Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Effiziente Algorithmen und wissensbasierte Systeme

Bachelor's Thesis

Placing Street Labels in Interactive Navigational Maps

Benjamin Morgan

October 30, 2012

Supervisors:
Prof. Dr. Alexander Wolff
Dr. Jan-Henrik Haunert

Contents

Chapter I

Introduction

1 Motivation

Personal Navigation Devices[1] started appearing in the mid-eighties and have been proliferating in recent years [VFW13]. According to a report published by Berg Insight [Ins12] in 2012, there are over 340 million such navigation devices in use today. Indeed, navigation devices can be extremely useful in today's high-mobility world.

Navigating in a foreign city, especially one of greater proportions, can nonetheless be a very daunting task even with navigational help. It is therefore important that the transfer of information from the navigation device to the user is efficient. Consider the confusion that can arise from inaccurate map data, bad routing algorithms, and significant lag. These components are always going to be there; the designer has a responsibility to minimize their effects as much as possible.

There are many aspects of these systems we could begin to optimize, in order to streamline the user experience: the graphical user interface, the accuracy of map data, the visualization of the map data, the voice of the assistant, the GPS technology, etc. Clearly, there is still room for improvement in each of these areas. One aspect, however, which has received relatively little attention is street labeling; this shall be the focus of my thesis.

Current navigation devices render a minimal number of labels on the display, perhaps on average five[2]. In some devices, labels are not shown at all except for important streets (such as the German *Autobahn*). If labels are shown, they are almost always rendered as an element parallel to the screen, without a *leader*. A

[1]The term personal navigation device is also known as "personal navigation assistant" or "GPS navigation device". Colloquial usage includes for example the terms "Navi" in Germany and "GPS" in the United States.

[2]It is not within the bounds of this thesis to perform an extensive empirical study of the current state-of-the-art personal navigation device technology.

leader is an element, such as a line, visually connecting the label to the object; for example, a triangular arrow can be a leader. If not rendered onto the map, the current street is almost universally shown in a bar at the bottom or top of the screen.

Placing labels outside of the scene, and connecting them to their elements with the use of leaders is called *boundary labeling*; important in boundary labeling is that the leaders to different points are not allowed to intersect [GHN11]. Boundary labeling has interesting applications, but has never really been used for navigation devices before.

Street labels are delicate elements, which may explain the conservative use of them. The difficulty of optimally placing and rendering labels rises considerably with the number of labels to be rendered. In addition, the more labels displayed, the more potential there is for the user to be overwhelmed and confused. There are many aspects which need to be considered when designing or choosing an algorithm for placing street labels.

2 Goal

The goal of this thesis is to explore ways and means to efficiently and dynamically manipulate and place street labels on the active route of a navigation system environment, such that the labels do not overlap with one another. In particular, we shall consider how varying the leader length affects the solution, and how a *force-directed* algorithm can achieve this goal (force-directed algorithms will be discussed in Section 3.4). It is not within the scope of this thesis to solve the problem of where and how many labels are *initially* placed.

Why is the goal not simply to efficiently, *optimally*, and dynamically manipulate and place street labels on the active route of a navigation system? The difficulty lies in the word "optimally." What is optimal? Measuring the quality of dynamic street-label placement is a highly subjective task; although it is very desirable, it is difficult to correctly define what the *optimum* is.

Even if we could define the optimum, the goal may still be effectively unreachable. We must be careful with trying to reach an "optimal" solution, because for many problems in computer science—such as NP-hard problems—attaining an optimal solution is computationally so difficult that it is considered unattainable. For example, given a list of cities and their pairwise distances, the *traveling salesman problem* consists in finding a shortest route where each city is visited exactly once. This problem is known to be NP-complete; in the worst-case, computational time rises exponentially with the number of cities [Pap77]. In Section 3, it shall become more evident how difficult the street-labeling problem is.

Nevertheless, having a definition of an optimal solution is desirable, because it provides us with a plumb-line against which we can measure any solution. For this reason, we shall below provide a suggestion for what might be considered an (important part of an) optimal solution. Let us first briefly consider the significance of the terms "environment" and "efficiency" in the above stated goal.

Environment

The environment is that of an interactive navigation system. The environment may have 3-dimensional (3D) elements, such as buildings, but the surface is 2-dimensional (2D). Even if there appear to be no 3D elements, the environment is still effectively 3D because of the need for readable labels with a changing perspective. The viewpoint in the environment is free, so perspective changes must be taken into account.

Efficiency

The software running in such navigation systems is required to render the environment in real-time and therefore should achieve frame rates of at least 24 fps. Navigation systems are embedded and have more limited access to resources than personal computers, so the required resources must be kept minimal.

A Suggestion for the Optimum

The label placement should come as close to optimal as possible. We define what is optimal by defining criteria which each label placement should fulfill and by imposing a penalty on each placement otherwise. Thus a label placement is optimal when the sum of all the penalties is minimal. The set of criteria might look like this:

1. Visual association between a label and its feature should be guaranteed. It should be clear which street the label *Würzburger Straße* is referring to.

2. Depth cues of the virtual world should be preserved. The label of a street 400 m away should not occlude or visually come before the label of a street only 50 m away.

3. Labels should support spatial orientation. If the perspective changes, the labels should still be readable (not flipped backwards or sideways).

4. Label movement between frames should be minimized. We don't want the inclusion of a new label to dislocate all the other labels already on the screen—that would irritate and confuse the user of the device.

5. The user of the system should
    (a) be able to recognize and process the information quickly and easily,
    (b) not be confused, and
    (c) not be overwhelmed.

3 Theory and Related Work

3.1 Labeling in Cartography

The classic label placement problem dates back to the early days of map-making. Map-making was practiced by the Babylonians as early as the 9th century B.C. [RT10], so this is not any kind of new territory we are covering. However, automated and computer-aided map-labeling is a relatively new area of research and development, and of great use: according to Cook and Jones [CJ90], labeling the different elements of a printed map took around 50% of the production time. Although the techniques used in map labeling are not the same that we will need for interactive street labeling, the theory behind map labeling will prove useful to us.

There are three different kind of *features* that are generally identified with label-placement in cartography: area features (such as countries, deserts or oceans), line features (such as rivers or roads), and point features (such as cities or ports) [CMS95]. We refer then for example to *line-feature label placement* (LFLP) and *point-feature label placement* (PFLP) as problems that one might want to solve.

Christensen et al. [CMS95] point out that although the task of optimally placing a label for a point feature is significantly different to that of optimally placing a label for a line or area feature, they all share a certain combinatorial aspect when there are multiple features to be labeled. The placement of a single label can have global consequences with respect to the labeling of the other features. It has been well established, that even simple cases of PFLP are NP-hard; although LFLP has received much less complexity analysis than PFLP, Christensen et al. continue in their discussion of algorithms for PFLP, noting that this is "without loss of generality." LFLP, we conclude, is computationally as difficult as PFLP. As street label placement is a special case of LFLP, and because of the combinatorial aspect, there is a high chance that it is also computationally too difficult to solve optimally. We shall therefore restrict ourselves to a heuristic or an approximation approach.

Navigation devices use maps—they navigate through maps—that is their *raison d'être*. So it seems reasonable to assume that static LFLP in cartography has everything to do with dynamic street labeling. After all, we are labeling line features! However, the differences and the constraints imposed through *interactive* use in a *3D* environment is enough to render the algorithms known for static PFLP and static LFLP inapplicable for our purposes. The goal of labeling in cartography is generally to maximize the labels displayed while minimizing the occlusion of elements and other labels. Neither of these criteria are directly compatible with the criteria for street labeling: we do not want to see as many labels as possible (that would confuse us); in some situations, a label might be allowed to occlude a more distant label (that would be a depth cue for us).

3.2 Imhof's Label-Placement Rules

In the "old days," cartographers would judge the quality of map labeling by their taste and experience. The foundation of automated label placement was first laid when the Swiss cartographer Eduard Imhof (and independently the French cartographer Georges Alinhac) formulated a set of rules, along with examples, which enabled one to objectively measure the quality of label placement on a map [Wol99]. This development let researchers who had little experience in cartography develop algorithms for automatic label placement.

Agarwal et al. [AKS+00] studied Imhof's label-placement rules applied to line labeling and categorized them into *hard* and *soft* constraints. Hard constraints represents the minimum requirements which must be kept, while soft constraints are those that should be kept (if possible). The constraints and the algorithm developed by Agarwal et al. are however primarily applicable to the static LFLP problem, and therefore less useful to us in this case.

3.3 Strijk's Label-Placement Rules

Strijk [Str01] "distills" an extensive set of street-label placement rules through studying how cartographers label city maps; these can be used for automatic static labeling. Strijk also makes some astute observations on different ways that text can be manipulated to make it easier to work with (or place in a smaller area):

- Long label text can be stacked to reduce the length. For example, the label "John F. Kennedy Street" can be written out in one line, or in up to three lines.
- Label text can be broken up or abbreviated. The above street name may also be recognized as "J.F.K. Street" or "J.F. Kennedy St."

Of course, this flexibility comes at a significant computational cost; it is therefore perhaps better applicable to static labeling, where there is no demand for interactivity.

Strijk sorts the labeling rules into three categories: association, visibility, and aesthetics. The following list is extracted and/or derived from Strijk's rules which are relevant to interactive street labeling.

Visual Association

1. A label is valid for the whole course of a street.
2. Label placement should avoid street intersections due to ambiguity.

Label Visibility

1. Labels should not overlap each other.
2. A label should not cross the edging of another street.

Aesthetics

1. Labels should never be upside-down.
2. Abbreviations should only be used when necessary.
3. Labels should not be repeated too often.

With some minor modifications, these rules can also be used in interactive street labeling. For example, while we want to avoid labels overlapping each other, it may not be as critical as it is for static labeling.

3.4 Force-Directed Labeling

Force-directed labeling methods were originally developed for drawing graphs [EKW03]. They view the input graph as a set of objects with repulsive forces between each other, as if each vertex were an electron that should avoid the other electrons. Giving objects low charges results in an appealing and structured visualization of the graph. These algorithms work well with medium-sized input, and are able to capture the natural symmetries present in the graph, without extra computation.

According to Ebner et al. [EKW03], force-directed methods for labeling maps were used as early as 1982, by Hirsch. Hirsch started with an initial label placement, and then through the summation of overlap vectors moved labels towards more open areas. All conflicting labels could then move successively towards better areas, through a series of iteration steps.

Ebner et al. [EKW03] identifies two primary methods of finding an equilibrium of forces in force-directed algorithms. When there is much freedom in where objects can move to, a vector of movement calculated directly from the summation of the forces can be used. Alternatively, a gradient-driven heuristic is employed, where the objects are in a preliminary stage moved into one of several pre-determined positions, often chosen at random; the net result of this new position is used to calculate a gradient, which determines where the label should be placed to reach the greatest improvement.

The primary drawback of force-directed labeling is that it is easily caught in local minima of the objective function. If labels are not allowed to overlap, then a label caught between other labels has no way of getting out into a potentially better position, unless it goes over other labels. This would cause an overlap however, and this is what the forces prevent.

In order to overcome this weakness of force-directed labeling, Ebner et al. [EKW03] use it in a combination with a technique called *simulated annealing*. Simulated annealing allows movement in directions other than that which the gradient or the force vector dictates; the solution is allowed to get worse before it gets better [CMS95]. The rate at which such inferior movements are allowed is controlled by a parameter called the *temperature of the solution*. The hotter the temperature, the more “anarchy” is tolerated. Thus the final labeling quality, as well as the computation time, can be affected by the speed of the cooling.

Simulated annealing is a useful technique for avoiding getting caught in local optima, but it is unsuitable for dynamic, interactive labeling, because it randomly allows movements that degrade the solution. Ebner et al. [EKW03] suggest force-directed labeling as an appropriate method for dynamic labeling, as it is very efficient. We are unaware, however, of any concrete studies performed in force-directed street labeling in navigation systems.

3.5 World and Screen Space

The way annotations are placed in a 3D world can be separated into two categories: *world space* (also called *object space*) and *screen space*. The difference is whether the labels themselves are subject to perspective transformations. The 3D world is projected through a particular perspective onto the 2D screen. World-space labels are part of the 3D world that is projected and are often embedded directly onto the object that they label, whereas screen-space labels appear to be placed on the screen after the projection. Figure 1 shows different labeling techniques as applied to street labeling.

Screen-space labels (see Figure 1a) are most commonly used when the representation is primarily 2D, such as a map. Screen-space labels are more legible, but they sometimes lose their association to the labeled object. This is where the use of a leader significantly helps maintain association. A further advantage of screen-space labeling is that the problem space is 2D—this makes it easier to detect

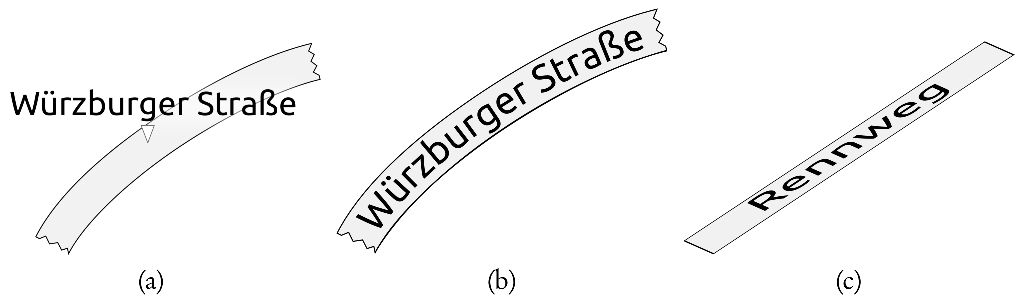


Figure 1: Different techniques for labeling a street: (a) typical screen-space label with leader, (b) screen-space label following the street, and (c) world-space label embedded in the street.

occlusions and position labels [DM07]. Sometimes, as in Figure 1b, screen-space labels are manipulated and adjusted to the objects in the projection.

World-space labels (see Figure 1c)—as the antipole to screen-space labels—maintain association to the labeled object well, but do not always remain legible. A solution that alleviates the legibility problem is the use of a technique called *billboarding*, where 2D graphical elements (often called *sprites*) have their front side always facing the camera. The graphics engine can simulate perspective changes, rotations, occlusions, and other effects with *billboards*, but the sprite is only viewed from a single angle. The use of billboarding is not uncommon: in games many artifacts and animations, such as trees, explosions, or falling leaves, are displayed using billboards [WQ10]. World-space labels used with the billboarding technique therefore are a viable alternative to screen-space labels [DM07]; they would resemble Figure 1a much more than Figure 1c, however.

3.6 Labels With Variable Leader Height

Maass and Döllner [DM06] considered the task of placing what they call *2.5-dimensional* annotations in 3D environments. These annotations are rooted in the 3D environment, but are by nature 2D, because they always face the camera.

They found that perspective views[3] of terrain-based scenes had two general characteristics: (a) anchor points of annotations near the user are generally lower in the screen space, and (b) annotations near the user are of higher interest than annotations farther away.

Based on these two observations, Maass and Döllner developed an algorithm to efficiently label point features in 3D environments, by labeling the nearest

[3]Perspective views are views that are not from the top looking down, but rather seem to be from within the scene.

points first followed by more distant points. The first labels would therefore be placed with a fixed leader length, while later labels would be placed higher as was necessary.

The algorithm proposed by Maas and Döllner [DM06] is excellent in its runtime and labeling quality for labels with only one degree of freedom. However, it is not as applicable to labeling in navigation systems, which requires fluid, continuous movement of labels in unexpected situations (as are brought about by a U-turn, for example).

3.7 Enhancing the Visibility of Labels

State-of-the-art navigation devices are increasingly including 3D graphical objects, such as landmarks or buildings, into the street map. When this is combined with street labeling, the problem of how to maintain good visibility of the labels becomes very significant. In this thesis, 3D graphical objects are not considered in any of the algorithms; however, Vaaraniemi et al. [VFW13] have grappled with this problem, and developed an interesting approach to solving it.

Simply rendering labels over 3D objects is unsatisfying. Their study concentrates on how the visibility of existing labels can be enhanced—in aesthetics, readability, and in visual associativity—in 3D navigation maps, while leaving the 3D environment intact. To this end Vaaraniemi et al. conducted a pre-study with multiple techniques, and finally implemented the two highest rated techniques: *transparency label aura* and *glowing streets*.

In the transparency label aura technique, labels are placed on the roads where they would be without 3D objects; then both road and label shine through 3D objects by use of a transparency aura. In the glowing streets technique, the entire street network shines through the 3D objects. Both techniques improve the visual association markedly.

Chapter II

Implementation

The street-labeling problem is by nature one that can only really be evaluated by humans. Formulation of hard and soft constraints, for example, help us to develop a correct and perhaps even efficient algorithm, however the result needs to be judged by the human eye. We have therefore implemented our street-labeling algorithms, and documented their results at every stage. For the sake of convenience we shall refer to the program implementing the algorithms as *StreetLabeler*.

In this chapter, the context of the street-labeling algorithms is described: where the actual labeling algorithms fit in this implementation, and the format of the data with which the algorithms work.

4 Groundwork of StreetLabeler

In this section we shall introduce the groundwork of the implementation of StreetLabeler: the general structure as well as the data structures of StreetLabeler.

In order to maintain brevity, the tools and libraries used for StreetLabeler are detailed in Appendix A. A brief summary is required, however: StreetLabeler is written in the programming language C++ and makes use of the OpenSceneGraph graphics library and the C library named Shapelib. Shapelib provides an interface for the reading of shapefiles, which contain street map data used for testing.

Often, implementing an algorithm is more difficult than developing the algorithm in the first place—the devil is in the details, so the idiom goes. This has been (at least somewhat) the case with StreetLabeler. The reasons for this will become clear in Chapter III.

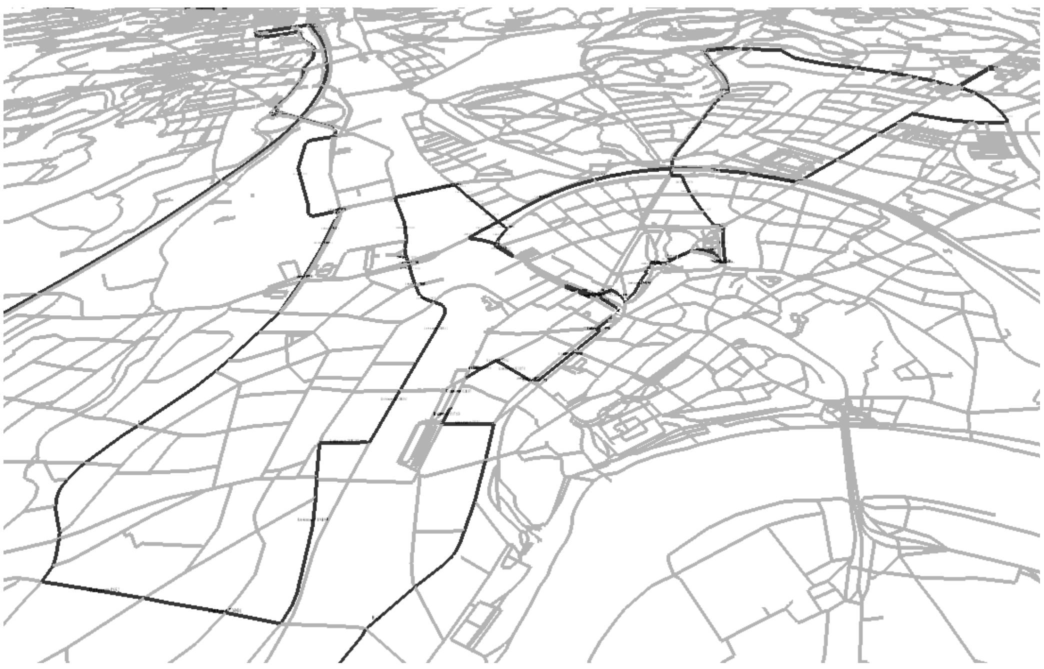


Figure 2: Bird's eye view of a plotted route through the street network in Lower Franconia. The seemingly random route was calculated using a depth-first-search algorithm with road data from OpenStreetMap.

4.1 Structure of StreetLabeler

In order to describe the algorithm for labeling the streets in the active route effectively, the context of the algorithm must be understood: this is the structure of StreetLabeler. StreetLabeler can be broken up into roughly five stages:

1. Read configuration data from commandline and configuration file.
2. Read a shapefile containing street data into a graph, where an edge of the graph is a list of 2D-coordinates, and the coordinate at each end of the list becomes a vertex in the graph.
3. Using a common graph search algorithm (such as *breadth-first-search*, which can find shortest paths in graphs which satisfy the triangle inequality), a route connecting a start and an end vertex is established.
4. The street-graph data structure, together with the route, is converted into a data structure suitable for use with OpenSceneGraph. Thus, the lists of 2D-coordinates are converted into 3D-coordinates and placed within appropriate classes.

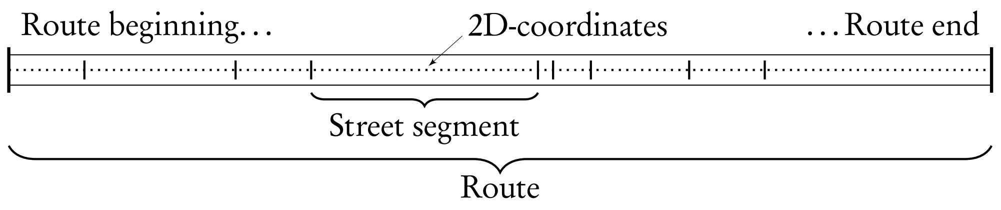


Figure 3: A route consists of a list of multiple street segments. Each street segment represents a stretch of road with a single street name.

5. OpenSceneGraph is then used to display the street map, together with two helper classes: one class moves a *pointer*, which resembles an arrowhead and indicates the current position in the active route; another class places labels for the streets in the active route.

The last two classes are of particular interest. The first one, the Pointer class, moves a pointer, followed by a camera, in the map. The camera following the pointer makes the pointer the center object in the map, which is common practice in most navigation systems. The second and final one, the Labeler class, is what contains the algorithm that places labels. In Chapter III, this algorithm will be closely examined.

Figure 2 shows the end result of these five stages. The route is drawn darker than the other streets. In OpenSceneGraph, the lines that are used to represent the roads maintain the same line width, regardless of the viewing angle or the zoom level used. In screenshots of StreetLabeler, close-ups will therefore seem to have surprisingly thin roads, while bird's eye views seem to have thicker roads.

4.2 From Shapefile to Street Route

There is no inherent organization or structure to the data in a shapefile. As explained in Appendix A, a shapefile simply consists of a set of vertex lists. Additionally, each list has a set of attributes. One of these attributes may be a street name; it is possible for some lists to have a name attribute, while others do not.

Normally, we might expect each physical street to have exactly one list of vertices. This is however not the case: we may see each street to be composed of multiple lists, or *segments*. Each segment acts as an edge in a graph, for the purposes of routing. After routing through this graph, we receive a *route*, consisting of an ordered list of street segments. (Segments are turned the right way around, if necessary.) Each segment consists of a list of vertices and the respective street name. If the street name is empty, StreetLabeler generates one for it.

Based on the names of segments, adjacent segments are merged if they have the same name. In the end, we have a route as depicted in Figure 3. The route in Figure 3 has 9 segments, and no adjacent segments have the same name. This is the kind of data the labeling algorithm must work with.

The data with which we are primarily working has very few streets named; only important streets such as “B26” or the *Autobahn* are named. This data represents Lower Franconia, and comes from OpenStreetMap. Unnamed segments receive auto-generated names, in the form “Unknown X”, where X is a number.

Chapter III

Labeling Algorithms

This chapter builds on the previous chapter; after having described the general structure of StreetLabeler, we shall continue by incrementally developing a force-directed algorithm for labeling the streets in the active route. We start with the base algorithm, and then build upon that.

While we would like to measure a labeling algorithm against the optimum (as we discussed in Section 2), it is helpful to see what difference the labeling algorithms actually make, by also measuring them to the simplest algorithm. We shall therefore start with static positioning of street labels on the active route. At the end of each section, we shall discuss the advantages, disadvantages, and implementation problems of each algorithm.

Before we elaborate on different algorithms however, a word must be said about the context of these algorithms. We split up a street labeling algorithm into two parts: the initialization stage and the update stage. This allows labeling algorithms to be bound in by registering callback functions in the Pointer object. At initialization, the algorithm receives a Route object, upon which it has the possibility of say, creating and placing all the labels at once. The update method of the algorithm is called every frame during which the Pointer object moves, and the current location of the Pointer in the route is given as the argument.

5 Static Street Labeling

Perhaps the most naïve street labeling algorithm is the simple static one: during initialization, labels are placed once in the center of their respective streets, and nothing more is done. This algorithm we shall call *static street labeling* (SSL). This shall be the basis for the force-directed street-labeling algorithm.

5.1 Mathematical and Algorithmic Description

Algorithm 1, the initialization of SSL, is quite simple: for each segment in the route, the middle of the segment, as measured by the euclidean distance, is calculated, and a label is created and placed at that point. (Please note that indices start with the number 1 in all cases, as is standard in mathematics.)

Let d_2 represent the euclidean distance metric. If the number of vertices in a segment is n, then the middle point q in the path is calculated in three steps. First, we sum up the distances between all consecutive vertices $v_i, v_{i+1} \in \mathbb{Q}^2$, in order to get the entire length t of the segment:

$$t = \sum_{i=1}^{n-1} d_2(v_i, v_{i+1}). \tag{5.1}$$

Then, we find the smallest l, such that

$$s = \sum_{i=1}^{l} d_2(v_i, v_{i+1}) > \frac{t}{2}, \tag{5.2}$$

and finally, we calculate the point q, which is $s - t/2$ beyond the vertex v_{l-1}, with a modification of the midpoint formula (see Equation 6.6 on page 21), where $v_{l-1} =: (x_1, y_1)$ and $v_l =: (x_2, y_2)$. Here, we calculate the ratio $r \in [0,1]$, which gives how far q is between v_{l-1} and v_l, and then using r we find q:

$$r = \frac{s - t/2}{d_2(v_{l-1}, v_l)} \tag{5.3}$$

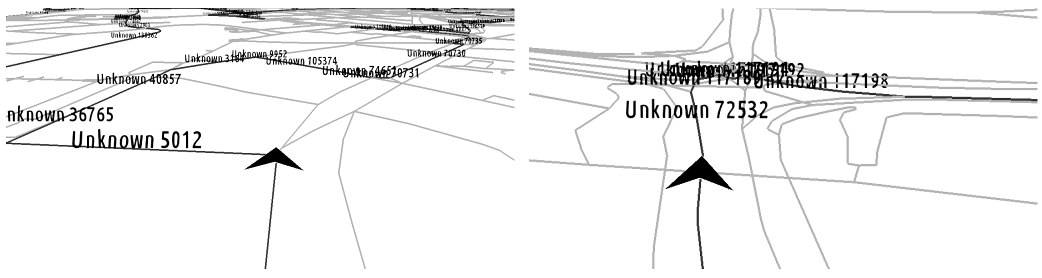
$$q = (x_1 r + x_2(1-r),\ y_1 r + y_2(1-r)). \tag{5.4}$$

Thus, we traverse all the vertices in the segment at most twice, resulting in a linear runtime in the length of the segment. The algorithm for the initialization is thus given:

Algorithm 1: Initialization stage of static street labeling

```
Input: Route $R$
1 foreach Segment $s \in R$ do
2     Label $\lambda \leftarrow$ createLabel(name($s$))
3     Point $p \leftarrow$ calculateMiddlePoint(vertices($s$))
4     placeLabel($\lambda, p$)                  // label is placed with a height of 0
5 end
```

The complexity of Algorithm 1 is therefore $O(n + N)$ in the size of the route, where n is the number of segments in the route and N is the number of vertices over all segments.



(a) (b)

Figure 4: Results of static labeling: (a) labels are placed in the middle of each street segment, without any regard to other labels or objects; and (b) in some inconvenient junctions, overlapping of labels renders almost all illegible.

5.2 Results and Evaluation

The algorithm is simple to implement when using the billboarding technique (as discussed in Section 3.5); the results can be seen in Figure 4.

The advantages of static street labeling are manyfold:

1. SSL is efficient, and only incurs a runtime cost at the initialization. Since there is no requirement of interactivity at initialization, this is acceptable.

2. Labels don't jiggle or move around, except to always face the user, and are therefore predictable.

3. The labels can be easily associated with the street segment they are labeling, because they are very close to the segment, in the center.

4. The labels act as depth cues, and so help to orient the user.

5. The labels automatically support spatial orientation, if implemented with the billboarding technique.

However, the previously stated advantages are outweighed by two major disadvantages of SSL:

1. Labels are placed without any regard to the dynamics of the route or to other labels. This often leads to mass overlapping, which renders many labels unreadable.

2. Only one label is placed per street segment. On any street, there should be a label near the beginning of the street, and its label should be repeated once it has gone out of sight.

The implementation of the algorithm reveals four issues. First, placing labels in the exact center of their respective segments is not necessarily the best method. It would be reasonable to put the label for a segment near the beginning of the segment, and then repeat that label once the first disappears, and so on. Second, it may be unclear where one segment ends and another one starts. The start of the segment could be colored differently to bring attention to the fact that a new segment is starting. Third, labels that overlap become often entirely unreadable. Making labels have an opaque instead of a transparent background might make the top label readable, but it would occlude the street too, which is also somewhat undesirable. The fourth issue is that the pointer simply drives over the labels; it would be better if the labels disappeared before the pointer came too close.

The focus of this thesis is the dynamic manipulation of labels by leader length adjustment; other approaches and variants to solve the problem of overlapping are discussed in Chapter IV. Although the questions of where to put labels, how they are rendered, as well as when they should appear and disappear, are important, they are too large to adequately handle in this thesis. We shall therefore dismiss these issues at this point. Although static street labeling is unsatisfactory by itself, we shall use SSL as the basis for the force-directed street labeling algorithm.

6 Force-Directed Street Labeling

The primary problem with solely using SSL is the uncontrolled label overlap. In order to combat these occlusions, we shall model (or rather imitate) certain forces in physics. In the same way that same magnet-poles push away from each other, so labels will also push away from each other by these forces, and by doing so, will avoid collisions. This we shall call *force-directed street labeling* (FDSL).

6.1 Mathematical Description

We cause labels to avoid collisions by assigning *repulsive forces* between labels, based on their proximity to each other. These forces push a label either up or down, depending on whether the other occluding labels originate below or above the label, respectively. Independently, each label tries to incrementally reach a balance of forces. In theory therefore, given enough time, each label will eventually find a local optimum, and so the entire system of labels will be in one of its local optima states.

Normally, the forces effected by other labels have two dimensions, however a label only has one degree of (vertical) freedom, namely in the length of its leader. In order to keep a label close to the street, the forces exerted by other labels are counteracted by the leader, which acts as a tension spring. Leader extension

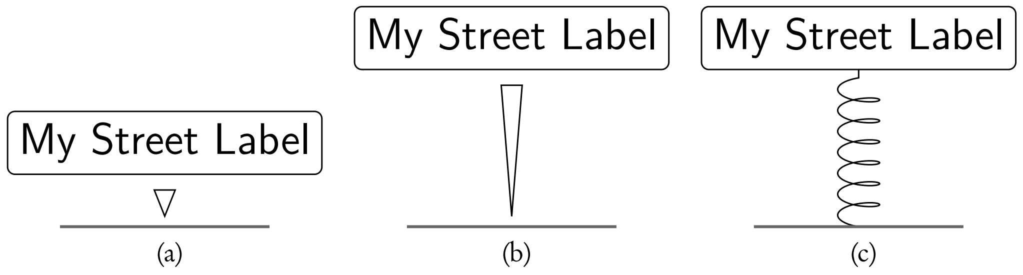


Figure 5: Spring analogy for label and leader behavior: (a) how it should look, (b) how it looks with an extended leader, and (c) how we imagine it to work.

translates into label height, which is a positive rational number. Any height greater than 0 effects a negative force upon the label, which pulls it down.

By adjusting the leader length in the right way, a label can reach an equilibrium of forces. This all has the effect that labels increase the length of their leader when necessary to avoid occlusions, and when not, they decrease it.

The physics that we shall model, seen in Figure 5, is that the leader represents a kind of spring, which can be extended, but at first opportunity, retracts again. The force with which the spring pulls a given label λ_l down can be expressed with Hooke's law:

$$\hat{F}_l = -k \cdot x_l, \tag{6.1}$$

where k is the *spring constant* and x_l is the current extension (or displacement) of the spring. A label $\lambda \in \mathbb{Q}^4$ on the screen has two components, the lower left corner, and the upper right corner; these coordinates represent the bounding-box rectangle on the screen.

There may be multiple labels above and beneath a given label λ_l, so the force exerted by those labels, pushing λ_l either up or down can be expressed by

$$\tilde{F}_l = c \cdot \sum_{i \in I \setminus \{l\}} \sigma(\lambda_l, \lambda_i) \frac{1}{d(\lambda_l, \lambda_i)^2}, \tag{6.2}$$

where c is the *repulsion constant*; d is the distance function; σ is the sign operator, which for $\sigma(j,i)$ is -1 if j is below i, and 1 otherwise; and I is the indices-set of all the labels displayed. The force exerted on the label is inversely proportional to the square of the distance, this dependency is analogous to electromagnetic forces in physics and ensures that labels only affect one another at short distances.

Given all this, we could find the equilibrium of forces for a label λ_l by solving the equation

$$\hat{F}_l + \tilde{F}_l = 0. \tag{6.3}$$

However, we must bear in mind that our aim is not to model physics but to place labels. For this purpose it is sufficient to incrementally approach equilibrium, not calculate it. So instead of solving Equation 6.3, we take the net force on the label

$$F_l = \hat{F}_l + \tilde{F}_l \tag{6.4}$$

for every iteration step and apply a black-box function $\beta(F_l)$ to get the leader length adjustment for that step.

We shall shortly cover the functions d, σ, and β in greater detail. For now it suffices to know that they all have a runtime complexity of $\mathrm{O}(1)$ in the number of labels displayed.

Since for each label we need to perform the distance and sign calculations for every other label, we limit the number of labels displayed on the screen to η labels. Thus for each one of η labels, we must calculate the distance between it and the other $\eta - 1$ labels. The complexity of the algorithm in total is therefore $\mathrm{O}(\eta \cdot (\eta - 1)) = \mathrm{O}(\eta^2)$.

The Functions d, σ, and β

The mathematical definitions of the laws of physics which we are modeling are themselves models of the behavior in the observed universe. In a given sense, they are therefore approximations. Hooke's law for example, given in Equation 6.1, does not model the spring's final extension length nor does it model the spring breaking. This is alright for modeling a label leader, because a leader does neither of these things. However, the electro-magnetic forces which we are modeling emanate from objects which cannot really overlap like labels do. A conventional distance metric ranges from 0 to infinity. This poses a significant problem, because labels can overlap and therefore do not logically fit in the conventional distance metric.

We therefore have to adjust the definitions of the functions d, σ, and β, so that they logically fit in with Equation 6.2. These functions could be defined in several different ways; the definitions here are not the only possible ones.

Let the function $\omega \colon \mathbb{Q}^4 \times \mathbb{Q}^4 \to [0, 1]$ measure the percentage of overlap of the first label λ_1 by the second label λ_2. Additionally, let $d_v \colon \mathbb{Q}^4 \times \mathbb{Q}^4 \to [0, \infty)$ measure the vertical distance between two labels λ_1 and λ_2; if λ_1 is not vertically above or below λ_2, then d_v results in infinity. We then define d as

$$d(\lambda_1, \lambda_2) = \begin{cases} 1 - \omega(\lambda_1, \lambda_2), & \text{if } \omega(\lambda_1, \lambda_2) \neq 0 \\ 1 + d_v(\lambda_1, \lambda_2), & \text{otherwise.} \end{cases} \tag{6.5}$$

This should effect what we expect from a distance function. It is continuous, resulting in a distance less than 1 when two labels overlap, and a distance equal to or greater than 1 when they are only near. This should cause them to repel each other increasingly strongly when they overlap, and otherwise only increasingly weakly.

The sign function σ in Equation 6.2 corrects the term behind it so that the force pushes the label the right way. Let $\mu \colon \mathbb{Q}^4 \to \mathbb{Q}^2$ calculate the middle point of a label $\lambda = (x_0, y_0, x_1, y_1)$ and be defined as

$$\mu(x_0, y_0, x_1, y_1) = \left(\frac{x_0 + x_1}{2}, \frac{y_0 + y_1}{2}\right). \tag{6.6}$$

This formula is called the midpoint formula, and is used in a modified form in Section 5.1 (see Equation 5.4 on page 16). With μ we then define σ as follows:

$$\sigma(\lambda_1, \lambda_2) = \begin{cases} -1, & \text{if } \mu(\lambda_1) <_y \mu(\lambda_2) \\ 1, & \text{otherwise} \end{cases} \tag{6.7}$$

where $<_y$ compares the y-components of the given points $\mu(\lambda_1), \mu(\lambda_2) \in \mathbb{Q}^2$.

The black-box function β is a function that takes the directional force exerted on a label, and scales that force into a extension or retraction of the leader for one iteration step. We provide only a crude definition of β in this thesis, in the next section we will discuss at greater length some of the issues surrounding it.

6.2 Algorithmic Description

In order to realize the mathematical description given in the previous section, the initialization stage of SSL must be modified (see Algorithm 1). We introduce a new constant η, the number of labels that will simultaneously be shown. Below, Algorithm 2 shows the initialization stage of FDSL.

Algorithm 2: Initialization stage of force-directed street labeling

```
Input: Route $R$
Input: Constant $\eta$                  // number of labels shown at a time
$i \leftarrow 1$
foreach Segment $s \in R$ do
    Label $\lambda \leftarrow$ createLabel(name($s$))
    Point $p \leftarrow$ calculateMiddlePoint(vertices($s$))
    placeLabel($\lambda, p$)           // label is placed visible with a height of 0
    if $i > \eta$ then hideLabel($i$)
    $i \leftarrow i + 1$
end
```

So while all labels are placed, only the first η labels are shown at initialization. In the update stage, the label visibility is shifted as the pointer reaches each label. The following algorithm represents the mathematical description of FDSL, in the update stage.

Algorithm 3: Update stage of force-directed street labeling

```
Input: Position $p$
Input: Integer $n$                          // total number of labels
Input: Constant $\eta$                      // number of labels shown at a time
Input: Constant $\epsilon$                  // distance at which a label disappears
Input: Constant $k$                         // spring constant
Input: Constant $c$                         // repulsion constant
/* 1. Hide labels as the pointer gets close, and show other ones.   */
$j \leftarrow$ currentSegment()
if $d_2$(getLabelPosition($j$), $p$) $< \epsilon$ then
    hideLabel($j$)
    if $j + \eta \leq n$ then showLabel($j + \eta$)
end
/* 2. Get the screen space coordinates of each label.   */
$S \leftarrow \varnothing$
foreach $\lambda \in$ displayedLabels() do
    $S \leftarrow S \cup$ getScreenSpaceCoordinates($\lambda$)
end
/* 3. Calculate the forces acting on each label and shift it.   */
foreach $\lambda_s \in S$ do
    $F \leftarrow 0$
    foreach $v \in S \setminus \{\lambda_s\}$ do
        $F \leftarrow F + \sigma(\lambda_s, v) \cdot \frac{1}{d(\lambda_s, v)^2}$
    end
    $F \leftarrow F \cdot c + -k \cdot$ getLeaderExtension($\lambda_s$)
    shiftLabel($\lambda_s$, $\beta(F)$)        // $\beta$ is the black-box function as discussed
end
```

In order to decouple the algorithms more from the implementation, the algorithms are displayed in a functional style in lieu of an object-oriented style. For example, given an instance p of a Point object with data members x and y, these would be accessed using the functions x(p) and y(p), and not the period operator: x.p and x.y.

The Black Box Function β

The question that β should answer is, given a force F exerted on a label λ, how should λ move in one iteration step?

If we were to more accurately model physics, we could give labels something like a charge or mass, which determines how they move. Labels could accelerate and decelerate, giving their movement a more natural feel. One advantage of this method would be that we could prioritize labels through their mass. A heavy label is more likely to retain its position and cause a lighter label to move upwards. This solution however, would require modifying the system on every level. For example, Equation 6.2 would look more like

$$\tilde{F}_l = c \cdot \sum_{i \in I \setminus \{l\}} m_l m_i \cdot \sigma(\lambda_l, \lambda_i) \frac{1}{d(\lambda_l, \lambda_i)^2},$$

where m_j is the mass of the label λ_j. Using this approach would open the possibility of replacing the extension-spring model by a gravitational force which pulls the label to the ground.

Alternatively, we could use just the black-box function β as an ad-hoc solution, and have it "intelligently" derive a label adjustment based on the force. In this (or perhaps any) case, the following questions would need to be answered:

1. What constant γ does the net force on a label have to exceed before it affects the label? Requiring the net force to satisfy $|F| > \gamma > 0$ would allow labels to eventually settle.

2. With which function does the movement distance scale with the force?

3. Does the number of iteration steps per second vary, such as frame rate varies?

4. How big should the adjustment be in one iteration step? What is the optimum between adjusting fast enough and not overshooting.

5. Should there be a maximum movement distance in one iteration step?

In order to simplify the association between label and street, labels together with their leaders are grafted into the 3D environment with the technique of billboarding. But because occlusions only make sense in screen space, we must extract the 2D screen space coordinates of the labels from the graphics library. However, if the leader is in the world space, then the effect of these forces on the screen space label must be converted back into a world-space adjustment of the leader height. This is not as trivial as the world-to-screen space conversion, and poses a challenge for the β function. The higher the perspective is, the more world-space height is required to raise the label by one pixel in the screen space, till it makes no difference at all, when the camera is looking straight down at the label. For this reason, it would be better for the leader's position to be rooted in the world-space, but itself be in the screen space, so that no such reverse conversion is required.

In the following section, the results of FDSL are presented. We have for lack of time opted for the ad-hoc solution with the label and leader in world space, and have tried multiple definitions of β. One of the definitions of β which was tried is where the force is simply scaled by a constant factor ζ:

$$\beta(F) = \zeta \cdot F. \tag{6.8}$$

In this definition, some of the above points are not considered; depending on the perspective and implementation, this may prove to be unsatisfactory.

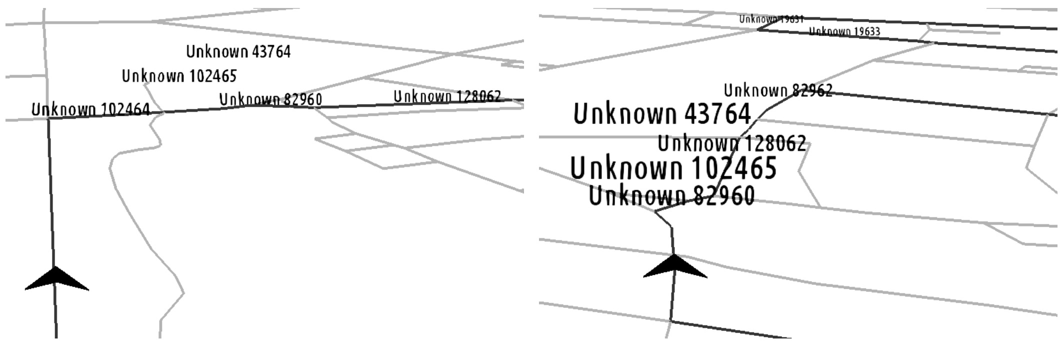
6.3 Preliminary Results and Evaluation

In theory, force-directed street labeling sounds very attractive. It is efficient, its runtime only quadratic in the number of displayed labels, and it is quite simple. In the average case, the solution provided by FDSL would seem to be both sufficient and aesthetic. Nonetheless, in its pure form it also seems to be incapable of providing an optimal solution in the worst case. This is because the algorithm only moves labels vertically, and does this without any regard to the positioning or importance of the labels. This however, is also an advantage.

Since this thesis does not concentrate on the visual aesthetics of a label, we restricted ourselves to the core of the algorithm. Due to time constraints, we did not implement actual visible label leaders, and we were unable to implement the algorithm in exactly all the ways described here. These preliminary results can be seen in Figure 6.

Force-directed street labeling has several advantages over other street labeling algorithms, and it overcomes the primary disadvantage of static street labeling:

1. Labels avoid each other by using repulsive forces, so that occlusions are quickly resolved.
2. The complexity of the algorithm is $\mathrm{O}\left(\eta^2\right)$ in the number of labels that are shown at a time. This has another benefit; the number of labels shown can be decreased to speed up the program.
3. Labels move predictably, because they imitate the movement of real world objects, which we are accustomed to, and because they only move vertically.



(a) (b)

Figure 6: Results of force-directed labeling: (a) labels avoid overlapping each other by extending their (here invisible) leaders, so that they grow in height; but (b) the forces that achieve this make it difficult for labels to "get by each other."

There are three inherent disadvantages to this algorithm, aside from the problem of having only one label per street segment:

1. Unless label mass is modeled, labels cannot escape their world-space vertical ordering; that is, they cannot get by each other. This effect can be seen in Figure 6b, where the label that is actually closer is higher than a label farther away. This is visually confusing, and distorts the sense of depth.

2. Labels all start with a height of 0; if a user sees an excessively labeled section in the map for the first time, he or she sees overlapping labels, such as in Figure 7b, which only then start to move away from each other.

3. If there are too many labels in one area that occlude each other, the final height in the end may be visually unappealing.

There are of course, as has been discussed throughout this section, a number of implementation challenges associated with this algorithm.

1. If labels together with their leaders are in the world space, then conversion between world and screen space must be performed. Additionally, perspective changes have a strong impact on how the forces affect the label.

2. The definition of the black-box function β is difficult.

3. There are at least four constants that must be correctly defined (two for label movement to be visually pleasing):

   η number of labels shown at a time

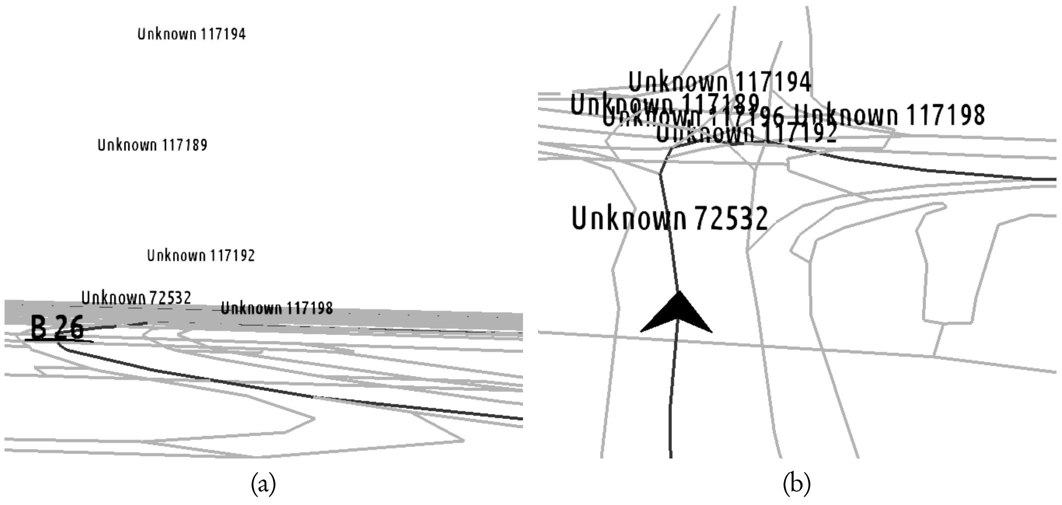


Figure 7: "Physical" constants must be finely tuned: (a) when the spring constant k is too small in comparison to c, there is no force to push the labels down; but (b) when k is too large (or c too small), the labels clump too much together.

ϵ	distance to a pointer at which a label disappears
k	spring constant (strength with which a spring pulls down)
c	repulsion constant (strength with which labels affect each other)

In addition, there are multiple constants that are encapsulated by β, such as γ which the net force must exceed to have an effect on a label. If we do not correctly define these constants or β, situations may arise as depicted in Figure 7, which are very unappealing. It is also possible for labels to bounce back and forth or jiggle, if these are not defined well.

4. What happens to labels that disappear because of camera rotation, but are still part of the η labels that are displayed? Do they retain their height? Do they receive perhaps only a fraction of available iterations? Or what happens to a label with a leader so extended, that it goes off the top of screen? Intuitively, one would want no change in behavior, but then the question arises: what are the screen space coordinates of that label?

5. If label leaders are drawn as larger elements than lines, such as triangles, do these leaders also contribute to label overlap? In what way? For example, given two labels λ_1 and λ_2, with λ_1 closer to the camera than λ_2, what should happen when the label λ_1 overlaps the leader L_2? This is unavoidable. Yet if λ_1 is higher than λ_2 in order that λ_2 remains visible, L_1 overlaps λ_2, and this is undesirable!

One can see that, even if the physical model is accurate and complete, there are a lot of additional special cases that need to be handled in the implementation.

In conclusion, although force-directed street labeling seems inadequate in its pure form, if the implementation issues are worked out and the main algorithm is complemented by helper algorithms, then the algorithm does a satisfactory job of keeping labels from overlapping. While it may not calculate any optimal label placements, the algorithm provides a way to label streets in such a way that labels remain readable and maintain their associativity to the street segment.

Chapter IV

Conclusion

Navigation devices started out simple, and it is even possible to find old devices without graphical user interfaces. In recent years however, these devices have been gaining in computational power, and today it is not uncommon to find 3D elements or even photo-realistic scenery being used. Despite these improvements, it would seem that features such as street labeling often receive less attention, perhaps due to the algorithmic complexity of placing labels in a near-optimal way.

In order to help bridge this gap, we have developed a simple force-directed approach to avoiding label overlapping in navigation systems. This method provides a basis for dynamically managing occluding labels, which can be extended in multiple ways; this is also attested to by Ebner et al. [EKW03].

This basis manages occluding labels in an unobtrusive way by incrementally adjusting their leader height, such that occlusions and collisions are avoided. Repulsive forces between labels, and forces restricting leader extension direct the movement of labels. Labels are thus kept close to the street, to maintain visual association, yet rise if necessary to avoid overlapping with other labels.

We implemented a simple static street-labeling algorithm, and a variant of the force-directed street-labeling algorithm presented in this thesis. Due to time constraints and implementation problems, we were unable to fully implement the latter algorithm; the principle showed itself effective however. While the static labeling-algorithm caused mass overlapping and illegibility of the labels in many sections of the map, the force-directed algorithm was able to maintain legibility of the labels.

Future Work

Nonetheless, we have also shown that this algorithm in its pure form is unable to satisfactorily manage street labels in their entirety; there are many areas for improvement and future research. Because there are so many possibilities, we shall list them here.

1. For clarity only the bare algorithm has been presented in this thesis—there is much room for optimization. For example, not all labels need to be explicitly tested for collisions.

2. The forces model together with the extension spring model could be augmented by the addition of charge or mass to the label. This represents a prioritization of the labels; by this method, the labels could be controlled in a simple and efficient way.

3. More research could be done on the effects of introducing the elements of acceleration and deceleration into the forces model.

4. Labels could be given a second restricted degree of freedom, by allowing them to slide either to the left or to the right of the leader. This could also be integrated into the forces model.

5. Labels leaders could be allowed to slant, which would allow labels to more effectively "follow" the direction which forces push them. In order to maintain labels above the streets, one could limit the leader slant to $\pm 45°$, for example.

6. More research could be done on techniques that would allow labels to escape local optima in a controlled fashion in order to reach a better solution. For example, labels closer to the camera might be allowed to slide down over labels behind.

7. Lastly, a final street labeling solution would require more intelligent initial placement of street labels, and the ability to dynamically reintroduce a label for a long segment.

These contain only a subset of the possible future work that could be done on the topic of force-directed street labeling.

Appendix A

Tools and Libraries

This appendix describes the tools and libraries used to write StreetLabeler, which is the program (described in Chapter II) that contains the implementation of the algorithms described in Chapter III.

StreetLabeler is written in the 1998 ANSI/ISO C++ standard (also known as C++98), with significant use of the Standard Template Library (STL), as well as other 3rd-party libraries: OpenSceneGraph (OSG), Boost, and the Shapefile C library (also known as Shapelib).

C++ is a compiled, multi-paradigm, general-purpose programming language, and is one of the most popular programming languages available on the planet today. It is mostly backward compatible with the C programming language, which makes it easy to use with C libraries. The STL—a central part of the current C++ standard—is a library of *generic* data containers and algorithms; they are not designed with a particular datatype in mind: a vector[1] may hold any kind of object, but that object type (such as string) must be specified when creating a vector, for example.

Shapelib reads, writes and updates ESRI *shapefiles*. A shapefile is a popular data format used in geographic information systems, which can store nontopological geometry and attribute information for point, line and area features. The data is then stored as a set of 2D vector coordinates [ESR98]. As such it is particularly capable of storing street maps; data from OpenStreetMap, for example, can be exported into the shapefile format. The information exported from OpenStreetMap contains many different shapefiles: there is a different shapefile for each category of feature. StreetLabeler uses Shapelib to read shapefiles that describe road features, these are usually aptly named "roads.shp". The shapefile format actually defines a set of at least three files that compose a shapefile. These files share the same basename (such as "roads"), but have varying suffixes: shp, dbf

[1]A vector is a dynamic array, with constant access time to any element in the list.

and shx. In this case, “roads.shp” contains the vector coordinates, and “roads.dbf” the attribute information, such as the street names. A description of Shapelib, as well as the API, is available at the official website: `shapelib.maptools.org`.

Boost is a large set of peer-reviewed portable C++ libraries. In fact, many of the new classes and features present in the newest C++ standard are directly derived from classes and features in Boost. According to Sutter and Alexandrescu [SA05], Boost is “one of the most highly regarded and expertly designed C++ library projects in the world.” StreetLabeler primarily uses the program_options library, for parsing options from the command-line and configuration files; and the Boost Graph Library, for initially storing and routing in the street map. For more information on Boost, consider visiting the official website: `www.boost.org`.

OpenSceneGraph is a high quality, open source, 3D-graphics toolkit based on the low-level OpenGL API, and is being used in many different fields, such as visual simulation, virtual reality and even games. OSG provides a convenient API abstracting upon the OpenGL API, albeit at the loss of some flexibility, to quickly develop 3D applications [WQ10]. OSG has thus been instrumental in providing the interactive visualization provided in StreetLabeler. One way to view OSG is that it is an *object oriented* approach to 3D-graphics programming, using different objects to store data and perform operations on that data. For more information on OpenSceneGraph, visit the official website: `www.openscenegraph.org`.

Appendix B

Deutsche Zusammenfassung

Die ersten Navigationssysteme waren sehr einfach ausgelegt. Man kann sogar Geräte ohne graphische Oberfläche finden. In den letzten Jahren haben diese Geräte allerdings sehr an Rechenleistung zugenommen, und heute ist es nicht ungewöhnlich, dass 3D Elemente oder sogar fotorealistische Landschaftsbilder benutzt werden. Trotz dieser Verbesserungen, scheint es, dass es immer noch Mängel im Bereich der Straßenbeschriftung gibt. Das liegt unter anderem daran, dass Algorithmen für die möglichst optimale Platzierung der Beschriftungen sehr komplex sein können.

Um dazu beizutragen, dass diese Lücke gefüllt wird, haben wir einen kräftebasierten Ansatz entwickelt, mit dem es möglich ist, das Überlappen von Straßenlabels zu vermeiden und aufzulösen. Diese Methode ist bereits in der Wissenschaft bekannt, und ist schon für die Beschriftung verschiedener Modelle verwendet worden, zum Beispiel von Ebner et al. [EKW03], um die Anzahl beschrifteter Elemente zu maximieren.

Das vorgestellte Verfahren bietet sich gut an als Basis, um Beschriftungen in einer dynamischen und interaktiven Art und Weise kollisionsfrei zu halten und kann auch leicht erweitert werden.

Abstoßende Kräfte zwischen den Straßenlabels und Kräften, die die vertikale Höhe des Labels einschränken, bestimmen wie das Label verschoben wird. Mit diesem Ansatz ist es möglich, das Überlappen von Straßenlabels auf unauffällige Weise zu vermeiden, indem die vertikale Höhe inkrementell angepasst wird. Dadurch bleiben Labels nah genug an der Straße, was eine optische Zuordnung erhält, werden jedoch gleichzeitig, wenn nötig, nach oben geschoben, um eine Überlappung mit anderen Labels zu vermeiden.

Wir haben einen einfachen, statischen Straßenbeschriftungs-Algorithmus und eine Variante des vorgestellten kräftebasierten Algorithmus implementiert. Aufgrund zeitlicher Beschränkung und Problemen bei der Durchführung, wie in Kapiel III geschildert, konnten wir den Algorithmus nicht genau wie beschrieben umsetzen. Dennoch haben sich die grundlegenden Prinzipien als effektiv erwiesen. Während durch den statischen Algorithmus viele Labels überlappten und dadurch Teile der Karte unleserlich wurden, waren mit dem kräftebasierten Algorithmus alle Labels auf der Karte lesbar.

Bibliography

[AKS$^+$00] Pankaj K. Agarwal, Lars Knipping, Tycho Strijk, Marc van Kreveld, and Alexander Wolff. A simple and efficient algorithm for high-quality line labeling. In Peter M. Atkinson and David J. Martin, editors, *Innovations in GIS VII: GeoComputation*, chapter 11, pages 147–159. Taylor & Francis, 2000.

[CJ90] Anthony C. Cook and Christopher B. Jones. A prolog rule-based system for cartographic name placement. *Computer Graphics Forum*, 9(2):109–126, 1990.

[CMS95] Jon Christensen, Joe Marks, and Stuart M. Shieber. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14(3):203–232, 1995.

[DM06] Jürgen Döllner and Stefan Maass. Efficient view management for dynamic annotation placement in virtual landscapes. In Andreas Butz, Brian Fisher, Antonio Krüger, and Patrick Olivier, editors, *Smart Graphics*, volume 4073 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, December 2006.

[DM07] Jürgen Döllner and Stefan Maass. Embedded labels for line features in interactive 3d virtual environments. In Hannah Slay, Stephen N. Spencer, and Shaun Bangay, editors, *Afrigraph*, pages 53–59. ACM, January 2007.

[EKW03] Dietmar Ebner, Gunnar W. Klau, and René Weiskircher. Force-based label number maximization. Technical Report TR-186-1-03-02, Institut für Computergraphik und Algorithmen, Technische Universität Wien, June 2003.

[ESR98] ESRI. Esri shapefile technical description. Technical report, ESRI, July 1998.

[GHN11] Andreas Gemsa, Jan-Henrik Haunert, and Martin Nöllenburg. Boundary-labeling algorithms for panorama images. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM-GIS'11)*, pages 289–298, 2011.

[Ins12] Berg Insight. Mobile navigation services and devices. Report 5, Berg Insight, January 2012.

[Pap77] Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[RT10] Kurt A. Raaflaub and Richard J. A. Talbert. *Geography and Ethnography*. Wiley-Blackwell, 2010.

[SA05] Herb Sutter and Andrei Alexandrescu. *C++ coding standards*. Addison-Wesley, Boston, 2005.

[Str01] Tycho W. Strijk. *Geometric Algorithms for Cartographic Label Placement*. PhD thesis, University of Utrecht, January 2001.

[VFW13] Mikael Vaaraniemi, Martin Freidank, and Rüdiger Westermann. Enhancing the visibility of labels in 3d navigation maps. In Jacynthe Pouliot, Sylvie Daniel, Frédéric Hubert, Alborz Zamyadi, William Cartwright, Georg Gartner, Liqiu Meng, and Michael P. Peterson, editors, *Progress and New Trends in 3D Geoinformation Sciences*, Lecture Notes in Geoinformation and Cartography, pages 23–40. Springer-Verlag, 2013.

[Wol99] Alexander Wolff. *Automated Label Placement in Theory and Practice*. PhD thesis, Freie Universität Berlin, 1999.

[WQ10] Rui Wang and Xuelei Qian. *OpenSceneGraph 3.0*. Packt Publishing Ltd., December 2010.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen Hilfsmittel und Quellen als die angegebenen benutzt habe. Weiterhin versichere ich, die Arbeit weder bisher noch gleichzeitig einer anderen Prüfungsbehörde vorgelegt zu haben.

Würzburg, den ____________________, ________________________________

(Benjamin Morgan)