeling for large, continuous, and non-overlapped cluster, while external labeling is used for small, disjoint, or overlapped clusters. Using multiple leader lines for a single label may be needed in case of disjointed clusters. The visual match between a label and a target cluster can be improved by highlighting the label by the color of the cluster.

Čmolík and Bittner [21] have developed a library for external label placement in semi-transparent objects. Input for the computation of label place-

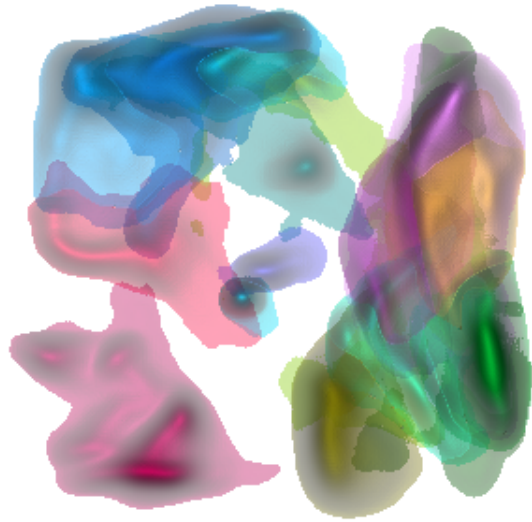


Figure 2.14: Too many clusters overlapping.

ment is 2D texture, where each pixel represents a bitmask. The value of each bit of the bitmask determines the presence of a corresponding cluster on the pixel. Based on four properties, the optimal placement of label anchors is evaluated from the texture. The properties are the opacity of the target, the opacity of objects overlapping the target, the number of objects overlapping the target, and the distance from the edge of the target. This would prevent an ambiguous placement of labels (leader lines in external labeling), which does occur in the original application. Evaluation of these properties may be influenced by methods of creation of the bitmask texture, such as changing point size, subsampling of the data, and use of the depth test. The temporal coherency of the labeling can be improved by anchoring the position of the labels and penalizing changes in label positions between frames [10]

■ 2.8 Application control

The application window is split into visualization view, and visualization configuration. The visualization view contains hierarchy visualization, point cloud visualization and area visualization. The views of point cloud and area visualizations are connected, meaning that zooming or panning in either of these two visualization is mirrored to the other one. The mouse cursor is also mirrored to the other visualization. This helps the user to exactly match positions between those two views. Cluster selection can be done in the point cloud and the hierarchy visualizations. In case the user gets lost after accidentally moving the view out of the scope of the data, the view position can be reset to the initial value in the view item in the menu bar.

The configurations allow the user to switch modes of visualization and configure them as needed. Some of the configurations are not intended to

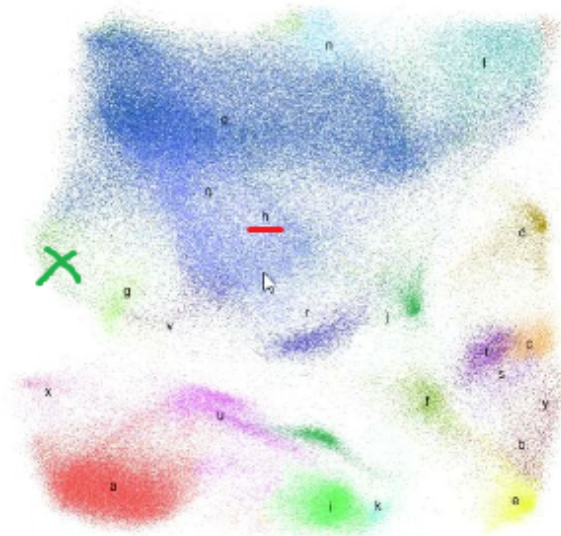


Figure 2.15: Confusing placement of label *h* (marked red); a more suitable placement is marked by the green cross.

provide another useful views on the data, but to compare successful and unsuccessful approaches. The configuration panel is divided into three sections. First one switches dark and light mode of the visualization. The visualization is designed for the use of the dark mode; light mode was only added for creation of printable figures. The distinction of colors and shapes is actually worse with the light mode.

The next section contains the configurations of the point cloud visualization. The value of the `point size` slider is used to multiply the size of the drawn points. The `opacity` slider sets the opacity of the points. When opacity is lower than one, the visualization enters the density mode, which means that the depth test and the use of the luminance channel are deactivated. The next four input fields set the zoom level at which coloring at a given level of hierarchy is activated. The `subsampling` checkbox activates the subsampling of the data points.

The remaining controls of this section allow us to view less successful approaches and therefore are not relevant to the user. The last section contains controls for the area visualization. The `contour mode` checkbox switches between contouring and relief shading of the density function. The `contour height` slider sets the density level where the density function is contoured. The `normalize relief` checkbox activates the normalization of the density in the relief shading. The `min density` and `max density` sliders crop the rendered relief by density by discarding fragments outside the density range specified by the sliders. The `opacity` slider sets the opacity level of the rendered reliefs. The minimal density setting allows increasing clarity of the visualization by cropping out areas of low significance. The maximal density setting is intended for the use case where opacity is set to one and then decreasing the maximal density will crop out the tops of the dense clusters,

allowing us to see beneath them.

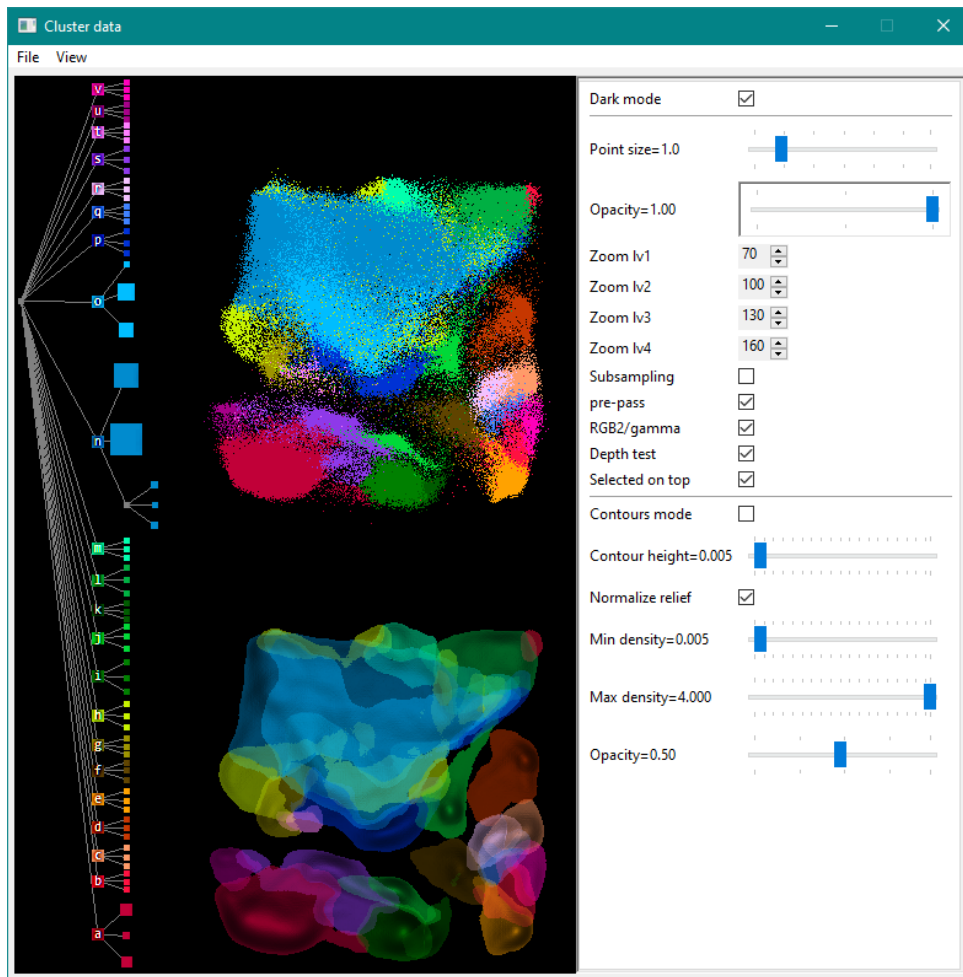


Figure 2.16: Application window

Chapter 3

Implementation

The application is written in the Java language in consideration of the possible use of the library for label placement [21], which is written in Java too. The JOGL library, which is Java binding for OpenGL graphical interface, was used to achieve real-time visualization of the datasets. Since Java bytecode is significantly slower than native code, the application was initially designed for utilization of the GraalVM JVM, which allows for the compilation of Java bytecode to native code. This led to the decision to use the SWT graphical toolkit because AWT and Swing are not stable when compiled on GraalVM. However, the performance of the visualization was revealed as sufficient after the implementation of the first prototype, reaching 60 FPS and the limiting factor being the GPU rather than the CPU. Therefore, the use of GraalVM was abandoned.

Since the data provided did not contain the hierarchical subdivision of the data points (only subdivision on the first level of the hierarchy was available, that is a linear list of data points and their population), it was necessary to create the hierarchy artificially for the testing purposes. The K-means++ algorithm was used to subdivide the data points into a three-level hierarchy. K-means++ is the heuristic approximation of hierarchical clustering and it is often used in statistical analysis of data.

3.1 Project structure

The project code is contained into three packages, `gui`, `data`, and `hclust`, except for the class `Main`, which is in root folder. The `hclust` package is contained in the `data` package. The project's resource contains folder `gui` with textures and GLSL shader source files. `hclust` contains the algorithm for hierarchical clustering, implemented by Behnke [4], but it is not used currently. Instead, the Kmeans++ algorithm is used, implemented in class `KMeans` in package `data`. This implementation is developed by Altschuler [2].

The entry point of the application is the `Main` class, which creates SWT application window, loads the default dataset and runs the SWT main loop.

■ 3.1.1 GUI

Package `gui` contains classes creating GUI, such as `VisualizationView` class, which creates a GUI panel that can be drawn on using OpenGL, and the `VisualizationConfig` class which creates a panel with controls for configuring the visualization parameters. Then there are helper classes for the visualization. These are the `HCL` class, providing conversion between HCL (polar LAB) and RGB, the `Chameleon` class for the dynamic color palette, and finally, the `Shader` class for loading and compiling shaders. Items, that are not the source code of the application, such as shader sources and textures are stored as resources.

■ Shader

This class loads the shader sources and compiles them in its `loadShaders` procedure. The shaders are represented as instances of this class and assigned to its static variables, which can be accessed from other classes through getters. There are seven shaders. Shaders `verts`, `tree`, `mask`, and `dens` draw data points transformed by the view matrix. The `tree` is used for rendering the hierarchy tree and the second mouse cursor. Shaders `tex`, `merge`, `gauss`, and `peel` are used for compositing and postprocessing of rendered textures. The geometry used by these shaders is a quad spanning over the whole viewport. Some of the shaders are multipurpose, that is they can be configured to different modes in order to perform different tasks. This is because switching shaders is expensive and fewer shaders means less switching per one frame. The multipurpose shaders are the `tex` and `tree` shaders.

verts. This shader draws the data points. When subsampling is used, this shader will discard fragments belonging to a higher LOD level than the visualization zoom level. Color of the fragment is given by vertex attribute `id` and the `color` shader storage buffer, which is indexed by the `id`.

tree. Used to draw the hierarchy tree visualization and the second mouse cursor in the connected view. The tree visualization consists of edges, labeled points, and unlabeled points. This shader needs to be configured separately to render each of these meshes.

This shader can use color from the shader storage buffer at the index given by vertex attribute `id`, just like the `vert` shader, or a gray color. The coloring method is switched by an uniform boolean parameter. The shader can also add texture color to the color or crop the rendered point to a rectangular shape by discarding fragments by their uv coordinate. These two features are also activated and deactivated by an uniform boolean parameter. The texturing coordinate can be transformed by an uniform vector variable. This allows using a single texture for every label. The texture contains all the letters in the alphabet and the uv coordinates are transformed so that just a single letter would appear on the label. The texture color is scaled to range $\langle -1, 1 \rangle$. This allows to use the texture to both subtract and add color.

mask. Renders the bitmask texture of cluster distribution from the data points. The bitmap is intended to be the input for label placement. The output of the fragment shader is a 128bit long mask, represented as a vector of four unsigned integer values. The vector is passed to the shader as an uniform variable. The bitmask texture is composed using logical or between fragment shader output and value accumulated in the texture. The operation is set by `glLogicOp(GL_OR)`.

dens. creates the density field of a cluster. Data points of the cluster are rendered using a Gauss kernel. This process is equivalent to convolution with Gauss kernel or smoothing. Density is composed using alpha blending, with the weights of both incoming fragment and accumulated value set to one by `glBlendFunc(GL_ONE, GL_ONE)`.

gauss. This shader can be used for creating the density texture of a cluster with the utilization of the linear separability of the Gauss kernel. This would reduce the computational complexity of the convolution from $\theta(nh^2)$ to $\theta(nh)$, where n is the number of pixels of the density texture and h is the size of the kernel. This shader is currently not used.

peel. Implements the depth peeling. The depth peeling requires two depth buffers, the second one is implemented in the fragment shader. The inputs of this shader are a density texture, depth texture, and uniform color vector. The fragment color is given by the uniform color. The fragment depth is negatively proportionate to density. Fragments with lower depth (higher density) than the one in the depth texture are discarded.

tex. This shader is used to create the final image for the visualization of the clusters using the density function. The shader has two modes. The first mode renders the contour of the density function. The density function is texture representing the density function of a single top-level cluster. The color of the contour is set by an uniform vector variable.

The second mode creates a semi-transparent relief visualization of the density function. The inputs are layer textures created by the depth peeling procedure. Each layer contains density and color. The density and color are used to render the relief. The normal vector of the relief is estimated from the density. The transparency level is set by an uniform variable.

merge. This shader is used for merging textures of cluster densities into a single texture using a virtual framebuffer. The depth and opacity of the output fragments are given by the input density texture. This shader is currently unused.

■ VisualizationView

This class is used to create a GUI panel for displaying the visualization and reading user input. This class extends the `swt.GLCanvas` class. The constructor of the `VisualizationView` class initializes the global state of

OpenGL for the visualization, loads and compiles shaders using the method `loadShaders` of the `Shader` class. The `VisualizationView` draws a given array of objects implementing an interface defined by the static abstract class `Drawable`, defined in the `VisualizationView`. This way, the implementation of `VisualizationView` is independent on the visualization itself. The class defines the methods `boolean needsRedraw(float view, float zoom)`, `void draw(float[] view, float zoom, GL4 gl)` and `public abstract void click(int[] pos, GL4 gl)`.

The array `view` represents the two-by-two row-major transformation matrix used for rendering data. The `zoom` parameter gives the zooming level, which is for setting the level of detail for subsampling, setting point size, and for determining which level of hierarchy should be visualized, that is at what level should clusters be colored with different colors. Both `view` and `zoom` are actualized by the mouse input. The return value of `needsRedraw` tells if there are any changes in the object, that would require redraw.

In case of change in any of the objects drawn on the canvas, all objects are redrawn, because the objects, in general, may overlap and thus redrawing of one object might interfere with objects drawn in the previous frame. The `click` method passes information about click on the canvas to the object.

■ VisualizationConfig

This class creates a GUI panel that exposes the visualization parameters to the user. The user can read and adjust the parameters with this panel. The implementations of `VisualizationView` and `VisualizationConfig` do not depend on each other in any way.

■ Chameleon

This class implements the dynamic color palette [16]. The constructor parameters are the number of color wedges and the maximal iteration count of the force model computation per one update. The force model update is implemented in the `recalculate` method. The `setWeights` method sets the weight of the individual wedges and updates the force model by calling the `recalculate` method. Therefore, if weights are set in every frame, it is not needed to call `recalculate` explicitly.

The class contains methods for calculation direction, distance, and signed distance between two wedges as if they were in a cyclical space. That means that distance between two wedges is a minimum of three distances: a simple distance and a distance where the border of the linear space is crossed. In the latter case, the order of the pair of color wedges cannot be handled just by changing the sign, therefore it is handled as two separate cases.

The color wedges are represented by the inner class `Node`. The member `pos` of the class is the center of the color wedge, the `outPos` member is the left bound of the wedge. The difference between `pos` and `oldPos` is used to detect convergence in the `recalculate` method. The class also maintains the radius and margin of the wedges.

The class contains an auxiliary `draw` method, that can be used to visualize the current state of the force model, from which the created color palette can also be inferred.

■ HCL

This class is used to convert from the perceptually uniform HCL color space to the physical RGB screen color space. The conversion takes place in several steps through the polar CIELAB, CIELAB, XYZ, and finally RGB color spaces, for which functions `polarLABtoLAB`, `LABtoXYZ`, and `XYZtoRGB2` (alternatively `XYZtoRGB`) are used. All the functions, excluding `XYZtoRGB2` which is sourced from the Catalano framework [5], are from the implementation by Zeileis et al. [20]. The `XYZtoRGB2` and `XYZtoRGB` have equivalent functions but are implemented differently and resulting conversions are also different. The difference in implementation is raising the RGB channels to a power of 0.4166 (possibly it is a gamma correction) in the `XYZtoRGB2` function, as shown in the sample of the implementation:

```
if ( rgb[0] > 0.0031308f ){rgb[0] = 1.055f * ( (float)Math.pow(rgb[0],
    0.4166f) ) - 0.055f;}
else{rgb[0] = 12.92f * rgb[0];}
```

Conversion with the use of `XYZtoRGB` creates darker color and performs worse in the property of isoluminance, which is observable even better, if luminance is equalized to the level of the conversion with the use of `XYZtoRGB2` (for comparison, refer to the section 2.1, Figure 2.3).

■ 3.1.2 Data

This package contains classes used for loading, representation, and visualization of the cluster data. There is the `Kmeans` class, that implements the clustering algorithm [2]. Next is the class `ClusterData`, which loads and parses the data from a file. The data are converted to a tree of clusters (abstract class `Cluster`). The `Cluster` class is implemented by classes `RootCluster`, which is also responsible for visualization, and the `ClusterNode` and `ClusterLeaf` classes.

■ Kmeans

This class contains the implementation of the Kmeans++ algorithm. The Kmeans++ is the improved version of Kmeans, delivering greater accuracy and lower running time [3]. The improvement lies in the method of choosing initial cluster centers. While with the Kmeans algorithm, initial centers are usually placed spatially uniform, the Kmeans++ chooses the first center uniformly random and the next one in a data point chosen randomly with the probability

$$P(x) = \frac{D(x)^2}{\sum_{x \in X} D(x)^2}, \quad (3.1)$$

where $D(x)$ is the distance between the point x and the closest of already placed initial centers. This is repeated k centers are placed and then the algorithm proceeds with the standard Kmeans algorithm. This means assigning every point to its closest cluster center and then repositioning the cluster center to the center of mass of the associated points. This step is repeated until the position of all cluster centers no longer changes.

ClusterData

This class loads cluster data and then uses the `RootCluster` class to create the data structure for the visualization, draw the visualization, and produce the output for the library for label placement. The class uses the singleton design pattern, meaning that there can only ever be one instance of this class. The class inherits from `VisualizationView.Drawable`, which allows to draw a visualization on the `GLCanvas2` canvas.

The class retrieves the data in its `getVertices` method in CSV format table, where each row represents one point. Point position projected into two-dimensional space is stored in the columns `EmbedSOM1` and `EmbedSOM2`, which are the fifth and sixth columns from the end of the table. The assignment of point to a cluster is given by the `Population` and `ClusterKey` columns, which are the last and pre-last columns of the table respectively. The information in these two columns is identical, the `Population` column identifies the cluster by letter of the alphabe and the column `ClusterKey` by number. The method returns an array of points having 2D coordinates and cluster identifiers `Population` and `ClusterKey`.

The data points are loaded from CSV and sorted according to their cluster in the `load` method. Since the number of clusters is small, the counting sort algorithm is used. Sorted points are used to initialize the `RootCluster` class. The clusters of points are then subdivided using the `subdivide` method of the `ClusterLeaf` class. Once the hierarchy construction is complete, the subclasses of `RootCluster` are initialized. Lastly, the `Subsampling` class is used to assign the points level of zoom at which they should appear when the subsampling option is activated in the application settings.

Cluster

Abstract class representing the nodes of the cluster hierarchy tree. The cluster tree consists of leaves, represented by the `ClusterLeaf` class, nodes represented by the `ClusterNode`, and the root, which is the `RootCluster`. All of these classes are subclasses of the `Cluster` abstract class. Each node and leaf of the tree has a unique identifier. The leaf identifier is written as an attribute in all vertices belonging to the given leaf. Based on this attribute, a color from a single color field is assigned in the vertex shader. The leaf identifier is also rendered by the `ClusterLeaf` class into the stencil buffer, allowing clusters to be selected by mouse click. The identifier is also used when generating output for the label placement library, which implies that the identifier can have a maximum of 128 different values (refer to section [2.6.2](#) for

details). The class contains abstract methods that must be implemented by all derived subclasses. These methods are `draw`, which draws a given cluster as a point cloud, `drawMask`, which draws the cluster as a mask for the label placement library, `drawDensity`, which draws the cluster as a density field, `getN`, which returns the total number of vertices in the cluster and all its descending subclusters, `getChildren`, which returns the array of the cluster subclusters, and finally, the `updateWeights`, which returns the total number of fragments drawn given cluster and all its subclusters. In the case of node (non-leaf) clusters, the method will also update the dynamic color palette of the node with the number of fragments drawn in its individual children.

■ ClusterLeaf

The `draw` of the class `ClusterLeaf` draws the part one vertex array shared across all leaves, which contain points belonging to this cluster. The method performs GL query on the number of fragments passed. The query result is read in the `updateWeights` method after rendering is completed. This prevents CPU to GPU synchronization after every cluster is drawn, increasing performance two-fold to three-fold.

The `ClusterLeaf` class also implements the method `subdivide`, which subdivides the leaf using the Kmeans++ algorithm [2] and returns node `ClusterNode` having the newly created leaves as its children. The new leaves are also assigned new unique identifiers, that are also written back into attributes of corresponding vertices.

■ ClusterNode

`ClusterNode` represents the inner nodes of the cluster tree. Each node subdivides its allocated color wedge into sub-wedges using the dynamic palette method. The `draw` method of the node calls `draw` on all its children, to which it also passes the allocated sub-wedges. The `updateWeights` method calls `updateWeight` on its children too. The children return the number of fragments rendered in their subtree. The node updates the state of its color palette with these counts and returns the sum to its parent, which then updates its color palette with this value.

■ RootCluster

`RootCluster` contains all the methods for data visualization. This class maintains the cluster tree data structure used to draw the clusters, and the visualization parameters, and also creates and renders the hierarchy tree visualization. The `RootCluster` drawing methods are called from the `ClusterData` class.

`RootCluster` creates a tree of depth 1 in its constructor, where each leaf corresponds to one cluster as defined by the *Population* attribute in the original data. A deeper hierarchy is created by subdividing the leaves using the Kmeans++ algorithm. It also creates a single vertex buffer object from

the ordered points, allowing for rendering on the GPU. The parts of the vertex buffers representing individual clusters are rendered in corresponding leaf nodes. The attributes of the vertex array object are four-element vectors containing 2D position, depth, identifier and a single float value containing the LOD, which is a zoom threshold, above which the point should be discarded. This value is the output of the subsampling algorithm [6][18] and the points are only discarded when the subsampling option is checked in the visualization configuration.

The point cloud visualization is rendered recursively using the `draw` method of the class. The method calls `draw` on all the children of the root node, similarly to `ClusterNode`. The method also draws the second cursor.

The `RootCluster` class also contains methods for the selection of clusters in the cluster hierarchy. The entry point of the selection procedure is the `click` method. This method is called by the `VisualizationView` class with the mouse position and a boolean parameter indicating whether the `shift` key is pressed. If `shift` is not pressed, the recursive function `select` will select subtree with given ID (that is, cluster with the given ID and all its subclusters). All other clusters are unselected. If `shift` is pressed, the recursive function `selectSubtree` will add subtree with given ID to the selection, if it is not selected, and remove it otherwise. Finally, the recursive function `subtreeState` is called. This function unselects all clusters that have none of their subclusters selected.

The `RootCluster` class also contains three subclasses. The first one, `TreeVisualizer`, visualizes the hierarchy tree, the second one, `DensityVisualizer` maintains the visualization of the data using the density function and the last one, `TypeMask` creates the bitmask, that could be used by the library for label placement in the future. These classes are initialized in their `init` methods, rather than in their constructors, because they are constructed in the `RootCluster`, but the final cluster hierarchy, which is necessary for the initializations, is created later by the `ClusterData` class. The `ClusterData` class constructs the `RootCluster`, then it extends the cluster hierarchy by subdividing the clusters, and then the `init` methods of the subclasses are called.

All of the subclasses draw to the same GL context because GL shaders, buffers and textures cannot be shared between different contexts and cloning each of these resources to every context is ineffective. The scissor test ensures the separation of the visualizations in a sense that rendering output or clearing of color or stencil of one visualization will not affect the others.

TreeVisualizer. Builds and draws the geometrical representation of the hierarchy tree. The nodes of the tree contain their own selection state. The selection state of the clusters and the tree visualization nodes is maintained and synchronized by the `RootCluster` class. The space between the leaves of the tree is proportional to the number of points in the leaves and so is the leaf size. The top-level nodes of the tree are labeled by the population of the corresponding cluster, as given by the visualized dataset.

DensityVisualizer. Renders the density function in its `drawDensity` method. The density function is rendered into a separate texture for each top-level cluster once at the first frame. The channel format of the textures is 32-bit float, because drawing the densities means adding a lot of very small values to a single value and the 8-bit per channel precision is not enough precise to compose the density correctly. The textures are then sampled by the `drawPeel` and `drawContours` methods. The `drawPeel` method implements depth peeling and relief shading. Output of the depth peeling are four textures with each fragment containing color and density. The area visualization is rendered by the `draw` method. This method sets the parameters of the shader `tex` and then uses either the `drawContours` or the `drawPeel` method to render the visualization based on the visualization configuration.

TypeMask. Creates input texture for the label placement in the `draw` method. The texture has four 32-bit unsigned integer channels. The value of the i -th bit marks the presence of a point belonging to the i -th leaf in the hierarchy of clusters. The method recursively draws the tree of clusters, similarly to the `draw` method of the `RootCluster` class, except it does neither count fragments nor allocate color wedges. Here, every leaf is drawn with a color equal to a 128-bit vector with a value of $1 \ll \text{LeafID}$. To compose the vectors when points with different IDs are drawn on a single pixel, new fragment value is composed using logical or between incoming fragment and accumulated value. The operation is set by `glLogicOp(GL_OR)`.

■ Subsampling

The `Subsampling` class implements the method of subsampling data points with the dart throwing technique to reduce the overdraw of the scatterplot [6][18].

The subsampling algorithm requires acceleration of spatial search of the data points and tracking of fill rates of vertices of individual classes. The data points are given in a form of a vertex array. The output is the zoom level of individual vertices written back into the vertex array as the fifth attribute in the vertex array. The vertices also have the "killed" state, meaning that they are currently set to not be part of the visualization. This state is maintained by setting the vertex zoom level to a very high value (1000.0f). All the vertices are initialized to this state.

The spatial search is accelerated using a uniform grid. Each cell of the grid is represented by the `IntVector` class. This class represents a dynamically resizable vector of integers. This way, the storage capacity at each cell is dynamically resized to accommodate the required number of data points as they are being sorted into the cells. The points are stored as indices to the vertex array. The data points are then assigned to one of three zoom levels. Not all points will be assigned even after the last level is built; many will remain killed after the end of the procedure.

The tracking of fill rates is done by the `ClassStat` subclass. There is one instance of this class per each top-level cluster in the dataset. The class

contains array `data` of all points belonging to a given cluster as indices to the vertex array. The vertex indices are sorted by the corresponding vertex state. The left part of the array contains the killed vertices, and the right part the accepted vertices, as shown in Figure 3.1. The class provides methods `accept` and `kill`, which will update the state of the vertex and move it to the part of the array according to its new state. The `kill` method is used by the algorithm to kill conflicting vertices as obtained by query to the uniform grid. The query result is an array of indices `q` to the vertex array, but we do not know where in the `data` to find the indices contained in `q`. Searching linearly the `data` array would be computationally expensive, therefore the `ClassStat` class contains helper array `keys` with the same length as the `data` array. The `keys` array is indexed by indices to the vertex array and its elements are indices to the `data` array. This way, the `keys` array is the inverse mapping of the `data` array. The indexing between the vertex array, the `data`, and the `keys` arrays is shown in Figure 3.1.

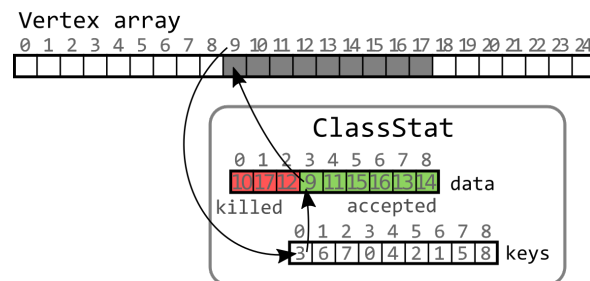


Figure 3.1: Schema of indexing. Indexing between vertex array and `keys` is linear, mappings between `keys` and `data` and between `data` and vertex array are permutations.

But the indexing of the `keys` array is not straightforward. Let us imagine a dataset with 100000 points and a top-level cluster with 100 points. The `data` array in this case would be 100 elements long, but the `keys` array is to be indexed by value up to 100000. Luckily, the points in the vertex array are already sorted by population, that is by the top-level cluster they belong into. This means, that the difference between the smallest and the largest index of points in the cluster equals the number of total points in the given cluster. The smallest possible index is known; therefore it can be subtracted from a given index. Now we have a value ranging from 0 to 99, that can be used to index the `keys` array. The vertex array contains the attribute `population`, that identifies the instance of `ClassStat` that manages the state of a given point.

The tracking of fill rates and spatial search acceleration is utilized during the building of the zoom levels. Each zoom level is built by the `buildLevel` method. The method takes as input the data points and attempts to add a given number of random yet unassigned points to the currently built level of zoom. Each of the attempts is performed k times. The attempt fails when the point when the chosen point p is closer than r to some of the already added points (checked by the `IsConflict` method) and the points too close

to p can not be removed (checked by the `conflictsRemovable` method). The conditions for removing conflicting points are described in the [2.5.3](#) section.

The subsampling algorithm is implemented in the constructor of the `Subsampling` class. Since the output of the subsampling algorithm is written back into the input vertex array, the return value of the constructor can be ignored.

Chapter 4

Results

An application visualizing clusters of real measured hierarchically grouped data in real-time was created. The visualized properties are the density of the clusters, their mutual overlaps, and the hierarchical structure. The application is written in the Java language and utilizes GPU accelerated rendering through the JOGL library, which is a Java binding for OpenGL. Running on AMD Ryzen™ 5 3500U APU with integrated Radeon™ Vega 8 GPU, we have measured 29 FPS on a dataset with 250 000 points and 68 leaf nodes in the hierarchy and 30 FPS on a dataset with 800 000 points and 50 leaf nodes. When rendering point visualization alone, the framerate was 60 on both datasets, but it was 40 and 45 FPS when rendering area visualization only, showing that area visualization is computationally more expansive than rendering of a point cloud.

This is because the area visualization is created by rendering n density textures over the whole visualization view, where n is the number of top level clusters. In the depth peeling procedure, four layers are created by the composition of the density textures. This means $4n$ draw commands are issued in each frame in total. The size of the actually rendered area the draw commands is cropped to the size of the area visualization view using the scissor test. With n being 22 and the view size being 304x349 after the scissor test, we get 9.3M fragments drawn per frame. For comparison, a 4K screen with a resolution of 3840x2160 has only 8.3M pixels. The number of fragments drawn per frame could be decreased further by cropping not only to the extent of the area visualization view but only to the extent of the currently drawn cluster. However, this is not possible with the current implementation of depth peeling, where fragments with zero cluster density write to depth texture. Therefore, even areas with no cluster occurrence cannot be neglected.

The `gluegen-rt` library, used by the JOGL library, contains a bug that causes an illegal reflective access operation. This is an operation, where perpetrating module uses Java reflection API to gain access to non-public methods of other modules at runtime. Since Java 9, such an operation will be detected and result in a warning, but in Java 17 and above, this is not allowed, and the application is terminated.

On some Intel GPUs, the GLSL layout qualifier `location` is ignored for

variables `zoom`, `view`, and `useLod` of the shader `vert` and a different location is assigned. Therefore, the location of these variables is queried at runtime using `glGetUniformLocation`.

Due to the unavailability of hierarchy data, cluster data were additionally divided into subclusters using the Kmeans++ method [2], resulting in three levels of hierarchy.

The point cloud visualization allows to distinguish individual clusters in a multilevel hierarchy, as shown in Figure 4.2, where zoom at three consecutive levels is depicted. Individual clusters at any level of the hierarchy can be highlighted (Figure 4.3) by clicking on the cluster of interest. To make selection easier and to allow the selection of an entire subtree or of a cluster overlapped by other clusters, a visualization of the hierarchy as a tree was added to the visualization (Figure 4.3). The selection is done by clicking on a node in the tree. Multi-selection is also possible.

The point cloud visualization is implemented in two modes: a discernability-based mode and a density-based mode. A comparison of these two modes is shown in Figure 4.1. The density-based visualization allows for a precise perception of the shapes of the two large blue clusters at the top left of the visualizations but at a cost of lower discernability of the clusters' category.

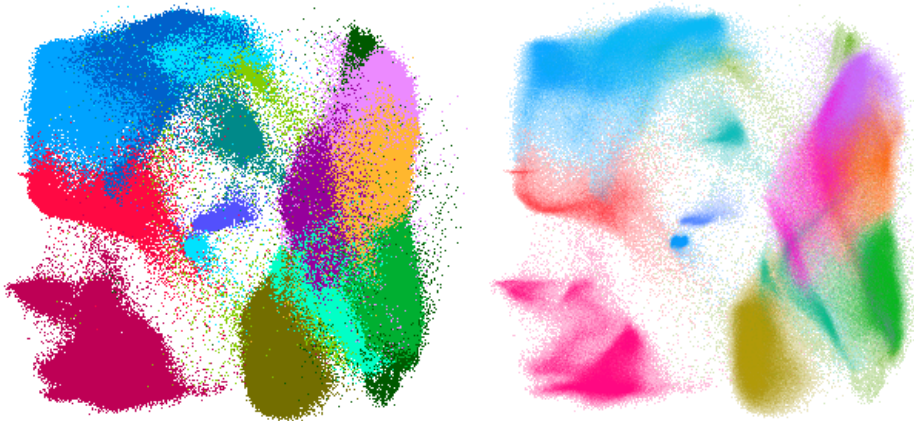


Figure 4.1: a) Discernability mode. b) Density mode.

The area visualization is also implemented in two modes, and these are contouring of the density function and relief shading of the density function. Both modes give a good overview of cluster overlap while maintaining color and density discernability. Changing contour height allows to blend between clear view with few cluster overlaps and a more precise, but also more cluttered view, as shown in Figure 4.4. The relief shading can show both relative and absolute changes in clusters' densities. Relative shading gives more detail about changes in density, but distorts the perception of clusters' absolute density. The approaches are compared on Figure 4.5. Semi-transparent relief allows to overview both overlaps and density to a limited extent (Figure 4.6).

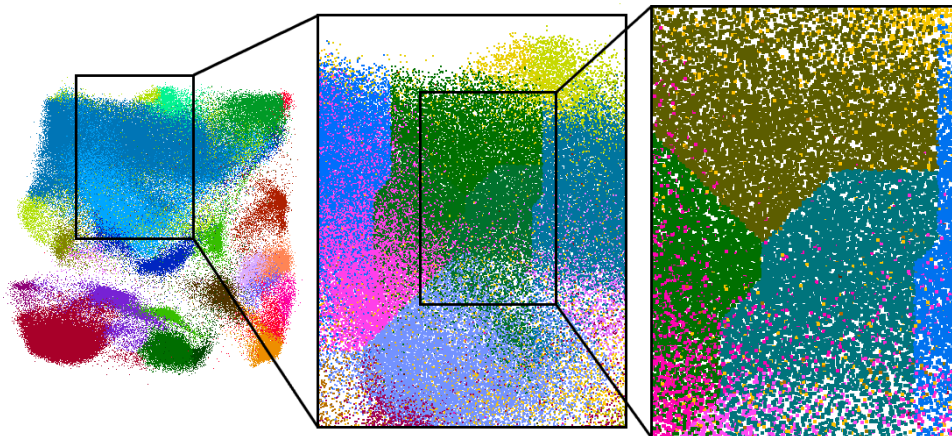


Figure 4.2: Hierarchical coloring

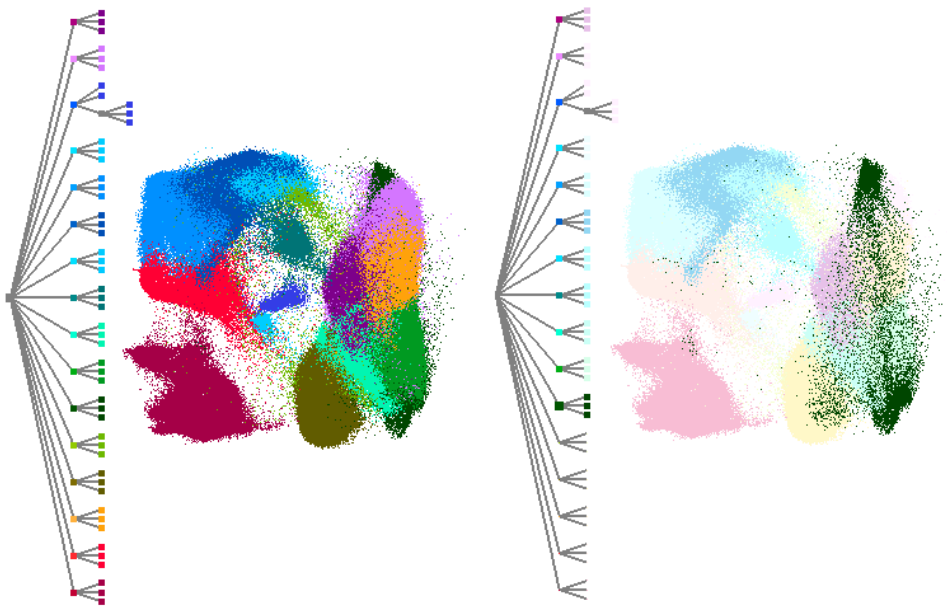


Figure 4.3: Cluster selection



Figure 4.4: Contours at different heights

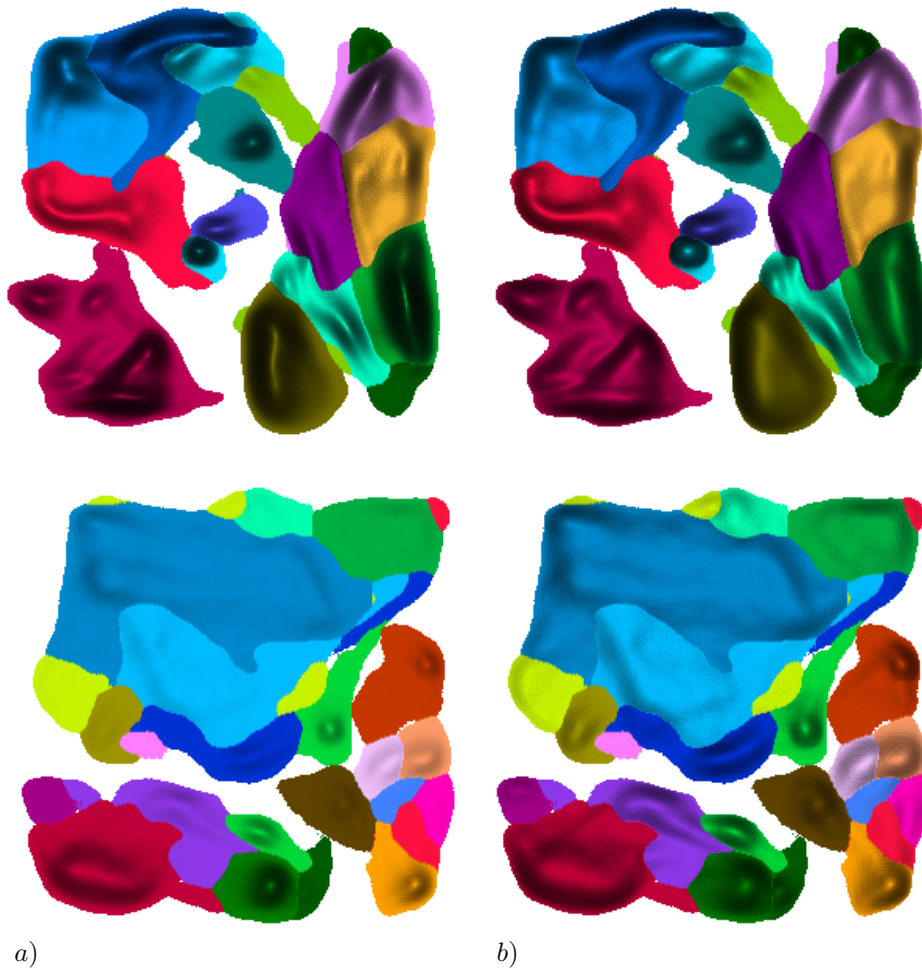


Figure 4.5: a) Absolute relief shading. b) Normalized (relative) relief shading.

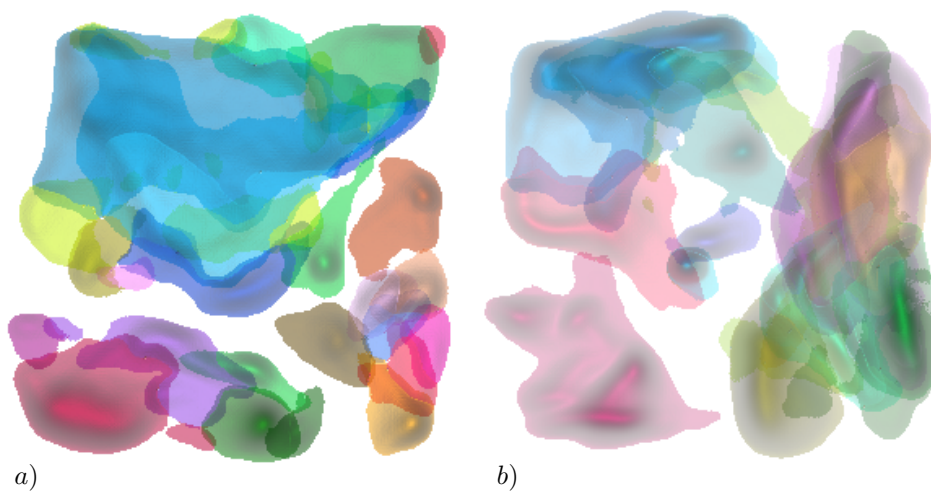


Figure 4.6: a) Low overlap. b) High overlap creates visual clutter.

Chapter 5

User evaluation

To evaluate the performance of our design, the visualization was tested with six participants. The participants were given a textual introduction to the application controls and a set of nine tasks. The text is provided both in Czech and in English in the attachments. The participants were then asked to rate the difficulty of the tasks and leave a comment on the visualization or the tasks. The tasks are chosen to cover the abstract task types (table 5.1) performed on scatterplots as categorized by Sarikaya and Gleicher [15]. The task types cover the identification of clusters, identification of cluster density, and overlaps, which are the goals of our visualization.

Types 3, 8, and 12 are not covered by our set of tasks. Types 3 and 12 could not be covered because the spatialization of the data point is abstract and does not encode any attributes, because the spatialization is the result of reduction of high dimensional position into 2D. Type 8 is partially covered by tasks 3 and 4 where the user is asked to find clusters with described distribution. Such a task requires to be able to characterize the distributions too, in order to match the clusters with the textual description of distribution. Tasks 1. to 5. were performed on the first dataset and the remaining tasks on the second dataset. The tasks are:

1. Are these clusters of same population? Do not use cluster selection.

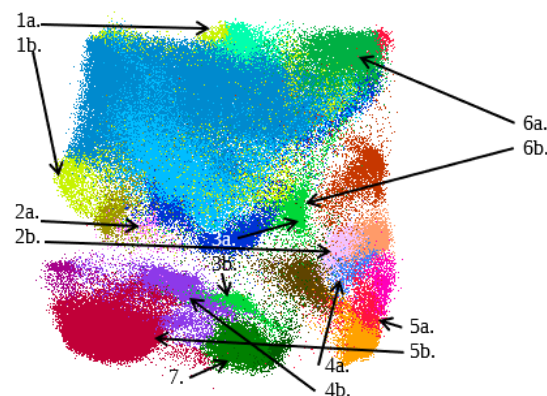


Figure 5.1: Image for the tasks 1. and 2.

	Task	Description
object-centric	1 Identify object	Identify the referent from the representation
	2 Locate object	Find a particular object in its new spatialization
	3 Verify object	Reconcile attribute of an object with its spatialization (or other encoding)
	4 Object comparison	Do objects have similar attributes? Are these objects similar in some way?
browsing	5 Explore neighborhood	Explore the properties of objects in a neighborhood
	6 Search for known motif	Find a particular known pattern (cluster, correlation)
	7 Explore data	Look for things that look unusual, global trends
aggregate-level	8 Characterize distribution	Do objects cluster? Part of a manifold? Range of values?
	9 Identify anomalies	Find objects that do not match the 'modal' distribution
	10 Identify correlation	Determine level of correlation
	11 Numerosity comparison	Compare the numerosity/density in different regions of the graph
	12 Understand distances	Understanding a given spatialization (e.g. relative distances)

Table 5.1: Classification of scatterplot analysis tasks. Adapted from "Scatterplots: Tasks, Data, and Designs" [15]

2. Find population of these clusters. Do not use cluster selection.
3. Points of some clusters are unevenly distributed, with most of the points located in small part of the cluster's area, while the remaining area is very sparse. These clusters can be characterised as a center and outliers. What clusters exhibit this property the most? Find top five. Sort descending.
4. Judging by average density, what are the densest and sparsest clusters? If densities of some clusters appear to be equal, write all of them.
5. What clusters are overlapping with cluster 'o'? Sort by area of overlap descending. Write how many subclusters of 'o' they overlap.
6. What clusters is the most prevalent at pointed position? What other clusters can be found there? Sort by density descending.

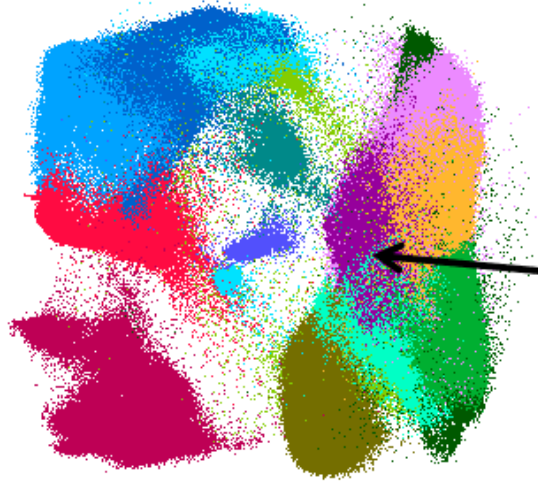


Figure 5.2: Image for the task 6.

7. Find clusters with unusual shape or density distribution, or multiple centers.
8. What cluster spatially correlates the most with the ‘f’ cluster?
9. Find groups of clusters with correlating shape, position, density distribution. There may be multiple such groups.

Among the participants, there were 3 graduate students (participants 1, 2, 6), one high school student (participant 3), one professor at a medical university (participant 5), and one software developer (participant 4). The

T\P	1	2	3	4	5	6	<i>correct</i>
1	ynnnyn	ynynnn	ynynnn	ynynnn	ynynnn	ynynnn	ynynnn
2	hrjnvli	htjqbl	htjnbli	htjsdl	htjqblk	asjqbli	htjqvli
3	djkgs	dgpkh (hkpgd)	djsep	djloa (aoljd)	darmn	jdias	
4	nai,op	n,h	a,h	i,tu	n,c	n,i	
5	nhpglj	hnp	nhlp	nhplgjd	np	nhlmjp	
6	ocp	phce	phcde	ocpf	mpojch	ope	ph({c e}do)
7	baefko	kfe	ahf	efo	m	a	
8	o	o	o	o	g	o	
9	jkl,cop	jkl,co	gh,co,hp	gh,ijkl, op,ef	kl,ik,op	gh,mn,op	

Table 5.2: Participant answers

participants were mostly using point visualization in discernability mode with the help of zooming for the first two tasks. Zooming in provided better color discernability due to the dynamic color palette increasing the color distance of the cluster in the center of the view from the other clusters. Most

participants were able to correctly compare the category of clusters (95% correct), but connecting clusters with labels proved to be more difficult (70% correct). Some of the users forgot to label the seventh cluster in the second task. This is not considered as an error. The success rate is calculated as the ratio of wrong answers to the number of given answers.

The participants approached the third task with various methods, the most favorite being the use of point visualization in density mode. Two participants used the relief visualization, from that one with deactivated normalization, and one participant point visualization in discernability mode with increased point size. Some users misunderstood the assignment and sorted the cluster in reverse order, that is, the most evenly distributed cluster first. The original answer of the users is given in parentheses.

Two equally popular methods were employed in the fourth task. About half of the participants used point visualization in density mode and the other half used the relief visualization with cropping by the min/max density settings. Some participants used cluster selection as well.

The main tool in the fifth task was cluster selection. It was used by all six participants. The participants were mostly inspecting the result of selection in the point visualization with one participant additionally zooming in and increasing the point size. Two participants used contour visualization.

In the sixth task, participants used mainly point density visualization and relief visualization. One participant used increasing point size in the point density visualization, and two participants used cropping in the relief visualization. One participant used the contour visualization and animating the contours by sliding the contour height setting. The participants who used cropping the relief function had the most correct answers for this task and Participant 6 rated the difficulty of the second task as 1 despite being only 50% correct. The success rates and difficulty ratings are shown in Table [5.3](#).

The error metric for the sixth task is calculated as the ratio of the number of errors to the number of clusters listed by the participant that are also present in the correct answer. The number of errors is the sum of the number of clusters that need to be moved in the sequence given by the participant to achieve the same ordering as in the correct answer, the number of clusters present in the participant-given list but not in the correct answer, and the number of strongly represented clusters that are not present in the participant-given list. The strongly represented clusters are the 'p' and 'h'. The representation of the clusters 'c' and 'e' is nearly equal, so misordering of these two clusters is tolerated.

In the seventh task, most participants again used point and relief visualizations, two participants used contours.

Animating contours by sliding the contour height was used by two participants in the eighth task. One participant used cluster selection while inspecting the contour visualization. One participant used subsampling, zooming and selection.

There was no single dominant method used in the last task. participants used point visualization and contours and relief visualizations too. Selection

of clusters and sliding contour height was used too. One participant used the relief max density setting to crop the tops of the clusters and see beneath the surfaces.

As can be seen in Table 5.3, columns min and max, the participant rating of the tasks difficulty varies strongly. The participant rating of difficulty often does not correspond to correctness of their answer. For example, in the first two tasks, participant 2 rated the difficulty as 4 despite being more correct than participants who rated the difficulty of this task as 1 or 2.

T\P	1	2	3	4	5	6	min	max	avg
1	2 (67%)	4 (100%)	1 (100%)	1 (100%)	2 (100%)	2 (100%)	1	4	2
2	1 (71%)	4 (83%)	1 (71%)	3 (67%)	2 (71%)	1 (57%)	1	4	2
3	3	1	2	2	3	4	1	4	2.5
4	5	1	5	3	4	3	1	5	3.5
5	3	3	2	2	4	3	2	5	2.83
6	3 (50%)	2 (100%)	3 (80%)	5 (25%)	4 (33%)	4 (33%)	2	5	3.5
7	2	4	1	2	5	5	1	5	3.17
8	4	1	3	3	3	4	1	4	3
9	4	3	5	2	5	5	2	5	4

Table 5.3: Task difficulty rating.

The participants independently found ways to use the features of the visualization to solve given tasks. The most popular was point cloud visualization, but there is a bias towards point cloud visualization induced by the application, where point visualization is in the top view, and by the task assignment, where pictures are made with the point cloud visualization. For example, all participants used the point cloud visualization to solve the first task, although there was no reason to not use the relief visualization. The other popular features of the visualization were relief and contour visualization including associated density limit parameters, cluster selection, and dynamic coloring. The participants only used the dynamic coloring to improve the discernability of the cluster at the top level of the hierarchy (tasks 1 and 2), not for determining the overlaps of subclusters (task 5), where cluster selection was the prevailing technique. Less popular were the settings of point size, subsampling, and absolute relief shading. Each of these were only used by a single participant.

The only reoccurring comment of the participants was to increase the suppression of unselected clusters. This was commented on by two participants. A control for setting the dimming rate of unselected clusters was added since the obstruction caused by unselected clusters varies per observing conditions (screen properties, light conditions), visualization mode, and use case. This setting is shared by all visualization modes.

Chapter 6

Conclusion

A visualisation of hierarchically clustered multicategorical data of medical measurements was created. Such visualisation is important for medical diagnoses and development. The visualised properties are the density of the clusters, their mutual overlaps and the hierarchical structure.

The nodes of the hierarchy are separated by color with color resolution of twenty-six clusters. Due to the large number of categories, it was necessary to use two color channels, hue and luminance, to achieve resolution. After zooming in, up to seventy clusters can be distinguished with the use of the dynamic color palette. It takes advantage of the fact that the vast majority of these seventy clusters are not visible on the screen when zoomed in. The size of the rendered points is in order to maintain color resolution increased proportionally to the degree of zoom when zooming in. It is possible to specify in the application at which zoom levels coloring with different colors should be activated at individual depths of the hierarchy.

An unsolved issue is color inconsistency at high zoom. In most cases, the color difference between adjacent subclusters is reasonable, but typically in very scattered clusters there are cases where the color shades of the subclusters are very far from each other and significant color changes occur when panning or zooming the view.

Visualization of cluster density based on the transparency of cluster points turned out to be at the expense of distinctness of individual clusters. Color mixing is a minor obstacle that only becomes apparent when the view is zoomed in, where the blending of the boundaries of the two clusters begins to take up a large portion of the screen. The main problem is the impossibility of using the luminance channel, which visually interferes with the transparency of the points and thus with the density of the clusters. For that reason, two visualization modes were created. Density-focused mode renders points with transparency, while discriminability-focused mode renders fully opaque points. When visualizing the density of clusters, the application enables rendering in two passes, which achieves good visibility of very sparse areas and at the same time distinguishable levels of densities within very dense clusters.

Both modes respect the mutual overlap of the clusters, i.e., that where several clusters overlap, the color distribution corresponds to the density of the individual clusters. In the case of opaque rendering of points, the overlap

visualization was achieved by activating the depth test and randomizing of the point depth.

The user evaluation has shown that all of the visualization techniques implemented proved to be useful in performing scatterplot tasks. Only exception may be the subsampling, which was only used by a single participant in a single task, and adding to that, participants who used other techniques, reported lower difficulty for the task. The dynamic color palette had only a limited success, as it was only used to improve discernability of clusters at the top level of the hierarchy. For operations at lower levels, cluster selection was preferred.

6.1 Further developments

The dynamic color palette may be extended to two dimensions or even three dimensions. This would improve the utilization of the available color space and the discernability of the clusters as well. There is multiple ways of designing multi-dimensional color palette to explore, especially in case of hierarchical coloring. The basic approach is that both hue and luminance would be assigned dynamically based on the situation. The hue channel would be allocated hierarchically and the luminance channel would be used to maintain noticeable differences between neighboring hue wedges. The hierarchical subdivision could be also extended to more dimensions, where allocated and subdivided units would be circles or spheres in 2D or 3D color space instead of hue wedges in a 1D space.

The connected views technique can be extended so that more than only two views would be displayed. The user would be able to add and remove views, select visualization technique in each of the views and select clusters displayed in the view – a single cluster or a group of clusters.

The visualization technique of parallel coordinates could be also implemented. The limitation are the color discernability of roughly twenty polylines crossing each other and overview of cluster distribution on polyline consisting of twenty or more segments. These limitations lead to displaying only subset of clusters or subset of dimension. Dimension reduction from original high-space to a lower dimensional space with five to ten dimensions could also be employed.

A mentioned but unexplored method is to place labels in the visualization, which would make it easier to distinguish and identify individual clusters. The labeling of the clusters needs to consider the density and overlapping of the clusters, as naive label placement would produce ambiguous or confusing labeling. Čmolík and Bittner [21] propose a method that evaluates places of possible label anchoring based on opacity salience, overlap salience, occlusion salience, and the distance from the edge of the labeled object.



Bibliography

- [1] Hany Alashwal, Mohamed El Halaby, Jacob J Crouse, Areeg Abdalla, and Ahmed A Moustafa. The application of unsupervised clustering methods to alzheimer’s disease. *Frontiers in Computational Neuroscience*, 13:31, 2019.
- [2] Jason Altschuler. Github - jasonaltschuler: Improvements on classical kmeans clustering, 2013. <https://github.com/JasonAltschuler/KMeansPlusPlus> [online; accessed 2023-01-05].
- [3] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics.
- [4] Lars Behnke. Github - lbehnke/hierarchical-clustering-java: Implementation of an agglomerative hierarchical clustering algorithm in java, 2014. <https://github.com/lbehnke/hierarchical-clustering-java> [online; accessed 2023-01-05].
- [5] Diego Catalano, Robert Theis, and Yuri Poudre. Github - diego catalano/catalano-framework: Framework, 2013. <https://github.com/DiegoCatalano> [online; accessed 2023-01-05].
- [6] Haidong Chen, Wei Chen, Honghui Mei, Zhiqi Liu, Kun Zhou, Weifeng Chen, Wentao Gu, and Kwan-Liu Ma. Visual abstraction and exploration of multi-class scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1683–1692, 2014.
- [7] Cass Everitt. Order-independent transparency. <https://developer.download.nvidia.com/assets/gamedev/docs/OrderIndependentTransparency.pdf>, 2001.
- [8] Ying-Huey Fua, M.O. Ward, and E.A. Rundensteiner. Hierarchical parallel coordinates for exploration of large datasets. In *Proceedings Visualization ’99 (Cat. No.99CB37067)*, pages 43–50, 1999.

- [9] Florian Heimerl, Chih-Ching Chang, Alper Sarikaya, and Michael Gleicher. Visual designs for binned aggregation of multi-class scatterplots, 2018.
- [10] David Kouřil, Ladislav Čmolík, Barbora Kozlíková, Hsiang-Yun Wu, Graham Johnson, David S. Goodsell, Arthur Olson, M. Eduard Gröller, and Ivan Viola. Labels on levels: Labeling of multi-scale multi-instance and crowded 3d biological environments. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):977–986, 2019.
- [11] Rolf G. Kuehni. Hue uniformity and the cielab space and color difference formula. *Color Research & Application*, 23(5):314–322, 1998.
- [12] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.
- [13] Kecheng Lu, Mi Feng, Xin Chen, Michael Sedlmair, Oliver Deussen, Dani Lischinski, Zhanglin Cheng, and Yunhai Wang. Palettailor: Discriminable colorization for categorical data. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):475–484, 2021.
- [14] Nathan Moroney. A hypothesis regarding the poor blue constancy of cielab. *Color Research & Application*, 28:371 – 378, 10 2003.
- [15] Alper Sarikaya and Michael Gleicher. Scatterplots: Tasks, data, and designs. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):402–412, 2018.
- [16] Nicholas Waldin, Mathieu Muzic, Manuela Waldner, Eduard Gröller, David Goodsell, Autin Ludovic, and Ivan Viola. Chameleon. *Eurographics Workshop on Visual Computing for Biomedicine*, 2016, 09 2016.
- [17] Jianxin Wang, Jun Ren, Min Li, and Fang-Xiang Wu. Identification of hierarchical and overlapping functional modules in ppi networks. *IEEE Transactions on NanoBioscience*, 11(4):386–393, 2012.
- [18] Li-Yi Wei. Multi-class blue noise sampling. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, New York, NY, USA, 2010. Association for Computing Machinery.
- [19] John Wenskovitch, Ian Crandell, Naren Ramakrishnan, Leanna House, Scotland Leman, and Chris North. Towards a systematic combination of dimension reduction and clustering in visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):131–141, 2018.
- [20] Achim Zeileis, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. colorspace: A toolbox for manipulating and assessing colors and palettes. *Journal of Statistical Software*, 96(1):1–49, 2020.

- [21] Ladislav Čmolík and Jiří Bittner. Real-time external labeling of ghosted views. *IEEE Transactions on Visualization and Computer Graphics*, 25(7):2458–2470, 2019.



Attachments

The application visualizes hierarchically clustered set of points. The points are separated into clusters first by category and then by mutual proximity.

The application window is divided into view and configuration. View contains visualization of hierarchy, point cloud visualization and density visualization. Visualizations of density and visualization of point are connected. View can be zoomed using the mouse wheel. Clusters can be selected in the point visualization and the hierarchy visualization. Multiple clusters can be selected using the shift key. The configuration is separated into three parts, first one contains settings shared by all three visualizations, second one settings of the point visualization and the last one settings of the density visualization.

The point visualization has two modes – mode of cluster discernability and for density discernability. In the cluster discernability mode, the luminance color channel is used and point are drawn in random order. This way, when two clusters overlap, points of both clusters are represented at the place of overlap. In the density mode, point are drawn with transparency. To be able to discern the density distribution, the point transparency needs to be set very low, around 0.1. Cluster can be selected by clicking into the point visualization or to the hierarchy visualization. Multiple clusters can be selected or unselected with clicking and holding the shift key.

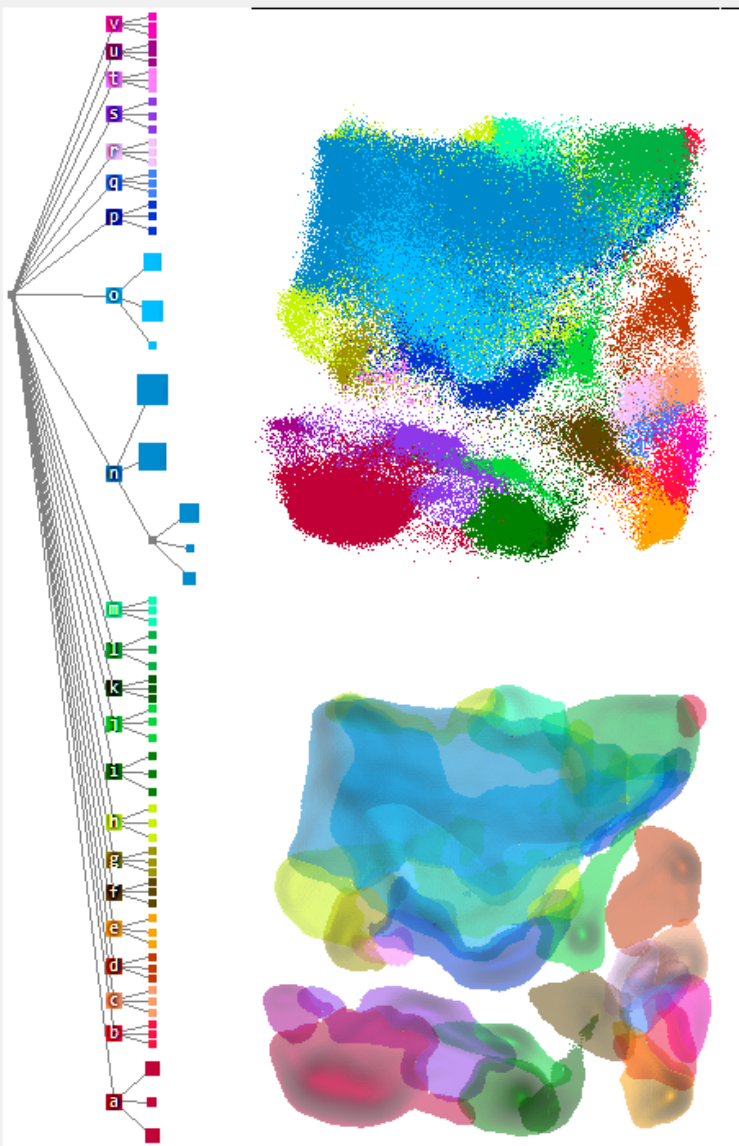
The settings of the point visualization allow to control the size of points and their transparency. If the transparency is set lower than 1, density mode is used, otherwise, the discernability mode is used. Next, there are settings of levels of zoom, where coloring at next level of hierarchy should be activated. Next is the setting of point subsampling. This will reduce the number of points drawn. The number of points will increase when zooming in. Other settings are only for development purposes.

Density visualization has two modes too – the relief shading mode and contouring mode. Relief mode shades the density function of individual clusters. Contouring mode will create contour line at given level of density. The modes can be switched using the Contours mode checkbox. The contour height and the density range drawn in the relief mode can be adjusted by sliders. The Normalize relief checkbox will set the shading to be proportionate to relative change of density rather to the absolute one. Normalization of relief will distort the perception of density, but will reveal more detail in the changes of the density.

There are two datasets available, Hammer_export.zip and Liechti_export.zip. The datasets to be used with individual tasks are stated in brackets at each task. Loading of the dataset liechti will take about a minute.

Get yourself familiar with the visualization and its configurations before you start working on the tasks.

File View



Dark mode

Suppression

Point size=1.0

Opacity=1.0

Zoom lv1 70

Zoom lv2 100

Zoom lv3 130

Zoom lv4 160

Subsampling

pre-pass

RGB2/gamma

Depth test

Selected on top

Contours mode

Contour height=0.004

Normalize relief

Min density=0.004

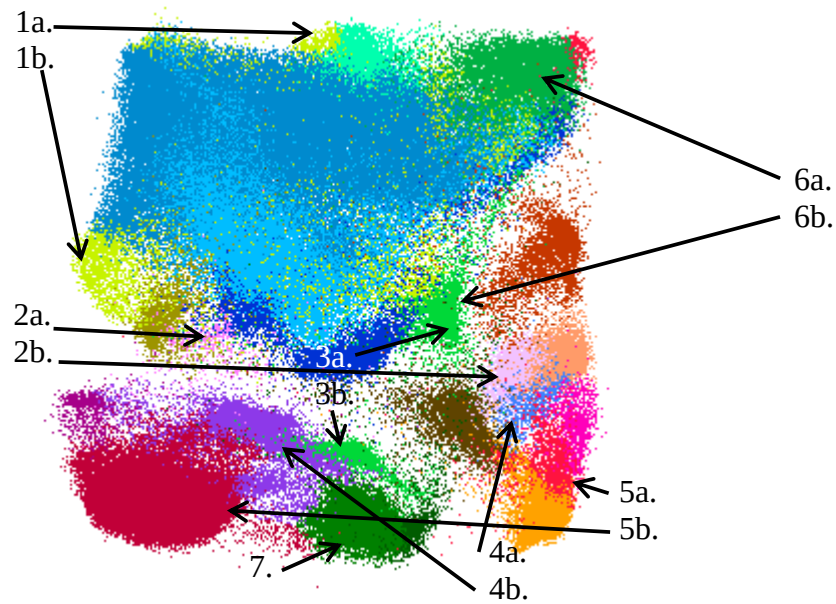
Max density=4.0

Opacity=0.5

Tasks

1. Are these pairs of clusters of same category? Do not use cluster selection. (Hammer_export)

- 1a, b.
- 2a, b.
- 3a, b.
- 4a, b.
- 5a, b.
- 6a, b.



2. Find category of these clusters. Do not use cluster selection. (Hammer_export)

- 1a.
- 2a.
- 3a.
- 4a.
- 5a.
- 6a.
- 7.

3. Points of some clusters are unevenly distributed, with most of the points located in small part of the cluster's area, while the remaining area is very sparse. These cluster can be characterised as a center and outliers. What clusters exhibit this property the most? Find top five. Sort descending. (hammer)

- 1.
- 2.
- 3.
- 4.
- 5.

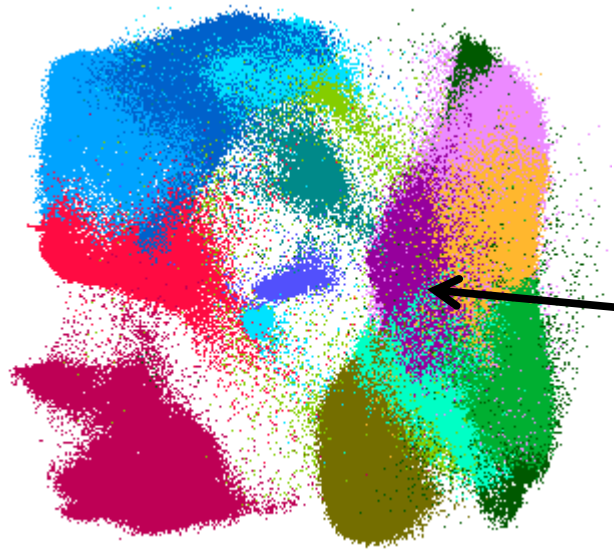
4. Judging by average density, what are the densest and sparsest clusters? If densities of some clusters appear to be equal, write all of them. (Hammer_export)

1.

2.

5. What clusters are overlapping with cluster of category 'o'? Sort by area of overlap descending. Write how many subclusters of 'o' they overlap. (Hammer_export)

6. What cluster is the most prevalent at the pointed position? What other clusters can be found there? Sort by density descending. (Liechti_export)



7. Find clusters with unusual shape or density distribution, or multiple centers. (Liechti_export)

8. What cluster spatially correlates the most with the 'f' cluster? (Liechti_export)

9. Find groups of clusters with correlating shape, position, density distribution. There may be multiple such groups. (Liechti_export)

Rate the tasks 1-9 by difficulty (1-easiest, 5-hardest)

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.

Your (optional) comment to the visualization or the tasks.

Aplikace vizualizuje hierarchicky seskupenou množinu bodů. Body jsou rozdělené do shluků nejdřív podle kategorie a potom podle vzájemné blízkosti.

Okno aplikace je rozděleno na náhled a konfiguraci. Náhled obsahuje vizualizaci hierarchie shluků, bodovou vizualizaci shluků vizualizaci hustoty. Vizualizace hustoty a vizualizace bodů jsou propojené. Pohled lze přibližovat a oddalovat kolečkem myši. Shluky lze označovat v bodové vizualizaci a ve vizualizaci hierarchie. S klávesou shift lze označit více shluků zároveň. Ve vizualizaci hierarchie jsou písmenem označeny kategorie shluků. Konfigurace je rozdělena do tří částí, první obsahuje nastavení společná pro všechny tři vizualizace, druhá nastavení pro bodovou vizualizaci, třetí pro vizualizaci hustoty.

Bodová vizualizace má dva módy – mód pro rozlišení shluků a mód pro rozlišení hustoty. V módu rozlišení shluků je využit barevný kanál jas a body jsou vykreslovány v náhodném pořadí. Tímto, prolínají-li se dva shluky, body obou shluků jsou v místě průniku reprezentované. V módu hustoty jsou body vykreslovány s průhledností. Aby byla hustota rozlišitelná, je potřeba nastavit průhlednost velmi nízko, okolo 0.1. Shluky lze označovat kliknutím do bodové vizualizace nebo kliknutím do vizualizace hierarchie. S klávesou shift lze označit nebo odoznačit více shluků najednou.

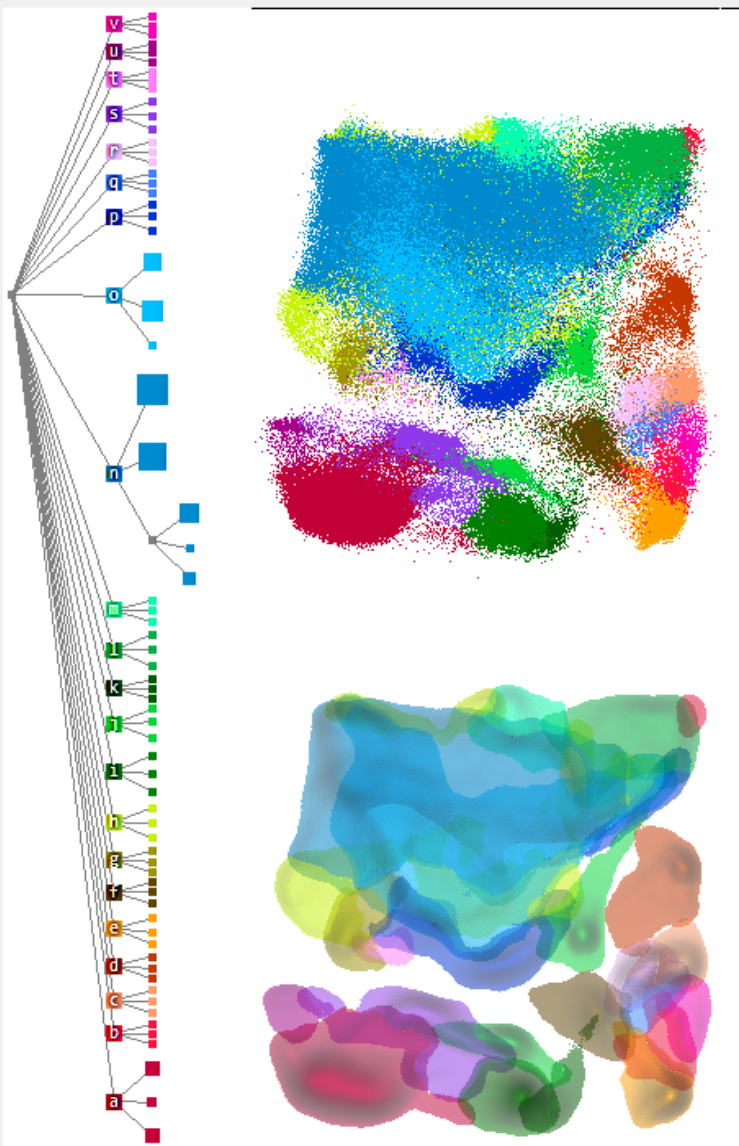
V nastaveních bodové vizualizace lze kontrolovat velikost bodu a průhlednost. Je-li nastavena průhlednost nižší než 1, je použit mód hustoty, jinak je použit mód rozlišení shluků. Je zde také nastavení úrovní přiblížení, kdy se má aktivovat vybarvení na další úroveň hierarchie. Další je nastavení podvzorkování dat. Při aktivování podvzorkování se vykreslí méně bodů větších bodů, s tím že bodů přibývá s přibližováním pohledu. Ostatní nastavení jsou pouze pro testovací účely

Vizualizace hustoty má také dva módy – mód reliéfního stínování a mód konturování. Reliéfní mód stínuje plochu tvořenou funkcí hustoty jednotlivých shluků. Mód konturování vytvoří vrstevnici funkce hustoty na zadané úrovni hustoty. Módy lze přepínat tlačítkem Contours mode. Výšku kontur i rozmezí výšek pro vykreslování reliéfního stínování jsou nastavitelné posuvníky. Tlačítko Normalize relief nastaví stínování aby bylo podle relativní změny hustoty namísto absolutní změny hustoty. Normalizování reliéfu zkreslí vnímání hustoty, ale umožní vidět i malé změny hustoty.

K dispozici jsou dvě datové sady, Hammer_export.zip a Liechti_export.zip. U každé úlohy je v závorce napsáno, jaká datová sada má být použita. Načtení datové sady liechti trvá okolo minutu.

Než začnete řešit úlohy, seznamte se s aplikací. Vyzkoušejte si konfigurace a módy vizualizace.

File View



Dark mode

Suppression

Point size=1.0

Opacity=1.0

Zoom lv1 70

Zoom lv2 100

Zoom lv3 130

Zoom lv4 160

Subsampling

pre-pass

RGB2/gamma

Depth test

Selected on top

Contours mode

Contour height=0.004

Normalize relief

Min density=0.004

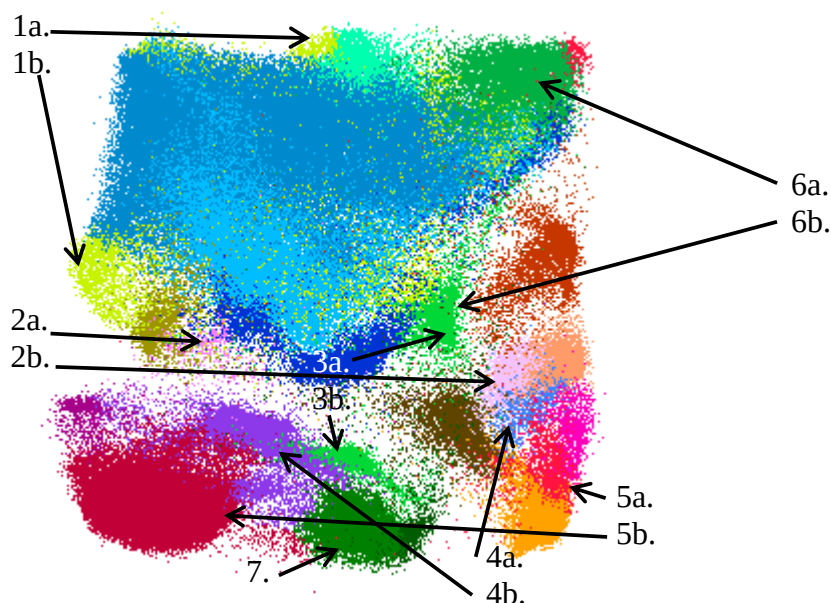
Max density=4.0

Opacity=0.5

Úlohy

1. Jsou následující dvojice shluků stejné kategorie? Shluky jsou označeny na obrázku. Nepoužívejte označování shluků. (Hammer_export)

- 1a, b.
- 2a, b.
- 3a, b.
- 4a, b.
- 5a, b.
- 6a, b.



2. Jaká je kategorie těchto shluků? Nepoužívejte označování shluků. (Hammer_export)

- 1a.
- 2a.
- 3a.
- 4a.
- 5a.
- 6a.
- 7.

3. Body některých shluků jsou distribuovány nerovnoměrně, s většinou jejich bodů soustředěných v malé části shluku, zatímco ostatní plocha je velmi řídká. Takové shluky lze charakterizovat jako střed a rozptýlené body. Které shluky tuto vlastnost projevují nejvíce? Najděte 5 nejvýznamnějších a seřaďte je sestupně. (Hammer_export)

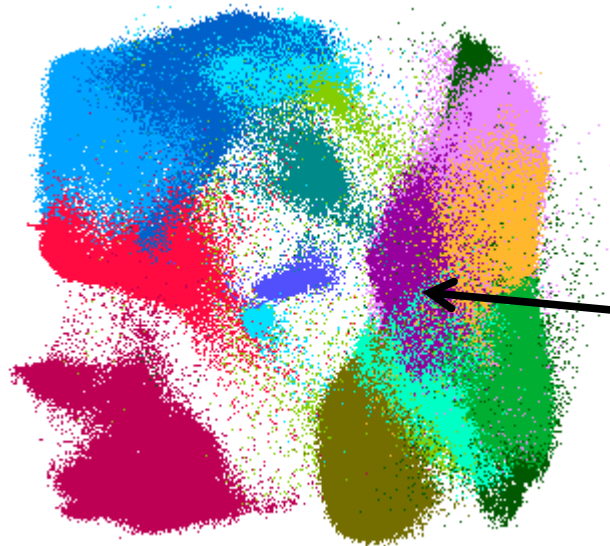
- 1.
- 2.
- 3.
- 4.
- 5.

4. Soudě podle průměrné hustoty, který shluk je nejřidší a který nejhustší? Jestliže se hustota některých shluků zdá být stejná, vypište je všechny. (Hammer_export)

- 1.
- 2.

5. Které shluky se překrývají se shlukem kategorie 'o'? Seřadte sestupně podle odhadované plochy průniku. S kolika podshluky shluku 'o' překrývají? (Hammer_export)

6. Který shluk má největší hustotu na vyznačeném místě? Jaké další shluky se zde vyskytují? Seřadte sestupně podle hustoty. (Liechti_export)



7. Nalezněte shluky s neobvyklým tvarem nebo distribucí hustoty nebo více středy. (Liechti_export)

8. Který shluk prostorově nejvíce koreluje se shlukem 'f'? (Liechti_export)

9. Najděte skupiny blízkých nebo překrývajících se shluků s korelujícím tvarem a rozdělením hustoty. Takovýchto skupin může být více. (Liechti_export)

Ohodnoťte úlohy 1-9 podle obtížnosti (1-nejlehčí, 5-nejtěžší).

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.

Případný komentář k vizualizaci, úlohám