# Labeling Streets in Interactive Maps Using Embedded Labels

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

Jan-Henrik Haunert
Institute for Geoinformatics and Remote Sensing
University of Osnabrück, Germany
http://www.igf.uni-osnabrueck.de/en/institute/staff/

## ABSTRACT

We consider the problem of labeling linear objects (such as streets) in interactive maps where the user can pan, zoom, and rotate continuously. Our labels contain text (such as street names). They are *embedded* into the objects they label, i.e., they follow the curvature of the objects, they do not move with respect to the map background, but they scale in order to maintain constant size on the screen. To the best of our knowledge, this is the first work that deals with curved labels in interactive maps.

Our objective is to label as many streets as possible and to select label positions of high quality while forbidding labels to overlap at street crossings. We present a simple but effective algorithm that takes curvature and crossings into account and produces aesthetical labelings. On average over all interaction types, our implementation reaches interactive frame rates of more than 85 frames per second.

## Categories and Subject Descriptors

H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*artificial, augmented, and virtual realities*

## General Terms

Algorithms, Experimentation

## Keywords

Automated label placement, interactive maps, visualization

## 1. INTRODUCTION

Whenever we visit a city where we have never been before, we appreciate tools that help us to find our ways. Some years ago, paper maps were the only such tool. They mainly guide us by means of textual annotations, i.e., by *labels*. In this work, we consider the problem of labeling streets. We require that a street label follows the curvature of its street. The challenge is to determine nice-looking label positions and, at the same time, to avoid overlaps of labels at crossings. We want to label many streets, but in this paper we do not aim at placing more than one label per street. (We motivate this decision in the related work section below.) That brings us to the *static* label-number optimization problem:

Given a set $P$ of objects (here: polygonal lines or, for short, polylines) in the plane and, for each $p \in P$, a set $L(p)$ of potential label positions and, for each $\ell \in L(p)$, its cost $c(\ell)$, find a subset $\mathcal{L} \subseteq \bigcup_{p \in P} L(p)$ of label positions that primarily maximizes the number $|\mathcal{L}|$ of placed labels and secondarily minimizes the sum of the costs $\mathcal{C} = \sum_{\ell \in \mathcal{L}} c(\ell)$ such that no two labels intersect and $|L(p) \cap \mathcal{L}| \leq 1$ for each $p \in P$, i.e., there is at most one label per polyline. The cost of a label position $\ell$ is meant to be small if $\ell$ is aesthetically pleasing, e.g., if $\ell$ has few bends. Note that the label-maximization problem for points [7] can be seen as a special case of our problem.

Most people nowadays rather use digital maps. Hence, we aim for placing street labels in a *dynamic setting* where the user can interact with the map by panning, zooming, and rotating continuously. Interactions change the visible part of the map. The additional challenge of a dynamic settings is to react to interactions properly; e.g., if a label leaves the visible part of the map. We stress that we only know the interaction that is currently executed.

**Our Model.** We define a time interval $[0, T]$ in which the user interacts. The content of the screen is redrawn repeatedly; the content between two updates is a *frame*. Accordingly, we discretize the time interval into a sequence $t_1, \ldots, t_m$ (with $t_1 = 0$ and $t_m = T$) of points in time that correspond to frames. At time $t_i$, the user can see a rectangular region $R_i$ of the plane, the *view*. When the user pans, $R_i$ is translated; when the user zooms, $R_i$ is scaled; when the user rotates, $R_i$ is rotated. Due to the change of the view, on a street, a potential label position that yields lower costs than the selected one can appear. Still, we prohibit that a label slides to a better position as we believe that this would irritate the map user.

This brings us to the following extension of the static line-labeling problem: For $i = 1, \ldots, m$, let $\mathcal{L}_i$ be the subset of labels (at most one per street) that are placed at time $t_i$ and intersect $R_i$ (with more than, say, half of their length); let $\mathcal{C}_i$ be the sum of the costs of the labels in $\mathcal{L}_i$. Our primary goal is to maximize the number of placed labels $\sum_{i=1}^{m} |\mathcal{L}_i|$ over all frames; our secondary goal is to minimize the sum of the costs $\sum_{i=1}^{m} \mathcal{C}_i$ over all frames.


*SIGSPATIAL'14,* November 04–07 2014, Dallas/Fort Worth, TX, USA

ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666494.

Several online map services use tile-based maps. Usually, such services precompute the labeling of several rows of tiles and cache the results. If a label overlaps two tiles, the placement of the two label pieces must be coordinated, otherwise the two pieces may not fit. By contrast, we use vector-based maps and do not cache labels.

**Related Work.** In general, there are three types of map objects that must be labeled: points (e.g., cities in a small-scale map), polygonal lines (e.g., streets in a large-scale map), and areas (e.g., countries). Wagner et al. [7] and others suggest to solve the general label-placement problem by first generating a sufficient number of potential label positions for any map object to be labeled and then having a high-level algorithm making a choice between these positions such that as many map objects as possible receive an overlap-free label. In this paper, we focus on the first step and use an ad-hoc method for the second step: we proceed incrementally, i.e., we go through the map objects (streets in our case) in an arbitrary order and greedily place labels.

In his seminal work, Imhof [2] listed rules for good labelings of point, line, and area objects. For linear objects, he states that labels should follow the curvature but avoid too strong bends. He recommends to not put too much white space between characters of the same label. (Our labeling algorithm observes these rules. We ignore, however, the rule that labels should be written as horizontal as possible—in our dynamic setting, the orientation of objects can change.) Finally, a label should be repeated, especially if two objects are connected and cannot be distinguished.

We ignore this rule due to the fact that it is NP-hard to maximize, in an overlap-free *static* labeling, the number of street segments that are *labeled* [1], in the sense that the label overlaps the street segment (so that the map reader understands the label–segment association). Moreover, it is known that maximizing the number of labeled streets without label–label overlaps is NP-hard [4]—even if every street is horizontal or vertical. Hence, our problem is NP-hard even for *one* frame.

Strijk [5] gave one of the few algorithms for (static) street labeling with embedded labels. His work has inspired our algorithm. To obtain an overlap-free and aesthetical labeling, Strijk computes potential label positions for each street. Next, he applies a heuristic for optimizing an evaluation function for the *entire* labeling (we, by contrast, decide street-wise). His function considers three criteria: street–label association, label visibility, and aesthetics.

So far two approaches for labeling streets dynamically have been suggested; one by Maass and Döllner [3] and one by Vaaraniemi et al. [6]. Both place only straight-line labels.

Maass and Döllner [3] label interactive 3D environments, i.e., maps with 3D objects (such as buildings). They prevent label–label and object–label occlusions. The authors support two modes of user interaction. In the first mode, the labeling does not change while the user interacts. When the user stops interacting, labels slide through the streets until they are visible. In the second mode, labels fade out when the user starts interacting with the map and fade in when the user stops the interaction. We argue that both modes cause high visual complexity.

Vaaraniemi et al. [6] gave a force-directed algorithm for placing straight-line labels onto curved streets. Additionally, they conducted an expert study. They presented three labeling styles: (a) Labels are placed horizontally. To maintain the label–object association, they used straight lines connecting label and object. (b) Straight labels have a similar rotation as the streets they label. (c) Labels follow the course of the street. Five out of six experts judged (c) as the most aesthetic style but also expected bad readability for irregular streets. This is exactly what we want to prevent with our approach for curved labels.

**Our Contribution.** We present the first algorithm that annotates streets with embedded curved labels in real time and deals with panning, zooming, and rotations (see Section 2). We guarantee that our labelings are overlap-free if the font height is bounded by the street width. Our labelings are aesthetical in that we punish label positions with strong bends. We have implemented our algorithm and tested it on real-world data (see Section 3). A video shows our labeling algorithm in action[1].

## 2. ALGORITHMS

Our algorithm repeatedly tests each unlabeled street in the view for potential label positions. Among these, we try to select an aesthetical one.

### 2.1 Finding Nice-Looking Label Positions

Any long-enough street contains an unbounded number of possible label positions. For finding a good position, we initially place a label at the start point of its street. Then, we push the label through the entire street. Simultaneously, we evaluate each label position by some *evaluation criteria* to obtain *costs*. Based on the costs, we select a good position.

We assume that a polyline $\langle s_1, \ldots, s_n \rangle$ is given by an ordered sequence of segments, where the end point of segment $s_i$ ($i = 1, \ldots, n-1$) and the start point of segment $s_{i+1}$ are the same. We call this point $b_i$ a *bend*. Start and end points of streets are not considered bends. In our case, costs only change at bends.

**Evaluation.** We first define the cost for each bend. The cost depends on several criteria, which are weighted by importance. We sum up the weighted costs of the criteria to obtain the cost of a bend; formally,

$$C(b_i) = \sum_{e \in E} w_e c_e(b_i)$$

where $e$ is an evaluation criterion from the set of criteria $E$, $w_e$ is the weight of $e$, and $c_e(b_i)$ is the cost of $e$ for bend $b_i$.

Next, we compute the cost of a label position. Consider a polyline $\pi = \langle s_1, \ldots, s_n \rangle$ and any label position $\ell$ at $\pi$. Let $\pi' = \langle s_j, \ldots, s_k \rangle$ ($1 \le j \le k \le n$) be the sequence of segments that $\ell$ occupies. Then we define the cost of $\ell$ to be

$$\sum_{i=j}^{k-1} C(b_i).$$

Suppose that $\ell$ does not start at bend $b_{j-1}$, the start of $s_j$, but anywhere else at $s_j$. Then $\ell$ has at least the same cost as a label of the same length that does start at $b_{j-1}$. So, it suffices to consider the bends as starting points for potential label positions. Let $|\ell|$ be the length of $\ell$. For

[1] `http://lamut.informatik.uni-wuerzburg.de/dynalinelab.html`


(a) $\alpha_i \in [0°; 180°)$ (b) $\alpha_i \in [0°; -180°)$

**Figure 1: How we measure angles.**

each $i = 1, 2, \ldots$, we define $k(i)$ (if such an index exists) such that $\sum_{j=i}^{k(i)-1} |s_j| < |\ell|$ and $\sum_{j=i}^{k(i)} |s_j| \geq |\ell|$. For each sequence $\langle s_i, \ldots, s_{k(i)} \rangle$, we compute its cost and determine the cheapest sequence. We center the label within its sequence. We now describe our evaluation criteria.

**Angles.** Let $\alpha_i \in (-180°, 180°)$ be the angle between segments $s_i$ and $s_{i+1}$ measured as in Figure 1. In the sequel, we identify $\alpha_i$ with its absolute value $|\alpha_i|$. Obviously, the larger $\alpha_i$, the worse a label looks if it contains bend $b_i$.

We want to punish large angles but also take into account that many small angles between very short segments basically also yield a large angle. Formally, let $\theta$ be a threshold, e.g., the average width of a character of the given font. For determining the angles that contribute to our evaluation, we take into account the length $|s_i|$ of $s_i$. Let $\alpha_0 = 0$. If $|s_i| \geq \theta$, let $k = 0$. Otherwise, we search for a $k > 0$ so that $\sum_{j=i}^{i+k-1} |s_j| < \theta$ and $\sum_{j=i}^{i+k} |s_j| \geq \theta$. The final cost for the subsequence $\langle s_i, \ldots, s_{i+k} \rangle$ is $(\sum_{j=i-1}^{i+k-1} \alpha_j)^2$. The cost of a label position is the sum of the costs of its subsequences.

Finally, we define a maximum angle $\alpha^\star$. If $\alpha_i > \alpha^\star$, we stop evaluating the current sequence and start a new sequence at $s_{i+1}$.

**Crossings.** As we want to place as many labels as possible, we try to avoid labels that pass a crossing. Otherwise, we might block a crossing for another label that must pass the crossing. For that reason, we charge a cost of $X$ for every crossing that a label contains. If we find a crossing while evaluating a sequence, we test whether it is already occupied. If so, we cancel the evaluation of the current sequence and start a new sequence after the crossing.

Observe that we visualize our streets with *spatial extent*. By contrast, our graph data structure for the street network consists of points (crossings) and polylines (street sections). For that reason, our evaluation takes into account the *visual* start and end points of the streets; see Figure 2.

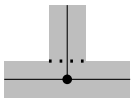

**Figure 2: Data structure (lines) and visualization (bars) of our street network. The dotted line indicates the visual end point of the vertical street.**

**The View.** In every frame, we collect for each polyline all its segments that intersect the view. If a polyline leaves and enters the view multiple times, we choose the longest visible part. As we permit labels to overlap the view boundary, we expand each polyline that leaves the view by some threshold $v$. We evaluate the (extended) visible parts of a polyline, but we punish sequences overlapping the view boundary with some additional cost $V$. If the selected sequence overlaps the view boundary, we do not center the label within its sequence but we try to maximize its visibility.

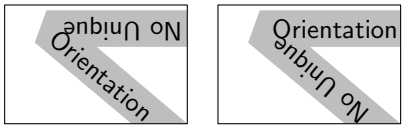


**Figure 3: Correcting the orientation of one part of a label might lead to a distortion of another part.**

## 2.2 Dealing with Interactions

In each frame, we do three things: we try to label unlabeled streets, we adjust the label positions while zooming, and we correct the labels' orientations while rotating. Since panning is simple, we sketch only zooming and rotations.

First, assume that the map user zooms out. (We treat zooming in inversely.) Since we require that the label size remains constant on the screen, labels grow with respect to their streets. In principle, our labels grow equally at both ends. If, however, a label is not centered, it first grows such that it becomes centered again. If the label overlaps the view boundary, the label grows into the direction of the view. We try to keep crossings free by growing labels away from crossings if possible. As before, we reject label positions that contain bends that exceed $\alpha^\star$.

Second, if the user rotates the map, the orientation of a label can become upside down. There are cases in which the orientation of the label is not unique; see Figure 3. As a rough guide, we determine the orientation by the gradient between the start point and the end point of the label.

# 3. EXPERIMENTS

We have implemented the algorithm of Section 2 using C++ with OpenSceneGraph 3.0[2] on Windows 7 with a 3.3-GHz AMD triple-core processor and 8 GB of RAM. We used the Microsoft Visual Studio 2010 Ultimate compiler in 32-bit release mode. For our experiments, we used OpenStreetMap data provided by Geofabrik[3] from which we extracted 620 streets of the downtown of Würzburg.

Our algorithm for placing letters of curved text is rather slow. Since there are open-source tools for OpenStreetMap data that render curved text (such as libosmscout[4]), we did not optimize this part of our implementation.

Table 1 gives an overview over the results of our experiments; we discuss them in the remainder of this section. Figure 4 depicts some screen shots of a map labeled by our algorithm. To save running time, we computed the lengths of street segments and the angles between them in a preprocessing. This preprocessing took less than one second.

As the computation time depends on the number of streets within the view, we executed our test for two different resolutions; one simulates the screen of a navigation system (*GPS*), the other one a computer monitor. On an average, computing the initial labeling took us 0.08 sec. for GPS and 0.27 sec. for the larger monitor screen size.

We tested several camera paths. There are four *path classes* in which we only execute one interaction type. Each of these path classes contains five different paths at various

[2] `http://www.openscenegraph.org/`, accessed 11-24-2013
[3] `http://download.geofabrik.de/`, accessed 5-20-2014
[4] `http://libosmscout.sourceforge.net/`, acc. 6-30-2014

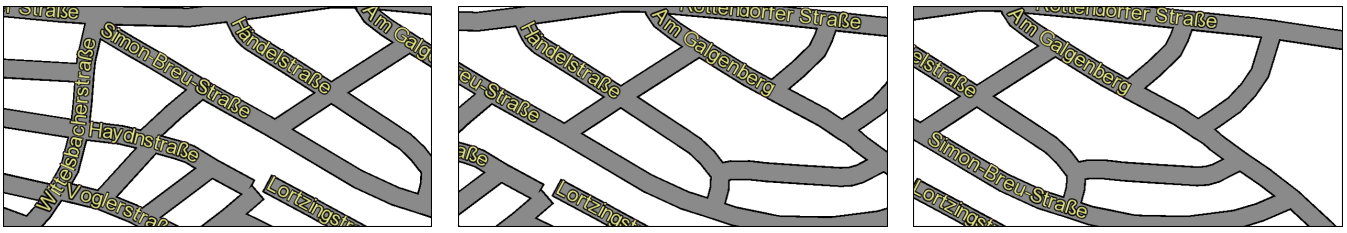


**Figure 4: From left to right: We pan to the right. Initially, the label "Simon-Breu-Straße" is completely visible; then it partly leaves the view. After moving it to a new position, it is completely visible again.**

scales. Each path lasts 14 sec. At the largest scale, there are, on average, 5 labels on the GPS display and 15 labels on the monitor; at the smallest scale, there are 15 and 30 labels, respectively. The fifth path class contains five multi-interaction paths. Each path mixes panning, zooming, and rotation operations for a total of 72 sec. We distributed the time for each of these three interaction types equally at three different scales (the largest and the smallest scale being the same as for the single interaction paths).

After some testing, we set the parameters of our algorithm as follows: $\theta = 0.57f$, where $f$ is the current font size, $X =$ 100,000, $v = |\ell|/2$, where $\ell$ is the label of the polyline, $V =$ 100,000, and $\alpha^\star = 90°$. We weigh all criteria equally.

Our implementation yields very good frame rates of 90–190 frames per sec. (*FPS*) for panning, rotating, and the multi-interaction paths. At first sight, the frame rates of 4–12 FPS for zooming are unacceptable. On closer inspection, we spent most of the time for placing the letters; a routine that we considered less important to improve as stated above. For panning and rotation operations, we only draw changes of the labeling. While zooming, we have to draw all the labels in each frame as, on the screen, the underlying streets change continuously. Switching off the drawing routine results in average frame rates of more than 155 FPS (independently of the interaction type).

**Table 1: Results. Our camera paths either executed only one interaction type or a combination (*multi*). We processed every path with a small and a large number of visible streets as in navigation systems (*GPS*) and on computer monitors (*mon.*), respectively. We recorded frame rates, (long enough) visible streets, and the percentage of placed labels.**

| | | ∅f-rate | all streets | | long enough | |
|---|---|---|---|---|---|---|
| type | size | [FPS] | #vis. | %lab. | #vis. | %lab. |
| pan | GPS | 182 | 26 | 34 | 11 | 78 |
| | mon. | 135 | 78 | 33 | 31 | 83 |
| zoom | GPS | 13 | 25 | 30 | 10 | 78 |
| out | mon. | 4 | 55 | 37 | 24 | 84 |
| zoom | GPS | 12 | 25 | 33 | 10 | 82 |
| in | mon. | 4 | 67 | 34 | 26 | 87 |
| rota- | GPS | 189 | 22 | 40 | 11 | 81 |
| tion | mon. | 139 | 64 | 35 | 27 | 85 |
| multi | GPS | 126 | 24 | 33 | 10 | 82 |
| | mon. | 90 | 68 | 35 | 27 | 86 |

Our algorithm labeled only about 34% of *all* visible streets, but 80% of all streets that are actually *long enough* to host their labels. Note that even in optimal solutions there can be unlabeled streets.

Finally, we compare our algorithm with the two other existing algorithms for labeling lines in interactive maps. Clearly, the frame rate depends on the number of visible streets, whether labels are curved (both existing approaches use straight-line labels), and on the hardware. In general, a frame rate of 24 FPS is qualified as fluent. Maass and Döllner [3] achieved frame rates of 17–22 FPS using an 2.93-GHz Intel Core 2 Duo processor with 2 GB of RAM. Vaaraniemi et al. [6] used GPU computation (on an NVIDIA GeForce 8600M GT with 256 MB of RAM). They labeled 512 objects of various types within 5.5 ms (which corresponds to 180 FPS).

We conclude that our algorithm is highly real-time capable in most situations. When zooming, the letter-placement is too slow. This should be improved.

# 4. REFERENCES


[1] A. Gemsa, B. Niedermann, and M. Nöllenburg. Label placement in road maps. In *Proc. 30th Europ. Workshop Comput. Geom. (EuroCG'14)*, 2014.

[2] E. Imhof. Positioning names on maps. *Amer. Cartogr.*, 2(2):128–144, 1975.

[3] S. Maass and J. Döllner. Embedded labels for line features in interactive 3D virtual environments. In *Proc. 5th Int. ACM Conf. Computer Graphics, Virtual Reality, Visualization and Interaction in Africa (AFRIGRAPH'07)*, pages 53–59, 2007.

[4] S. Seibert and W. Unger. The hardness of placing street names in a Manhattan type map. In G. Bongiovanni, G. Gambos, and R. Petreschi, editors, *Proc. 4th Italian Conf. Algorithms Complexity (CIAC'00)*, volume 1767 of *LNCS*, pages 102–112. Springer, 2000.

[5] T. Strijk. *Geometric Algorithms for Cartographic Label Placement*. PhD thesis, Utrecht University, Department of Computer Science, Jan. 2001.

[6] M. Vaaraniemi, M. Treib, and R. Westermann. Temporally coherent real-time labeling of dynamic scenes. In *Proc. 3rd Int. Conf. Comput. Geospatial Research Appl. (COM.Geo'12)*, pages 17:1–17:10. ACM, 2012.

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