Dynamic Label Placement in Practice

Nadine Schwartges

Chair of Computer Science I, University of Würzburg
http://www1.informatik.uni-wuerzburg.de/en/staff

Abstract. We consider the problem of labeling dynamic (3D) maps: we develop real-time algorithms that attach non-overlapping annotations to objects on maps which the user can pan, zoom, and rotate continuously. Existing algorithms either label static maps or only a small number of the objects.
We consider the problem to label streets and to label points. We label streets either internally or externally. For internal labels, we compute potential label positions for each street, compare them, and decide for one. For external street labels, we apply a force-directed approach; labels behave like electrons. The first algorithm for labeling points allows labels to slide along their reference points; it makes use of visibilities between labels. The other approach precomputes for each point an interval of zoom levels at which the label can be placed; labelings are queried at run time.

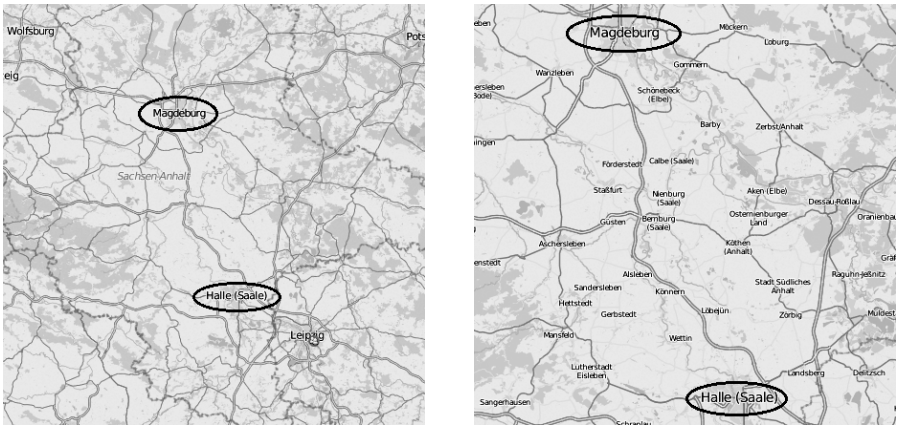
Keywords: Automated map labeling, dynamic maps, interactive maps, sliding labels, street labeling, point labeling, POI labeling, billboards.

1 Introduction

We deal with the problem of *labeling dynamic maps*. In general, a label is an annotation of an object. So, we assign labels to map objects like cities or streets. We are especially interested in labeling objects of dynamic maps. Dynamics maps are maps with which the user can interact; that is, the user can pan, zoom, or rotate the map, for example, by clicking or dragging the mouse. We particularly consider dynamic maps that provide continuous zooming. Some dynamic maps additionally allow for a 3D view.

In order to react to user interactions (and the associated changes of the map) immediately at run time, we need to developed real-time algorithms for updating labelings.

We completely neglect any type of preselection of the objects to label; our aim is rather to label many of the effectively given objects. Neither we want to compute dynamically changing labelings in the sense of labelings that are attuned to each single user at each point in time (as the user's interests might change as well). This is, we use *static* weights to determine the importance of a label. Our algorithms must be understood as a final step in the process of map generation.



(a) initial scale; sparse labeling (b) more labels after zooming in

Fig. 1. Two maps [1]. The map on the left has a sparse labeling, although there are many cities that could be labeled (confer map on the right). Obviously, there is enough space to label (at least some of) them.

A comprehensive literature review (for a very small one, see Section 2) shows that there has been an abundance of work on *static* label placement, for example, for printed maps. Publications dealing with labeling dynamic maps are, however, rare; those additionally dealing with a 3D view are even rarer. This is surprising as commercial products such as navigation systems already use such algorithms. The algorithms in these commercial products are not satisfactory, though, since there are often large areas that contain interesting objects but no labels; see Figure 1. This is due to the fact that most systems are very conservative: they block large areas around a labeled point in order to avoid that labels overlap when the user interacts with the map.

To sum up, we want to solve the problem of labelings with few labels (such as generated by many common navigation system) while preventing occlusions of labels as well. That is, we develop algorithms and data structures for placing many, but occlusion-free, labels on dynamic maps in real-time. We also plan to test the real-time capabilities of these algorithms and data structures. Moreover, we should try to find out whether users accept these approaches and if they benefit from them; especially, we should try to learn if placing as many occlusion-free labels as possible helps or rather confuses the users.

In the following, we describe four work packages dealing with map labeling, namely embedded street labeling (see Section 3), active street labeling (see Section 4), city labeling (see Section 5), and POI labeling (see Section 6). For each of these packages, we characterize the problem, have a look at its contribution, outline an algorithm, and, finally, discuss the current state.

2 Related Work

There are many papers dealing with the problem of labeling static maps. There are, however, only few papers for labeling dynamic maps. Thus, most of our approaches are extensions of static labeling algorithms.

Strijk [9] investigates the problem of labeling streets on static 2D maps. The author aims for a labeling such that labels look like they are painted on the streets, each street has at most one label, and there are no label–label overlaps. In this work, he first studies typically labeled (European) street maps. Based on these findings, he formulates some rules. He translates these rules into mathematic expressions and combines them in a cost function in order to evaluate different labelings of the same map. He selects the labeling with the best score.

Maass and Döllner [7] mainly consider the same problem as Strijk [9] but for a dynamic 3D environment. This means, that scene elements (such as buildings) additionally can occlude labels. Indeed, the authors allow such overlaps but, nevertheless, they require that overlaps of scene elements and labels are reduced to a minimum. To solve the problem, the authors place labels directly into the 3D scene with a placement that is roughly similar to the one of Strijk. For each street, the authors determine several discrete label positions. They evaluate each position with the help of a cost function. Finally, the authors give different values for the variables of the function, depending on the desired output (for example, for centering labels within their objects).

Furthermore, Maass and Döllner [6] describe a point labeling algorithm for dynamic 3D environments using labels that always stay oriented to the user. Their algorithm places labels at varying heights in order to prevent overlapping labels. To this end, in each frame, it rasterizes the map, places a label, blocks all overlapped cells of the raster for other labels, and continues placing further labels. To maintain the correct label–object association, the authors use lines to connect a label with its reference point.

Van Kreveld et al. [5] change the common idea of point labeling problems. Instead of using a fixed point at which a reference point and a label must touch (for example, the bottom left corner), they use labels that are allowed to slide with their lower edges along their reference points. The authors describe two algorithms; one permitting and one prohibiting sliding. With the help of experiments, they show that the sliding approach labels up to 15% more points than the approach with fixed touching points.

Been et al. [2] provides a paper for labeling points on pannable and (continuously) zoomable maps. They insist that a label must not change its size or position over all scales. Besides, labels must not overlap. For this, they introduce the notion of *active ranges*. An active range is an interval of scales at which a label is visible. Their aim is to maximize the sum over the lengths of all active ranges. They compute the active-range data structure in a preprocessing step. At run time, they only query the current labeling. This allows their approach to run in in real-time. Additionally, Been et al. [3] refine this work from a more theoretical point of view. The authors show that the problem of maximizing the sum over the length of all active ranges is $\mathcal{NP}$-hard, that is, there is probably

no algorithm that solves the problem optimally in suitable running time. Furthermore, they give labeling algorithms handling different label forms. The one dealing with the form of congruent squares has an *approximation ratio of 4*, that is, the total sum over the lengths of the active ranges is at least $OPT/4$, where OPT is the value of an optimal solution.

Finally, Gemsa et al. [4] complement the work of Been et al. [2,3] by taking rotation operations into account. Now, an active range is an interval of angles. Again, the aim is to maximize the sum over all intervals whereas labels must not overlap. The authors show that this problem is still $\mathcal{NP}$-hard. By means of an approximation algorithm, nearly-optimal solutions can be computed quickly, though.

3 Embedded Street Labeling

An *embedded label* looks and behaves as if it were painted on the street; see Figure 2. Placing embedded labels is a 2D problem. (We can use the 2D algorithms for a 3D view without any changes.) We require that each label is easy to read: it should have a suitable reading direction and a bounded curvature. Moreover, it must not change its size on the screen at run time due to a zooming operation. This work package is interesting as, to the best of our knowledge, there is only one paper dealing with dynamically placing embedded labels to streets. Our approach is different to the existing approach of Maass and Döllner [7] as, on run time, they allow labels to move within the street; so, labels do not behave as if they were painted on the streets. This movement can be distracting.

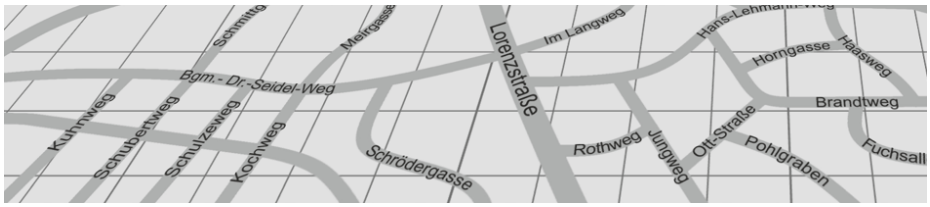
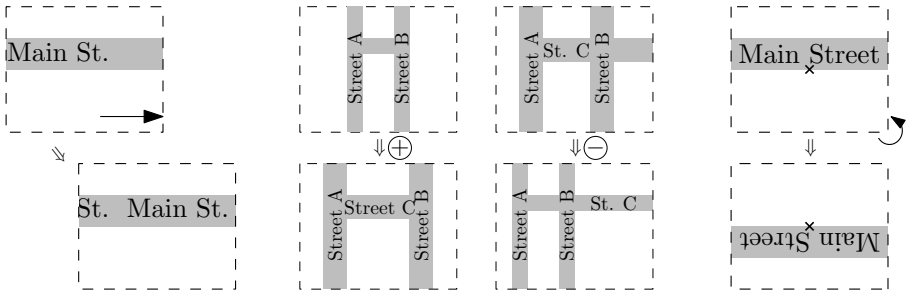


Fig. 2. Embedded labels look like they are painted on the street. Distortions can make labels crabbed.

In order to save running time, we only label the currently visible part of the map, the *view* (or probably a slightly larger part). Any interaction with the map results in changes of the view. As a consequence, we have to check for updates to the labeling incessantly. For example, a label could leave the view due to a panning operation (see Figure 3(a)). If no further label for the corresponding street is currently displayed, we have to place a new one. If the user zooms in (see Figure 3(b)), streets grow on the screen (while labels maintain their sizes). This creates space to place further labels. On the other hand, a street shrinks if the user zooms out. Then, we have to test if the street is still large enough to host



(a) panning: add new label if current label leaves the view

(b) zooming: add label if street gets large enough; remove or replace label if street gets too small

(c) rotation: correct reading direction if necessary

Fig. 3. Reactions to interactions.

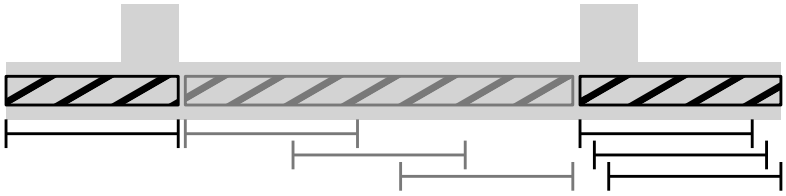


Fig. 4. The street (gray) is subdivided into three candidate strips (hatched). For each candidate strip, we have chosen three specific candidates: at the beginning, the center, and the end of each strip. A candidate becomes a label if it is selected as label position.

the already-placed label. If not, we have to test if there is another visible part of the street at which we can place the label instead. Finally, rotating the map results in reversed labels (see Figure 3(c)). We have to correct their orientations.

Our algorithm works roughly as follows. In the first phase of the algorithm, we compute an initial labeling. In the second phase, we react to interactions. Both phases subdivide streets first into *candidate strips* and afterwards into *candidates*. A candidate is a part of a street that represents a position for a label. A candidate strip is a part of a street that groups together candidates with equal properties. In this case, a candidate strip is the grouping of candidates that overlap the same crossings; see Figure 4 for these notions.

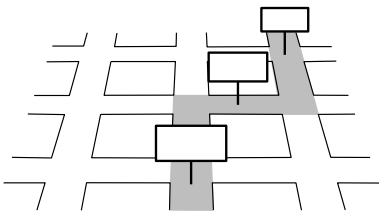
For each street in the view, we determine several candidate strips. Then, we choose for every street at most one candidate strip that does not overlap any other selected strip. Lastly, we subdivide each strip into candidates, evaluate each candidate by its quality (for example, by the number of bends), and place the label at a candidate with the best quality value.

We have partly implemented this algorithm. Thus far, we generate and select candidate strips for the initial and for the dynamic phase. The current implementation runs in real-time on multi-core processors (we have not test single-core processors yet).

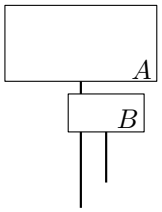
The subdivision of candidate strips into candidates and the implementation of the cost function is missing, though. This is complicated, as there are many (weighted) rules that could be included in the cost function. For example, we could take into account the number of bends, the angles of these bends, the centrality of a candidate on the street or the view, and so on. For all these properties, we have to find costs and weights such that the algorithm yields aesthetic labelings. We should conduct a user study in order to verify the usefulness and aesthetics of the outputs of our approach.

4 Active Street Labeling

In a navigation system, the user has the possibility to select a target. If so, the system computes a route to the specified target and highlights the corresponding streets. We call such a street (as long as the user has not passed through it completely yet) an *active street*. In this work package, we want to label streets in a 3D view. The idea is to use an embedded labeling (see Section 3) for inactive streets and to label active streets via *billboards* (see Figure 5(a)). A billboard is a label that is always oriented towards the user and usually placed with some distance to the referenced object. Therefore, we use *leaders* (for example, straight lines) to visualize the correct label–object association. We refer to the rectangle that contains the label text as *head*. Currently, we require that the leader touches the head in the middle of its the bottom edge. The reaction of billboards to interactions as well as the 3D view are the challenging aspects of this work package. We especially have to ensure correct depths of leaders and heads; see Figure 5(b).



(a) active streets labeled with billboards; leaders (centered at heads) maintain label–object associations



(b) violated depths; although the reference point of A is nearer, B overlaps A's leader

Fig. 5. Using billboards.

As far as we know, there is no work discussing the problem of labeling streets via billboards in a 3D view. There is, however, one single approach that labels points via billboards [6]. With this approach, leaders can suddenly change their lengths. Hence, incorrect depths of leaders and heads can emerge.

For labeling active streets, we use a *force-directed* approach, that is, reference points attract their corresponding heads and heads repel each other. Briefly worded, the algorithm varies the length of the leaders at run time in order to maintain an occlusion-free labeling while the user moves through the map.

We have implemented this approach. Indeed, it works but it has some weaknesses; for example, labels shiver or, without any visual reason, a leader gets abruptly longer. It remains to test if this is a conceptual or an implementation issue. Moreover, there are many variations of the problem we can additionally take into account; for instance, we could allow several contact points.

We expect that the mixed-type of labelings will support the user's orientation significantly; not least because billboards prevent the distortion of label texts. We should test this assumption in a user study, though. In a pilot study with static pictures, we already found out that the mixed-type could be accepted.

5 City Labeling

We can distinguish two ways to display cities. On the one hand, in large-scale maps, cities are displayed as areas. On the other hand, in small-scale maps, cities are represented by single points. Here, we consider the point labeling problem. In order to place a large number of labels, we allow any label to slide on its bottom edge at running time. This problem is interesting in the 2D case as well as in the 3D case. To the best of our knowledge, there is no other work that establishes an algorithm for the dynamic version of that problem. Van Kreveld et al. [5] experimentally showed that, in static maps, up to 15% more labels can be placed if sliding labels are permitted.

Our algorithm for the 2D case works roughly as follows. To save running time, we again only label the view (or probably a slightly larger part). This is why we have to test incessantly if there are further (so far unlabeled) reference points visible and which city labels have left the view. To make room for placing another occlusion-free label, we allow already-placed labels to slide. For that purpose, we build a simple data structure. For each label ℓ, we store all of the labels that lie left to ℓ and all of the labels that lie right to ℓ. Additionally, we store the distance between these neighbor labels and ℓ. With the help of this data structure, we can quickly query if a new label overlaps an already-placed one and which labels are affected if we let labels slide (in favor of the new label). In this way, we can decide either to place or to dismiss the new label.

We have so far implemented the data structure, but have yet to implement an algorithm for sliding labels. Currently, the (partial) implementation runs in real-time on multicore processors although it does not react to interactions by updating the data structure; instead it recomputes the data structure completely in each frame. In order to save running time, we could determine, for each type of interaction, a point in time at which we have to launch the update of the data structure. To reduce the running time even further, we have implemented a *waiting function*. This means that the algorithm does not check in *each* frame

if an unlabeled point in the view can be labeled but the algorithm waits *several* frames before it checks this again.

For the 3D case, we have some initial ideas which we have not entirely developed yet. They are mainly an adaption of the 2D approach.

Both the 2D and the 3D algorithm are probably fully real-time capable. We should be caseful about the user acceptance, though. Surely, this approach is a nice add-on for the *map mode*, that is, when the user navigates the map on his or her own. By contrast, in the *navigation mode*, the navigation system provides the interactions. In this navigation mode, moving labels could possibly confuse the user rather than help. We should test this in a user study. If users do not accept sliding labels, we just can switch off the movement. This simplified approach is even faster and, probably, will still place more labels than algorithms of current commercial systems. We plan to verify this.

6 POI Labeling

The result of the following work package has been published at the 16th ICA Generalisation Workshop [8].

A *point of interest* (POI for short) is a place of special interest, for example, a hospital, a restaurant, or a gas station. In navigation systems a POI annotation is typically represented by an *icon*. An icon is a pictorial label; see Figure 6. Normally, an icon maintains the same size on the screen all the time. So, if the user zooms the map out, the distance between two points of the map gets smaller on the screen or, regarded the other way round, the labels grow with regard to the map background. This generally results in overlapping labels. (Other interaction types do not substantially influence the current labeling.)

We investigate this problem under the constraint of circular icons with equal diameters. Additionally, we aim for a consistent labeling. By this we mean that once a label has vanished, it can only reappear if the user changes the zooming direction. The idea is to build a data structure in a preprocessing step and to

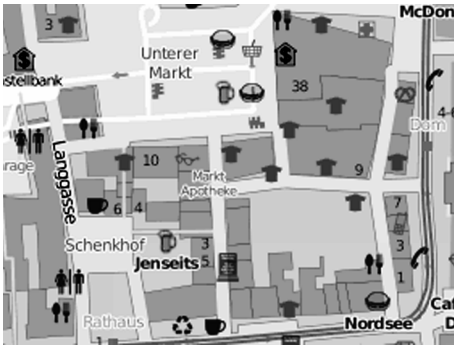
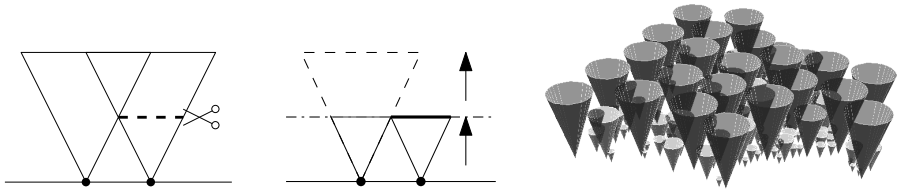


Fig. 6. Map with some POIs [1].



(a) schematic representation of shrinking-cones heuristic (2D view)

(b) schematic representation of growing-cones heuristic (2D view)

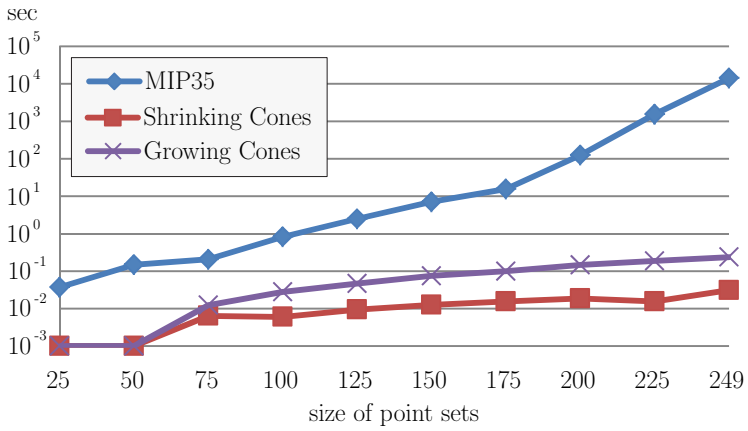
(c) visualized data structure (computed by shrinking-cones heuristic)

Fig. 7. Computing a data structure for POI labelings.

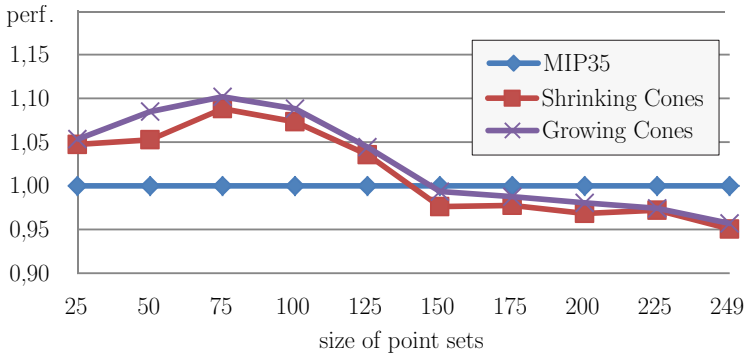
query label positions (from this data structure) at run time quickly. In the data structure, we represent each label by an interval of zoom levels at which the label is displayed. We can visualize the intervals (over zoom levels) as (disjoint) cones; see Figure 7(c). In order to maximize the number of placed labels, the goal is to maximize the sum over the lengths of all intervals. This problem is $\mathcal{NP}$-hard [3], that is, there is probably no algorithm that solves the problem optimally in suitable running time. For that reason, we have developed and implemented *heuristics*. Heuristics are fast but not necessarily optimal algorithms. Additionally, we have established a *mixed integer linear program* (MIP for short) for computing optimal solutions in order to evaluate the quality of the heuristics.

Our algorithms work roughly as follows. The first one, see Figure 7(a), starts with (overlapping) cones of equal heights. In order to prevent overlapping POI labels, we just cut the cones so that no two cones intersect. This approach is an adaption of an algorithm introduced by Been et al. [2,3]. The second heuristic, see Figure 7(b), works the other way round. We start with cones of height zero. We let the cones grow until two cones touch. At this point, we decide which cone stops growing and which cone grows further.

We compared the results of our heuristics with optimal solutions (computed by the MIP). Due to the high running time of solving the MIP, we only computed the MIP35, that is, we stopped computations as soon as the result of the MIP was not worse than 35% of an optimal solution. Figure 8 shows that our heuristics are much faster than the MIP35 and still yield labelings with a nearly-optimal number of labels. Indeed, we can prove that the growing-cones heuristic yields an approximation ratio of 1/5, that is, the sum over all intervals of an optimal solution is at most five times larger than the sum of any solution computed by the heuristic. To round off, we can also enhance our algorithms such that they can deal with priorities and even with labels of different sizes. Then, however, the proof for the approximation ratio of the growing-cones heuristic does not hold anymore.



(a) absolute running times in seconds (log-scale!)



(b) total active range height relative to MIP35

Fig. 8. Results of our experiments for synthetic data sets. We uniformly distributed 249 points in the unit square $[0,1]^2$. For each point set of size $n = 25, 50, \ldots, 225$ we randomly selected n point of the point set. For each n, we averaged the sum of all heights as well as the running time over five trials. For $n = 249$, we did only one trial.

7 Conclusion

We deal with the problem of labeling dynamic maps in an occlusion-free manner; we are especially interested in dynamic maps that allow for continuous zooming. The aim is to place as many labels as possible. In this work, we introduced four work packages, namely, embedded street labeling, active street labeling, city labeling, and POI labeling, sketched some solutions, and described the current state of each of the packages. All in all, wide parts of the concepts are already finished. Mainly, further implementation work is required (for example, speeding up the programs or combining the algorithms). We ideally should conduct a study dealing with the user acceptance (for example, to answer the question if sliding labels does disturb the user). The four work packages are intended to reduce the gap between theory and practice; between papers dealing with static labeling problems and navigation system that already use (poor) algorithms for labeling dynamic maps.

References

1. Extracts from OpenStreetMap, URL: `http://www.openstreetmap.de/`. Accessed 10/31/2013.
2. Been, K., Daiches, E., Yap, C.: Dynamic map labeling. IEEE Transactions on Visualization and Computer Graphics 12(5), 773–780 (2006)
3. Been, K., Nöllenburg, M., Poon, S.H., Wolff, A.: Optimizing active ranges for consistent dynamic map labeling. Computational Geometry: Theory and Applications 43(3), 312–328 (2010)
4. Gemsa, A., Nöllenburg, M., Rutter, I.: Consistent labeling of rotating maps. In: Dehne, F., Iacono, J., Sack, J.R. (eds.) Proceedings of the 12th International Symposium on Algorithms and Data Structured (WADS'11). vol. 6844, pp. 451–462 (2011)
5. van Kreveld, M., Strijk, T., Wolff, A.: Point labeling with sliding labels. Computational Geometry: Theory and Applications 13(1), 21–47 (1999), `http://dx.doi.org/10.1016/S0925-7721(99)00005-X`
6. Maass, S., Döllner, J.: Efficient view management for dynamic annotation placement in virtual landscapes. In: Butz, A., Fischer, B., Krüger, A., Oliver, P. (eds.) Proceedings of the 6th International Symposium on Smart Graphics (SG'06). Lecture Notes in Computer Science, vol. 4073, pp. 1–12. Springer-Verlag (2006)
7. Maass, S., Döllner, J.: Embedded labels for line features in interactive 3D virtual environments. In: Slay, H., Spencer, S.N., Bangay, S. (eds.) Proceedings of the 5th International ACM Conference on Computer Graphics, Virtual Reality, Visualization and Interaction in Africa (AFRIGRAPH'07). pp. 53–59. ACM (2007)
8. Schwartges, N., Allerkamp, D., Haunert, J.H., Wolff, A.: Optimizing active ranges for point selection in dynamic maps. In: Proceedings of the 16th ICA Generalisation Workshop (ICA'13) (2013), 10 Pages
9. Strijk, T.W.: Geometric Algorithms for Cartographic Label Placement. Ph.D. thesis, University of Utrecht (Jan 2001), chapter 9