This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

Rapid Labels: Point-Feature Labeling on GPU

Václav Pavlovec, Ladislav Čmolík

Abstract— Labels, short textual annotations are an important component of data visualizations, illustrations, infographics, and geographical maps. In interactive applications, the labeling method responsible for positioning the labels should not take the resources from the application itself. In other words, the labeling method should provide the result as fast as possible. In this work, we propose a greedy point-feature labeling method running on GPU. In contrast to existing methods that position the labels sequentially, the proposed method positions several labels in parallel. Yet, we guarantee that the positioned labels will not overlap, nor will they overlap important visual features. When the proposed method is searching for the label position of a point-feature, the available label candidates are evaluated with respect to overlaps with important visual features, conflicts with label candidates of other point-features, and their ambiguity. The evaluation of each label candidate is done in constant time independently from the number of point-features, the number of important visual features, and the resolution of the created image. Our measurements indicate that the proposed method is able to position more labels than existing greedy methods that do not evaluate conflicts between the label candidates. At the same time, the proposed method achieves a significant increase in performance. The increase in performance is mainly due to the parallelization and the efficient evaluation of label candidates.

Index Terms—Label placement, Point-feature labeling, GPU.

1 Introduction

Labels, short textual annotations, are an important component of data visualizations, illustrations, infographics, and geographical maps. The main function of the labels is to provide annotations, for which we can easily see to which visual features they relate. The annotations (e.g., names of the visual features) are crucial in identifying the visual features, especially in information visualization and geographical maps where all visual features have the same or similar shape.

The label placement was identified as one of the most important problems in Discrete Computational Geometry [3]. The label placement is usually divided according to the feature type that is being labeled into point-feature labeling, line-feature labeling, and area-feature labeling. In this work, we are focusing on the point-feature labeling where the features are points or small areas that can be approximated with points.

The point-feature labeling problem also denoted as *label number maximization* problem is an optimization problem. The goal is to label maximum number of point-features possible with labels of given dimensions such that each label can be unambiguously associated with the labeled point-feature and that no label overlaps with other labels or important visual features (e.g., labels of point-features in a line chart will not overlap the line). The problem does not have an easy solution as it is NP-hard [2, 6, 11, 18].

Oftentimes, the point-feature labeling problem is extended to consider several groups of point-features where the priority of each group is given by order of the groups. In a such case, the goal for each group is to maximize the number of positioned labels while not decreasing the number of positioned labels in groups with higher priority.

The vast majority of point-feature labeling techniques, discussed in detail in Section 2, position the label in close proximity to the labeled point-feature. Most often, the techniques choose the label from four or eight predetermined label candidates around the point-feature. Less often, the techniques use the sliding model where the label candidate is allowed to slide around the point-feature. Only several techniques [13, 16] allow positioning the label further away from the point-feature. They associate the label with the labeled point-feature with a leader - a line or a curve that connects the label with the point-feature.

• *V. Pavlovec, L. Čmolík are with Faculty of Electrical Engineering, Czech Technical University in Prague. E-mail: { pavlova1 | cmolikl}@fel.cvut.cz.*

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

In interactive applications, especially in interactive data visualizations and interactive geographic information systems, we often obtain previously unknown configurations of point-features after loading a dataset or performing a dynamic query [24]. In such situations, to support the user in identifying the point-features, the label layout needs to be found in a very limited time.

In this work, we propose a screen-space greedy method that is running on GPU. The proposed method takes a list of point-features with their coordinates and labels as the input. Further, the method takes as the input a raster image containing the important visual features that cannot be occluded by the labels and a raster image into which the labels will be drawn. This way, any application that is able to produce an image can be easily integrated with the proposed method. The proposed method allows to label a large number of point-features at interactive framerates without the need for a pre-processing of the input data. Alternatively, the proposed method can be utilized to quickly pre-process the input data to generate a label layout that supports zoom and pan. As the method is greedy, we do not guarantee that the proposed method will find the optimal label layout. We highlight our four main contributions:

1. In contrast to existing methods that position the labels sequentially, the proposed method positions several labels in parallel. Yet, we guarantee that the positioned labels will not overlap, nor will they overlap important visual features.

2. The proposed method is evaluating overlaps with important visual features, considering conflicts with label candidates of other point-features, and evaluating ambiguity when it is determining the position of a label. This makes the proposed method more flexible than the existing greedy methods that either evaluate overlaps with important visual features [12, 16] or consider conflicts with label candidates of other point-features [20]. Further, none of the existing greedy methods evaluate ambiguity when it is determining the position of a label.

3. The proposed method uses Summed Area Table [5] to evaluate overlaps of a label candidate with important visual features, to evaluate conflicts of the label candidate with label candidates of other features, and to evaluate ambiguity of the label candidate. Due to the Summed Area Table, the evaluation is done in constant time independently of the number of important visual features, labeled point-features, and resolution of the created image. Please note that there is a performance cost associated with the creation of the Summed Area Table, but the overall acceleration is greater than the cost.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

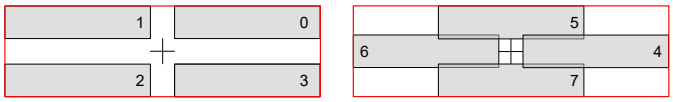


Fig. 1. The eight predetermined label candidates around the point-feature (indicated with the cross). The horizontal and vertical offset of each label candidate from the point-feature is given by the offset o. For each label candidate, we depict its preference as a number (lower is better). The conflict rectangle enclosing all label candidates is indicated with red color.

4. We have compared the performance and the number of labeled point-features of the proposed method with two existing greedy methods, Particle-based labeling of Luboschik et al. [16], and Fast Labels of Kittivorawong et al. [12]. To assess the impact of the greedy approach used in the proposed method on the number of labeled point-features, we have compared the proposed method with a modified meta-heuristic method of Zoraster [33] based on simulated annealing. Our measurements indicate that the proposed method achieves a significant increase in performance compared to the other methods. At the same time, the proposed method is able to label more point-features than the two greedy methods but fewer point-features than the modified method of Zoraster.

2 Related Work

The point-feature labeling problem has been extensively studied, and many optimization strategies were used to solve it. Often, the strategies are using the same concept called conflict graph [25] to represent possible conflicts (e.g., overlaps) among label candidates. Within this concept, the objective of the point-feature labeling problem is to find the maximum independent set of the conflict graph.

Many optimization strategies have been applied to the problem, including simulated annealing [22, 33], tabu search [1, 32], genetic algorithms [8, 28, 29], ant colonies [23], and 0-1 integer programming [9, 17, 19]. For an extensive list of the existing methods, please see the website of Wolff and Strijk [31], who catalogs the map labeling techniques.

All the above-described methods provide high-quality label layouts for various applications. However, only several methods [9,14] consider overlaps among labels and other important visual features. Further, all the above-described methods share a common problem. The computation time ranges from seconds for small datasets to minutes or hours for large datasets. Consequently, these methods cannot be used in interactive applications where the positions of point-features can change in time or where the label layout of a previously unknown configuration of point-features is needed quickly.

Petzold et al. [21] addressed the problem of high computation time for a special case of zooming in a map. They divided the calculation into a time-demanding pre-processing stage and fast labeling phase. In the pre-processing stage, they determine a modified conflict graph to resolve label conflicts at any given zoom level of the map. Due to the time-demanding pre-processing stage, the method is not applicable in scenarios where the point-features cannot be pre-processed in advance.

Been et al. [2] explored the labeling of dynamic maps. They propose a method that can position the labels consistently with panning and zooming operations. Interactive computation times are again achieved by a heavy pre-processing phase.

Greedy methods are also addressing the problem of high computation time, but for a general case of the point-feature labeling problem. Greedy methods do not guarantee to find the optimal label layout. Generally, they will find a worse label layout than the above-described methods, but they are able to calculate the label layout in milliseconds which makes them applicable in situations where heavy pre-processing is not possible.

Mote [20] introduced a greedy method based on a so-called trellis strategy. The method divides the screen area into a 2-dimensional grid and determines a simplified version of a conflict graph for the given point-features. Only point-features in neighboring cells need to be checked for conflict with a given point-feature. The technique assigns a cost to every label candidate and selects the least expensive set of non-conflicting candidates. However, overlaps of labels with important visual features are not evaluated. Subsequently, the labels can occlude important visual features.

Lubostchik et. al. [16] introduced Particle-based labeling, where the particles represent both the already positioned labels and the visual features that the labels cannot overlap. To reduce the number of particles that need to be tested for overlaps with a label candidate, they introduce a local neighborhood data structure similar to the Trellis strategy of Mote [20]. In contrast to the method of Mote, Particle-based labeling does not evaluate conflicts among the label candidates, which leads to label layouts with a lower number of positioned labels when the labels are positioned to the close proximity of the point-features. They compensate it with so-called distant labels positioned further away from the point-features. The relationship is established with straight leaders that connect each distant label with the corresponding point-feature. The distant labels also allow for the labeling of dense clusters of point-features. Unfortunately, the straight leaders often cross the point-features, the labels, and one another, which generates a significant visual clutter. The performance of Particle-based labeling depends heavily on the number of used particles. The method is significantly slower for cases with complex visual features that the labels cannot occlude.

The performance problem of Particle-based labeling was addressed by Kittivorawong et al. [12] with occupancy bitmask. The occupancy bitmask allows faster evaluation of the overlaps of the label candidates with the visual features that cannot be occluded. However, the time required to determine overlap for a label candidate still depends on the size of the label candidate and the screen resolution.

Recently, Lhuillier et al. [13] proposed an approach to determine the distant labels without the visual clutter. They calculate a density map of point-features. The positions of labels are calculated by gradient descent of the density map with leaders corresponding to the trajectory. Therefore, the leaders are guaranteed to be crossing-free. They incorporate an obstacle map to avoid overlaps of labels and other important visual features. However, the method is not suitable for large datasets due to high computation time.

As our proposed method uses Summed Area Table [5] to evaluate the label candidates, we mention that recently Čmolík et al. [4] used Summed Area Table to evaluate label candidates of area-features.

3 Labeling of Point-Features on GPU

In this work, we propose a greedy method that allows to provide visual features that the labels cannot overlap and evaluates conflicts between the label candidates of point-features. Further, we propose how to evaluate the ambiguity of the label candidates. We demonstrate that the evaluation of ambiguity further improves the resulting label layout. Finally, in contrast to other existing methods, the proposed method positions labels of several point-features in parallel.

The proposed method is choosing the label for each point-feature from eight predetermined label candidates around the point-feature, see Figure 1. Each label candidate has its preference (lower is better). The proposed method is choosing the label based on the preference of the label candidates and other properties of the label candidates discussed later. The horizontal and vertical offset of the label candidates from the point-feature is given by the offset o. All label candidates of a point-feature are enclosed in the *conflict rectangle* with the width $w = 2 \cdot w_L + 2 \cdot o$ and height $h = 2 \cdot h_L + 2 \cdot o$, where w_L is the width of the label and h_L is the height of the label.

3.1 Parallel Point-Feature Labeling

To allow labeling of multiple point-features at once, we utilize distance between the point-features. We can label a set of point-features in parallel if the vertical distance $d_v \geq w_{max}$ and the horizontal distance $d_h \geq h_{max}$ for each pair of point-features in the set, where w_{max} and

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

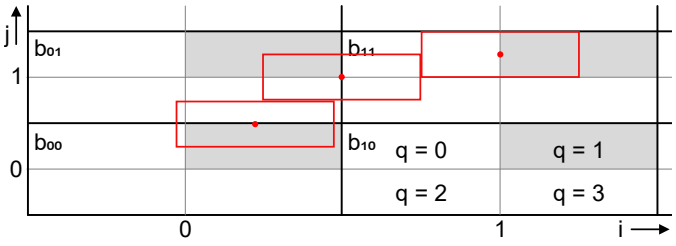


Fig. 2. An illustration of four blocks b_{00}, b_{01}, b_{10}, and b_{11} and their quadrants. The quadrant of each block with quadrant id $q = 1$ is highlighted in gray color. Three point-features at extreme positions and their conflict rectangles are depicted in red color to show that in a worst-case, the conflict rectangles will touch but not overlap.

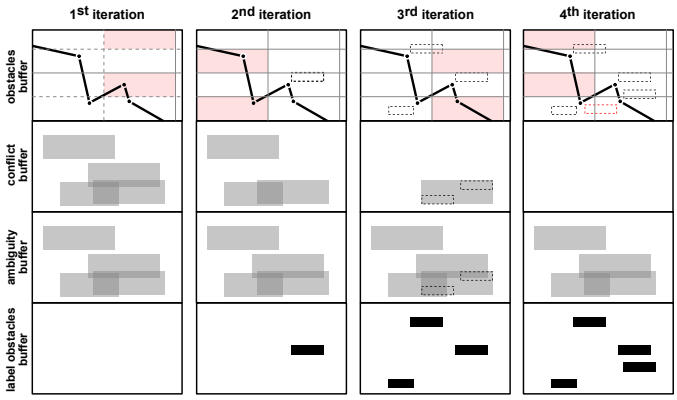


Fig. 3. An illustration of changes in the *evaluation buffer* during the first four iterations of the proposed algorithm. For each iteration we depict the *obstacles buffer*, the *conflict buffer*, the *ambiguity buffer*, and the *label obstacles buffer*. In the *obstacles buffer*, we indicate the processed quadrant with light red color and the labels positioned in the previous iterations. In the first iteration, a label of one point-feature is positioned. The conflict rectangle of the point-feature is removed from the *conflict buffer*, and the label is added to the *label obstacles buffer*.

h_{max} are the maximum with and height of conflict rectangles of all point-features calculated from the provided labels and font size.

Finding such a set of point-features is relatively easy. We divide the space into grid a of blocks of width $2 \cdot w_{max}$ and height $2 \cdot h_{max}$, see Figure 2 for an example. We denote each block as b_{ij}, where i and j are x and y coordinates of the block in the grid. Then, we divide each block into four quadrants of width w_{max} and height h_{max}. We assign quadrant id $q \in \{0,1,2,3\}$ to each quadrant according to its position in the block, see Figure 2 for an example. Finally, by choosing one point-feature from the quadrant with given q of each block, we obtain the set of point-features that can be labeled in parallel. In Figure 2, we highlight the quadrant with $q = 1$ in each block with gray color and depict three point-features with extreme positions and with the width $w = w_{max}$ and height $h = h_{max}$ of their conflict rectangles to show that in the worst case the labels will touch but not overlap. Please note that if certain quadrants do not contain any point-features then the condition that the labels will not overlap is still met.

3.2 Overview of the Algorithm

The proposed algorithm is a screen-space technique that operates in image space. The algorithm works with 2D buffers. A 2D buffer is similar to a 2D raster image, but the values stored in a 2D buffer do not need to be colors; they can be any desired values.

Our algorithm takes the list of point-features and two buffers as the input. For each point-feature, the $[x,y]$ coordinates and the label string are provided. The *obstacles buffer* is a buffer that contains a mask of the important visual features that the labels cannot overlap. Essentially, the pixels of the *obstacles buffer* that the labels cannot overlap have the value 1 while the remaining pixels have the value 0. The *color buffer* contains the image that will be overlaid with the labels.

The output of the algorithm is the *color buffer* overlaid with the positioned labels. Alternatively, the output of the algorithm can be the *labels buffer*, a buffer of the width equal to the number of point-features and of the height 1 containing the coordinates and dimensions of the positioned labels. The x coordinate in this buffer corresponds to the index of the point-feature in the input list of point-features.

The algorithm uses other buffers internally. The *conflict buffer* is used to evaluate conflicts between the label candidates of different point-features. The *ambiguity buffer* is used for the evaluation of ambiguity of the label candidates. The *label obstacles buffer* is used to evaluate overlaps of label candidates with already positioned labels. All these buffers have the same size as the input *obstacles buffer*. In fact, we combine the *obstacles buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer* into the *evaluation buffer*. The *evaluation buffer* is a buffer with four channels (similar to an RGBA image), where the *obstacles buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer* are each represented as one channel. Further, the algorithm uses the *block buffer* that represents the blocks of the grid. The size of the buffer corresponds to the size of the grid of blocks, i.e., there is one pixel of the *block buffer* for each block of the grid. Finally, the algorithm uses the *priority buffer* of the same size as the *block buffer*. The *priority buffer* is used to find the point-feature with the highest priority for each block of the grid.

Below, we describe the overview of the proposed algorithm. We explain the details in the following sections. We indicate the corresponding sections in the overview. Figure 3 depicts how the *evaluation buffer* will be modified during the first four iterations of the algorithm for a simple line chart with four point-features. For a graphical version of the overview with all used buffers, please see Figure 4.

1. Establish the set of quadrant ids $Q = \{0,1,2,3\}$ and set the quadrant id $q = 0$.
2. Create *evaluation buffer*, see Section 3.3 for details.
3. Create *priority buffer*, *block buffer*, and *labels buffer*.
4. While the set of quadrant ids Q is not empty:
   (a) Set quadrant id $q = (q+1)\%4$.
   (b) If the quadrant id $q \notin Q$ then go to Step 4.
   (c) Set the value of each pixel in the *priority buffer* to 0. Clear each pixel in the *block buffer* to contain no label and point-feature.
   (d) Do in parallel for each unlabeled point-feature f in the quadrant with id q:
      i. Determine the coordinates $[i,j]$ of the block b_{ij} which contains the point-feature f.
      ii. Determine the label L and the priority p of the point-feature f using the *evaluation buffer*, see Section 3.4 for details. If the point-feature f cannot be labeled, then the processing of the point-feature f ends.
      iii. Atomic operation: If the priority p is greater than the priority at position $[i,j]$ in the *priority buffer*, then set priority at position $[i,j]$ in the *priority buffer* to p and store the position and dimensions of the label candidate l and position and id of the point-feature f in the *block buffer* at position $[i,j]$.
   (e) Do in parallel for each block b_{ij}:
      i. If a point-feature in block b_{ij} was labeled, then update the *evaluation buffer*, see Section 3.5 for details.
   (f) If no point-feature was labeled for quadrant id q, then remove the quadrant id q from the set of quadrant ids Q.
   (g) Otherwise, do in parallel for each block b_{ij}:

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

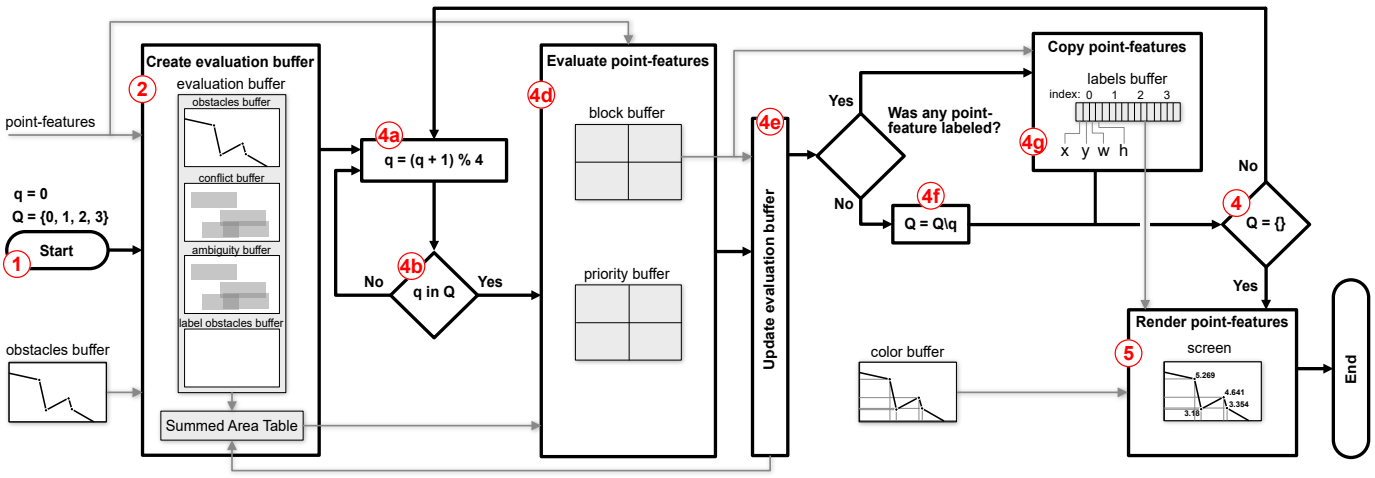


Fig. 4. Graphical overview of the proposed algorithm. Black nodes and arrows represent the state diagram of the algorithm. Inside each node, we depict the created buffers. Gray arrows incoming to a black node indicate that the node is reading from the buffer. A gray arrow outgoing from a black node indicates that the node is writing into the buffer. In the red circles, we provide the numbering of the steps in the description of the algorithm.

i. If a point-feature in block b_{ij} was labeled, then copy the position and dimensions of the chosen label into the *labels buffer* at the position given by the id of the point-feature.
ii. Mark the point-feature as labeled.

5. Do in parallel for each point-feature:

   (a) Draw the label on the screen.

The atomic operation used in the algorithm means that the whole step is executed at once for each point-feature without interruption.

3.3 Creating the Evaluation Buffer

We create the *evaluation buffer* by combining the *obstacles buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer* into a single buffer similarly to how the RGBA image combines red, green, blue, and alpha channels. We will express the value of a pixel of the *evaluation buffer* as $\mathbf{e} = [e_o, e_c, e_a, e_l]$, where the e_o, e_c, e_a, and e_l are values of the corresponding pixel in the *obstacles buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer*, respectively.

Now let us describe how we create the individual buffers. The *obstacles buffer* is the input of the algorithm, and the remaining buffers are created with the same size. The value of each pixel in the *conflict buffer* is set to 0. For each point-feature, we render its conflict rectangle and add 1 to each pixel in the *conflict buffer* inside the conflict rectangle. After this, each pixel of the *conflict buffer* contains the number of conflict rectangles that it is inside of. The *ambiguity buffer* is created exactly the same as the *conflict buffer*. This is because the content of the *conflict buffer* is changing during the labeling, while the content of the *ambiguity buffer* remains the same. Please see Figure 3 for an example. Lastly, the value of each pixel in the *label obstacles buffer* is set to 0.

After we create the *evaluation buffer*, we calculate the Summed Area Table [5] of all blocks b_{ij} in the buffer in parallel. Calculating the Summed Area Table for blocks b_{ij} instead of the whole evaluation buffer makes the calculation dependent on the size of the blocks which is smaller than the size of the whole buffer. Further, it allows us to update the Summed Area Table more efficiently. We describe the update process in detail later in Section 3.5. To calculate the Summed Area Table, we use the method of Hensley et al. [10] tailored for the parallel calculation on GPU.

In the Summed Area Table, each pixel p of the buffer inside of block b_{ij} contains sum $\mathbf{S} = [S_o, S_c, S_a, S_l]$ of the values of pixels in the rectangle with the bottom left corner at the bottom left corner of the block and top right corner at the position of pixel p. Again, the values S_o, S_c, S_a, and S_l are sums of values of pixels in the corresponding rectangle in the *obstacle buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer*.

Summed Area Table allows to obtain the sum of values in a given rectangle in the *evaluation buffer* in constant time. In our case, the maximum size of the given rectangle is equal to the size of one quadrant of the block. In other words, the given rectangle can be inside of a single block, overlap two blocks horizontally, overlap two blocks vertically, or overlap four blocks of the grid. In the following text, we show that in the worst case we need nine reads from the Summed Area Table of the evaluation buffer to obtain the sum of values in the given rectangle.

If the given rectangle is inside of a single block, we can obtain the sum of values in the rectangle with reading values in the four corners of the rectangle. In Figure 5(a), we depict the rectangle together with the bottom and left border of the block. To obtain the sum S of values in the rectangle, we calculate

$$\mathbf{S} = \mathbf{S}_3 - \mathbf{S}_1 - (\mathbf{S}_2 - \mathbf{S}_0) = \mathbf{S}_0 - \mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3, \tag{1}$$

where $\mathbf{S}_0$, $\mathbf{S}_1$, $\mathbf{S}_2$, and $\mathbf{S}_3$ are sums in the Summed Area Table in the corners of the given rectangle depicted in Figure 6.

In the case that the given rectangle overlaps two blocks vertically, we need six reads from the Summed Area Table, see Figure 5(b). For the part of the rectangle in the bottom block, we can use Equation 1. For the part of the rectangle in the top block, we can calculate the sum by subtracting the sum $\mathbf{S}_4$ from the sum $\mathbf{S}_5$. This gives us the equation

$$\mathbf{S} = \mathbf{S}_0 - \mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 - \mathbf{S}_4 + \mathbf{S}_5. \tag{2}$$

Similarly, in the case that the given rectangle overlaps two blocks horizontally, we need six reads from the Summed Area Table, see Figure 5(c). For the part of the rectangle in the left block, we can use Equation 1. For the part of the rectangle in the right block, we can calculate the sum by subtracting the sum $\mathbf{S}_6$ from the sum $\mathbf{S}_7$. This gives us the equation

$$\mathbf{S} = \mathbf{S}_0 - \mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 - \mathbf{S}_6 + \mathbf{S}_7. \tag{3}$$

Finally, in the case that the rectangle overlaps four blocks of the grid, we need nine reads from the Summed Area Table, see Figure 5(d). For the part of the rectangle in the bottom left block, we can use Equation 1. For the part of the rectangle in the top left block, we need to subtract sum $\mathbf{S}_4$ from sum $\mathbf{S}_5$. For part of the rectangle in the bottom right block, we need to subtract sum $\mathbf{S}_6$ from sum $\mathbf{S}_7$. And for the part of the rectangle in the top right block, the sum equals to sum $\mathbf{S}_8$. Adding the values for all parts of the rectangle gives us the equation

$$\mathbf{S} = \mathbf{S}_0 - \mathbf{S}_1 - \mathbf{S}_2 + \mathbf{S}_3 - \mathbf{S}_4 + \mathbf{S}_5 - \mathbf{S}_6 + \mathbf{S}_7 + \mathbf{S}_8. \tag{4}$$

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

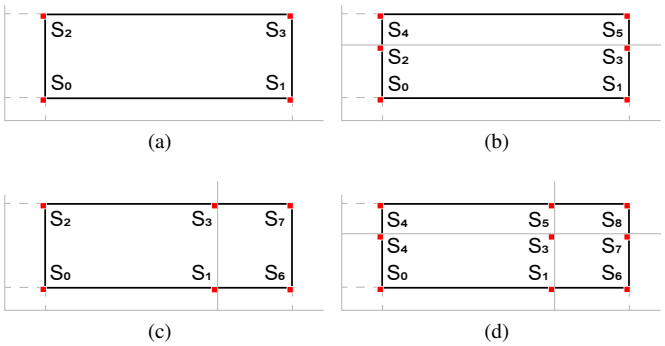


Fig. 5. The pixels (highlighted in red color) of the Summed Area Table from which we need to read in order to calculate the sum of values in the rectangle contained in a single block (a), overlapping two blocks vertically (b), overlapping two blocks horizontally (a), and overlapping four blocks (d).

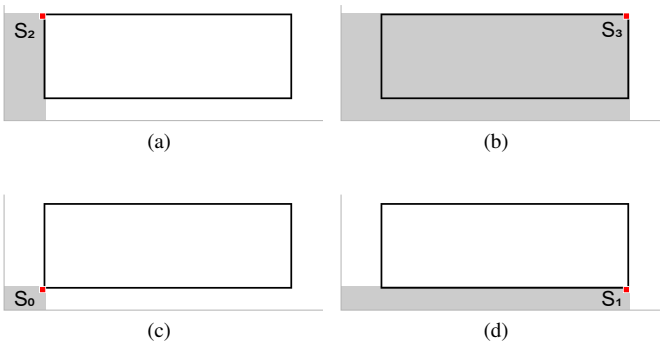


Fig. 6. The sum of values (indicated with gray color) that is stored in a pixel (highlighted in red color) of the Summed Area Table. We depict the sum of values for the four pixels in the corners of a rectangle.

3.4 Evaluating the Point-Features

This section describes how we determine the label L and the priority p of a point-feature. The evaluation is based on the Summed Area Table of the *evaluation buffer*. To simplify the following description, we will not distinguish between label candidates completely inside in a single block, label candidates overlapping two blocks vertically, label candidates overlapping two blocks horizontally, and label candidates overlapping four blocks. Instead, let us consider that the correct formula from the previous section is used to obtain the sum of values inside of the label candidate in the evaluation buffer.

In the following text, we describe an algorithm that determines label L of a given point-feature from the label candidates of the point-feature and assigns priority p to the point-feature. If the point-feature cannot be labeled then, the label L is not set. In the algorithm, the label candidates of the point-feature are evaluated sequentially in ascending order according to their preference.

To evaluate a label candidate l, we obtain the sum $\mathbf{S} = [S_o, S_c, S_a, S_l]$ inside of the label candidate l from the Summed Area Table of the *evaluation buffer*. First, we test that the candidate does not overlap important visual features encoded in the *obstacles buffer* or already positioned labels encoded in the *label obstacles buffer*. If that is the case then $S_o + S_l = 0$.

For the satisfactory label candidate l, we evaluate the conflicts of the label candidate l as a ratio of the area of the label A to the sum of conflicts S_c inside of the label candidate. Please note that $A/S_c = 1$ only if the label candidate does not overlap the conflict rectangle of any other unlabeled point-feature (i.e., the label candidate overlaps only the conflict rectangle of its point-feature). If the label candidate overlaps conflict rectangles of other unlabeled point-features then $A/S_c < 1$.

If there are multiple label candidates of a point-feature with the same ratio A/S_c (this will happen most often when point-feature has several label candidates without conflicts), then we prefer the label candidate with the lowest ambiguity. We evaluate the ambiguity of label candidate l as a ratio of the area of the label A to the sum of ambiguity S_a inside of the label candidate. The sum of ambiguity is similar to the sum of conflicts, but also considers the conflict rectangles of already labeled point-features. Please note that the higher the ratio, the lower the ambiguity. Again, $A/S_a = 1$ only if the label candidate does not overlap the conflict rectangle of any other labeled or unlabeled point-feature. A label candidate with such a ratio is not ambiguous because it is not located in the proximity of any other point-feature. The lower the ratio A/S_a, the closer is the label candidate to other point-features. In Figure 3, we have depicted two label candidates with dashed lines in the *conflict buffer* and the *ambiguity buffer* in the algorithm's third iteration. The evaluation of the conflicts alone will not allow to prefer one of the candidates while the evaluation of ambiguity will. Choosing the label candidate further away from other point-features will make the label point-feature association easier. However, the evaluation of ambiguity usually will not improve the label layout in areas with densely packed point-features, as such point-features typically will not have multiple label candidates with the same ratio A/S_c.

If there are multiple label candidates of a point-feature with the same ratio A/S_c and with the same ratio A/S_a, then we prefer the label candidate with the highest preference.

Further, we use two rules of Wagner et al. [30] to simplify the dependencies between the point-features. We have extended the second rule to consider the ambiguity of the label candidates. (1) If a point-feature has only one available label candidate left, then we use the candidate, and the point-feature is assigned the highest priority possible. (2) If a point-feature has label candidates that are not in conflict with any label candidate of any other point-feature, then we use the least ambiguous of the label candidates, and the point-feature is assigned second highest priority.

The first rule will have the following effect. In the processed quadrant of blocks, we will always label a point-feature f with only one available label candidate first. It is important to label the point-feature f as soon as possible because the space needed for the last available label candidate could be used by a label of another point-feature, and then we would not be able to label the point-feature f. The second rule allows to simplify the evaluation of conflicts between the label candidates. Please note that after a point-feature is labeled, its conflict rectangle is removed from the channel of the *conflict buffer* in the Summed Area Table of the *evaluation buffer* which increases the chance of finding new point-features with label candidates that are not in conflicts in the next iteration.

The complete algorithm is as follows:

1. Set the number of available candidates $N = 8$.
2. Set best conflicts $c = 0$.
3. Set best ambiguity $a = 0$.
4. For each label candidate l:
   (a) Calculate area A of the label candidate.
   (b) Obtain the sum $\mathbf{S} = [S_o, S_c, S_a, S_l]$ inside of the label candidate l from the Summed Area Table of the *evaluation buffer*.
   (c) If $S_o + S_l = 0$:
      i. If $A/S_c > c$ or ($A/S_c = c$ and $A/S_a > a$): Set the label $L = l$, set the priority $p = 0.9 \cdot A/S_c$, set best conflicts $c = A/S_c$, and set the best ambiguity $a = A/S_a$.

      Otherwise: $N = N - 1$
   (d) If $N = 1$: Set the priority $p = 1$.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

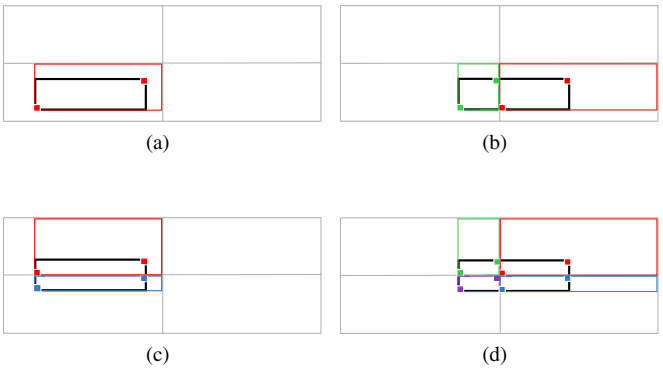


Fig. 7. The update rectangles (in red, green, blue, and purple color) with highlighted pixels c_0 and c_1 for a conflict rectangle inside a single block (a), overlapping two blocks horizontally (b), overlapping two blocks vertically (c), and overlapping four blocks (d).

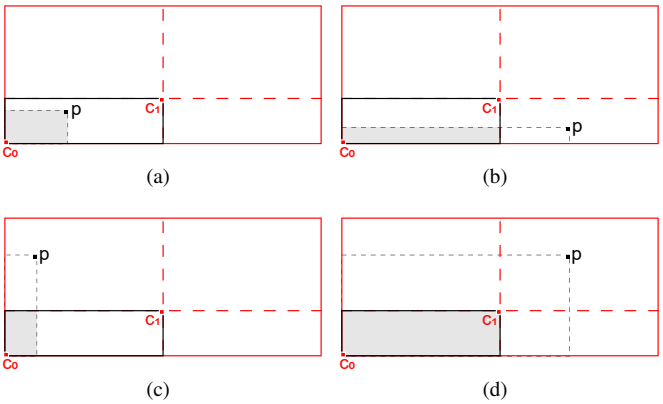


Fig. 8. The sum of values (indicated with gray color) that is subtracted from pixel p in the update rectangle (red rectangle) when we are removing the conflict rectangle (black rectangle) of a labeled point-feature from the Summed Area Table. We depict the sum pixel p in four quadrants of the update rectangle.

Please note that in Step 4(c)i, we use the multiplication factor 0.9 to calculate the priority p to make sure that the priority of the point-feature with the last available label candidate (Step 4d) is the highest. If there is no point-feature with the last available label candidate, then the multiplication will not have any effect on the resulting label layout.

3.5 Updating the Evaluation Buffer

In each iteration, after we have positioned a label for one point-feature in certain blocks, we need to update the information in the Summed Area Table of the *evaluation buffer*. One option is to remove the conflict rectangle of each labeled point-feature from the conflict buffer, add each positioned label into the *label obstacles buffer*, and calculate the Summed Area Table of the evaluation buffer again. Unfortunately, this approach leads to poor performance. Therefore, we update the Summed Area Table of the evaluation buffer directly.

To remove a conflict rectangle of a labeled point-feature from the *conflict buffer*, we need to subtract 1 from each pixel in the *conflict buffer* inside the conflict rectangle of the labeled point-feature. Based on this, we can update the Summed Area Table of the *evaluation buffer* directly.

First, let us consider a conflict rectangle that is completely inside of a single block. For such a conflict rectangle, we need to update the rectangle from the lower-left corner of the conflict rectangle to the top right corner of the block, see Figure 7(a) for an example. We will denote the rectangle as update rectangle. For the sum S_c in each pixel p in the update rectangle, we need to subtract the number of pixels C that are inside of the intersection of the positioned label and the rectangle with lower-left corner c_0 and top right corner p. In Figure 8, we depict the intersection for four pixels in various quadrants of the update rectangle. To calculate the number of pixels C for each pixel p, we use the equation

$$C = (\lceil \min(x_p, x_{c_1}) \rceil - \lfloor x_{c_0} \rfloor) \cdot (\lceil \min(y_p, y_{c_1}) \rceil - \lfloor y_{c_0} \rfloor). \quad (5)$$

If we consider that the conflict rectangle overlaps two or four blocks, see Figures 7(c), 7(b), and 7(d), all we need to do is to determine the update rectangle for each block, and for each update rectangle determine the positions of pixels c_0 and c_1. We depict the positions for each update rectangle in Figure 7. Then, we can use Equation 5 to update all pixels in each update rectangle.

Similarly, to update the *obstacles buffer*, we need to add 1 to each pixel in the *obstacles buffer* inside the label of the labeled point-feature. Based on this, we can again update the Summed Area Table of the *evaluation buffer* directly. The approach is the same as for the conflict rectangle, but instead of the conflict rectangle, we use the positioned label. Further, instead of subtracting from the S_c in each pixel, we are adding to S_l.

Now let us discuss why we are calculating the Summed Area Table of the *evaluation buffer* for the individual blocks instead of for the whole buffer. Let us consider that the Summed Area Table was calculated for the whole buffer. In consequence, the update rectangles will be much larger. Each rectangle will be from the lower-left corner of the conflict rectangle (or the label) of the positioned point-feature to the top-right corner of the buffer. Therefore, we need to update many more pixels of the buffer. As a GPU has a fixed number of shader units, updating more pixels will result in sequential processing of several pixels by each shader unit, and thus the performance will decrease dramatically.

3.6 Supporting Priority Groups of Point-Features

In many real-world situations, we have several groups of point-features with various priorities (e.g., capital cities and regular cities). To achieve this with the proposed method, we establish a list of the groups where the groups are sorted according to their priority in descending order. Then, we sequentially run the algorithm for each group. We recommend using the same grid of blocks for all runs of the algorithm.

For each run of the algorithm, we need to set up the *obstacles buffer*, *conflict buffer*, *ambiguity buffer*, and *label obstacles buffer* correctly. The *conflict buffer* should contain conflict rectangles of point-features that are only in the processed group. On the other hand, the *ambiguity buffer* should contain conflict rectangles of all point-features. This way, the algorithm will evaluate the ambiguity of label candidates with respect to all point-features. The *label obstacles buffer* should contain the labels positioned by all preceding runs of the algorithm. Finally, the *obstacles buffer* can be the same for all runs of the algorithm. Please note that by providing different *obstacles buffer* for each group of point-features, we can control which important visual features will not be occluded by the labels of the point-features in the group. For example, the labels of capital cities cannot overlap other capital cities, but can overlap the regular cities and the regular cities cannot overlap both capital cities and other regular cities.

3.7 Supporting Zoom and Pan

In interactive scenarios, users typically use zoom and pan to navigate in the environment (e.g. in a 2D map). During zooming and panning, the labeling of the point-features needs to be consistent. Been et al. [2] provide four rules for consistent point-feature labeling: (R1) labels are not vanishing when zooming in or appearing when zooming out, (R2) position and size of a label is changing continuously under the pan and zoom operations, (R3) labels are not vanishing or appearing during panning, and (R4) the placement of any label is a function of the current zoom and pan state (i.e., it is not influenced by previous states).

In order to support the zoom and pan scenario and fulfill all the rules for consistent labeling, we need to sequentially calculate the label layout of point-features at increasing discrete zoom levels. Further, for each point-feature we need to store the zoom level z_l at which its label

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

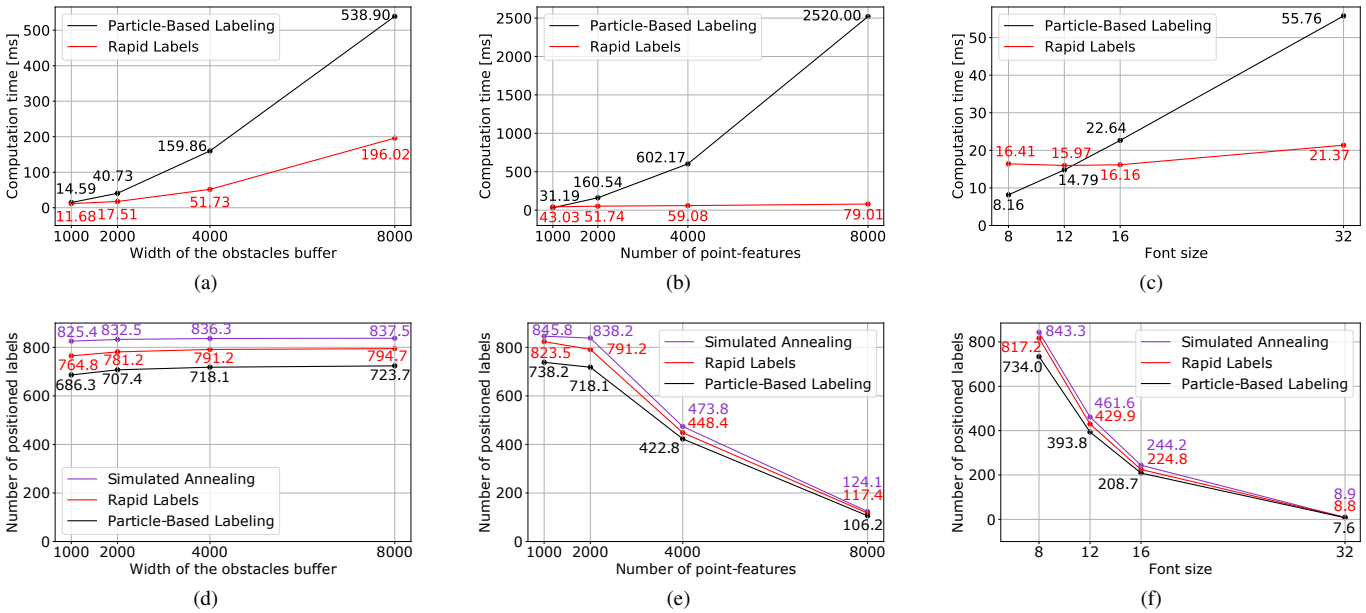


Fig. 9. (top) The measured computation time in dependence on the resolution of the *obstacles buffer* (a), the number of point-features in the dataset (b), and the font size (c). (bottom) The number of positioned labels in dependence on the resolution of the *obstacles buffer* (d), the number of point-features in the dataset (e), and the font size (f). In charts (a) and (d), we show on the x axis only the width of the *obstacles buffer* as the size of the buffer was always in the $4:3$ ratio.

was positioned. Please note that during the label layout calculation, we do not scale up the environment. Instead, with the increasing zoom level, we scale down the labels with $1/z_l$.

We start with zoom level $z_l = 1$ and subsequently multiply the zoom level z_l with zoom factor z_f until the desired maximum zoom level is reached. In our case, we are using zoom factor $z_f = 1.05$. Before we calculate the label layout for zoom level $z_l > 1$, we render the labels of point-features labeled on lower zoom levels to the *label obstacles buffer* and calculate label positions only for the remaining point-features.

Finally, during the rendering of the labels, we render only the labels with zoom level $z_l \leq z_r$, where z_r is the rendering zoom level.

4 Results and Discussion

In this section, we present the results of the proposed method. Further, we compare the performance and the quality of the resulting label layouts of the proposed method with existing methods. To assess the quality of the resulting label layouts, we use the number of labeled point-features. For the comparison, we have used the implementation of the proposed method in Java and OpenGL.

We compare the proposed method with the available Java implementation [15] of Particle-based labeling [16]. For one dataset, we compare the proposed method also with the improved Particle-based labeling by Kittivorawong et al. [12]. The comparison is made based on their reporting of performance improvement over Particle-based labeling for the particular dataset. To assess the impact of the greedy approach used in the proposed method on the number of labeled point-features, we compare the proposed method with the modified meta-heuristic method of Zoraster [33] based on simulated annealing. We have modified the method to consider the *obstacles buffer* and eliminate label candidates that overlap the important visual features.

Please note that comparing the performance of the proposed method with existing methods is complicated as the proposed method is running on GPU while the existing methods are running on CPU. We have approached the comparison from the point of the end-user. Therefore, we have measured the performance of all methods on the same computer equipped with Intel Xeon E3-1275 V2 running at 3.5 GHz, 16 GB of RAM, and NVIDIA GeForce GTX 1660 Ti with 6 GB of RAM and 1536 unified shaders.

For the comparison, we also consider the visual representation of the point-features. In other words, each point-feature can be visually represented as a small circle, square, or any other shape. This is usual in visualization, geographic maps, geographic information systems, and many other areas. Please note that while we are rendering the point-features as small circles (or any other shape), we are still labeling them as points. To ensure that the labels will not overlap the visual representations of the point-features, the visual representations need to be treated as important visual features that the labels cannot overlap. Therefore, we provide the same *obstacles buffer* as an input to all compared methods.

First, we have measured the time needed to calculate the label layout and the number of positioned labels in dependency on the number of labeled point-features, on the precision with which we evaluate the overlaps with important visual features (e.g., the point-features rendered as small circles) and already placed labels, and on the size of the font used for the labels. Please note that in all compared methods, the precision depends on the resolution of the input *obstacles buffer*.

To measure the dependencies, we rendered the given number of pseudo-randomly distributed point-features into the *obstacles buffer* of the given resolution, calculated the sizes of the labels based on the given font, and provided the point-features together with the label sizes and the *obstacles buffer* as the input of all compared methods.

For each given number of point-features, we repeated the measurement 100 times, that is 10 times for each of the 10 variants of distributed point-features. From these measurements, we calculated the average computation time and the average number of positioned labels that we report. In the supplemental material, we depict a cut-out from one of the ten random datasets used for the performance test.

We depict the computation times of the methods and the number of positioned labels in dependency on the resolution of the input *obstacles buffer* in Figures 9(a) and 9(d). The number of point-features was fixed at 2000. For the *obstacles buffer* resolution 1000×750, the font size was 8 pt. The font size was increasing with the increasing resolution of the *obstacles buffer*. This way, the same configuration of point-features and label candidates was evaluated but with increasing precision. Our measurement indicates that the computation time of the proposed method increases with the resolution of the *obstacles buffer* more slowly than the computation time of Particle-based labeling. For

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

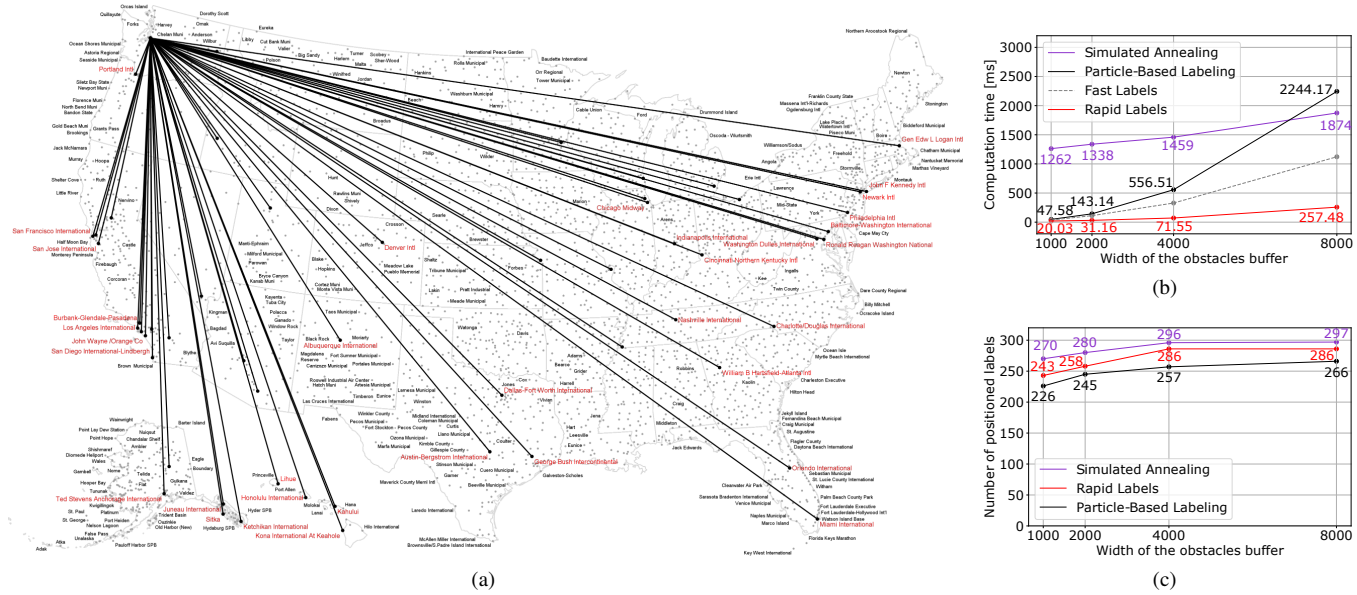


Fig. 10. (a) The US airports dataset labeled with the proposed method. (b) The measured computation time in dependence on the resolution of the *obstacles buffer*. (c) The number of positioned labels in dependence on the resolution of the *obstacles buffer*.

the highest measured resolutions of the *obstacles buffer*, the proposed method needed 36% of the computation time of Particle-based labeling. We do not depict the computation time for the method of Zoraster as it required from 1 s for the resolution 1000×750 to 1.5 s for the resolution 8000×6000. Further, the proposed method positioned labels for more point-features than Particle-based labeling, but less than the method of Zoraster. Please note that the slight increase in the number of positioned labels with the increasing resolution of the *obstacles buffer* for all methods is due to the increasing precision of the evaluation.

In Figures 9(b) and 9(e), we show the computation times of the methods and the number of positioned labels in dependency on the number of point-features. For this measurement, the resolution of the *obstacles buffer* was fixed at 4000×3000 and the font size was fixed at 8pt. The measurement indicates that while Particle-based labeling is faster for a lower number of point-features ($n = 1000$), the computation time of the proposed method increases with the number of point-features much more slowly than the computation time of Particle-based labeling. For the highest measured number of point-features, the proposed method needed only 3% of the computation time of Particle-based labeling. Particle-based labeling was faster for a lower number of point-features ($n = 1000$) due to the cost associated with the Summed Area Table calculation that is resolution-dependent. For higher numbers of point-features, the speed up in evaluation of the label candidates outweighs this cost. Again, we do not depict the computation time for the method of Zoraster as it required 0.9 s for 1000 point-features and 4.7 s for 8000 point-features. The proposed method positioned more labels in dependency on the number of point-features than Particle-based labeling, but less than the method of Zoraster. The reason for the decreasing number of positioned labels with the increasing number of point-features is less free space for the labels with the increasing number of point-features.

We depict the computation times of the methods and the number of positioned labels in dependency on the font size in Figures 9(c) and 9(f). For this measurement, the resolution of the *obstacles buffer* was fixed at 2000×1500 and the number of point-features was fixed at 1000. The measurement indicates that the proposed method is slightly font-size-dependent. This comes from the fact that the number and dimensions of blocks are dependent on the font size. Particle-based labeling was faster for small fonts, but the computation time of the proposed method increased with the font size much more slowly than the computation time of Particle-based labeling. For the highest measured font size, the proposed method needed 38% of the computation time of Particle-based labeling. Again, we do not depict the computation time for the method of Zoraster as it required 0.8 s for the font size of 8 pt and 0.7 s for the font size of 32 pt. As expected, the number of positioned labels decreased with the increasing font size, as every label takes up more space. Still, the proposed method was able to position more labels than Particle-based labeling, but less than the method of Zoraster.

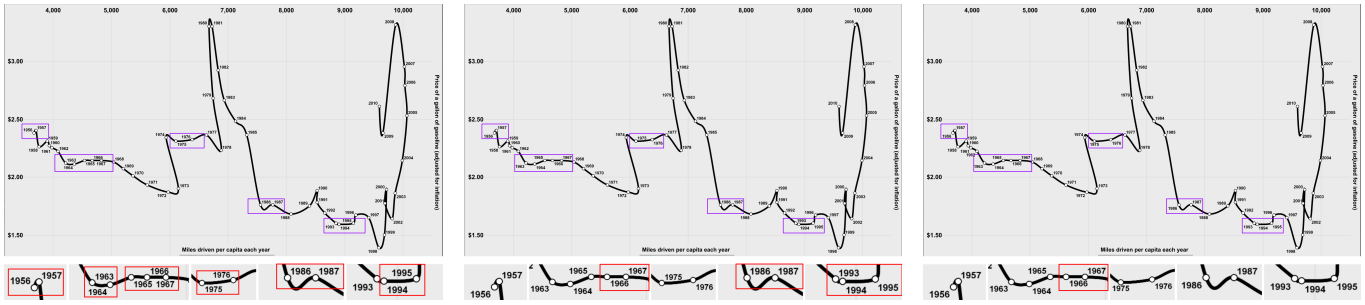
From the measurements, we can see that the performance of the proposed method decreases the most with the increasing resolution of the *obstacles buffer*. This is due to the calculation of the Summed Area Table, and its update as the performance of these steps decreases with increasing resolution of the *obstacles buffer*. For more details, please see the supplementary material.

Next, we have measured the computation time and the number of positioned labels on a real-world dataset of 3340 airports in the USA [27]. For this dataset, we measured the computation time and the number of positioned labels in dependency on the resolution of the input *obstacles buffer*. Again the font size was decreasing or increasing with the resolution of the *obstacles buffer* to preserve the configuration of point-features and label candidates, but increase the precision of the evaluation. For the measurement, we have divided the input point-features into two priority groups. In the first priority group were 56 airports with direct flights to Seattle-Tacoma airport. The remaining airports were in the second priority groups. The point-features in the first group were labeled with red font of size 26 pt. The point-features in the second group were labeled with black font of size 19 pt. These font sizes were used for the *obstacles buffer* resolution 4000×3000. Please see Figure 10(a) for the result of the proposed method. The labels of the point-features in the first group (in red color) were not allowed to overlap the connecting lines and visual representations of the point-features in the first group. The labels of the point-features in the second group (in black color) were not allowed to overlap the connecting lines, visual representations of the point-features in both groups, and the boundaries of US states.

We depict the performance of the methods and the number of positioned labels in dependency on the resolution of the input *obstacles buffer* in Figures 10(b) and 10(c). Our measurement indicates again that the computation time of the proposed method increases with the resolution of the *obstacles buffer* much more slowly than the computation time of Particle-based labeling. For the highest resolutions of the *obstacles buffer*, the proposed method needed 11% of the computation

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854



(a) (b) (c)

Fig. 11. Connected scatter-plot dataset labeled with the proposed method. To evaluate the label candidates, we have used the overlaps and preference of the label candidates (a), the overlaps, conflicts, and preference of the label candidates (b), and the overlaps, conflicts, ambiguity, and preference of the label candidates (c).

time of Particle-based labeling. Interestingly, the method of Zoraster was faster than Particle-based Labeling for the highest resolution. This was due to the extreme number of label candidates removed based on the *obstacles buffer*. The proposed method positioned labels for more point-features than Particle-based labeling, but less than the method of Zoraster. The slight increase in the number of positioned labels with the increasing resolution of the *obstacles buffer* for all methods is again due to the increasing precision of the evaluation.

In Figure 10(b), we also indicate the performance of the Fast Labels method by Kittivorawong et al. [12] based on their reporting of a performance increase for the same dataset. The computation time of the proposed method increased with the resolution of the *obstacles buffer* much more slowly than the computation time of the Fast Labels method. Kittivorawong et al. [12] report that the Fast Labels method positions labels of slightly fewer point-features than Particle-based labeling. Therefore, we can expect that the proposed method will position labels of more point-features than the Fast Labels method.

Further, we used the proposed method to support zoom and pan for the US airports dataset and the GapMinder dataset [7]. The proposed method calculated label positions for all 62 zoom levels between 1 and 20 (zoom factor $z_f = 1.05$) in 1644 ms for the US airports dataset and in 756 ms for the GapMinder dataset for the year 2008. Please see the supplementary material for images of the results and the supplementary video for the live capture of interaction with the results.

In Figure 11, we demonstrate the effect of the evaluation of conflicts between the label candidates and the evaluation of ambiguity of the label candidates on the connected scatter-plot dataset [26]. In Figure 11(a), the labels were positioned without evaluating conflicts and ambiguity, the label candidates were evaluated based on overlaps with important visual features and already positioned labels and their preference. In the figure, we have magnified five areas (purple boxes) where the label layout can be improved (red boxes). In Figure 11(b), the labels were positioned based on overlaps, conflicts, and preference. The evaluation of conflicts between the label candidates improved the label layout in two of the five areas. In Figure 11(c), the labels were positioned based on the overlaps, conflicts, ambiguity, and preference. The evaluation of ambiguity of the label candidates further improved the label layout in two more areas. In the figure, we have highlighted with a red box the single problematic part of the label layout. As it can be seen, the evaluation of conflicts and ambiguity improves the label layout. Please note that the single problematic label for the year 1966 was not centered below the point-feature due to the overlapping conflict rectangles of the previous and next point-features in that region. In consequence, the algorithm considers such positions as more ambiguous.

To assess the impact of the evaluation of ambiguity on areas with densely positioned point-features, we labeled the US airports dataset with and without the evaluation of ambiguity. Please see the supplementary material for the results. The evaluation of ambiguity had a small effect on the resulting layout. In both cases, the proposed method positioned labels for the same 939 point-features, and only 13 labels were positioned at different positions.

4.1 Limitations

In this section, we discuss the limitations of the proposed method. Foremost, it is a greedy algorithm that cannot recover from a local minimum/maximum. Therefore, we cannot guarantee that the algorithm finds the optimal solution nor that the found solution will be close to the optimal solution.

We are using the grid of blocks to label multiple point-features in each iteration. However, we cannot guarantee that a point-feature with a low priority from the processed quadrant will not use the space of a point-feature with a higher priority in the remaining quadrants. In consequence, this may lead to a lower number of labeled point-features. Fortunately, the evaluation of the conflicts and the ambiguity for each label candidate helps to decrease the problem.

The proposed method works well only with rectangular labels. Labels of different shapes (e.g., circles) need to be enclosed by rectangles which may lead to inefficient utilization of the space available for labels.

If the difference in the size between the smallest label and the largest label is large, then the used blocks may contain many small labels, which will lead to a higher number of iterations of the algorithm and to a decrease in performance.

5 Conclusions and Future Work

In this work, we propose a screen space greedy method to solve the point-feature labeling problem. In contrast to other existing methods, the proposed method positions labels of several point-features in parallel based on a grid of blocks. When determining the position of the label of a point-feature, the proposed method evaluates overlaps of the label candidates with important visual features and already positioned labels, conflicts of the label candidates with label candidates of other point-features, and ambiguity of the label candidates.

We have demonstrated that the proposed method supports point-features divided into several priority groups, that the proposed method can be utilized in the zoom and pan scenario, and that the evaluation of conflicts and ambiguity of the label candidates improves the label layout. Further, we have compared the proposed method with Particle-based labeling of Luboschik et al. [16] and with the modified meta-heuristic method of Zoraster [33]. Our measurements indicate that for the number of point-features greater than 1000 and the font size greater than 10 pt, the proposed method achieves significantly lower computation times while positioning labels for more point-features than Particle-based labeling, but fewer point-features than the modified meta-heuristic method of Zoraster.

Acknowledgments

This research has been supported by MEYS of Czechia OP VVV grant No. CZ.02.1.01/0.0/0.0/16_019/0000765 – Research Center for Informatics.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Visualization and Computer Graphics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TVCG.2021.3114854

REFERENCES

[1] A. C. Alvim and É. D. Taillard. Popmusic for the point feature label placement problem. *European Journal of Operational Research*, 192(2):396–413, 2009.

[2] K. Been, E. Daiches, and C. Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006.

[3] B. Chazelle and N. Amenta. Application challenges to computational geometry. Technical Report TR-521-96, Computational Geometry Impact Task Force, 1999.

[4] L. Čmolík, V. Pavlovec, H. Y. Wu, and M. N ollenburg. Mixed labeling: Integrating internal and external labels. *IEEE Transactions on Visualization and Computer Graphics*, Early access, 2020. doi: 10.1109/TVCG.2020.3027368

[5] F. C. Crow. Summed-area tables for texture mapping. *SIGGRAPH Comput. Graph.*, 18(3):207–212, Jan. 1984. doi: 10.1145/964965.808600

[6] M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, SCG '91, p. 281–288. ACM, New York, NY, USA, 1991. doi: 10.1145/109648.109680

[7] Gapminder Foundation. `https://www.gapminder.org/data/`. Accessed 2021-06-20.

[8] S. P. Gomes, L. A. N. Lorena, and G. M. Ribeiro. A constructive genetic algorithm for discrete dispersion on point feature cartographic label placement problems. *Geographical Analysis*, 48(1):43–58, 2016.

[9] J.-H. Haunert and A. Wolff. Beyond maximum independent set: An extended model for point-feature label placement. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 41, 2016.

[10] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24(3):547–555, 2005. doi: 10.1111/j.1467-8659.2005.00880.x

[11] C. Iturriaga and A. Lubiw. Np-hardness of some map labeling problems. Technical Report CS-97-18, University of Waterloo, Waterloo, ON, Canada, 1997.

[12] C. Kittivorawong, D. Moritz, K. Wongsuphasawat, and J. Heer. Fast and flexible overlap detection for chart labeling with occupancy bitmap. In *2020 IEEE Visualization Conference (VIS)*, pp. 101–105, 2020. doi: 10.1109/VIS47514.2020.00027

[13] A. Lhuillier, M. van Garderen, and D. Weiskopf. Density-based label placement. *The Visual Computer*, 35(6):1041–1052, 2019.

[14] L. Li, H. Zhang, H. Zhu, X. Kuai, and W. Hu. A labeling model based on the region of movability for point-feature label placement. *ISPRS International Journal of Geo-Information*, 5(9):159, 2016.

[15] M. Luboschik. Java implementatin of particle-based labeling. `https://sourceforge.net/projects/fpf-labeling/`. Accessed: 2021-03-31.

[16] M. Luboschik, H. Schumann, and H. Cords. Particle-based labeling: Fast point-feature labeling without obscuring other visual features. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1237–1244, 2008. doi: 10.1109/TVCG.2008.152

[17] A. Marın and M. Pelegrın. Towards unambiguous map labeling-integer programming approach and heuristic algorithm. *Expert Systems with Applications*, 98:221–241, 2018.

[18] J. Marks and S. Shieber. The computational complexity of cartographic label placement. Technical Report TR-05-91, Harvard University, Cambridge, MA, USA, 1991.

[19] G. R. Mauri, G. M. Ribeiro, and L. A. Lorena. A new mathematical model and a lagrangean decomposition for the point-feature cartographic label placement problem. *Computers & Operations Research*, 37(12):2164–2172, 2010.

[20] K. Mote. Fast point-feature label placement for dynamic visualizations. *Information Visualization*, 6(4):249–260, 2007.

[21] I. Petzold, G. Gröger, and L. Plümer. Fast screen map labeling – data-structures and algorithms. In *Proc. 21th Internat. Cartographic Conf.(ICC'03)*, pp. 288–298, 2003.

[22] R. L. Rabello, G. R. Mauri, G. M. Ribeiro, and L. A. N. Lorena. A clustering search metaheuristic for the point-feature cartographic label placement problem. *European Journal of Operational Research*, 234(3):802–808, 2014.

[23] M. Schreyer and G. R. Raidl. Letting ants labeling point features [sic.: for'labeling'read'label']. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, vol. 2, pp. 1564–1569. IEEE, 2002.

[24] B. Shneiderman. Dynamic queries for visual information seeking. *IEEE Software*, 11(6):70–77, 1994. doi: 10.1109/52.329404

[25] T. Strijk, B. Verweij, and K. Aardal. Algorithms for maximum independent set applied to map labelling. Technical Report UU-CS-2000-22, Dept of Computer Science, Utrecht University, 2000.

[26] UW Interactive Data Lab. Connected scatter-plot dataset. `https://vega.github.io/editor/#/examples/vega/connected-scatter-plot`. Accessed: 2021-03-31.

[27] UW Interactive Data Lab. Us airports dataset. `https://vega.github.io/editor/#/examples/vega-lite/geo_rule`. Accessed: 2021-03-31.

[28] S. Van Dijk, D. Thierens, and M. De Berg. Using genetic algorithms for solving hard problems in gis. *GeoInformatica*, 6(4):381–413, 2002.

[29] O. V. Verner, R. L. Wainwright, and D. A. Schoenefeld. Placing text labels on maps and diagrams using genetic algorithms with masking. *INFORMS Journal on Computing*, 9(3):266–275, 1997.

[30] F. Wagner, A. Wolff, V. Kapoor, and T. Strijk. Three rules suffice for good label placement. *Algorithmica*, 30(2):334–349, 2001. doi: 10.1007/s00453-001-0009-7

[31] A. Wolff and T. Strijk. Map labeling bibliography. `http://i11www.iti.uni-karlsruhe.de/map-labeling/bibliography`. Accessed: 2021-03-31.

[32] M. Yamamoto, G. Camara, and L. A. N. Lorena. Tabu search heuristic for point-feature cartographic label placement. *GeoInformatica*, 6(1):77–90, 2002.

[33] S. Zoraster. Practical results using simulated annealing for point feature label placement. *Cartography and Geographic Information Systems*, 24(4):228–238, 1997.

© 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.