Road Data Simplification with Circular Arcs and Straight Lines

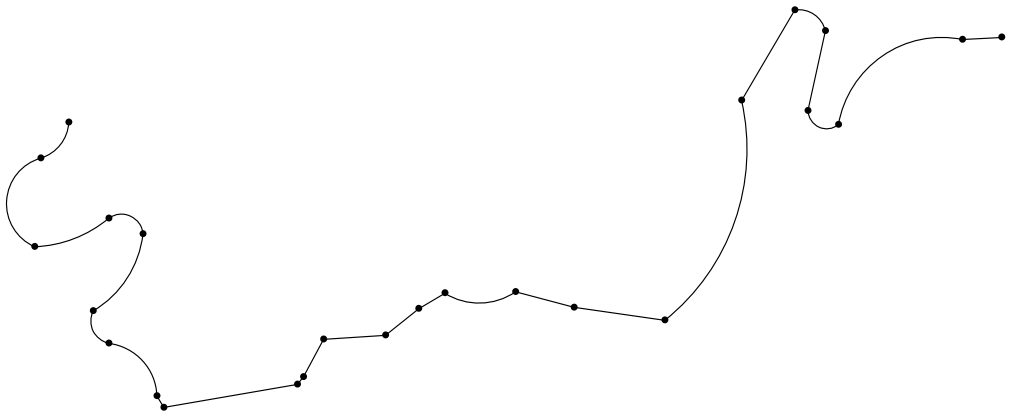
inf/scr-07-106

B.J. Spaan

0493775

January 20, 2009

Master's Thesis



Utrecht University[1]
TomTom, Amsterdam[2]

Supervisors:
Marc van Kreveld[1]
David Martens[2]
Martin Wolf[2]

Universiteit Utrecht

Abstract

Every new navigation device TomTom releases comes with even more functionality and larger maps than previous, older versions. Although the memory capacity of those new devices increases too, still little of this memory is ever left unused. Storing map data as efficient as possible is therefore still very important.

Highly detailed maps are provided by external companies, too large to fit completely on the device's memory card. Among other techniques, maps can be made smaller by removing superfluous detail in road data. This is currently done by a polygonal line simplification algorithm which removes insignificant points but preserves the overall shape of the road. Good compression is achieved by this algorithm, but resulting road approximations easily get jagged and unrealistic.

This master's thesis proposes a new method to approximate road data with a combination of circular arcs and straight lines, which will prove not only to store roads more data-efficient, but as well to produce approximations more aesthetically pleasing.

Contents

Chapter 1

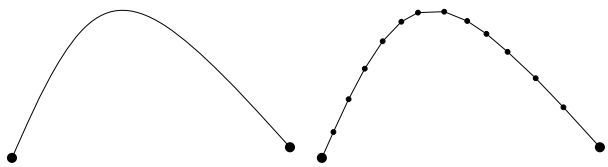
Introduction

1.1 About TomTom

TomTom is one of the world's leading providers of navigation devices for cars and motorcycles. The software of these navigation products is easy to use and runs on integrated TomTom devices, PDAs and mobile phones. Founded in 1991 in Amsterdam, TomTom has grown to a company that sells its products in over 30 countries in Europe, North America and Asia Pacific.

Every new navigation device TomTom releases has more functionality than previous models; all those features — such as custom voices and speech recognition — take up lots of space on the device's memory card. Also, maps are getting more detailed and provide more information: outlines of forests and buildings are shown and meta-data (e.g. speed limits and lane information) is stored for most map objects. Together with the number of Points of Interest (*POI*) like cinemas, gas stations and even dentists, that are available for each map, one can easily imagine that memory space is a major concern. Where smaller, regional maps (such as Benelux or Germany maps) would still fit on the high capacity memory of the TomTom devices, larger maps (e.g., Europe or Northern America) would not, even though maps are compressed before being stored on the device. Since compressing those larger maps is not enough to store larger maps on the TomTom devices, some map data has to be removed before compression takes place. To make clear where this data removal can be accomplished, the TomTom map process can be summarized in the following four steps:

1. Maps are acquired from digital map providers such as Tele Atlas and Navteq,
2. Original maps are processed to remove unneeded data and calculate TomTom data structures. It's in this step where data removal takes place.
3. Processed maps are compressed with a *lossless*[1] compression algorithm,
4. Finally, compressed maps are shipped preloaded on TomTom devices, or sold separately, in stores and online.



(a) Original road section (b) Digitized road section

Figure 1.1: Road digitization

1.2 Road Data

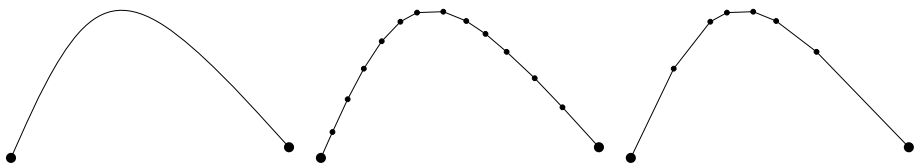
This thesis focuses on the simplification of road data in order to save memory. TomTom's map providers have special cars equipped with GPS devices, driving around the whole world sampling and digitizing road data. The coordinates of the roads are stored as discrete points at intervals of a couple of meters. Those points are divided into road sections that are stored separately as chains of points. Those road sections have, evidently, two end points and can have a number of *intermediate points*. The shape and location of the road section is defined by the end points and the intermediate points together, as is shown in figure 1.1 (the digitized road section in this example has 2 end points and 12 intermediate points). Road sections begin and end in points where other roads branch off or intersect, or at points where properties of the road change. For every road segment, besides the coordinates of the end points and intermediate points, meta data such as phonetic data, information about adjacent house numbers, maximum speed, road type and the street name is stored. At points on roads where, for example, the maximum speed changes, a new road segment starts. This meta data is, just as the begin and end points are, very important for the route information and planning algorithm of the TomTom device. Those points and their meta data may therefore not be deleted, some exceptions notwithstanding.[2] Simplifying the shape of the road by removing intermediate points is the only way to reduce the amount of road data. To do this, an algorithm which simplifies *polygonal lines* — or polylines, in short — is used in the TomTom map process.

1.3 Problem Statement

All algorithms that simplify polygonal lines have two conflicting goals. First of all, the simplification must consist of as few points as possible. In the second place, contradicting to the first requirement, the simplification must resemble the original data as much as

[1]The glossary in appendix A explains terms and abbreviations used throughout the text. Glossary items are *emphasized* on their first occurrence.

[2]Road segments from the source data are joined when the properties that change are unimportant for TomTom maps. This joining involves the conversion, and thus deletion, of end points to intermediate points.



(a) Original road section (b) Digitized road section (c) Simplified road section

Figure 1.2: Road simplification

possible. A good simplification algorithm produces data efficient results that are both good looking and realistic.

Removing intermediate points that do not contribute much the overall shape of road sections will result in road segments consisting of fewer points, and thus taking up less memory space. This cannot be done carelessly – of course – since removing too many intermediate points will result in unrealistic and unusable maps. Curved roads in particular need a lot of points to approximate the original road section, even after simplification. As an example, figure 1.2 shows an original road segment together with its digitized source and simplified variant. Although the simplified segment uses fewer points, still the simplification contains most of the original points in the curved part of the segment to maintain a smooth curve. It is on curved road sections such as the one in the example where little compression is achieved by the current algorithm.

This thesis researches a new method that not just deletes intermediate points, but replaces certain *subsections* of road sections with *circular arcs*. This way, curved road segments can be stored more efficient while not compromising on map quality. The advantage of using circular arcs over other ways to approximate curved polygonal lines is discussed in the next chapter.

Chapter 2

Polygonal Line Simplification

2.1 Existing Algorithms

Straight Lines

Different techniques exist to simplify polygonal lines. The simplest (and fastest) algorithms only remove certain points from the original polygonal line; the simplified result will therefore as well consist of only straight lines.

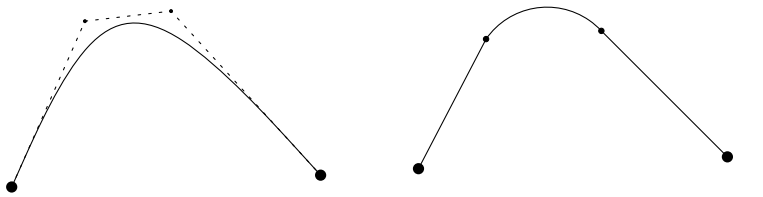
The first simplification algorithm used by TomTom to simplify its road data was the Douglas-Peucker algorithm, a very fast and simple simplification technique [6]. Its speed and simplicity come at a price, though: the algorithm does not always find the optimal simplification.

When this algorithm soon proved not to be good enough, a new algorithm was introduced in the TomTom map process. This algorithm, which is able to find the optimal simplification of every polyline [4, 9, 8], is still used by TomTom and is explained in detail in section 2.2.

Curved Lines

Especially when curved road segments need to be simplified, simplification techniques that only use straight lines will not be able to produce very data efficient simplifications. Simplified or not, still many straight lines are necessary to approximate a smoothly curved road. In such cases, using a simplification algorithm that uses *splines* or circular arcs might be a good alternative.

Algorithms exist that can approximate and simplify polygonal lines using splines [11]. Spline simplifications can be completely smooth (just like the original road sections) and can efficiently simplify curved polylines. Unfortunately, spline simplification algorithms require complex mathematics (also for drawing them, which requires changes in the software on the TomTom navigation devices), which makes using them in the TomTom map process difficult to incorporate.



(a) A spline with two control points

(b) Combination of two straight lines and one circular arc

Figure 2.1: Simplification with curved lines

Using a combination of circular arcs and straight lines can solve those problems, while still keeping the smooth characteristics of a spline approximation. Figure 2.1 compares spline simplification with circular arc simplification. The circular arc simplification still uses points from the original polyline, while it only needs one circular arc to approximate the curved part of the original road.

In [10], a simple algorithm is explained to simplify any given road with a set of circular arcs using least squares fitting, an easy and much-researched optimization method. When used for fitting a circular arc to a set of points, it tries to minimize the total sum of squared distances from each point to the arc. Unfortunately, this algorithm is too slow and inefficient to be effectively used in the TomTom map process.

Chapter 3 explains circular arcs in more detail, and attempts to find an efficient simplification algorithm that uses circular arcs to combine smoothness and simplicity.

2.2 Current Algorithm

This section describes the simplification method that is currently used to fit large maps on TomTom devices. The algorithm, described in [4, 9, 8], removes intermediate points from the original road segment, it does not add new points to the simplification; only points from the original polyline are used.[1] This way, straight line *shortcuts* are created, bypassing the removed points. Imagine a polyline with n points $p_0, \ldots, p_{n-1}$. A *line segment* $\overline{p_i p_j}$ with $i < j$ (and $i \geq 0, j < n$) is called a shortcut; it bypasses all points between point p_i and point p_j.

A line segment $\overline{p_i p_j}$ is a valid shortcut for subsection $p_i, \ldots, p_j$ if the maximum distance from $\overline{p_i p_j}$ to every point p_k with $i \leq k \leq j$ is at most ϵ, a predefined maximum error

[1] Adding new points could lead to more efficient simplifications, using less points while being more accurate. The computation of such simplifications, however, is very complex.

distance. When around each point p_k a disk with radius ϵ is constructed, all valid shortcuts bypassing p_k must intersect with this disk. This disk is called the *error disk* D_k of p_k.

The algorithm first fills a directed acyclic graph G with node set $V = \{p_0, \ldots, p_n\}$ with all valid shortcuts $\overline{p_i p_j}$, with $i < j$. Thereafter, the shortest path is computed using *Dijkstra's algorithm*.[5] Because all the segments of the original polyline are valid shortcuts in G, a path from p_0 to p_n is always possible.

Given two points, p_i and p_{i+1}, the end point p_k of all valid shortcuts $\overline{p_i p_k}$ with $k > i+1$ lies between the two tangents of D_{i+1} through p_i. The distance of those tangents to p_{i+1} is equal to ϵ. The distance of all shortcuts not between these tangents will be larger than the maximum error, and hence invalid. This notion is illustrated in figure 2.2. All shortcuts starting in p_0 must end in the area between the two tangents.[2] This area is called the *wedge*, and is shown in gray. The wedge of p_0 and D_1 is denoted as $W_{D_1}^{p_0}$.

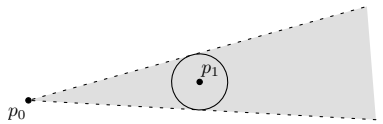


Figure 2.2: Wedge of allowed shortcuts from p_0 through D_1

A shortcut $\overline{p_i p_j}$ is a valid approximation of polyline $p_i, \ldots, p_j$ if p_j and all error disks D_k with $i < k < j$ intersect with $\mathcal{W}_{i,j}$, where

$$\mathcal{W}_{i,j} = \bigcap_{k=i+1}^{j} W_{D_k}^{p_i}$$

This makes it possible to find the longest valid shortcut $\overline{p_i p_k}$ of a subsection $p_i, \ldots, p_j$ in linear time. While $\mathcal{W}_{i,k} \neq \emptyset$, the algorithm proceeds to the next point (i.e. sets $k \leftarrow k+1$). If intersecting $\mathcal{W}_{i,k}$ with the wedge of the next point $W_{D_{k+1}}^{p_i}$ yields an empty set, all further intersections will be empty as well. This *stop criterion* permits the algorithm to stop calculating more intersections after a empty set is encountered; $\overline{p_i p_{k-1}}$ is the longest possible shortcut of subsection $p_i, \ldots, p_j$.

One only has to run the above described steps starting on all n points of polyline $p_0, \ldots, p_{n-1}$ to completely fill shortcut graph G with all possible shortcuts. This again is a linear process, which leads to the complete algorithm having a $O(n^2)$ time complexity. Running Dijkstra's shortest path algorithm on the resulting graph G will find the smallest set of shortcuts approximating the original polyline. This is done as well in $O(n^2)$ time. The running time of

[2] When p_0 lies close enough to p_1, it may be possible that D_1 intersects with p_0. In such case, the wedge $W_{D_1}^{p_0}$ is equal complete plane containing p_0 and p_1: all shortcuts starting in p_0 and bypassing p_1 are valid.

Dijkstra's algorithm is $O(|V|^2+|E|)$, where $|V|$ is the number of vertices in G and $|E|$ is the number of edges. Since G has n vertices and at most n^2 shortcuts, $O(|V|^2+|E|) = O(n^2)$.

An example of how the current algorithm works is given in figure 2.3:

(a) Point p_1 lies in the wedge, so the shortcut $\overline{p_0p_1}$ is valid,

(b) The wedge is reduced. Point p_2 lies outside of the reduced wedge, so shortcut $\overline{p_0p_2}$ is not accepted,

(c) The wedge is left unreduced, because $\mathcal{W}_{0,2} \subset W_{D_3}^{p_0}$. $\overline{p_0p_3}$ is accepted as a valid shortcut,

(d) The shortcut $\overline{p_0p_4}$ is invalid, and an empty wedge shows that no other shortcuts $\overline{p_0p_i}$ with $i \geq 4$ will be accepted.

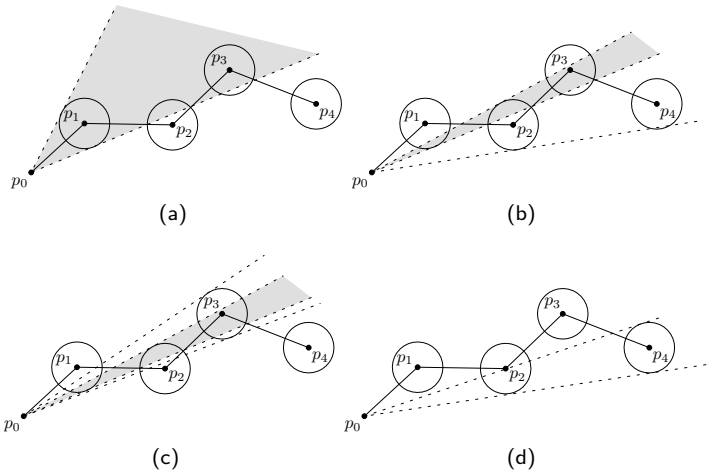


Figure 2.3: Current algorithm

Both $\overline{p_0p_1}$ and $\overline{p_0p_3}$ are added to the set G containing the valid shortcuts of polyline $p_0, \ldots, p_n$. In figure 2.4, the shortcut graph is shown of the polyline $p_0, \ldots, p_4$ from the above example.

Figure 2.5 shows another example of a shortcut graph, this time of a polyline with five points $p_0, \ldots, p_5$. Running Dijkstra's algorithm on this graph will find one of the two

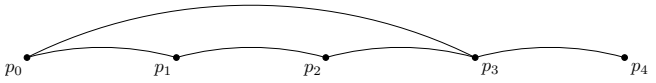


Figure 2.4: Shortcut Graph of polyline from the example in figure 2.3

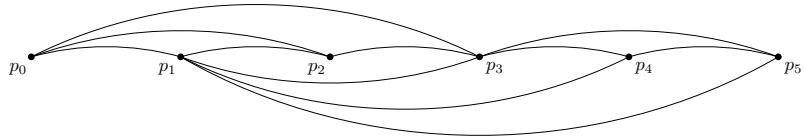


Figure 2.5: Another example of a shortcut graph

shortest paths, that approximate the original polyline with two line segments: p_0, p_3, p_5 or p_0, p_1, p_5.

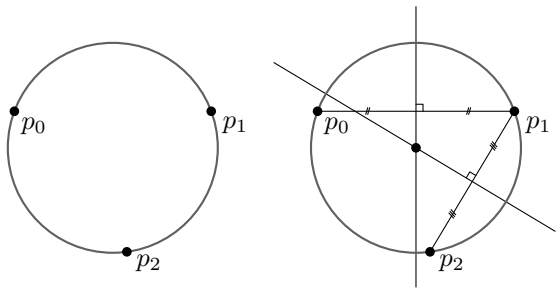
Chapter 3

Circular Arcs

Now that we have seen how the current algorithm works, we can start constructing an algorithm which uses circular arcs to smoothly simplify polygonal lines. First of all, it is important to define what a circular arc exactly is: a circular arc is a segment of the circumference of a circle.

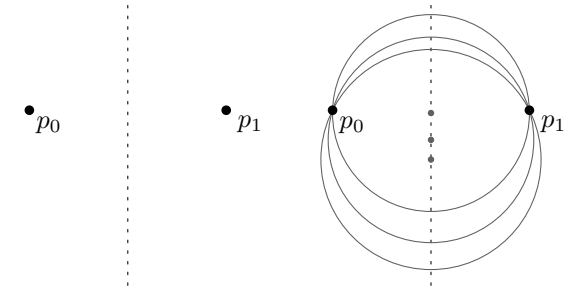
An important property of a circle is that any three non-collinear points uniquely define a circle, as can be seen in figure 3.1(a). From those three points, the center of the circle can be calculated. The center of the circle defined by p_0, p_1 and p_2 lies on the intersection of the perpendicular bisectors of line segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$. This is shown in figure 3.1(b).

When given fewer than three points, no unique circle can be defined. However, when only two points p_0 and p_1 are known, the perpendicular bisector of $\overline{p_0p_1}$ can be computed, on which all circles going through points p_0 and p_1 have their centers. This bisector is the *locus* of the centers of all circles defined by p_0 and p_1. Figure 3.2 shows this bisector, as well as three possible circles and their centers.



(a) Unique circle (b) Center

Figure 3.1: Unique circle defined by three points



(a) Locus of circle center (b) Three arbitrary circles

Figure 3.2: Two points define locus of circle center

Altogether, this yields an easy way to define a circular arc. When given two points p_0 and p_1, a number d stores the distance between the arc and the *vector* $\overrightarrow{p_0p_1}$. The position of the arc relative to the vector specifies its sign: when the arc lies above $\overrightarrow{p_0p_1}$, d is positive, d is negative when the arc lies underneath $\overrightarrow{p_0p_1}$. The special case $d = 0$ would specify an arc with an infinite radius, and consequently defines a straight line segment $\overline{p_0p_1}$.

A circular arc between two points p_0 and p_1 will from now on be denoted as $\overset{\frown}{p_0p_1}$. As an example, figure 3.3 shows four different arcs $\overset{\frown}{p_0p_1}$ between p_0 and p_1, all with different values of d.

3.1 Problem of Apollonius

Given two points p_0 and p_1, the wedge $W_{D_1}^{p_0}$ specifies two lines through p_0 tangential to D_1. The directions of all valid shortcuts from p_0 bypassing p_1 are bounded by those two lines. To find such a boundary for circular arcs instead of straight lines, we first need to know about the tangency of circles, lines and points. On this subject, much research was done by Apollonius of Perga, a Greek geometer and astronomer from the third century before Christ [3]. The problem of Apollonius, a mathematical problem named after him, is to find a circle that is tangential to a given set of three points, lines or circles, or a combination thereof. Such a circle is called an *Apollonius circle*. For three circles, there are eight possible solutions (all shown in figure 3.4), while two points and one circle yield two possible Apollonius circles (figure 3.5). We already have seen in section 3 that for three points, there is just one possible Apollonius circle.

We need to find all circles that intersect with a given point p_0 and the error disk of another point p_1. Just as two points define a locus of the centers of circles intersecting those points, such a locus is defined by a point p_0 and a circle D_1. Because two points and a circle define two Apollonius circles, the locus is two-branched: one branch b_0 closest to the point p_0 for the set of centers of circles tangential to the side of D_1 facing p_0, and one branch b_1 for

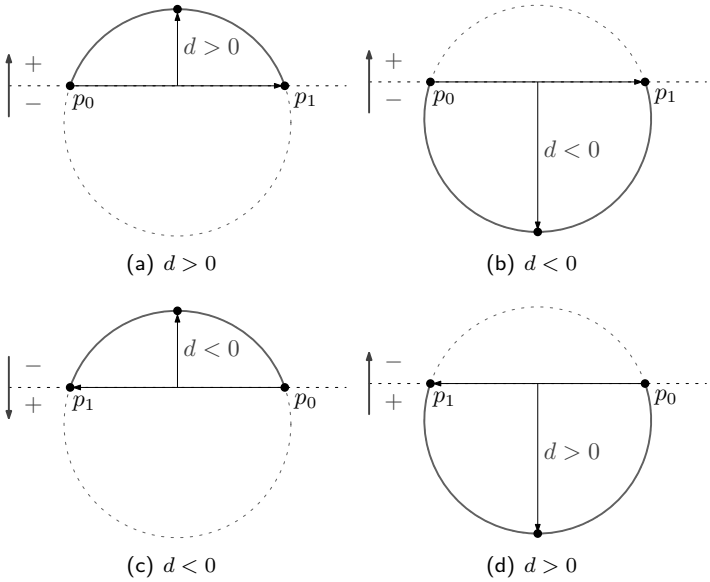


Figure 3.3: Examples of arcs with different d values

those tangential to the side opposing p_0 (as shown in figure 3.6). All circles through p_0 with their centers on the left side of b_0 will be too small to touch or intersect D_1, and all circles through p_0 with their centers right of b_1 will be too big; from this we can conclude that all circles through p_0 with their centers on or right of b_0 but on or left of b_1 either are tangential to or intersect D_1. All those circles have their center in the area between the two branches of the locus.

This locus happens to be a *hyperbola*, a type of conic section described in more detail in the next two sections. We have seen that all circles through p_0 with their centers on either one of the branches of the locus is tangential to D_1. Moreover, every circle through p_0 its center inside the area described by the two branches of the locus intersects with D_1. From now on, this hyperbola is denoted as $h^{p_0}_{D_1}$, the area as $H^{p_0}_{D_1}$.

3.2 Intersection of Hyperbolas

In section 3.1 we have seen that the position of the center of a circle through a point p_0 and a disk D_1 is restricted to the area enclosed between the two branches of a hyperbola. We will now add another point p_2 and its error disk D_2. Every circle through p_0 and D_1 must have its center in $H^{p_0}_{D_1}$, while every circle through p_0 and D_2 must have its center in $H^{p_0}_{D_2}$. From those two facts, we can draw an important conclusion: every circle through p_0, D_1 and D_2 must have its center in both $H^{p_0}_{D_1}$ and $H^{p_0}_{D_2}$, i.e. in $H^{p_0}_{D_1} \cap H^{p_0}_{D_2}$. This

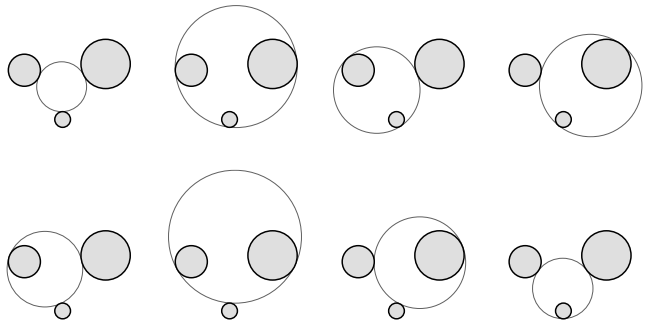
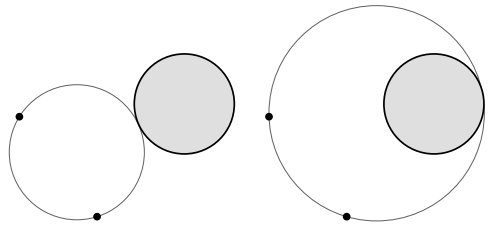
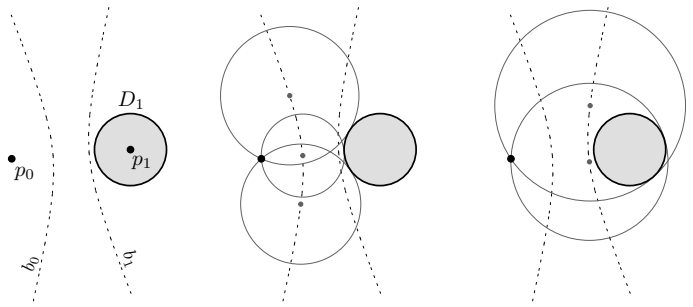


Figure 3.4: Apollonius circles: three circles



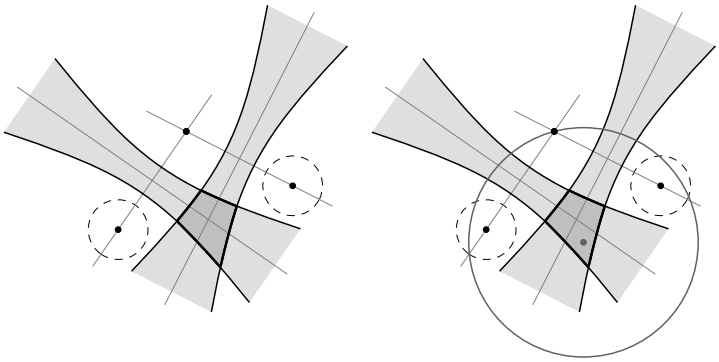
(a) Tangential to inside (b) Tangential to outside

Figure 3.5: Apollonius circles: two points & one circle



(a) Two-branched locus (b) Tangential to inside (c) Tangential to outside

Figure 3.6: Locus of circle center



(a) Intersection (b) Circle with center in intersection

Figure 3.7: Hyperbola intersection

intersection of two hyperbolas is illustrated in figure 3.7.

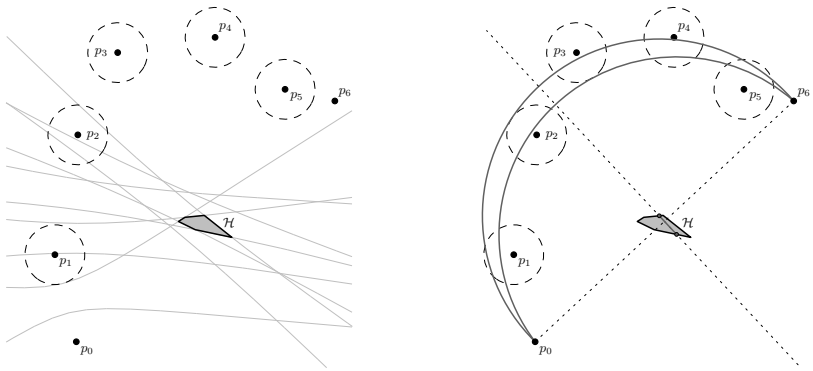
Given a polyline $p_i, \ldots, p_j$, the center of the circle through p_i and the error disks $D_{i+1}, \ldots, D_j$ of all the other points $p_{i+1}, \ldots, p_j$ has to be inside the area $\mathcal{H}_{i,j}$ with

$$\mathcal{H}_{i,j} = \bigcap_{k=i+1}^{j} H_{D_k}^{p_i}$$

When to this polyline $p_i, \ldots, p_j$ another point p_{j+1} would be added, we only have to compute the intersection of $\mathcal{H}_{i,j}$ and $H_{D_{j+1}}^{p_i}$ to see whether a circle is possible through p_i and $D_{i+1}, \ldots, D_{j+1}$. This circle is possible when

$$\mathcal{H}_{i,j} \cap H_{D_{j+1}}^{p_i} = \bigcap_{k=i+1}^{j+1} H_{D_k}^{p_i} \neq \emptyset$$

However, this does not mean that a valid circular arc shortcut is possible starting in p_i, through $D_{i+1}, \ldots, D_j$, and ending in p_{j+1}. Such an arc requires to end exactly in the last point p_{j+1}, while a circle may only intersect the error disk D_{j+1} and not the point itself. A circular arc $\overset{\frown}{p_i p_{j+1}}$ is possible if and only if its perpendicular bisector $\perp p_i p_{j+1}$ intersects with $\mathcal{H}_{i,j}$. The center of the arc must lie on the line segment (or segments) defined by the intersection of the $\perp p_i p_{j+1}$ and $\mathcal{H}_{i,j}$. Figure 3.8 shows how the intersection of a set of hyperbolas $H_{D_1}^{p_0}, \ldots, H_{D_5}^{p_0}$ define a line on which the centers of all possible arcs $\overset{\frown}{p_0 p_6}$ lie, as well as both the smallest and largest possible arc. On short lines, lines with low curvature or when a large distance threshold is used, it can sometimes happen that the intersection $\mathcal{H}$ consists of two parts. An example of such an intersection polygon is shown in figure 3.9.



(a) Intersection of hyperbolas $H_{D_1}^{p_0}, \ldots, H_{D_5}^{p_0}$

(b) Smallest and largest possible arc and their centers on $\mathcal{H}_{0,6} \cap \perp p_0 p_6$

Figure 3.8: Hyperbola intersection and circular arc placement

In this example, it would mean that the center of an arc through p_0, D_1, D_2 and p_3 may lie on the intersection of $\mathcal{H}$ and $\perp p_0 p_3$, either on the left or right side of $\overline{p_0 p_3}$.

The hyperbola intersection works the same way as the wedge structure used by the current algorithm, and enables us to define a stop criterion. By replacing the wedge $W_{D_k}^{p_i}$ from the current algorithm by the area of hyperbola $H_{D_k}^{p_i}$, we can construct an algorithm that computes circular arc shortcuts in the same way the current algorithm computes straight line shortcuts. However, the intersection of wedges only involves the subtraction of two angles, whereas intersecting hyperbolic arcs involves more complex mathematics.

3.3 Mathematics of a Hyperbola

A hyperbola is a *conic section* which consists of two symmetrical branches (see figure 3.10(a)) [2]. Conic sections are the intersections of a plane with a *circular double cone*. A circular double cone is a cone with a circular base, consisting of two *nappes*. Figure 3.10(b) shows such a cone. Depending on the position and orientation of this plane, the intersection will either be a circle, an ellipse, a parabola, a hyperbola or — when the plane intersects the cone in the *apex* — a line, a point or two lines. Figure 3.11 shows two of those conic sections: a plane perpendicular to the cone's slope yields a circular intersection, the curve produced by a plane intersecting both nappes is a hyperbola.

A hyperbola can also be described mathematically: a hyperbola is the locus of points for

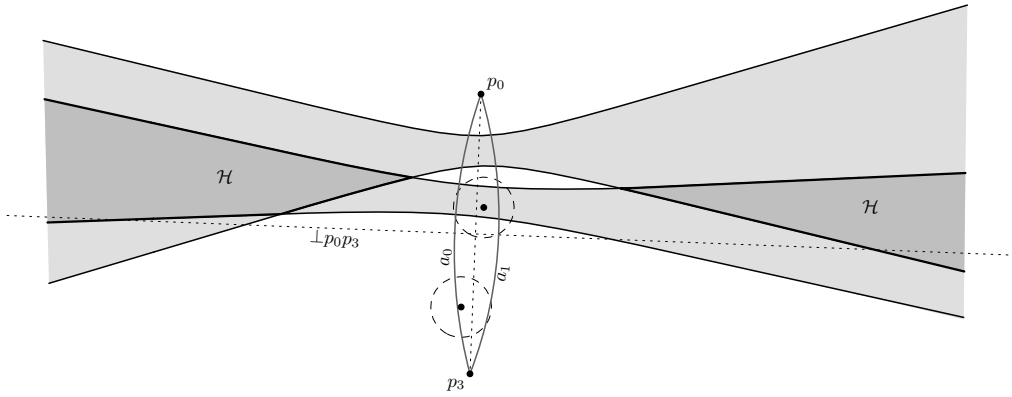


Figure 3.9: $\mathcal{H}_{0,1}$ consists, just like $\mathcal{H}_{0,1} \cap \perp p_0p_3$, of two parts

which the difference of the distances from two fixed points F_1 and F_2 (the *foci*) is a constant k. Just as an ellipse, a hyperbola has two focal points, a *major axis* and a *minor axis*. The major axis passes through the foci, the minor axis is perpendicular to the major axis. A hyperbola, centered at $(0, 0)$, can be defined by the length of the semimajor axis a and semiminor axis b (half the length of the major and minor axis, respectively):

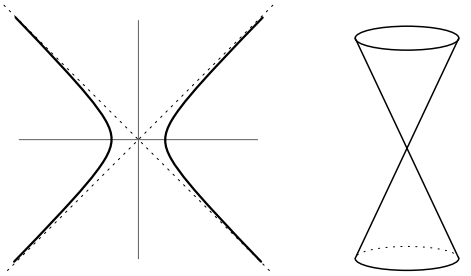
$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

From the positions of both the focal points and the length of the semimajor axis, the length of the semiminor axis can be calculated:

$$b = \sqrt{\left(\frac{|F_1F_2|}{2}\right)^2 - a^2}$$

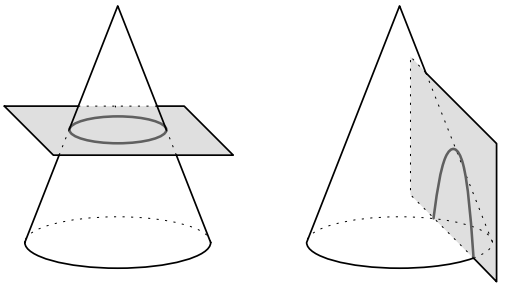
Figure 3.10(a) shows the hyperbola with $a = 1$ and $b = 1$: $x^2 - y^2 = 1$. The hyperbola intersects the x-axis at $(-a, 0)$ and $(a, 0)$. The distance between the x-intercepts is equal to the constant k, i.e., $2a = k$. The hyperbola $h^{p_0}_{D_1}$ which is the locus of the centers of all circles through p_0 and tangential to D_1, is computed with the following values:

$$F_1 = p_0$$
$$F_2 = p_1$$
$$a = \frac{1}{2}\epsilon$$



(a) Hyperbola $x^2 - y^2 = 1$ (b) Double cone

Figure 3.10: Hyperbola & Double Cone



(a) Circle (b) One branch of a hyperbolic conic section

Figure 3.11: Conic sections

3.4 Hyperbola Approximation

Although it is possible to compute the intersections of conic polygons [1], this would only add unneeded complexity to the algorithm. By using a polygonal approximation h' of the hyperbola h this complexity can be avoided. All hyperbolas have two important properties, which can be used to approximate any hyperbola with a simple polygon. For a hyperbola centered at $(0, 0)$ and its major axis parallel to the x-axis, those properties are the following (illustrated in figure 3.12):

1. Two asymptotes, both through $(0, 0)$:
$$y = \pm\frac{b}{a}x$$
2. Two tangents parallel to y-axis at x-intercept:
$$x = \pm a$$

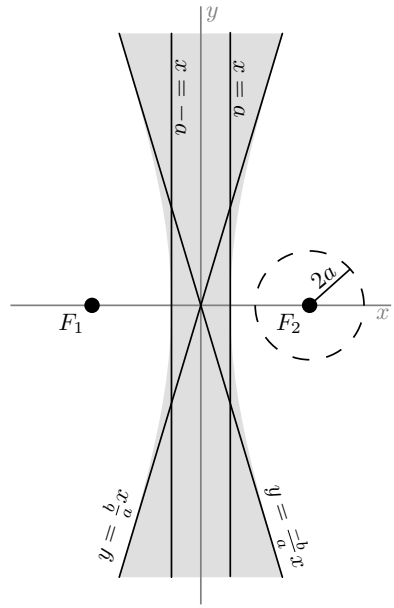
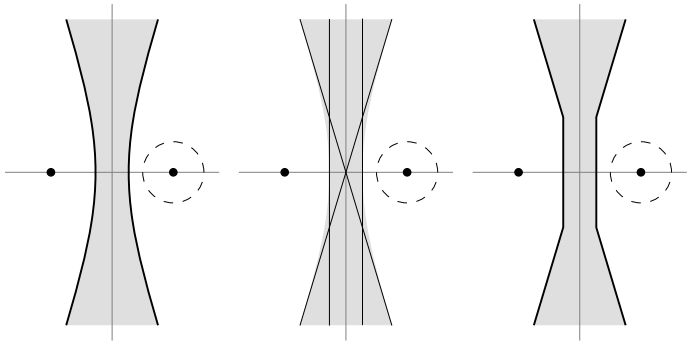


Figure 3.12: Equations of hyperbola approximation

The above equations are only valid for axis-parallel hyperbolas, centered at the point $(0, 0)$ between the two focal points F_1 and F_2. The equations for hyperbolas with different center points can easily be derived by translation. The same holds for hyperbolas with major axes not parallel to the x-axis; the asymptote and tangent equations for such hyperbolas can be constructed rotating the above listed equations.

With those four line equations, a simple polygon can be created which can approximate any given hyperbola. In figure 3.13, this process is illustrated. Using such an approximation comes at a small cost: the approximation has a slightly smaller area than the original hyperbola, so some shortcuts may be falsely rejected. Figure 3.14 shows us that a circle with its center in the intersection of two hyperbola approximations intersects with p_0, D_1



(a) Hyperbola H (b) Asymptotes & tangents (c) Approximation H'

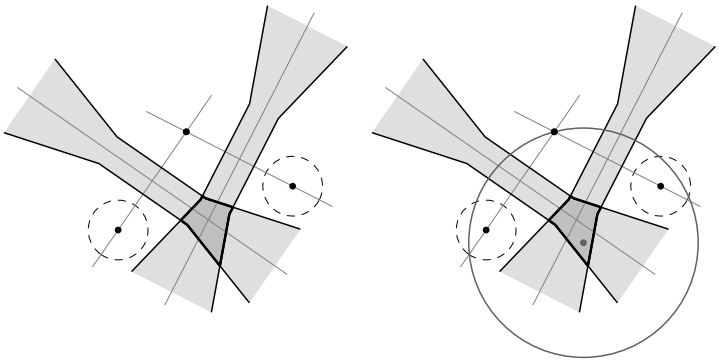
Figure 3.13: Hyperbola approximation

and D_2. The approximation is a subset of the original area, so never will false positives be accepted:

$$H' \subset H$$
$$p \in H' \Rightarrow p \in H$$

The error of the polygonal hyperbola approximation — equal to $H - H'$ — is shown in black in figure 3.15. By increasing the number of hyperbola tangents (e.g. adding more sides) used to create the approximation, the error can be decreased. All results in this thesis were created by the above described approximation, using only two tangents together with the two asymptotes.

As can be seen in figure 3.13(c), the polygonal approximation built up from the above mentioned lines will have an infinite size, while polygonal intersection algorithms usually only work on closed polygons. A variable $\mathcal{L}$ is introduced to specify the length of the approximation. This length is measured along one of the asymptotes, starting in the point where the asymptotes cross. Choosing $\mathcal{L}$ very large will ensure all intersections are computed correctly.



(a) Approximation (b) Circle with center in intersection

Figure 3.14: Approximation intersection

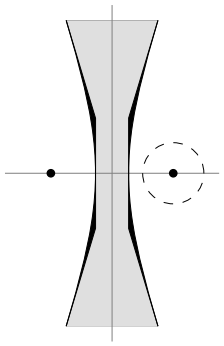


Figure 3.15: Error of hyperbola approximation: $H - H'$

Chapter 4

Circular Arc Algorithm

4.1 Algorithm on Subsections of Polygons

The algorithm on the next page checks whether it's possible to approximate a given subsection $p_i, \ldots, p_j$ with a circular arc, and, if so, computes the d value of this arc. For a given subsection $p_i, \ldots, p_j$, hyperbola intersections are computed until the stop criterion is met, i.e. when the intersection polygon $\mathcal{H}_{i,k}$ is empty. The stop criterion ensures that no further circular arc shortcuts $\overset{\frown}{p_i p_l}$ with $k < l \leq j$ are possible.

If a valid circular arc $\overset{\frown}{p_i p_k}$ can be constructed, the center of that arc lies on l, the intersection of area $\mathcal{H}$ and line $\perp p_i p_k$, the perpendicular bisector of $\overline{p_i p_k}$. If the area and line intersect, the resulting intersection will be a non-empty set of line segments. In that case, a circular arc is possible between p_0 and p_k, intersecting all intermediate error disks. The possibility of a multi-part intersection was explained earlier on page 15, such intersections only occur on low-curvature subsections, which can be simplified with a circular arc with a very large radius, either with a positive or a negative d value. If a subsection allows simplification with a circular arc with a positive d value as well as a negative, it also allows simplification with a straight line. Such subsections will be assigned a $d = 0$, since a straight line can be seen as an circular arc with an infinite radius, as chapter 3 showed us. We can therefore always choose $l[0]$, the first line segment in the non-empty set of intersecting line segments l.

We now know the line segment on which the center of the circular arc shortcut has to lie, but we do not know where exactly on this line segment $l[0]$ the center has to be placed. Figure 3.8(b) showed that placing the center on one end of the line segment gives the smallest possible circular arc shortcut, while placing the center on the other end gives the largest. A better fitting circular arc shortcut, on average closer to all bypassed points, would probably have its center somewhere halfway the two end points of the line segment $l[0]$, which is why line 9 sets p_c to be the middle point of the line segment $l[0]$.

Chapter 3 and figure 3.3 made clear that two points p_i, p_k and a value d define a circular

Algorithm 1 ComputeArcs

Input: A set of points $p_i, \ldots, p_j$ and maximum error ϵ
Output: The set $S[i, \ldots, j]$ contains the value of d when an arc is possible, or **false**

1: Set all values of S to **false**
2: $\mathcal{H} \leftarrow H^{p_i}_{D_{i+1}}$ *//Computes the hyperbola of the first two points*
3: $k \leftarrow i + 1$
4: **while** $\mathcal{H} \neq \emptyset$ and $k \leq j$ **do**
5: $k \leftarrow k + 1$
6: $l \leftarrow \mathcal{H} \cap \perp p_i p_k$ *//l is a collection of line segments*
7: **if** l contains 1 segment **then**
8: *//Circular arc $\widehat{p_i p_k}$ is possible*
9: $p_c \leftarrow$ MidPoint $(l[0])$ *//$l[0]$ is the first (and only) line segment in l*
10: $a \leftarrow \lfloor \frac{1}{2}(i + k) \rfloor$
11: $S[k] \leftarrow$ ArcFromCircle (p_i, p_a, p_k, p_c)
12: **else if** l contains more than 1 segment **then**
13: *// $p_i, \ldots, p_k$ can be simplified with a straight line*
14: $S[k] \leftarrow 0$
15: **end if**
16: **if** $k < j$ **then**
17: $\mathcal{H} \leftarrow \mathcal{H} \cap H^{p_i}_{D_{k+1}}$
18: **end if**
19: **end while**

arc. Until line 11 in the ComputeArcs algorithm, however, only the center of the circular arc is known, not the arc's d value. It is not possible to compute d from only the begin point p_i, end point p_k and center p_c of the arc, we also need to know on which side of $\overrightarrow{p_i p_k}$ the circular arc has to lie, otherwise the algorithm would not know to return which one of the two possible d values – one positive, one negative (the former would define an arc above the vector $\overrightarrow{p_i p_k}$, the latter an arc underneath). To specify which of the two possible arcs is meant, ArcFromCircle expects a point p_a on the side of $\overrightarrow{p_i p_k}$ on which the arc lies, besides points p_i, p_k and center p_c. Note that ArcFromCircle only uses p_a to determine the sign of d; p_a has to be on the right side of $\overrightarrow{p_b p_e}$ but need not the lie exactly on the computed circular arc.

Algorithm 2 ArcFromCircle

Input: Three points p_b, p_a, p_e on a circle, and its center p_c
Output: The value of d of $\widehat{p_b p_e}$, on the same side of $\overrightarrow{p_b p_e}$ as p_a

1: $s_{Arc} \leftarrow$ SideOfLine (p_b, p_e, p_a)
2: $s_{Center} \leftarrow$ SideOfLine (p_b, p_e, p_c)
3: **return** Distance $(\overline{p_b p_e}, p_c) * s_{Center} +$ Distance $(p_b, p_c) * s_{Arc}$

To determine on what side of the segment $\overrightarrow{p_i p_k}$ the points $p_{i+1}, \ldots, p_{k-1}$ lie, we could

calculate the side of each point $p_{i+1}, \ldots, p_{k-1}$. When most points lie above $\overrightarrow{p_i p_k}$ d will be positive, or negative otherwise. This operation would take $O(n)$ time, while a simple $O(1)$ operation would suffice: the point in the middle of the original subsection $p_i, \ldots, p_k$ would in most cases probably be a good candidate for p_a. This point will most likely lie the furthest away from both the begin and end point, and thus has a high change to lie on the same side of $\overrightarrow{p_i p_k}$ as most of the other points. The index of the middle point of $p_i, \ldots, p_k$ is equal to $\lfloor \frac{1}{2}(i+k) \rfloor$, so $p_a = p_{\lfloor \frac{1}{2}(i+k) \rfloor}$. In the ComputeArcs algorithm, a is computed on line 10. We can use figure 3.8(b) as an example to illustrate lines 6 to 11 from ComputeArcs that compute p_a:

6: $l \leftarrow \mathcal{H} \cap \perp p_0 p_6$

9: $a \leftarrow \lfloor \frac{1}{2}(0+6) \rfloor$

10: $p_c \leftarrow$ MidPoint (l)

11: $d \leftarrow$ ArcFromCircle (p_0, p_a, p_6, p_c)

Algorithm SideOfLine calculates whether point p_k lies above or underneath $\overrightarrow{p_b p_e}$ and returns 1 or -1, respectively. Distance calculates the distance between two points or a point and a line segment.

The HyperbolaApproximation algorithm that computes a polygon approximating any given hyperbola, is explained and shown in appendix B.

4.2 Threshold Values

Two threshold values are used to ensure that the difference between the original road and the simplification stays within acceptable margins; differences in distances and angles between line segments of the original and simplified polylines are restricted. The way this is done is explained in the next two subsections.

Distance Threshold

All line segments in the simplifications produced by the current algorithm are always within the maximum distance ϵ of the original road. This is different for the circular arc algorithm, as an example in figure 4.1 shows. All three points p_1, p_2 and p_3 are within the maximum distance ϵ (the radius of the error disks) from the circular arc shortcut $\overset{\frown}{p_0 p_4}$, but we can see that the maximum distance between $\overline{p_1 p_2}$ and $\overset{\frown}{p_0 p_4}$ is much larger. Although the shortcut is incorrect, such a shortcut could be calculated by the ComputeArcs algorithm and would be accepted if there would be no separate distance threshold check for circular arcs.

Enforcing such a distance threshold would be easy enough for straight line segments. To check whether the line segment $\overline{p_i p_j}$ is a valid approximation of subsection $p_i, \ldots, p_j$, we only have to ensure that the distance of each point $p_{i+1}, \ldots, p_j$ to $\overline{p_i p_j}$ is smaller

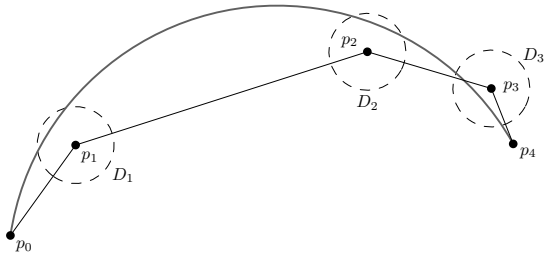


Figure 4.1: Incorrect circular arc shortcut

or equal to ϵ. For circular arcs, things are less straightforward. For example, we could construct two points p and q, as well as line segment $\overline{pq}$ and circular arc $\widehat{pq}$ between those points, as is shown in figure 4.2(a). The distance between $\overline{pq}$ and $\widehat{pq}$ is clearly not the maximum of the distances between their end points and begin points: those distances both equal zero. The distance is the distance between $\overline{pq}$ and the point on $\widehat{pq}$ where $\widehat{pq}$ has the same slope as $\overline{pq}$. It is of course more probable that the arc and the segment don't share both their start and end points. In such a case, the distance between $\widehat{pq}$ and $\overline{rs}$ is equal to the maximum of the following values, illustrated in figure 4.2(b):

δ_1: The distance between the begin point r of the segment and the segment $\widehat{pq}$,

δ_2: The distance between the end point s of the arc and the segment $\widehat{pq}$,[1]

δ_3: The distance between the point m on $\widehat{pq}$ where $\widehat{pq}$ and $\overline{rs}$ have the same slope, and $\overline{rs}$.[2]

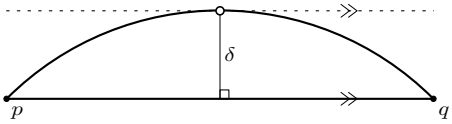
Figure 4.3 shows an example of the distance between a polyline $p_0, \ldots, p_8$ and its approximation $\{p_0, p_4, p_8\}$ consisting of a circular arc $\widehat{p_0p_4}$ and a line segment $\overline{p_4p_8}$. In this example, δ_k is the distance between $\overline{p_kp_{k+1}}$ and the shortcut bypassing that segment.

Angle Threshold

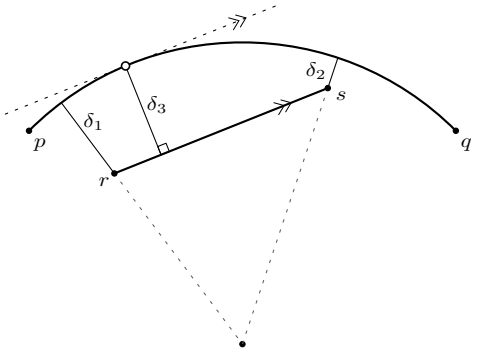
The angle threshold, just as the distance threshold, prevents big differences between the original road and the simplified version. For every point in the simplification, the angle threshold restricts the angle between the incoming and outgoing shortcuts and the edges they replace. This is important as well for navigation on the TomTom device. The device will base many of its instructions — like "Go left after six hundred meters" — on the angles between lines. Instructions will stop making sense when large angle differences are allowed.

[1] The distance from a point to a circular arc is the difference between the radius of the arc and the distance from the point to the center of the arc, or the distance to an end point of the circular arc (whichever is closest).

[2] There could me two of those points if $\widehat{pq}$ is more than a semi-circle. In such case, the point m with the largest distance to $\overline{rs}$ is chosen.



(a) Similar begin and end points



(b) Different begin and end points

Figure 4.2: Line-arc distance

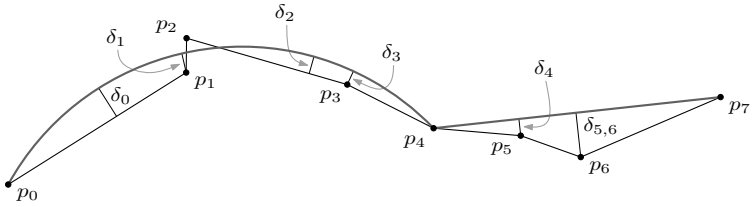


Figure 4.3: Distance threshold

Figure 4.4 shows an example of the angles between a polyline $p_0, \ldots, p_8$ and its approximation $\{p_0, p_4, p_8\}$ consisting of a circular arc $\widehat{p_0p_4}$ and a line segment $\overline{p_4p_8}$. In this example, α_k is the angle between $\overline{p_kp_{k+1}}$ and the circular arc or straight line bypassing that segment.

4.3 Polygon Intersection

The boolean intersection operation on two polygons $\mathcal{P}_1$ and $\mathcal{P}_2$ can be done in linear time[7]. The time complexity of this algorithm is $O(m+k)$, where m is the total number of vertices and k is the number of points of the intersection (i.e. the size of the output polygon). Figure 4.5 shows an example of intersecting two polygons $\mathcal{P}_1$ and $\mathcal{P}_2$. In this example, $m = 14$ and $k = 6$.

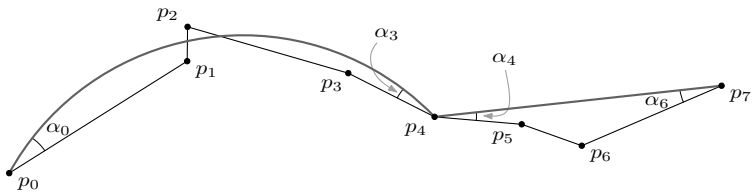


Figure 4.4: Angle threshold

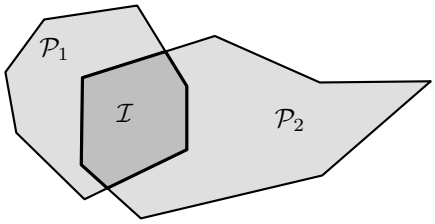


Figure 4.5: Intersection $\mathcal{I} = \mathcal{P}_1 \cap \mathcal{P}_1$

4.4 Data Representation

Storing a circular arc takes up more memory space than storing a straight line segment. As section 3 shows, it takes two points (i.e. four coordinates) to store a straight line segment, while it takes those two points and an additional number d to define a circular arc. For an arc $\widehat{pq}$, d is the distance in meters from the line segment $\overline{pq}$ to the arc. We can assume that steps of 10 cm will suffice to accurately store the details of all circular arcs. TomTom uses 32 bit signed integers to store the coordinates of all map data. Using such an integer to store the value of d would allow d to admit very large dimensions in both directions. Table 4.4 compares the maximum value of d for a number of data types, all with a resolution of 10 centimeter.

Type	Resolution	Maximum d	Cost for circular arc[3]
Signed 8 bit integer	10 cm	$\pm$ 0.03 km	9 bytes
Signed 16 bit integer	10 cm	$\pm$ 3.28 km	10 bytes
Signed 24 bit integer	10 cm	$\pm$ 838.86 km	11 bytes
Signed 32 bit integer	10 cm	$\pm$ 21474.36 km	12 bytes

Table 4.1: d data type and cost

Using an appropriate data type for storing d is very important. Not only will it save device

[3]Per segment, only the start point is stored, hence two integers. For each chain of segments, there is only one end point. The same holds for a chain of circular arcs.

memory, it is also important for distinguishing the data cost of a straight line and a circular arc when calculating the shortest path (i.e. the path with the least cost) in the shortcut graph. Looking at table 4.4 will show that using 32 bits to store d is by far too much. Never will there be roads that can be simplified with circles with d anywhere near as large as would be possible with this data type. As an 8 bit integer does not provide enough space, a signed 16 bit integer was chosen to do all tests and compute all results with.

This thesis (including all tests and results) did not take the extra data into account that would be needed to store whether a computed shortcut is a straight line or an arc.[4]

4.5 Shortcut Graph with Circular Arcs

The ComputeArcs algorithm tries to find the longest simplification of a given subsection using only circular arcs, while have also seen that it is cheaper to store straight lines than it is to store circular arcs. The algorithm will find large radius arcs as shortcuts for subsections where it would be cheaper to use a straight line. To avoid this, we can run both the current algorithm and the circular arc algorithm, combine their shortcut graphs, and afterwards compute the shortest path. Section 4.4 showed the difference in storage costs between circular arcs and straight lines; therefore, arc shortcuts (e.g. shortcuts with $d \neq 0$) will be assigned higher costs in the shortcut graph. This will ensure that Dijkstra's shortest path algorithm will only use circular arcs when they amount to a data efficient simplification.

The ShortcutGraph algorithm which computes a shortcut graph combining circular arcs and straight lines is shown below. After the two shortcut graphs A_i and C_i are computed, they are merged on line 4. This merge operation will choose the shortcut with the lowest cost if shortcuts bypassing the same subsection exist in both shortcut graphs. Figure 4.6 illustrates this with an example, where Dijkstra's algorithm would find p_0, p_1, p_5 as a one shortest path in this shortcut graph, consisting of straight line segment $\overline{p_0p_1}$ and circular arc $\widehat{p_1p_5}$. (Straight line shortcuts are drawn solid, thin and black, circular arcs dotted, thick and gray.)

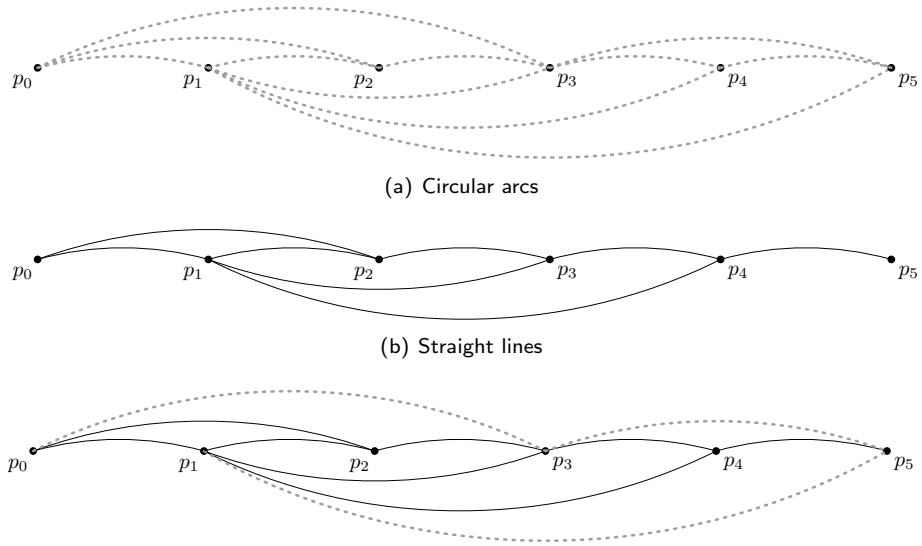
Just like ComputeArcs , CurrentAlgorithm returns a set $S[i, \ldots, n]$ containing all possible shortcuts from p_i. However, this set can only contain two values: 0 if a straight line shortcut is possible, or false otherwise.

[4] This could, for example, be done by adding one bit for each shortcut (e.g. 0 for a straight line and 1 for a circular arc), but also a more complex storage scheme could be used.

Algorithm 3 ShortcutGraph

Input: A polyline $p_0, \ldots, p_n$ and maximum error ϵ
Output: The shortcut graph G of $p_0, \ldots, p_n$

1: **for** $i = 0$ to n **do**
2: $A_i \leftarrow \text{ComputeArcs}\ (p_i, \ldots, p_n, \epsilon)$
3: $C_i \leftarrow \text{CurrentAlgorithm}\ (p_i, \ldots, p_n, \epsilon)$
4: $S_i \leftarrow A_i \cup C_i$
5: Merge S_i into G
6: **end for**
7: **return** G



(a) Circular arcs

(b) Straight lines

(c) Circular arcs and straight lines merged

Figure 4.6: Example of merging circular arc and straight line shortcut graphs

4.6 Time Complexity

Current Algorithm

The table below lists the parts of which the the current algorithm, described in section 2.2, consists. As there are at most $\binom{n}{2}$ shortcuts, the angle threshold check has a time complexity of $O(n^2)$.

Step	Complexity
Find shortcuts	$O(n^2)$
Angle threshold check	$O(n^2)$
Dijkstra's algorithm	$O(n^2)$
Total	$O(n^2)$

$+$

Circular Arcs

Imagine a road segment consisting of n points. In the worst case, the complexity of the intersection can, after n intersection operations, become $O(n^2)$. During the simplification of a road segment with n points, the intersection algorithm is called n times by ComputeArcs , which itself is called n times as well; everything combined, the time complexity of the computing a shortcut graph with circular arcs is $O(n^4)$.

Invalid shortcuts that violate a threshold value need to be removed from the shortcut graph produced by the ShortcutGraph algorithm from the previous section just before Dijkstra's algorithm is run on the shortcut graph. Computing the angle between all $O(n^2)$ shortcuts and the bypassed subsection can be done in $O(n^2)$ time, since each for each shortcut, exactly two angles have to be computed. Enforcing the distance threshold is more complicated, because each shortcut can bypass at most n line segments — and there are $O(n^2)$ in the worst case. Computing the distance between n line segments and $O(n^2)$ shortcuts can be done in $O(n^3)$ time.

Step	Complexity
Current algorithm	$O(n^2)$
Find circular arcs	$O(n^4)$
Join Dijkstra graphs	$O(n^2)$
Angle threshold check	$O(n^2)$
Distance threshold check	$O(n^3)$
Dijkstra's algorithm	$O(n^2)$
Total	$O(n^4)$

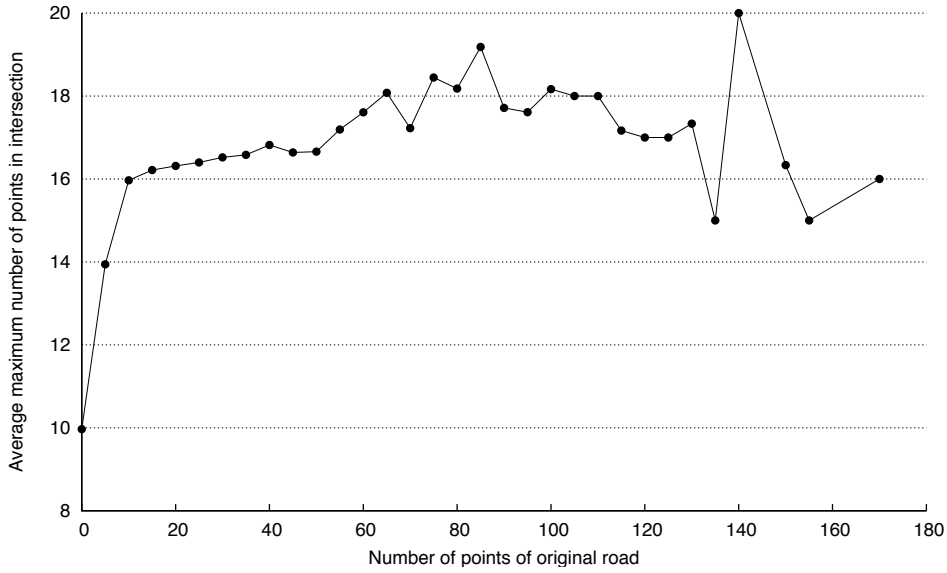
$+$

Intersection Complexity

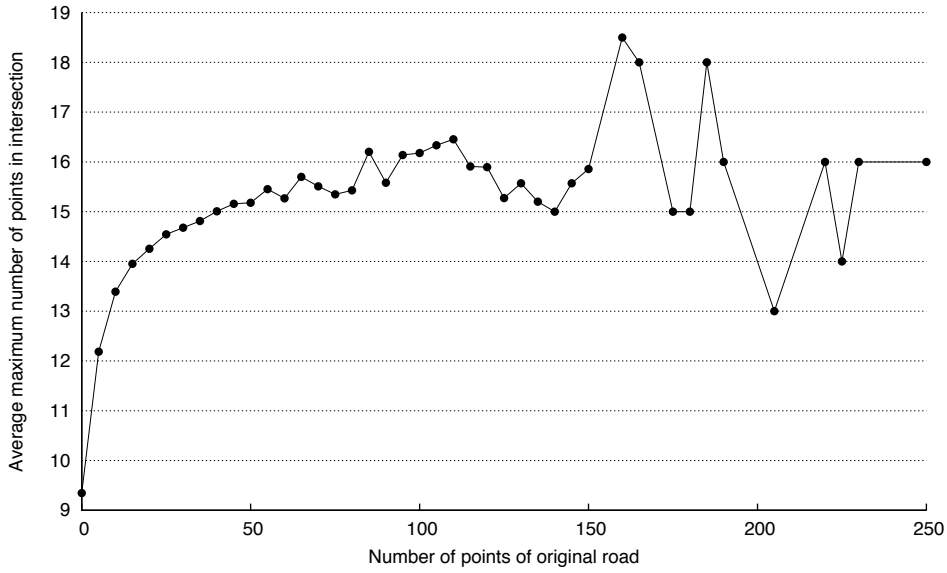
Suppose we have a subsection $p_0, \ldots, p_n$ that the algorithm can simplify with one circular arc. In theory, the resulting intersection polygon $\mathcal{H}'_{0,n}$ can have $O(n^2)$ points after n intersections, with

$$\mathcal{H}'_{0,n} = \bigcap_{k=0}^{n} H'^{p_0}_{D_k}$$

Each of the n intersection operations can add new points to $\mathcal{H}'_{0,n}$ resulting in a polygon with $O(n^2)$ points. However, in reality the complexity of $\mathcal{H}$ is more likely to stay constant than to grow with every step. We can prove this by counting the maximum number of points in intersection polygons for each road. For a road with n points, the ComputeArcs algorithm is called n times. We keep track of the number of points in the intersection polygon in each execution, and compute the maximum afterwards. The graphs below, created from complete country road sets, show the relation between road length (in points) and the maximum number of points in $\mathcal{H}$. One can imagine that, when a complete country is processed, most road lengths are not unique. For road lengths encountered more than once, the mean is taken. The two graphs in figure 4.7 compare the number of points in the complete set of roads of the Netherlands and Wisconsin with the average maximum number of points in the intersection polygon used in that road (because there are few roads with more than 100 points, the graphs get more and more spiky for longer roads). The graphs show us that there is no linear relation between the length of a road, and the number of points in the intersection polygon. Even for very long roads with more than 150 point, the intersection polygon will have at most 20 points.



(a) The Netherlands



(b) Wisconsin

Figure 4.7: Intersection complexity compared to road length in points

Chapter 5

Results

5.1 Simple Shapes

First of all, some simple shapes were created to show the basic functionality of the circular arc algorithm. Both simplification algorithms were run on all shapes, of which the results are shown in figure 5.1. The advantages of the circular arc algorithm are obvious: curves are preserved better, and the algorithm needs much fewer line segments to simplify the shapes than the current algorithm.

5.2 Real-World Road Data

TomTom uses a distance threshold of 5.5 meter and an angle threshold of around $18°$ for most maps. To prove that the above described advantages still hold when more complex, real-world road data is used, figures 5.2 to 5.5 compare the two algorithms for those threshold values.

The fixed distance threshold of 5.5 meters was chosen because allowing larger distances between the original and simplified road would result in intersections with other map objects (especially in towns and cities). We cannot use a higher distance threshold value, and using a lower one is useless, since this will only make compression rates go down. We can change the angle threshold, however. Figure 5.6 does just that, and shows the effects of using different angle threshold values for both simplification algorithms.

A more detailed comparison is done in appendix C. The complete road data set of some selected countries was simplified by both the algorithms. The appendix shows the results of those simplifications.

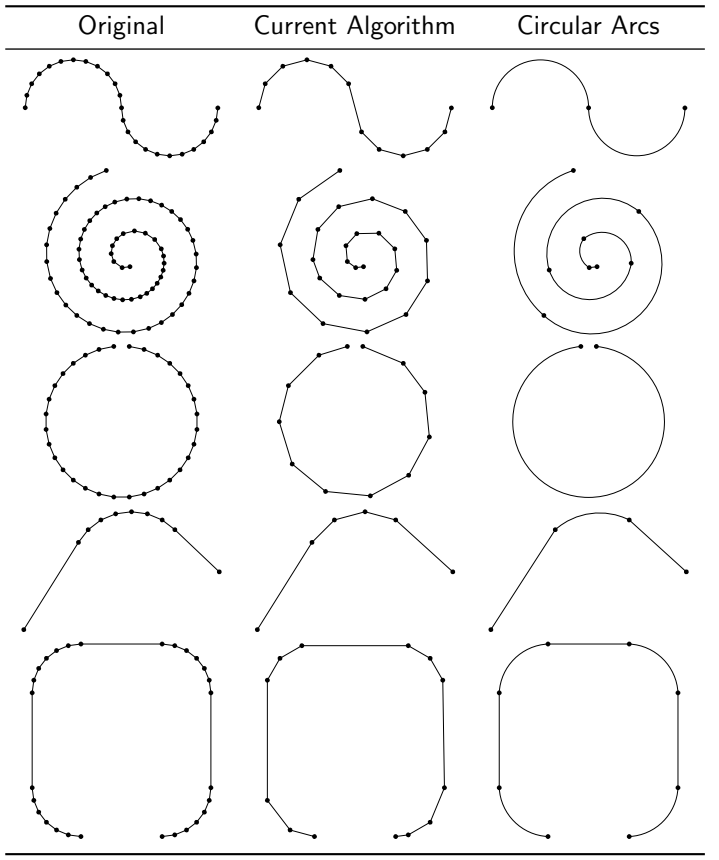
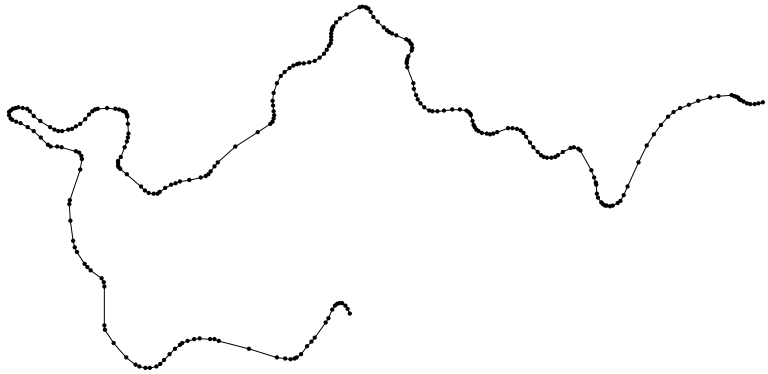
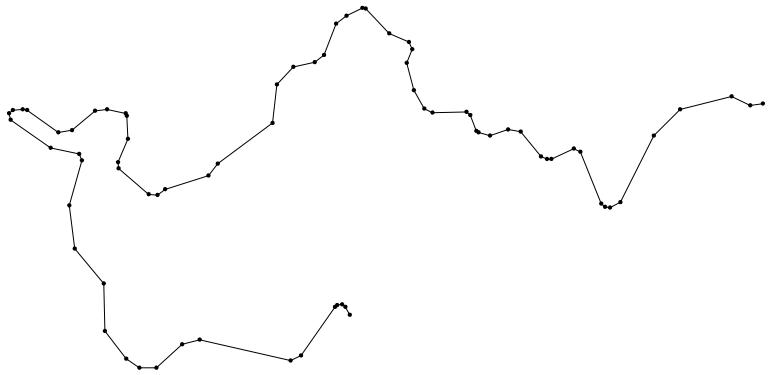


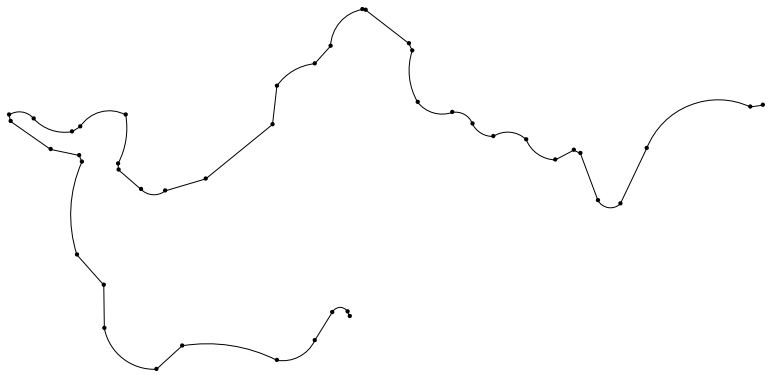
Figure 5.1: Results for both algorithms run on simple shapes



(a) Original

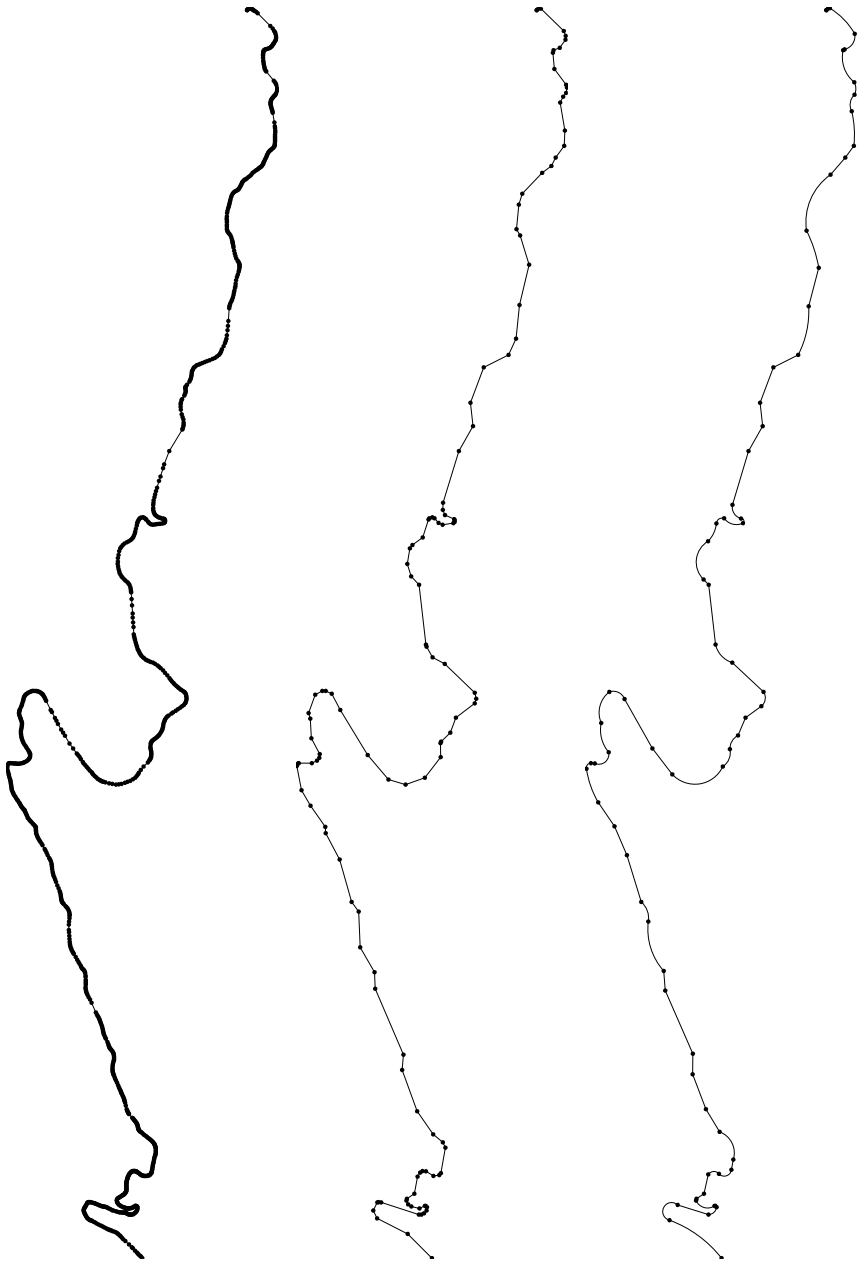


(b) Current Algorithm



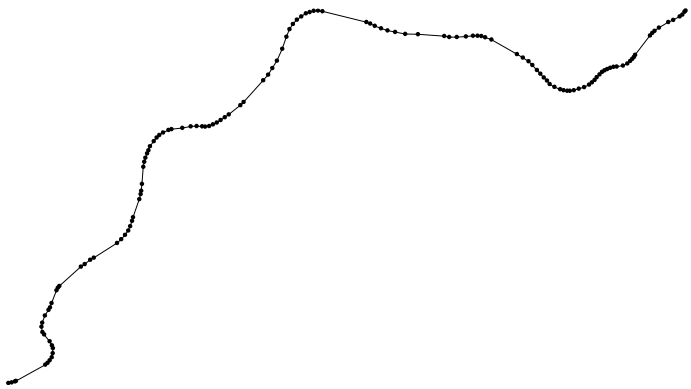
(c) Circular Arcs

Figure 5.2: Algorithm comparison with real-world road data (1)

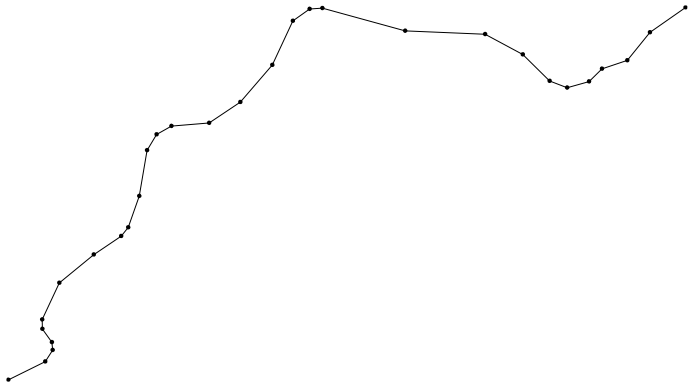


(a) Original (b) Current Algorithm (c) Circular Arcs

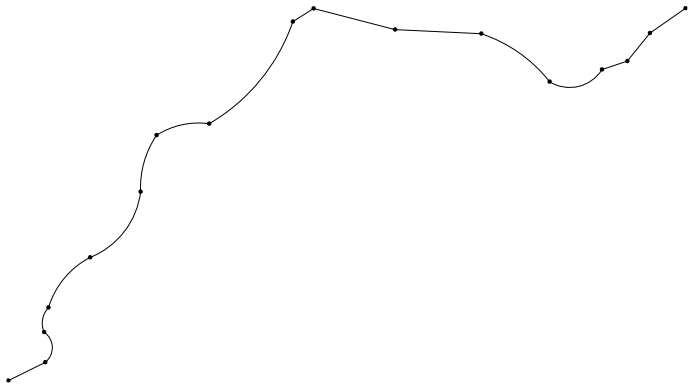
Figure 5.3: Algorithm comparison with real-world road data (2)



(a) Original

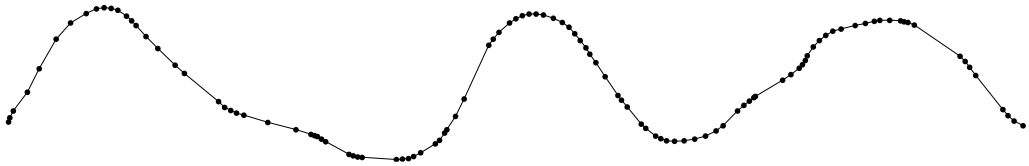


(b) Current Algorithm

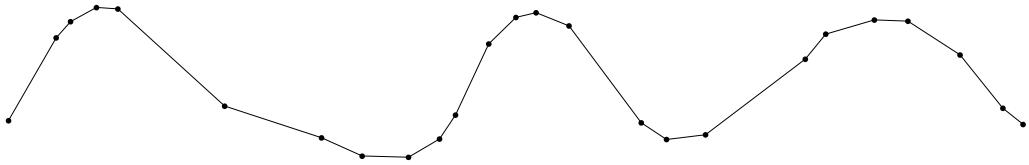


(c) Circular Arcs

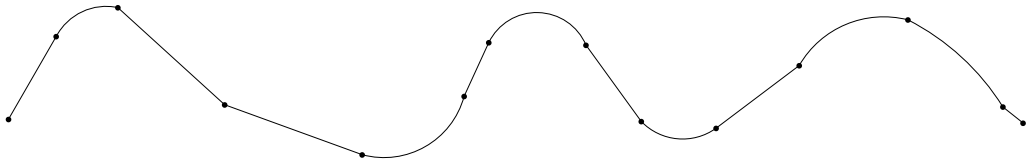
Figure 5.4: Algorithm comparison with real-world road data (3)



(a) Original



(b) Current Algorithm



(c) Circular Arcs

Figure 5.5: Algorithm comparison with real-world road data (4)

α	Current Algorithm	Circular Arcs
6°	94 points 46.8%	50 points 24.9%
12°	56 points 27.9%	28 points 13.9%
18°	45 points 22.4%	24 points 11.9%
24°	35 points 17.4%	23 points 11.4%

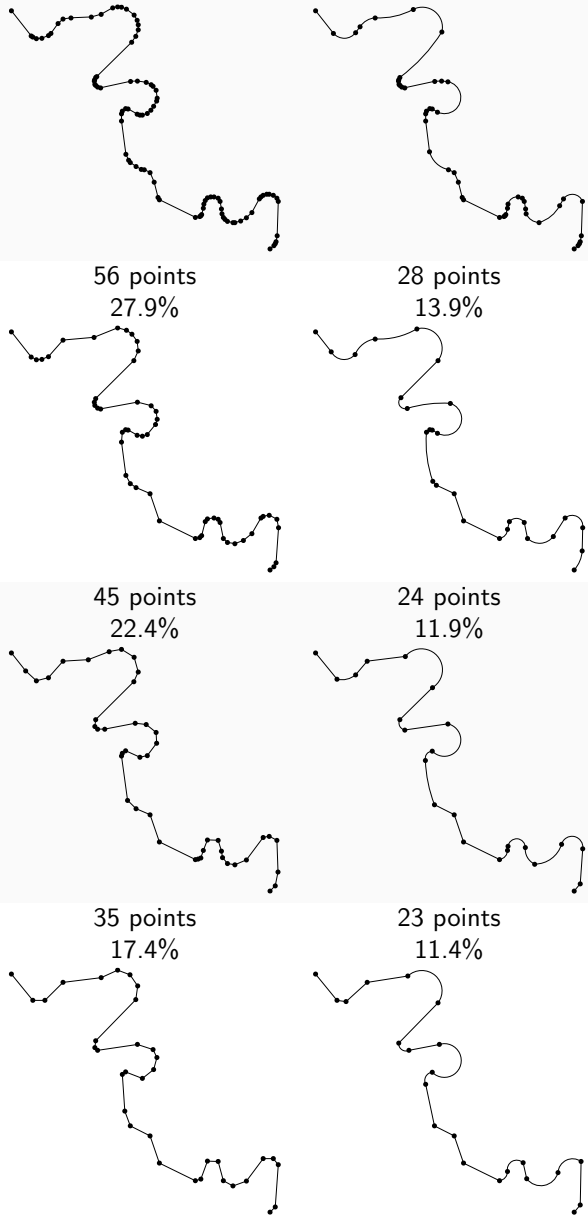


Figure 5.6: Comparison with different angle thresholds for a road section with 201 points

Chapter 6

Conclusion

The circular arc simplification technique presented in this thesis produces approximations that not only are more data-efficient than the currently used algorithm, but also look better, as we have seen in the previous chapter. Roads simplified with the circular arc algorithm are smoother and preserve the shape of the original road while using fewer points.

The way both the current algorithm and the circular arc algorithm work is very similar. Both algorithms iteratively compute intersections, controlled by a stop criterion. This analogy makes it easy to implement the circular arc algorithm in the current TomTom map process.

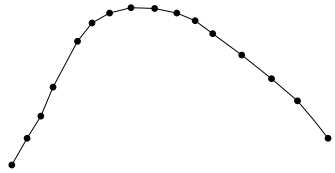
The above described advantages come at a price, unfortunately. The circular arc algorithm has a not so good time complexity, and performs therefore much slower than the current algorithm does (because only prototype code was used to compute the presented results, no exact timing results are given in this thesis). But, where new maps are simplified and compressed only once on fast and dedicated computers during the TomTom map process, every single navigation device sold will benefit from the compression and quality improvements.

Chapter 7

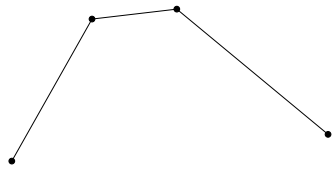
Future Work

Many things can be improved on the speed and efficiency of the circular arc algorithm, as well as on the quality of the simplifications. This chapter suggest some possible improvements:

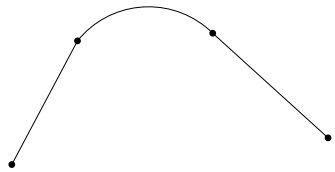
- We have seen that ComputeArcs (see section 4.1) exits when the hyperbola intersection becomes empty. Further computation is sometimes also unnecessary when the intersection polygon lies far away from the processed subsection. We can then assume that only a circular arc with a large radius can be used to approximate the subsection, while it will be cheaper to use a straight line instead.
- TomTom maps are sometimes simplified using larger distance thresholds outside built-up areas. In areas with a high map information density an error of 5.5 meters at most is needed to ensure that simplified road segments do not intersect adjoining roads or buildings. In lower density areas, the threshold values can increase. Using higher threshold values will result in simplifications with fewer points and improve compression.
- The current implementation, as seen in section 4.1, always chooses the middle of the line intersecting the hyperbola intersection polygon as the arc center. However, it might occur that choosing the middle of the intersection line will lead to an invalid simplification that does not pass angle or distance threshold tests, while choosing the arc center on another point on the intersection line would lead to a valid simplification. Better (e.g. more data efficient, smoother) simplifications may be found when the whole intersecting line is stored, and the center is computed later.
- Not always is the shortest path in the shortcut graph the best simplification. Although Dijkstra's algorithm will always find the most data efficient (i.e. cheapest) simplification, slightly longer paths often are smoother and better looking. It would be very difficult, however, to decide which of the short paths is the best choice. Figure 7.1 illustrates this: the shortest path algorithm will find the cheapest simplification of the original road, while sometimes the smoothness of the result can be improved at only little cost.



(a) Original road



(b) Cheapest simplification, shortest path in shortcut graph



(c) Slightly more expensive, but better looking simplification

Figure 7.1: Line or Arc

- Instead of the simple eight-sided polygon approximations currently used, more complex approximations could be used. This would reduce the error of the hyperbola approximation, and hence the number of missed arc shortcuts.

Chapter 8

Acknowledgments

I would like to thank the following persons for their help during this project:

At Utrecht University:

- **Marc van Kreveld**, for supervising me during the project, and providing me with ideas and help.

At TomTom:

- **David Martens** and **Martin Wolf**, for providing me with direct help on project and TomTom related questions,
- **Sergei Kucheiko**, for taking me into his team,
- **Joost Voogt**, for proof-reading this thesis.

References

[1] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Kurt Mehlhorn, and Elmar Schömer. A computational basis for conic arcs and boolean operations on conic polygons. In *In ESA 2002, Lecture Notes in Computer Science*, pages 174–186. Springer-Verlag, 2002.

[2] Robert Bix. *Conics and Cubics: A Concrete Introduction to Algebraic Curves*. Springer, New York, 1998.

[3] C.B. Boyer. *A History of Mathematics*. Wiley, 1968.

[4] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In *ISAAC '92: Proceedings of the Third International Symposium on Algorithms and Computation*, pages 378–387. Springer-Verlag, 1992.

[5] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, second edition, 2001.

[6] D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10:112–122, 1973.

[7] Ulrich Finke and Klaus H. Hinrichs. Overlaying simply connected planar subdivisions in linear time. In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 119–126. ACM, 1995.

[8] H. Imai and M. Iri. Polygonal approximations of a curve - formulations and algorithms. In G.T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier Science Publishers, 1988.

[9] M. van Kreveld M. de Berg and and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257, October 1998.

[10] Paul L. Rosin and Geoff A. W. West. Segmentation of edges into lines and arcs. *Image Vision Computing*, 7(2):109–114, 1989.

[11] E. Saux and M. Daniel. Data reduction of polygonal curves using b-splines. *Computer-Aided Design*, 31:507–515, 1999.

Appendices

Appendix A

Glossary

Apex The topmost single point of a cone. In a double cone, the apex is the point connecting the two nappes.

Circular arc A connected part of the circumference of a circle. A circular arc between points p and q is denoted as $\widehat{pq}$.

Cone A geometric shape consisting of a base, a single point (the apex), and all lines connecting those two. A circular cone is a cone with a circular base, assuming that the apex is positioned in such a way that a line through the apex and the center of the circular base is perpendicular to the plane containing the circular base.

Error disk A disk centered at a point with radius ϵ. The error disk of a point p_i is denoted as D_i. All points within a distance ϵ from p_i are within the disk D_i.

Hyperbola A two-branched conic section with two focal points F_1 and F_2. The hyperbola with focal points p and q and major axis length $\frac{1}{2}\epsilon$ is denoted as $h^p_{D_q}$.

Intermediate point A point from a polygonal line, not being the first or last point (i.e., the begin or end point).

Line segment Finite part of infinite line bounded by two end points. A line segment between points p and q is denoted as $\overline{pq}$.

Locus The set of points satisfying some condition.

Lossless compression A manner of data compression which allows the exact original data to be computed from the compressed data, as opposed to lossy compression which does not allow this.

Nappe One half of a double cone, split at the apex.

Perpendicular bisector A line segment perpendicular to $\overline{pq}$, going through the point halfway p and q. This perpendicular bisector is denoted as $\perp pq$.

POI Point of Interest. A map location of possible interesting place or point, such as hospitals, train stations, hotels or gas stations.

Polyline Short for polygonal line: a continuous line composed of one or more line segments.

Shortcut Line or arc between two points p_i and p_j, with $i < j$, bypassing all points p_k with $i < k < j$ from the original polyline.

Spline A smooth continuous curve passing a given set of (control) points.

Stop criterion A criterion which stops continuation of a loop or process after it is met.

Subsection Set of points $p_i, \ldots, p_j$, being a subsection of polyline $p_0, \ldots, p_n$ with $0 \leq i < j \leq n$.

Vector A vector from point p to point q, denoted as $\overrightarrow{pq}$.

Vertex A point of a polygonal line or polygon.

Wedge The area between the two half-lines (i.e. rays) starting in point p and tangential to the error disk D_q of point q. This wedge is denoted as $W^p_{D_q}$.

Appendix B

Hyperbola Approximation Algorithm

The HyperbolaApproximation algorithm, shown on the next page, calculates the approximation H' of any hyperbola. As input variables, the algorithm expects the two foci F_1, F_2, maximum error ϵ and the length of the approximation $\mathcal{L}$. Figure B.1 explains the points and lines the algorithm uses to come to its final result.

Depending on the orientation of F_1 and F_2, lines 23 to 36 make sure that the points e_k and i_k with $1 \leq k \leq 4$ are in the right order, before creating H'.

The algorithm uses a series of sub-algorithms:

Line (p_1, p_2) returns a line through points p_1 and p_2,

Line (p, b) returns a line through point p with slope b,

Intersection (l_1, l_2) returns the point where lines l_1 and l_2 intersect,

Rotate (l_1, p, l_2) rotates l_1 around p, over the angle of line l_2 with the x-axis,

Vector (l, s) returns a vector with the slope of line l and length s,

Swap (a, b) swaps the contents of objects a and b.

Algorithm 4 HyperbolaApproximation

Input: Focal points F_1, F_2, maximum error ϵ and approximation length $\mathcal{L}$

Output: A polygonal approximation H' of a hyperbola with focal points F_1, F_2 and major axis length

1: $m \leftarrow \text{MidPoint}\ (F_1, F_2)$
2: $a \leftarrow \frac{1}{2}\epsilon$
3: $v \leftarrow \text{Vector}\ (\overline{F_1F_2}, a)$
4: $t_1 \leftarrow \text{Line}\ (m + v, \perp F_1F_2)$ //*Calculate tangents*
5: $t_2 \leftarrow \text{Line}\ (m - v, \perp F_1F_2)$
6: $c \leftarrow \frac{1}{2}\ \text{Distance}\ (F_1, F_2)$ //*Calculate hyperbola variables*
7: $b \leftarrow \sqrt{c^2 - a^2}$
8: $a_1 \leftarrow \text{Line}\ (m, b/a)$ //*Calculate asymptotes of y-axis parallel hyperbola*
9: $a_2 \leftarrow \text{Line}\ (m, -b/a)$
10: $\text{Rotate}\ (a_1, m, \overline{F_1F_2})$ //*Rotate asymptotes*
11: $\text{Rotate}\ (a_2, m, \overline{F_1F_2})$
12: $i_1 \leftarrow \text{Intersection}\ (t_2, a_1)$ //*Calculate intersections*
13: $i_2 \leftarrow \text{Intersection}\ (t_2, a_2)$
14: $i_3 \leftarrow \text{Intersection}\ (t_1, a_1)$
15: $i_4 \leftarrow \text{Intersection}\ (t_1, a_2)$
16: $e_1 \leftarrow m + \text{Vector}\ (a_1, \mathcal{L})$ //*Calculate end points*
17: $e_2 \leftarrow m - \text{Vector}\ (a_2, \mathcal{L})$
18: $e_3 \leftarrow m - \text{Vector}\ (a_1, \mathcal{L})$
19: $e_4 \leftarrow m + \text{Vector}\ (a_2, \mathcal{L})$
20: $s_1 \leftarrow \text{SideOfLine}\ (i_2, i_3, e_2)$ //*Calculate relative positions*
21: $s_2 \leftarrow \text{SideOfLine}\ (i_1, i_2, e_2)$
22: $s_3 \leftarrow \text{SideOfLine}\ (i_1, i_2, e_1)$
23: $\mathcal{B} \leftarrow$ **false** //*Correct order of points*
24: **if** $s_1 = -1$ **then**
25: $\quad \text{Swap}\ (e_1, e_2)$
26: $\quad \text{Swap}\ (e_3, e_4)$
27: $\quad \mathcal{B} \leftarrow$ **true**
28: **end if**
29: **if** $s_2 = -1$ **then**
30: $\quad \text{Swap}\ (e_1, e_4)$
31: $\quad \text{Swap}\ (e_2, e_3)$
32: $\quad \mathcal{B} \leftarrow$ **true**
33: **end if**
34: **if** $s_3 = -1 \wedge \mathcal{B} =$**false then**
35: $\quad \text{Swap}\ (e_1, e_3)$
36: **end if**
37: $H' \leftarrow i_1, i_2, e_2, e_3, i_3, i_4, e_4, e_1$ //*Fill H' with points*
38: **return** H'

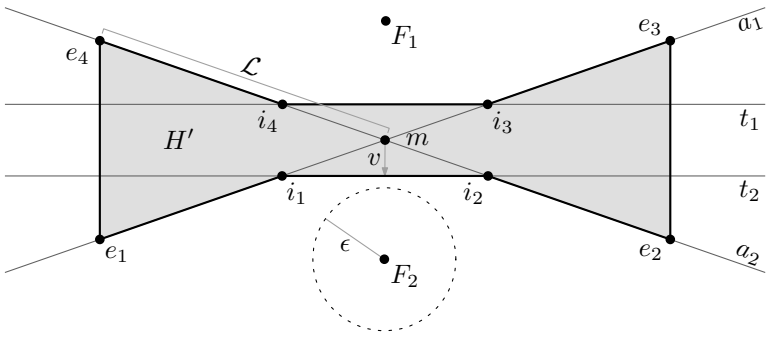


Figure B.1: Points and used in HyperbolaApproximation

Appendix C

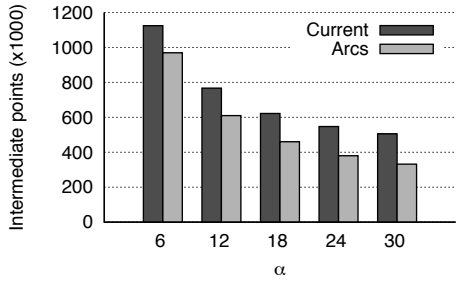
Results for Complete Road Data Sets

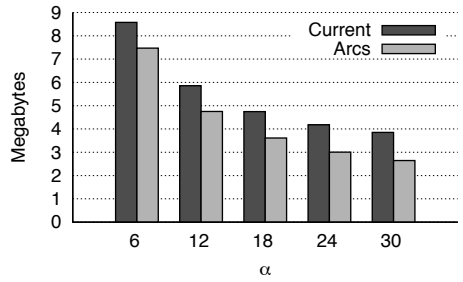
The following sections compare the current algorithm and the circular arc algorithm for the complete road data sets of a selected number of countries with the following threshold values: $\epsilon = 5.5$ meter and $\alpha = 6°, 12°, 18°, 24°, 30°$.

C.1 The Netherlands

Number of roads: 2,518,170
Intermediate points: 2,600,330

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	1,124,443	43.2%	969,444	37.3%	8,995,544	43.2%	7,831,625	37.6%
12°	767,342	29.5%	610,288	23.5%	6,138,736	29.5%	4,979,810	23.9%
18°	621,926	23.9%	460,209	17.7%	4,975,408	23.9%	3,788,843	18.2%
24°	547,630	21.1%	379,795	14.6%	4,381,040	21.1%	3,150,560	15.1%
30°	505,103	19.4%	332,180	12.8%	4,040,824	19.4%	2,772,047	13.3%

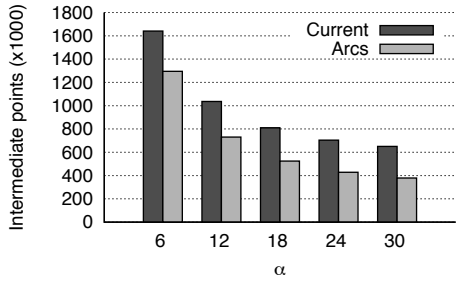


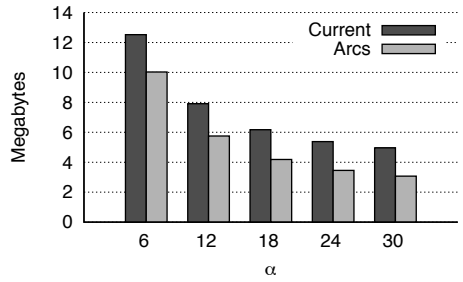


C.2 Switzerland

Number of roads: 1,424,450
Intermediate points: 4,415,682

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	1,641,136	37.2%	1,295,326	29.3%	13,129,088	37.2%	10,513,246	29.8%
12°	1,036,380	23.5%	730,081	16.5%	8,291,040	23.5%	6,028,325	17.1%
18°	809,265	18.3%	524,535	11.9%	6,474,120	18.3%	4,390,792	12.4%
24°	704,665	16.0%	428,576	9.7%	5,637,320	16.0%	3,623,111	10.3%
30°	650,750	14.7%	378,633	8.6%	5,206,000	14.7%	3,221,070	9.1%

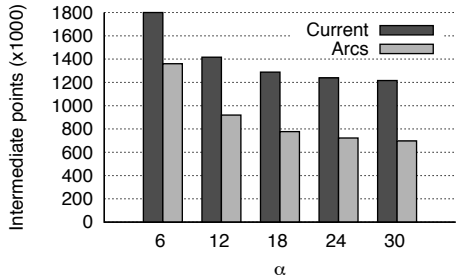


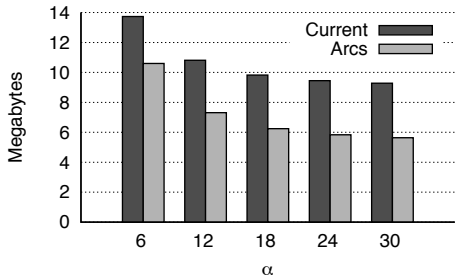


C.3 Colorado

Number of roads: 1,170,480
Intermediate points: 3,298,935

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	1,799,693	54.6%	1,360,117	41.2%	14,397,544	54.6%	11,111,858	42.1%
12°	1,416,612	42.9%	919,812	27.9%	11,332,896	42.9%	7,671,520	29.1%
18°	1,287,752	39.0%	777,076	23.6%	10,302,016	39.0%	6,547,252	24.8%
24°	1,238,488	37.5%	722,359	21.9%	9,907904	37.5%	6,112,241	23.2%
30°	1,215,929	36.9%	697,605	21.1%	9,727,432	36.9%	5,913,584	22.4%

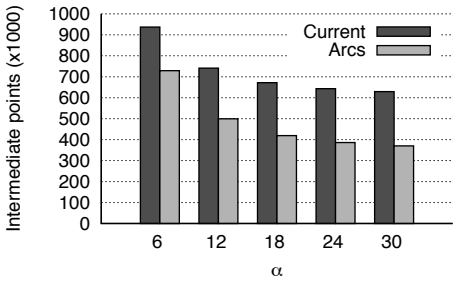


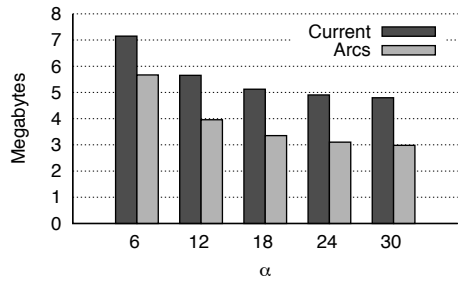


C.4 Wisconsin

Number of roads: 1,183,662
Intermediate points: 1,884,713

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	937,015	49.7%	729,072	38.7%	7,496,120	49.7%	5,946,746	39.4%
12°	740,889	39.3%	499,562	26.5%	5,927,112	39.3%	4,152,060	27.5%
18°	671,490	35.6%	419,149	22.2%	5,371,920	35.6%	3,518,948	23.3%
24°	643,053	34.1%	386,188	20.5%	5,144,424	34.1%	3,256,954	21.6%
30°	629,285	33.4%	370,229	19.6%	5,034,280	33.4%	3,129,269	20.8%

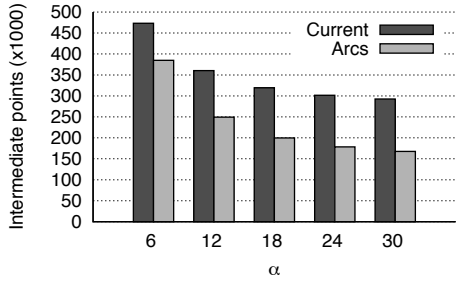


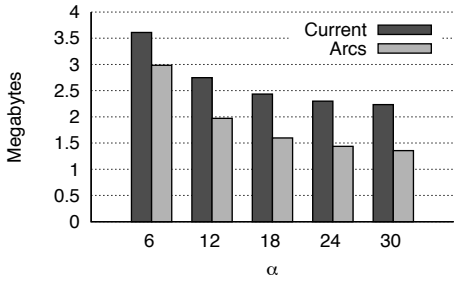


C.5 Hungary

Number of roads: 1,097,784
Intermediate points: 1,053,409

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	47,3201	44.9%	38,4829	36.5%	3,785,608	44.9%	3,129,816	37.1%
12°	36,0279	34.2%	24,9342	23.7%	2,882,232	34.2%	2,066,967	24.5%
18°	31,9236	30.3%	19,9749	19.0%	2,553,888	30.3%	1,676,871	19.9%
24°	30,1619	28.6%	17,8294	16.9%	2,412,952	28.6%	1,507,233	17.9%
30°	29,2682	27.8%	16,7751	15.9%	2,341,456	27.8%	1,423,254	16.9%





C.6 Luxembourg

Number of roads: 95,890
Intermediate points: 425,506

	Intermediate points				Bytes			
α	Current		Arcs		Current		Arcs	
6°	124,389	29.2%	90,933	21.4%	995,112	29.2%	743,371	21.8%
12°	78,838	18.5%	51,932	12.2%	630,704	18.5%	432,620	12.7%
18°	63,271	14.9%	38,720	9.1%	506,168	14.9%	326,612	9.6%
24°	56,621	13.3%	32,808	7.7%	452,968	13.3%	278,973	8.2%
30°	53,305	12.5%	29,707	7.0%	426,440	12.5%	253,974	7.5%