Labeling Streets Along a Route in Interactive 3D Maps Using Billboards

Nadine Schwartges, Benjamin Morgan, Jan-Henrik Haunert, and Alexander Wolff

Abstract We consider the problem of labeling linear objects, such as streets, in *interactive 3D maps*, where the user can continuously pan, zoom, and rotate a perspective view of the scene. We dynamically annotate streets that belong to a user's route, assuming that the future course of the route, within the currently visible part of the map, is known or well predicted. We use *billboards* as annotations, that is, each label is a rectangle holding the annotation text, is oriented towards the user, placed at some distance above the midpoint of the street to be labeled, and connected to the point by a vertical line segment, the *leader*.

Our goal is to maintain an overlap-free labeling that reacts to changes of the view in real time. To this end, we dynamically vary the lengths of the leaders. In order to achieve that labels move smoothly, we do not strictly forbid label–label overlaps. We present a force-directed algorithm that applies forces to labels to cause overlapping labels to repel each other, while keeping leaders as close to their desired length as possible. On real-world data, with a realistic number of labels, we obtain frame rates of more than 400 frames per second, while drastically reducing the total overlapped area per frame, compared to an algorithm with fixed leader lengths.

Key words: Dynamic maps, Interactive maps, Map labeling, Street labeling, Billboards

N. Schwartges (✉) · B. Morgan · A. Wolff
Chair of Computer Science I, University of Würzburg, Germany
URL: http://www1.informatik.uni-wuerzburg.de/en/staff

B. Morgan
e-mail: benjamin.morgan@stud-mail.uni-wuerzburg.de

J.-H. Haunert
Institut für Geoinformatik und Fernerkundung, University of Osnabrück, Germany
URL: http://www.igf.uos.de/en/institute/staff

F. Bação et al. (eds.): AGILE 2015, *Geographic Information Science as an Enabler of Smarter Cities and Communities*, LNG&C, 2014.

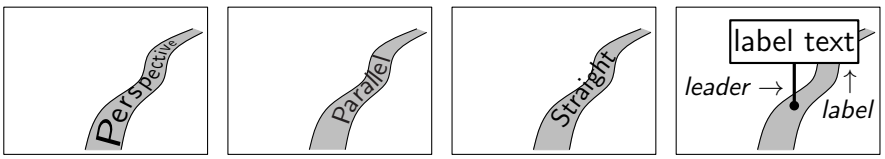
1 Introduction

Interactive maps commonly allow users to zoom, pan, and rotate the map. Examples of such maps are Google Maps[1] or digital devices with navigation software. Some interactive maps provide only a two-dimensional (2D) view whereas others, for instance, Google Earth[2], provide a perspective 3D view. Such tools help users to orient themselves in an unknown environment.

Mobile devices with interactive maps commonly provide a *map mode* with which users can freely move about (literally, users travel with their fingers on the map). Additionally, the same devices offer a *navigation mode*, a route planner, that leads users from their current locations to specified destinations. Sometimes, it is even possible to interact with the map in navigation mode. To aide users in orienting themselves, independent of the mode, map objects are usually annotated by textual or pictorial *labels*.

There are three types of objects represented in maps: *points*, such as a cities (in small-scale maps) and points of interest; *lines*, such as streets and rivers; and *areas*, such as countries and lakes. In this work we consider the problem of labeling streets, but our results can also be used for labeling point features (without any changes) or area features (assuming that we are given a suitable point in each area).

In printed large-scale maps, streets are commonly labeled *embedded*, that is, the label is placed inside the area occupied by the street and follows the curvature of the street. In interactive 3D maps, embedded labels are rendered with perspective distortion, as in Fig. 1(a), or they are rendered parallel to the view plane, as in Fig. 1(b). Sometimes, labels are placed straight—neglecting the exact course of the street—as in Fig. 1(c), in order to save computation time and to improve the readability of the label text. It is also quite common to make use of *billboards*; examples can be found in some built-in car navigation systems. Our billboards consist of (i) a *label*, that is, a rectangle that is oriented towards the user and holds the *label text*, and (ii) a *leader* that connects the point to be labeled, the *reference point*, with the label. The leader of a billboard can be a line, as in Fig. 1(d), or a more complex object, such as a triangle or arrow. This paper focuses on placing billboards on streets in interactive maps that are in navigation mode.



a perspective **b** parallel **c** straight and parallel **d** billboard

Fig. 1 Different ways of labeling streets in a large-scale 3D map. Our approach uses billboards

[1] `https://maps.google.com/`, accessed Oct. 1, 2014

[2] `https://earth.google.com/`, accessed Oct. 1, 2014

When the navigational device leads a user to a destination, the route which the user has to follow is usually highlighted. We call this route and the corresponding streets *active*. Accordingly, we refer to streets that are not contained in the active route as *inactive*. By using billboards instead of embedded labels for the active route, we highlight active streets. Moreover, horizontally oriented text can be read faster than rotated text (Larson et al, 2000; Wigdor and Balakrishnan, 2005). We thus improve the readability of those labels that are, at any given time, the most important ones for the user. For labeling the remaining streets, our algorithm can be combined with our algorithm that embeds labels into streets in large-scale interactive maps (Schwartges et al, 2014).

Our Model We consider a dynamic scenario where the user follows a route in an interactive navigation mode. To this end, we consider a time interval $[0,T]$ in which either the navigational device automatically manipulates the map, or the user manually interacts with it. During this time interval, the content of the display of the navigational device is redrawn repeatedly; the content between two updates is a *frame*. Accordingly, we discretize the time interval into a sequence $t_1,t_2,\ldots,t_k$ (with $t_1 = 0 < t_2 < \cdots < t_k = T$) of points in time that correspond to frames. At any given time t_i, the user can see a trapezoidal region R_i of the map. This is the image of a projection (whose center is the camera) of the rectangular, vertical screen on the horizontal map. When panning, R_i is translated on the map; when zooming, R_i is scaled; when rotating, R_i is rotated; and when changing the camera angle of the 3D view, we transform R_i perspectively, more precisely, if we change the camera angle such that we can see more of the map at the horizon, the edge of R_i that corresponds to the bottom edge of the view gets shorter, the other base edge gets longer, the two angles at the shorter base edge of R_i gets larger, and the legs get longer; and vice versa.

For our model, we lean on the physical principle of a thermodynamic equilibrium. We assume that in each frame there might be a label–label overlap, because either a new label has come into the view, or an overlap of the preceding frame has not been completely solved, or a solved overlap from the preceding frame has caused another overlap. Each label–label overlap induces a force F_{overlap}. Moreover, we define a desired leader length. A leader that is too long or too short induces a force F_{leader}. As we assume that we only know the currently visible part of the map, we solve overlaps only from one frame to the next. We can translate the goal of establishing an overlap-free labeling in one frame to the goal of minimizing the acting force

$$F = \sum |F_{\text{overlap}}| + \sum |F_{\text{leader}}|$$

in that frame. More precisely, we aim to minimize each individual force, as some forces of the same label might cancel each other out, even though there is still a label–label overlap (which happens, for example, if a label A pushes label B down, but B is also pushed up by the force from having a too short leader). With the application of a physical model, we expect the movements of labels to look natural—as if they were subjected to physical laws.

Our Contribution We start with a motivation why it is reasonable to design an algorithm for placing billboards to streets (see Sect. 3). To this end, we present the results of a survey asking for the aesthetic and practicability of such labelings. Next, we present our force-directed algorithm for labeling streets with billboards in an interactive navigation mode (see Sect. 4). Throughout the navigation, the algorithm maintains an aesthetic and practical labeling; it uses repelling forces for resolving overlaps and both attracting and repelling forces for keeping leaders as close to their desired length as possible. All label movements are smooth. Tests of the implementation of the algorithm on real-world data show that our algorithm yields interactive frame rates of more than 400 FPS (see Sect. 5). A video that shows our algorithm in action is available at `http://lamut.informatik.uni-wuerzburg.de/dynaroutelab.html`.

2 Related Work

In his seminal work on label placement (for printed maps), the Swiss cartographer Imhof (1975) establishes many rules for good label placement. His two most important rules are that labels should be legible (R1) and always yield a correct label–object association (R2). We fulfill these rules since we align labels horizontally, we connect the label and the reference point by a leader, and we avoid overlaps.

Imhof also says that a label should reflect its object's importance (R3). In our setting, where we want to label streets on the active route, Imhof's rule is automatically fulfilled since our perspective view draws the closer (and, hence, in that moment more important) labels in the foreground larger than the distant (and less important) labels in the background. Further, Imhof states that labels should occlude the map background as little as possible (R4). Currently, we do not take this rule into account. This can, however, be achieved by making labels semi-transparent (at the cost of reducing legibility). In order to avoid label clusters, Imhof suggests carefully selecting the objects to be labeled (R5). In our navigation-mode scenario, we simply select the next n streets on the route to be labeled. Among these, we show all labels that fall into the current view; we avoid clusters by changing the leader lengths.

For drawing graphs aesthetically, Eades (1984) introduces an algorithm which is based on a physical model using forces. He considers a drawing aesthetic if the edges of the graph have similar lengths and the graph is as symmetric as possible. To this end, the vertices of the graph may move in any direction. Adjacent vertices are supposed to keep a certain distance from each other, non-adjacent vertices repel each other. In our model, by contrast, the reference point of a label is fixed and the point where leader and label touch can move only vertically. Similar to the edges in Eades' approach, our leaders try to have a certain length. The author states that he does not use Hooke's law (as we do) but a logarithmic function to obtain the edge lengths because a logarithmic function works better for vertices that are far apart. Eades' algorithm computes the forces and thus the new positions of the vertices several

times. Our approach is also iterative: in each frame, we recompute the lengths of the leaders if the corresponding labels overlap or are not at their desired lengths.

Table 1 gives an overview about our algorithm and some of the related work that we discuss in the following.

Vaaraniemi et al (2012) give a force-directed algorithm that is, to some extent, similar to ours. Their algorithm labels points and areas horizontally. Depending on the current perspective, a street label is either placed horizontally or is aligned to a straight line that approximates the course of the street. In contrast to our approach, their algorithm operates in what we call map mode and they allow any leader direction. The main difference between the two approaches is that while we always display all labels that fall into the view, Vaaraniemi et al remove labels in two situations, namely if a label moves too fast or if a label is overlapped such that the forces acting on it cancel each other. While we label only the active streets, Vaaraniemi et al label all types of objects within the view. As they resolve overlaps by moving and selecting labels, we expect a lot of changes on the screen, which may be distracting (for example, for a car driver using a GPS). In terms of speed, Vaaraniemi et al report that their algorithm computes the layout for 512 objects within 5.5ms (this corresponds to 180 FPS *without* rendering.)

Gemsa et al (2013) investigate the off-line version of a point-labeling selection problem (in 2D) with respect to an active route. They assume that the entire active route is given in advance and fixed (while it may change in our case). They do not use leaders, but they assume that each label has a fixed position relative to the point it labels. As in our setting, they have a fixed camera above the active route and a map that turns and translates such that the direction of movement appears to be upwards/North. Each label may be visible during several intervals. In order to reduce flickering, for each of these intervals, Gemsa et al allow for selecting a *connected* (sub)interval in which the corresponding label is finally visible. The authors aim for an overlap-free labeling for the entire route that maximizes the total length over all selected (sub)intervals.

The authors show that their problem is NP-hard. They present an exact algorithm (an integer linear program that solves the problem optimally; in exponential time). They test their algorithm on 1,000 active routes at three different scales. On average, they need less than a second for optimizing the labeling of routes with about 162 labels and less than six seconds for routes with about 313 labels. A few routes,

Table 1 Our approach compared to some related work

	interaction types	history	computation time	mode	technique
Vaaraniemi et al	pan, zoom, rotate, 3D	considered	5.5ms per update	map	force-based
Maass and Döllner	*unknown*	with workaround	”real time“	map	ILP, greedy
Gemsa et al	pan, rotate	considered	sec to min	GPS	greedy
our approach	pan, zoom, rotate, 3D	considered	$>$ 400 FPS	GPS	force-based

GPS navigation mode, *FPS* frames per second, *sec* seconds, *min* minutes, *greedy* greedy algorithm, *ILP* integer linear program.

however, took them several minutes. The authors also give efficient approximation algorithms, but do not include any test results.

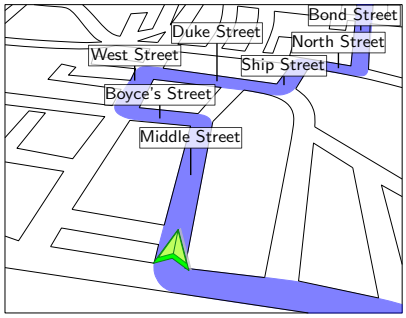
Maass and Döllner (2006) place billboards in interactive 3D maps that also allow for 3D objects (such as buildings). They require that the further away the labeled object is from the user, the smaller the label and the higher the leader. Their algorithm subdivides the view plane into a grid and places labels incrementally. Each placed label blocks several surrounding grid cells for other labels. The algorithm does not consider the *history* of the labeling, that is, the labeling of frame f_i does no take the labeling of frame f_{i+1} into account but computes a new labeling from scratch. In order to avoid jumps that might confuse the user, the labeling does not change while the user interacts with the map. If the user stops interacting, labels smoothly move to their new positions. In contrast, we immediately react to user interaction while taking into account the labeling of the preceding frame.

We also pursue another concept to label streets: we introduce an incremental algorithm for placing embedded labels into streets in interactive maps (Schwartges et al, 2014). Our aim is to maintain an overlap-free labeling where as many streets as possible are labeled. We avoid placing labels on street parts that contain strong bends. To this end, we introduce a cost function with which we evaluate any interesting label position on each of the visible streets. Such an embedded labeling complements the labeling of active streets.

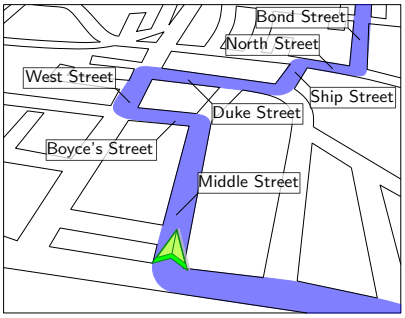
Maass et al (2007) present the results of a user study dealing with labelings in 3D maps using billboards. The authors examine the problem of leaders inducing wrong depth cues; for instance, a leader, whose reference point obviously lies behind a building, is drawn over the building. Most of the participants of the user study judge depth cues that are accurate as more comfortable. The authors finally suggest introducing a parameter that measures the perceivable perspective disturbance. If the parameter is applied in labeling algorithms, it is intend to improve the label placement, but, simultaneously, it sometimes permits wrong depth cues. In our approach, we always aim for correct depth cues.

Similarly, Vaaraniemi et al (2012) describe an expert study. The first result is that labels in a 3D view should shrink with distance to the user, in order to create a better understanding of the depth. We take this result of their expert study into account in our implementation.

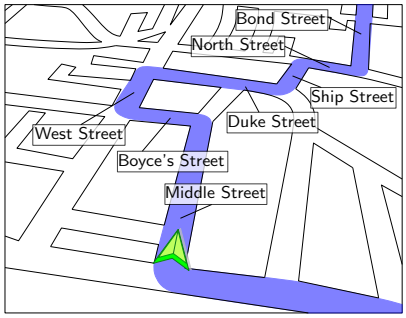
Moreover, Vaaraniemi et al ask the experts if streets should be labeled embedded, straight-line aligned, or horizontally. Four of six experts judge horizontally-placed labels as very legible, although they also note that a high search time is required to associate a label with its object. Again, we follow this judgement by using horizontal labels; we improve the label–object association by using leaders. On the other hand, five of six experts like the embedded labeling, but also point out that if labels are situated in strong bends, they might be badly readable. Three of six experts observe that embedded labels yield a good label–object association.



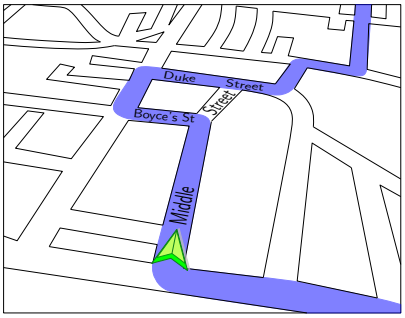
a vertical leaders



b leaders are rotated by $k \cdot 45°,\ k \in \mathbb{N}$



c arbitrarily-rotated leaders

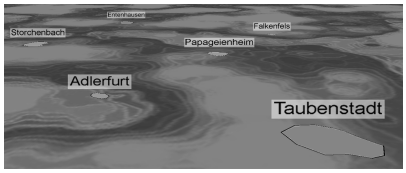


d embedded labels

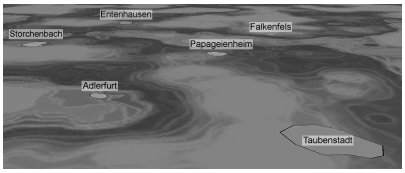
Fig. 2 Figures similar to those we have shown the participants of our survey. (We do not have permission to publish the original figures. They differ in that we removed the map background and the embedded labels of inactive streets)

3 Survey

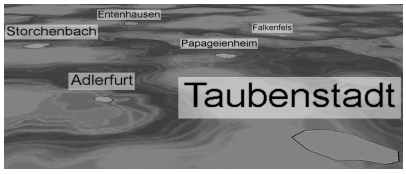
We conducted a small survey in order to decide which variant of our concepts to implement first. In total, we had 19 participants (one female), aged between 19 and 26 years, all of them studying a technical subject. We asked the participants which of the labelings shown in Fig. 2 they like most and which of them they think is the most practicable for navigation systems. Moreover, we asked which of the labelings shown in Fig. 3 they like most. We found out that 47% preferred the labeling in Fig. 2(d) and that this solution is also considered most practicable by 53%. The next best rated possibility was Fig. 2(a) with 32% for both questions. As we have already published our work about labeling streets in interactive maps using embedded labels (Schwartges et al, 2014), in this work, we tackle the problem of labeling streets using billboards. To conclude, 47% preferred the labeling in Fig. 3(a). We give the remaining numbers in Tab. 2.



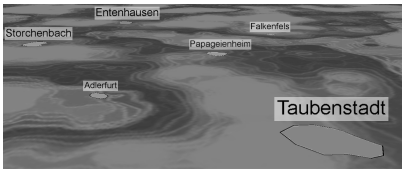
a constant size in world space, distance dependent in screen space



b constant size in screen space



c population dependent in world space, distance dependent in screen space



d population dependent in screen space

Fig. 3 Figures we showed to the participants of our survey

Table 2 Results of our survey

	aesthetics	practicability
	%	%
Fig. 2(a)	32	32
Fig. 2(b)	21	11
Fig. 2(c)	0	5
Fig. 2(d)	47	53

	aesthetics
	%
Fig. 3(a)	47
Fig. 3(b)	16
Fig. 3(c)	0
Fig. 3(d)	37

4 Algorithm

In this section, we propose a simple force-directed algorithm for preventing labels from *occluding*, that is, overlapping, other labels. There are several auxiliary algorithms that are needed: setting up the environment, reading and routing through a street map, and so on; we will not cover these algorithms in depth.

Consider a street map which contains an active route from a starting location A to a destination location B. The active route is a sequence of *street polylines*, where each street polyline is only a subsection of the entire street, in the sense that one traverses a street for only so long as to reach the intersection that leads to the next street. We shall use the terms street polyline and street interchangeably. The route is traversed by a *pointer* π which represents the user, and is typically modeled as a triangle or vehicle (see Fig. 4). The camera is placed at some distance behind the pointer.

Our goal is to label the individual streets and prevent labels from occluding each other. We place labels (or rather billboards) in world space, each above the refer-

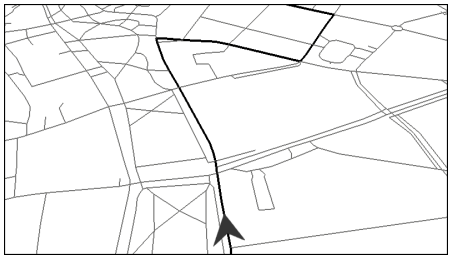


Fig. 4 A street network with a route and a pointer

ence point of its street. For the sake of simplicity, we assume that each street has only one reference point, namely at the midpoint of the polyline. Hence we label a street only once, regardless of its length. (It is of course not difficult to overcome this restriction.) Recall that we connect each label to its reference point with a vertical leader whose length can dynamically vary. We denote by h_0 the *default leader height*, which is the desired height. A label can move only vertically, by extending or contracting the leader. To simplify further discussion, let us consider each billboard as a complete entity, denoted by λ_i, where i denotes the i-th reference point on the route. The actual height of the leader of the billboard λ_i at any given time t is denoted by $h_i(t)$.

Given the route R, let $N := |R|$ denote the number of streets along the route. It makes little sense to display every label for every street in the route at the same time, as the user is primarily interested in the next few streets ahead. Therefore, we upperbound the number of *placed* labels at any time to a constant $n \leq N$. Let $I = \langle l, \ldots, m \rangle$ denote the queue of currently placed labels; we have $|I| \leq n$. (We have $|I| < n$ if the remaining part of R consists of fewer than n streets.) When the distance of the pointer π from the reference point q_l of label λ_l falls below a threshold ε, then label λ_l is dequeued from I, and label λ_{m+1} (if it exists) is enqueued. Note that the number n of placed labels is sometimes greater than the number of *visible* labels, that is, the labels within the view. We also place labels that lie outside the view, in addition to visible labels, for two reasons. First, we can elide checking if a label is about to enter the view. Second, when labels do enter the view, they do not disturb the visible labeling much, as they have already been considered by the algorithm.

4.1 Force-Directed Approach

We propose a simple force-directed algorithm. Forces are exerted on each label by other labels and by its own leader; these forces cause the label to move in a way minimizing the aggregate force. The leader acts as a spring, keeping the label close to its reference point, while the labels repel each other much in the same way that same-pole magnets do. Excess aggregate force is mapped to a change in leader height. To prevent a label from oscillating strongly between two other labels, the

force is scaled by a temperature, which is reduced when the aggregate force changes its *sign* from one iteration to the next.

While the labels themselves live in 3D *world space*, we see them in a projection onto 2D *screen space* (see Fig. 5). Certain calculations, such as determining whether two labels overlap, are therefore performed in the screen space. The screen-space representation of a label is a rectangle; we exclude the leader from this rectangle. We determine if two labels overlap by inspecting whether their screen-space projections overlap. To ease further discussion, we refer to the label in screen space simply as the label projection and again denote it by λ_i.

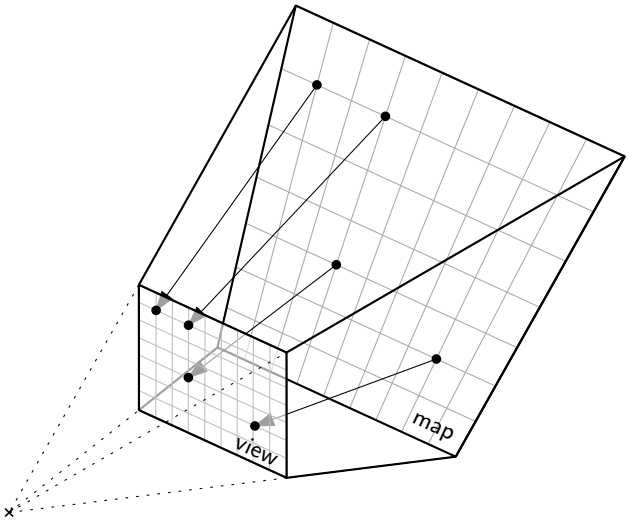


Fig. 5 Projecting points lying in the currently visible part of the map from world space to screen space

One last note before we move on: we say the algorithms in this section are *frame-based*. In each new frame, pointer and camera may have moved, labels may be seen from a different perspective, labels may have been moved up or down by their leaders, auxiliary algorithms take the new data into account, and the force-directed algorithm runs. Frames may coincide with rendered frames or they may be timer-based. The only requirement is that changes in leader height are reflected in the screen-space projection in the next frame. Since the information the algorithms need is primarily for the current frame, we elide the time t, so that for frame-bound value, say $h_i(t)$ for example, we write $h_i := h_i(t)$.

In the following, we discuss how the various forces are computed and how they change the leader height.

4.2 Spring Force

The leader of a label is modeled as a simple tension spring, which has a default height of h_0. A spring can undergo extension as well as contraction, which is negative extension. The force is given by Hooke's law, and is simply a spring constant k multiplied with the leader extension:

$$F_i^{\mathrm{s}} := -k \cdot (h_i - h_0)\,. \tag{1}$$

Due to the way the spring force flows into the overall force acting upon a label, the main effect k has is to affect the speed at which labels return to their default height: the greater k is, the stronger the force.

4.3 Aggregate Repulsive Force

Principally, the aggregate repulsive force F_i^{r} for the label λ_i consists of the sum of all repulsive forces $r(i,j)$ between λ_i and every other placed label λ_j:

$$F_i^{\mathrm{r}} := \zeta \cdot \sum_{j \in I \setminus \{i\}} r(i,j). \tag{2}$$

This is a slight simplification, as we shall see in Sect. 4.4, but for initial understanding it is sufficient. The constant ζ serves to weight the aggregate repulsive force.

The function r relies on several concepts we will introduce first: the relevance of labels, the sign of labels, the distance metric, and the interplay between labels.

Sign Let $\mu(i)$ be the midpoint of the projection of λ_i and let $\mu_y(i)$ be the y-coordinate of $\mu(i)$. Let the sign $\sigma(i,j)$ of a label λ_i denote whether it is above or below another label λ_j, $i \neq j$:

$$\sigma(i,j) := \begin{cases} -1 & \text{if } \mu_y(i) < \mu_y(j), \text{ and} \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

We can categorize each λ_j in one of three categories in regard to λ_i: either the midpoint of λ_j is below, above, or at the same height as that of λ_i. The function σ however, handles only two cases. One might ask: might not two labels, in case of midpoint equality, move upwards at the same rate and thus arbitrarily far from their reference point? We think this cannot happen, because multiple factors affect the change in height, and these factors cannot all be the same, unless $\lambda_i = \lambda_j$, which we disallow. Hence one label shall travel further, and in the next frame the midpoints are different.

Distance Metric We can model the repulsive force between two labels as a function of their distance to one another. This is an attractive model, as it is easy to reason about, but it leaves us with the problem of defining a useful distance metric δ.

Not all labels need repel each other. We have already ascertained that labels have only one degree of freedom, namely in the vertical axis (y-axis). It is reasonable therefore to allow labels to only affect each other when a movement would make sense. This occurs only when labels intersect in the horizontal axis (x-axis). In our distance metric then, labels that do not intersect in the horizontal axis have a distance of ∞.

What when the labels overlap? Distance between typical objects is measured starting from 0, when the objects are right next to each other. To deal with this, we can redefine a distance unit to equal the height of a label projection. Let $\omega(i,j) \in [0,1]$ denote the percentage that λ_i is occluded by λ_j, that is, $\omega = 1$ iff λ_i is completely covered by λ_j. Note that ω is asymmetric. Further, let $\gamma(i,j)$ denote the y-offset between λ_i and λ_j, in the case that they do not overlap. This leads us to the following definition of δ:

$$\delta(i,j) := \begin{cases} \infty & \text{if } \lambda_i \text{ and } \lambda_j \text{ do not intersect in } x\text{-axis,} \\ 1-\omega(i,j) & \text{if } \omega(i,j) > 0, \text{ and} \\ 1+\gamma(i,j)/h_i & \text{otherwise.} \end{cases} \tag{4}$$

Note that this definition is somewhat inconsistent, because $\delta \in [0,1]$ is a measure of area, while $\delta > 1$ a measure of height. This is intentional. If two labels overlap by one pixel in the horizontal axis, they cannot by the above Eq. 4 have a distance of 0. This behavior is useful, as it differentiates between full and partial occlusions.

Resulting Force When the label λ_i is fully occluded by λ_j, we let the force be a constant F_{limit}; when $\delta = 1$, we let the force equal F_i^s. The force grows quadratically for $\delta < 1$, and linearly when $\delta \in [1,2]$. Finally, we restrict that labels that have $\delta > 2$ do not affect each other. Any label farther away need not have an effect, and if it should come closer, then at some point $\delta \leq 2$ will hold. We describe the entire algorithm that computes the resulting force $r(i,j)$ that λ_j exerts upon λ_i in Algorithm 1.

The function r is monotonic and continuous, despite potentially flattening out before $\delta = 0$. Fig. 6 shows the graph of r. We could define the function to approach F_{limit} when $\delta \to 0$, however it would provide only questionable benefit while requiring more computational power.

This function definition has the fortunate property that the force acting upwards on label λ_i from a lower label λ_j will equal the spring force pulling λ_i down precisely when λ_i is right next to λ_j. This provides just the right amount of force to prevent occlusions, while allowing labels to return to their default height when possible.

Algorithm 1: ResultingForce

input : labels λ_i and λ_j
output: force that λ_j exerts upon λ_i

$d \leftarrow \delta(i,j)$
if $d = 0$ **then**
  return $\sigma(i,j) \cdot F_{\text{limit}}$
else if $d \leq 1$ **then**
  $v \leftarrow r \cdot \left(1/\delta(i,j)^2 - (1 - F_i^{\text{s}})\right)$
  if $v > F_{\text{limit}}$ **then**
    return $\sigma(i,j) \cdot F_{\text{limit}}$
  else
    return $\sigma(i,j) \cdot v$
else if $d \leq 2$ **then**
  return $\sigma(i,j) \cdot F_i^s \cdot (1 - \delta(i,j))^2$
return 0

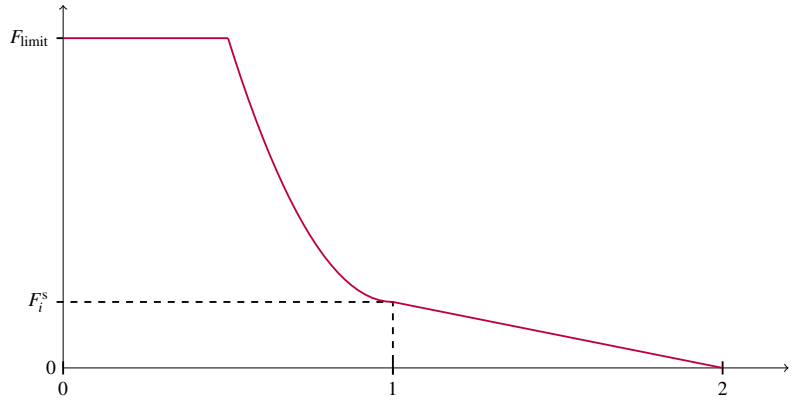


Fig. 6 The graph of function r, with the distance between two labels in the x-axis

4.4 Aggregate Force

Intuitively, the overall force F_i acting on a single label λ_i consists of the sum of the repulsive forces F_i^{r} (with constant factor ζ) and the spring force F_i^{s}:

$$F_i := F_i^{\text{s}} + F_i^{\text{r}}. \tag{5}$$

This turned out to be too simplistic. What happens to λ_2 when three labels λ_1, λ_2, and λ_3 are stacked? If λ_1 exerts the same force on λ_2 as λ_3 does, then the forces cancel each other out and the remaining force is F_2^{s}, which causes λ_2 to descend into λ_3, thereby causing a collision.

Hence we need to keep positive forces F_i^+ and negative forces F_i^- separate, so that we can handle this case specially:

$$F_i^+ := \sum_{j \in I \setminus \{i\},\, r(i,j)>0} r(i,j)$$
$$F_i^- := \sum_{j \in I \setminus \{i\},\, r(i,j)<0} r(i,j).$$

Since if $r(i,j) = 0$ it affects nothing, we can ignore it in the above definitions. With now two sets of repulsive forces, we include F_i^s iff at least $F_i^+ = 0$ or $F_i^- = 0$. The formula is now

$$F_i := \begin{cases} F_i^s + F_i^+ + F_i^- & \text{if } F_i^+ = 0 \text{ or } F_i^- = 0, \text{ and} \\ F_i^+ + F_i^- & \text{otherwise.} \end{cases} \tag{6}$$

If a label is surrounded by two labels, the spring force is thus ignored.

4.5 Temperature

For each label λ_i, we define a temperature T_i that acts as a factor in scaling the force. The default temperature is defined by a constant T; this is the temperature that labels have when first placed. We track the aggregate force of the label between two frames; if the sign of the force changes, we reset the temperature to a constant T_{base}. To prevent two labels from jumping too quickly away from each other, a negative force (pushing the label downwards) counts as a sign change from a force of 0. If the sign remains the same, we multiply the temperature by a constant T_{step}, thereby increasing the temperature slightly. In order to prevent the temperature from becoming arbitrarily large, we limit it to a constant T_{limit}. It is assumed that $T, T_{\text{base}} > 0$ holds.

4.6 Leader-Height Change

The combination of aggregate force and temperature let us derive either an extension or contraction Δ_i of the leader of label λ_i, which we can (again) scale by a constant factor χ:

$$\Delta_i := \chi \cdot T_i \cdot F_i. \tag{7}$$

Because of rounding error, it is unlikely that F_i will ever reach an equilibrium between different forces. To prevent labels from always moving, we only apply a leader-height change when F_i is greater than a constant F_{min}.

4.7 Complexity and Runtime

The setup time complexity for all the algorithms is linear in the size of the input. We consider the complexity of the algorithm for a single frame. The auxiliary algorithms

require at most $O(n)$ time, where n is the maximum number of placed labels in a frame.

The force directed algorithm considers each of the labels in I, and performs for each the following:

1. Calculates the spring force in $O(1)$ time (Eq. 1).
2. Calculates the repulsive force in $O(n)$ time (Eq. 2 and Alg. 1).
3. Calculates the leader change and applies it in $O(1)$ time (Eq. 7).

Thus we have a runtime of $O(n^2)$ in total. Since we control the value of n, we can determine the maximum runtime of the algorithm in each frame. This is especially useful for embedded applications. In practice, letting $n = 10$ seems sufficient for us and has next to no negative impact on performance. As we show in our experiments, even with a quite large n, we obtain interactive frame rates.

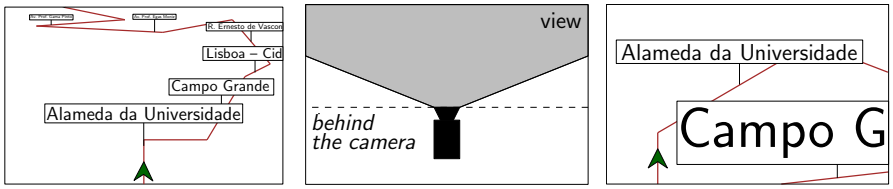
4.8 Implemented Improvements

In our description of the force-directed algorithm, we constrained ourselves to the essential essence of the algorithm. Here we present two additional modifications. None of the modifications change the complexity of the algorithm.

Margins In our problem definition, labels must, at least, not overlap. Sometimes it is desirable that they maintain a margin of separation from each other. To this end, we give each label a bottom margin of ν by extending the screen-space projection of the label by ν units (for example pixels). This margin is not visible to the user.

Relevant Labels The queue I of placed labels is maintained by an auxiliary algorithm. However, there are labels for any given frame which are not necessarily relevant to the force-directed algorithm or even might cause undesired behavior. We remove such labels from I. Any label that is removed has its values reset to its defaults. Finally, in each iteration, we use $\hat{I} \subseteq I$ as the set of *relevant* labels; we define $n' := |\hat{I}|$.

Some labels may be so far away or out of sight, that to include them in force calculations would seem unnecessary (see Fig. 7(a)). We define the following optimization then, in which we effectively ignore from I any label that has an area less than ξ. The label is still rendered, but it is not considered in any force calculation. Further recall that our labels are placed in world space. To this end, some labels lie behind the camera (see Fig. 7(b)). This might cause problems when projecting them from world space to screen space. We remove labels lying behind the camera from I and we do not render them. Last, if a future part of the route is very near to the user, some labels become undesirably large (see Fig. 7(c)). Whenever the area of a label becomes larger than some value α, we remove the label from I and do not render it.



a labels in the far back are too small to read

b labels behind the camera might cause wrong projections

c an adverse course of the route causes a very large label

Fig. 7 Relevant and nonrelevant labels (note: these are no screenshots!)

5 Experiments

We have implemented our force-directed algorithm from Sect. 4, a testing environment, and a *static* algorithm for comparisons. For our implementations and tests, we used C++ with OpenSceneGraph 3.1[3] and Boost C++ Libraries 1.56[4] on Linux 3.16 with a 2.5-GHz Intel dual-core processor, 8 GB of RAM, and an Intel HD3000 integrated graphics card. We applied the GCC-4.9.2 compiler to produce 64-bit binaries with compiler optimizations. For the testing environment, we used an OpenStreetMap data set provided by Geofabrik[5] from which we extracted the downtown street network of Würzburg, a town of 120,000 inhabitants in southern Germany. In order to simulate the navigation mode, we determined three different routes through Würzburg along which we created a camera path each. We tested $n = 10$, 25, and 50. We think, however, that $n = 10$ is the most reasonable value for the rather small displays of navigational devices. Our virtual navigation system had a resolution of 1,366×768 pixels. In order to maintain n placed labels for each frame and to guarantee paths of equal lengths, we stopped the camera paths as soon as $|I| < 50$, that is, when the remaining part of the route had less than 50 labels left. In total, we placed $N = 69$, 96, and 131 labels in the first, second, and third path, respectively. (Note that number of actually placed labels is $N - 50 + n$.) The paths needed between 18s and 68s. In our tests, we only panned and rotated the map whereas we rotated 13% of the total time. (We do not expect that the frame rate drops for the remaining interactions as there is no special handling for the different interaction types—neither in our algorithm nor for rendering.) In the camera paths for our tests, the pointer had a constant position and direction on the screen (as in Fig. 7). For rotations, the pointer stopped its drive and rotated; then it continued the drive.

Our set of configurable values yielding nice-looking, smoothly moving labelings is as follows: $h_0 = 5.0$ units in world space[6], $k = 0.25$, $\zeta = 1.0$, $F_{\text{limit}} = 5.0$, $\chi = 0.2$,

[3] `http://www.openscenegraph.org/`, accessed Nov. 28, 2014

[4] `http://www.boost.org/`, accessed Dec. 4, 2014

[5] `http://download.geofabrik.de/`, accessed Nov. 28, 2014

[6] For comparisons: we set the font size of the label text in world space to 3 units.

$(T, T_{\text{base}}, T_{\text{step}}, T_{\text{limit}}) = (1.0, 0.1, 1.05, 5.0)$, $F_{\min} = 0.1$, $\xi = 100$ pixels, $\nu = 5$ pixels, $\alpha = 0.25$ of the total resolution, $\varepsilon = 10.0$ units in world space.

For the static algorithm, we just fixed the leader lengths to h_0. We ran the same camera paths as for the dynamic labeling algorithm.

Figure 8 shows some screenshots of our algorithm. When we start the algorithm, the leader lengths are equal. In this example data set, the overlap resolves within two seconds. In Tab. 3, we summarize our results for the force-directed algorithm and the static algorithm, both applied to the real-world data set.

Fig. 8 Screenshots of our program in an artificial data set. *In reading direction*: Within 1.5s, the overlap of the initial label placement is resolved. On the very first subfigure, the total overlap is 6773 pixels

Table 3 Results of our experiments for the static and the force-directed algorithm. For the frame rate, we divided the number of frames by the running time in seconds; for the remaining values, we averaged the numbers over the number of frames. For each measurement, we assured that at least n streets were still ahead

	static			**force-directed**	
	frame rate	overlap		frame rate	overlap
n	FPS	px	n'	FPS	px
10	453	3,190	8	452	49
25	444	4,930	19	444	91
50	435	5,980	34	431	116

n placed labels, n' relevant labels, *FPS* frames per second, *px* pixels.

In order to evaluate the processing time of our algorithm, for each path and each value of n, we measured the number of totally drawn frames as well as the total running time. By these two values, we computed the *frame rate* as the number of

drawn frames per second (*FPS*). Table 3 shows the frame rates for each value of n, averaged over the three different paths.

Our algorithm yields very good frame rates of more than 400 FPS when it places $n = 50$ labels or less. If we only render the active route, the billboards, and the pointer (in other words, we do not render the street network), the frame rate increases by about 54 FPS. On the other hand, Tab. 3 shows that the frame rates of the static algorithm are only better by a few frames compared to the force-directed algorithm. This is due to the fact, that, for the static algorithm, we have stretched the limits of what OpenSceneGraph combined with the integrated graphics card can achieve. To verify this, we also tested for *one* map the frame rate while OpenSceneGraph idled: without any computations or rendering but with loading the map and the route, we reached frame rates of about 450 FPS at our system.

For each path and each value of n, we recorded also the number of overlapped pixels. We counted only the overlaps of relevant labels in the view. We divided the total number of overlapped pixels by the total number of frames. Table 3 shows that, compared to the static algorithm, we could reduce the number of overlapped pixels per frame by about 98% by applying our force-directed algorithm.

Unfortunately, we cannot compare our results to Maass and Döllner (2006) as they only state that their algorithm "operates in real time". Similarly, we cannot compare to Gemsa et al (2013) as they compute the entire labeling in advance, that is, the frame rate is determined by the rendering algorithm only but not by the labeling algorithm. Vaaraniemi et al (2012) state that their algorithm has a frame rate of 180 FPS for computing label positions if they disable the rendering. If we only render the important part of our visualization, that is, the route, the labels, and the pointer, for $n = 50$, we obtain frame rates of almost 500 FPS. In general, however, a frame rate of 24 FPS is qualified fluid. Thus we conclude that our algorithm for placing billboards to active streets in an interactive navigation mode for 3D maps yielding frame rates of more than 400 FPS is highly real-time capable and it is really worth trying to combine it with an algorithm that labels the remaining streets embedded.

6 Conclusion and Future Work

We have introduced a force-directed algorithm for placing billboards with leaders of dynamically varying lengths to active streets in interactive 3D maps. In our approach, each reference point tries to keep its corresponding leader at a desired length; overlapping labels repel each other. From frame to frame, we minimize the unbalanced forces. This yields labelings that avoid label–label overlaps of billboards that are near to the user but accepts overlaps in the background. Our algorithm directly reacts to changes of the current view by smoothly moving overlapping labels. In our tests on real-world data with a realistic number of labels, our implementation reached interactive frame rates of more than 400 FPS and reduced the number of overlapped pixels compared to the static algorithm by about 98%.

In the future, we plan to support multiple labels per street, different anchor points, and different leader directions. Most importantly, we plan to combine our algorithm for placing billboards along the active route with our algorithm that embeds the labels of the remaining streets into these streets. The challenge will be to make sure that the combined algorithm is fast enough for real-time interaction. It would also be interesting to verify the findings of our survey (in which we used static figures) by means of a user study that provides interactive scenarios.

References

Eades P (1984) A heuristic for graph drawing. Congressus Numerantium 42:149–160

Gemsa A, Niedermann B, Nöllenburg M (2013) Trajectory-based dynamic map labeling. In: Cai L, Cheng SW, Lam TW (eds) Proc 24th Int Symp Algorithms Comput (ISAAC'13), Springer, LNCS, vol 8283, pp 413–423

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

Larson K, van Dantzich M, Czerwinski M, Robertson G (2000) Text in 3D: Some legibility results. In: Begole J (ed) Proc 18th ACM Conf Human Factors Comput Syst (CHI'00), pp 145–146

Maass S, Döllner J (2006) Efficient view management for dynamic annotation placement in virtual landscapes. In: Butz A, Fischer B, Krüger A, Oliver P (eds) Proc 6th Int Symp Smart Graphics (SG'06), Springer, LNCS, vol 4073, pp 1–12

Maass S, Jobst M, Döllner J (2007) Depth cue of occlusion information as criterion for the quality of annotation placement in perspective views. In: Fabrikant SI, Wachowicz M (eds) The European Information Society – Leading the Way with Geo-Information, Springer, Lect Notes Geoinform Cartogr, pp 473–486

Schwartges N, Wolff A, Haunert JH (2014) Labeling streets in interactive maps using embedded labels. In: Proc 22nd ACM SIGSPATIAL Int Conf Advances in Geogr Inform Syst (ACM-GIS'14), 4 pages

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

Wigdor D, Balakrishnan R (2005) Empirical investigation into the effect of orientation on text readability in tabletop displays. In: H Gellersen et al (ed) Proc 9th Europ Conf Comput Supported Cooperative Work (ECSCW'05), Springer, pp 205–224