Novel drawing algorithms and application of texture mapping for 2D cartographic line symbolization

Diss. ETH No. 17439

Novel Drawing Algorithms and Application of Texture Mapping for 2D Cartographic Line Symbolization

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by

Fei He

M.Sc. computer science, XIOPM, Xi'an China

Date of birth
June 12, 1975

citizen of
People's Republic of China

accepted on the recommendation of

Prof. Dr. Lorenz Hurni, examiner
Prof. Dr. Liqiu Meng, co-examiner
Dr. Hansruedi Bär, co-examiner

2008

Table of Contents

Acknowledgments

I would like to express my sincere thanks to those who helped me to make this thesis possible.

Prof. Dr. Lorenz Hurni, the director of my thesis,

for his full support and scientific instructions during my work in his research group, and also for his careful reading and editing of the thesis. He is not only an experienced supervisor, but also a good friend in life. He always gives the necessary help at best time.

Prof. Dr. Liqiu Meng, the co-examiner of my thesis,

for her careful reading and editing of the thesis, kindly encouragement and help.

Dr. Hansruedi Bär, the co-examiner of my thesis,

for his scientific instructions and discussions during the progress of this work, the careful reading, correction and editing of the thesis.

Many thanks also go to my colleagues, especially Peter Sykora, Michael Cooper, Marion Werner, Tobias Clemens Dahinden, Olaf Schnabel, Sonja Gauch, Margrit Péquignot, and all the previous and present members of the institute, for their support during my living and work in Zurich.

Special thanks will give to two friends, Qingfu Zeng and Qingyong Meng, for their encouragement and medical suggestions. Without their kindly help, I have no way to escape from the torture of depression and finish this thesis.

Last but not least, I thank my husband, Dr. Wei Liu and my mother, Ms. Hao Li, for their caring love, encouragement, and support during the preparation of this thesis.

Summary

A high quality map is characterized by the appropriate presentation of the basic graphic elements, including points, lines and areas. At present, most of the commonly used graphic software was designed for general purpose of digital image production; many cartographic principles are not completely satisfied. As a result, a time-consuming modification work is typically required for a convincing cartographic product.

In this thesis, two approaches were studied to improve this situation by focusing on one graphic element, the line. The first approach is dealing with cartographic elaborations of line geometries. New drawing algorithms were developed to generate lines better satisfying the cartographic requirements. Many unsatisfied presentations that often occur at line ends, turns and intersections, can be avoided during the rendering process.

The second approach aims to enrich the line symbols, covering both categories and visual qualities. First, efficient drawing methods were developed to draw some complex lines, including both single and multiple lines. Second, the computer graphic technique of texture mapping was the first time introduced into cartography; which turns out to be an interesting tool to greatly improve line symbolizations.

These achievements lead to the production of a new program, SmartLine. It contains a flexible editorial environment, allowing map makers to adjust the graphic attributes of line symbols easily. This work aims to improve the

generation of line symbols, which is not only useful for cartographers but also for general users of the interactive map products.

Zusammenfassung

Eine qualitativ hochstehende Karte ist durch eine gute Lesbarkeit und grafisch aufeinander abgestimmte Kartenelemente im Kartenbild charakterisiert. In der digitalen Kartografie unterstützen Computerprogramme und -techniken den Kartenautor in vielen Fällen bei dieser Arbeit. Trotzdem ist auch heute noch ein Grossteil der häufig genutzten Grafikprogramme nur für allgemeine gestalterische Arbeiten geeignet, spezielle kartografische Wünsche wie komplexe Doppel- oder Dreifachlinien werden nicht berücksichtigt. Daher müssen aufwändige Modifikationen einzelner Linienobjekte in der Karte vorgenommen werden.

Diese Arbeit will einen Beitrag zur Verbesserung der Liniendarstellung leisten. Daher wurden in einem ersten Ansatz die kartografischen Darstellungen und Zeichenalgorithmen des Grafikelements Linie analysiert. Auf dieser Basis wurden bestehende Zeichenalgorithmen für Linien entsprechend den kartografischen Anforderungen verbessert. Dazu wurden zum Beispiel komplexe Linien wie Mehrfachlinien mit Hilfe von Primitiven beschrieben, was zu einer Steigerung der Effizienz der Methoden führte. So konnten viele unbefriedigende Effekte an Linienenden, -ecken und -kreuzungen während des Renderingprozesses eliminiert werden.

In einem zweiten vollautomatischen Ansatz wurden erstmals in der Kartografie ein Musterabbildungsalgorithmus entwickelt, der Texturen auf Linien projiziert

und effizient anordnet. Dieser Ansatz eröffnet durch die Flexibilität der Anwendung neue Darstellungsmöglichkeiten für Linien.
Um die verbesserten Zeichenalgorithmen und deren Ergebnisse zu validieren, wurde ein flexibles Programm zur Visualisierung und Editierung von Linienstilen entwickelt. Damit können eigene Geodaten und Linienstile eingelesen und über GUI-Elemente wie Schieberegler in ihrem Aussehen einfach modifiziert werden. Die resultierenden Linienstile werden als XML-basierte Beschreibung exportiert und stehen für weitere Anwendungen zur Verfügung.

Chapter 1 Introduction

1.1 Motivation

1.1.1 Cartographic line symbolization

The geo-spatial information stored on a map is transferred to map users through the map language, symbolization. It has been a central task of cartography to study how to abstract the physical world (reality) into map symbols, which are built up by points, lines and areas.

As a kind of basic graphic primitive, lines are used to express an extensive range of geo-spatial referenced phenomena. Many physical objects can be symbolized as line symbols, like rivers, traffic paths, cables, pipelines and so on. They are also used to mark outlines, like those of lakes, mountains and coasts. In addition, abstract information, such as wind direction, political borders and contour lines, can also be depicted as line symbols. Accordingly, a great variety of line symbols are needed to meet these requirements. In map design, an important strategy to mark different line symbols is to provide them with diversified patterns. A single line can be drawn in a solid, dashed, dotted or dash-dotted manner. Double line, triple line or other lines drawn with complex components are also frequently used to depict subjects that need to be highlighted. Combined with other graphic attributes, including color, size, texture etc, a large amount of line symbols could be created.

However, a successful cartographic symbolization is more than just to label the different objects; a more sophisticated task is to achieve a balance between the clarity and the specificity of map symbols. Otherwise, it might generate ambiguous information or even mislead the readers. Regarding the line symbols, special attention should be paid on non-solid lines, including dashed lines, dotted lines and dash-dotted lines [Hurni, 1995]. Fig. 1.1 lists the comparison of some appropriate and inappropriate line symbols. A main problem of the unsatisfied symbols is that gaps should not be presented at the critical positions on line geometries, like sharp turns, intersections and line ends (Fig. 1.1 a, b, c). For lines drawn with complex components, the arrangement of the drawing units is also important to avoid confused information (Fig. 1.1 d).

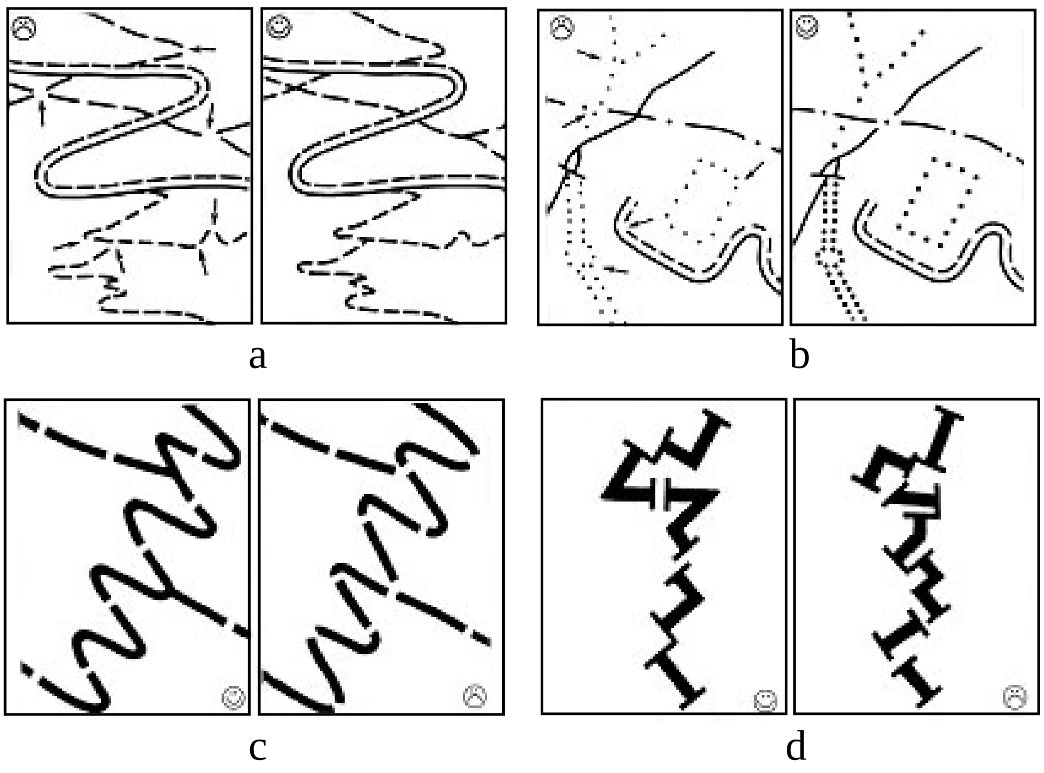


Fig. 1.1 Examples of appropriate vs. inappropriate cartographic line symbols © [Spiess, 1995a]

Cartographers have made a lot of efforts to study these aesthetic treatments of the graphic symbols and find the optimal way for display. For a long time, such efforts involved artistic intuition and hard works. Since the introduction of computer techniques into cartography, most of the drafting work can be performed using computer graphics programs.

1.1.2 Limitations of current graphics software for cartographic line symbolization

In modern cartography, the process of symbolization is mostly carried out by using two kinds of computer graphics programs: general graphics software and automated cartographic software. General graphics software includes vector graphics products like Macromedia Freehand, Adobe Illustrator, and CorelDraw etc. They contain powerful functions to draw and edit digital images, and provide flexible platforms for cartographers to test appropriate strategies of symbolization. But these programs are mostly developed for general purposes to handle digital graphics; cartographic drawing principles and functions are only partially integrated. As a result, the preliminary results created by these tools may contain some inappropriate effects (examples shown in Fig. 1.1), although they might not be problematic for other, non-cartographic applications. Such unaesthetic results must therefore go through careful interactive modifications before publication, which is often a time consuming work.

GIS and other automated map-producing software also provide some functions to manipulate graphics. Once the geo-spatial data is processed and sent to the rendering engine, map symbols could be created automatically in a standardized way. But compared with the specialized graphics software mentioned above, the graphic engines in these programs have relatively weaker functions to generate high quality graphic output.

Another shortcoming of the present graphics tools is that they can only provide a limited number of line patterns. Fig. 1.2 shows some dashed lines and dash-dotted lines provided by Macromedia FreeHand 10. The dash length and gap length are fixed. We can only choose line styles of this kind out of the collection provided by the program. Although double lines and triple lines are frequently used as map symbols, until now there is no efficient method available to draw these kinds of lines. At present, these symbols are mostly drawn indirectly, which involves sophisticated modifications (which will be discussed in chapter 4). New drawing methods are therefore needed to improve this situation.

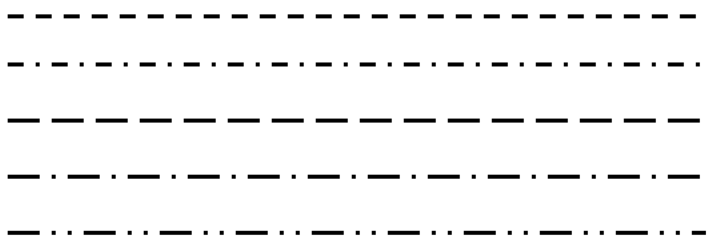


Fig. 1.2 A reflection of dashed and dash-dotted lines provided by Macromedia FreeHand 10

1.1.3 Developing new drawing methods for cartographic linear symbols

To meet the increasing demand of interactive map products, computer programs for automated cartographic symbolization with high graphic qualities are highly desired. With an intention to contribute to this effort, this thesis aims to improve line generalization regarding two aspects. The first is the elaboration of line patterns to meet cartographic requirements. The second is to enrich linear symbols, covering both pattern styles and visual quality. Ideally, a good program should be able to create different line patterns as possible, and as efficiently as possible. Cartographic principles of line symbolization, especially for non-solid lines, should be integrated into the drawing engines in order to obtain results without, or with less further cartographic modifications necessary. A friendly, flexible, interactive environment is desired for the customer-oriented designing process.

1.2 Methodology and outcomes

1.2.1 New line drawing algorithms

In traditional drawing methods, non-solid lines are drawn as a sequence of consecutive drawing units; each unit contains both gaps and solid. Once a line pattern is specified (dash length, gap length, dot size, etc.), the line is drawn between two end points in a continuous way. Therefore, it is possible that gaps may occur at corners or intersections (Fig. 1.3).

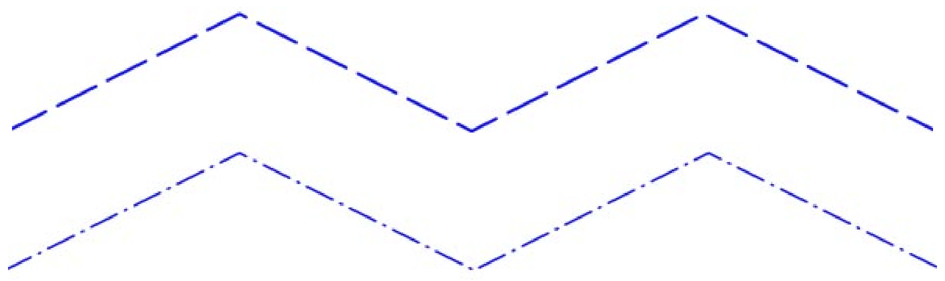


Fig. 1.3 A dashed line and a dash-dotted line drawn using the traditional method

The new method proposed in this dissertation treated each pair of adjacent vertices as two end points of a line segment. For dashed lines, the dash length will be adjusted automatically so that each line segment contains only integer number of dashes and gaps. It starts and ends with two half dashes. A complete dashed line element is composed of a series of connecting shorter dashed lines. For dotted lines and dash-dotted lines, the principles are essentially the same as for dashed lines, except at start and end points, where dots will be placed. The new drawing algorithms will produce only solid parts with roughly balances arms at corners or crosses (Fig. 1.4).

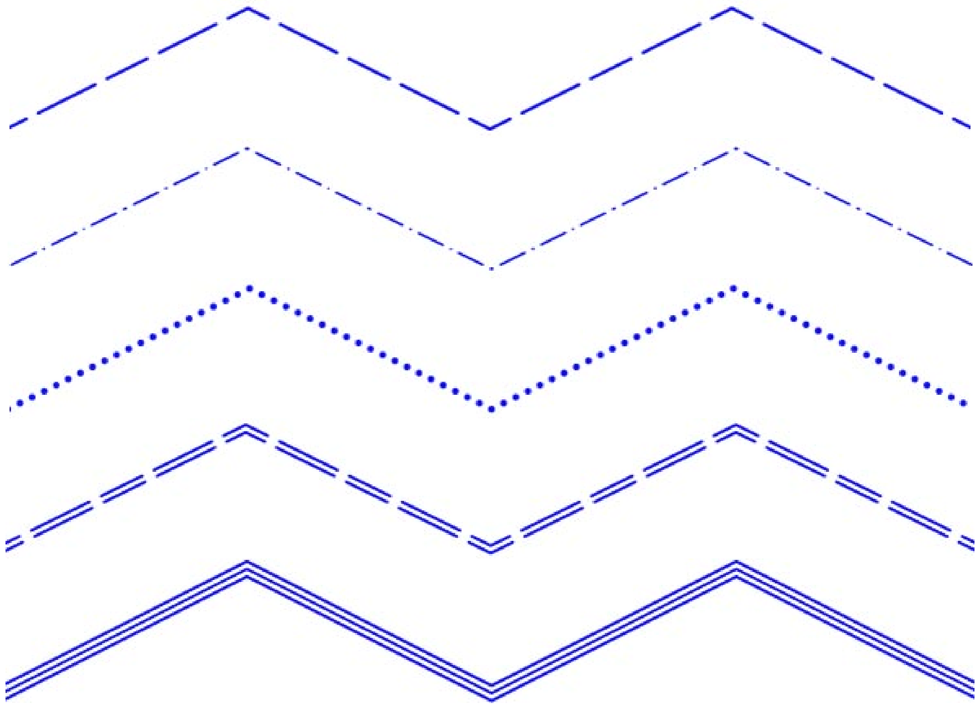


Fig. 1.4 Lines created using the new methods developed in this dissertation

Efficient rendering algorithms were developed to create double lines and triple lines. The methods follows the principle is that two or three single lines are drawn in parallel using the given geometric vertices.

1.2.2 Application of texture mapping in line symbolization

To enhance the visual effects and construct new line patterns, the computer graphic technique of texture mapping is introduced into cartographic symbolization for the first time. Its main aim is to add realism to the objects' surfaces during digital image synthesis. A realistic view normally requires fine surface information like textures, subtle color gradations, shadows, reflections etc. Instead of performing complex modeling and a large amount of computations to describe this detailed information, in texture mapping an already existed image, which could originate from the real world (a photography) is “pasted”, onto the target object surface. The target object is outlined as an empty polygon, which is “filled” by the texture image (Fig. 1.5). Depending on the properties of texture sources, this technique has been successfully used to enrich the image performance defined by parameters like surface color, specularity, reflection, normals, transparency, etc.

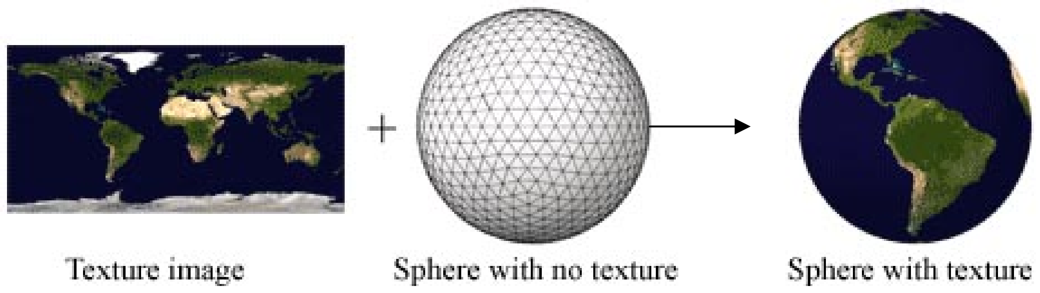


Fig. 1.5 The principle of texture mapping (Modified from http://en.wikipedia.org/wiki/Texture_mapping)

Cartographers are familiar with traditional graphic primitives, like points, lines and areas. Actually the texture image may also be treated as a kind of graphic primitive. This idea was proposed more than one decade before [Haeberli and Segal, 1993], but it has not been studied thoroughly since then.

In this work, the application of texture mapping was studied to create textured lines. A collection of texture images for linear symbols is constructed. The textured lines can be drawn in two patterns: solid line patterns and dotted line patterns. It serves as a new tool to add realism and to provide a potentially unlimited source of line patterns (Fig. 1.6).

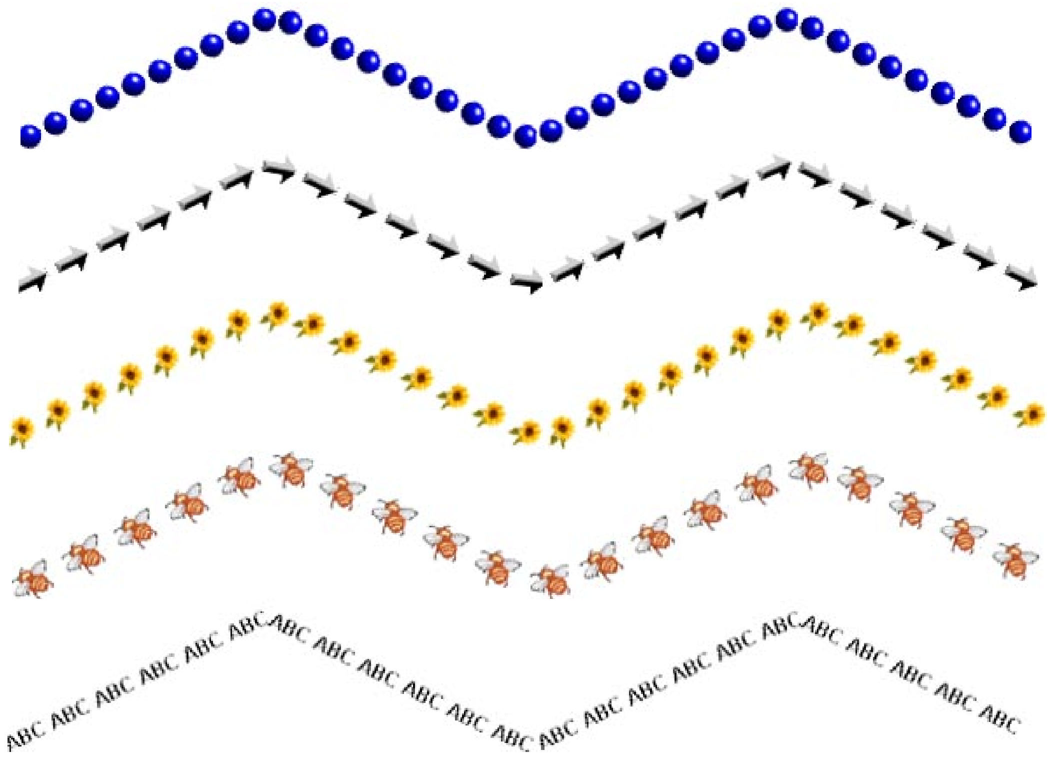

ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC ABC

Fig. 1.6 Examples of textured lines

1.2.3 An interactive working environment

Cartographic line symbolization might contain many abrupt turns if it has more than one line with irregular geometries. Different lines may frequently interact with each other or with other kinds of symbols. It is therefore challenging to develop an automated solution to cover all problem cases. In this work, an interactive controlling environment is developed allowing flexible adjustments of linear symbols.

In the new program, the parameters line width, dash length, gap length, line cap and line joint can controlled individually by user. For double lines and triple lines, we can also adjust the features of each parallel line individually. This flexible control functionality allows for delicate adjustments of linear symbols to balance the overall clarity and specificity. In addition, it also provides an interactive platform for users to design their favorite linear symbols. The objectives and approaches of this work are illustrated (Fig. 1.7).

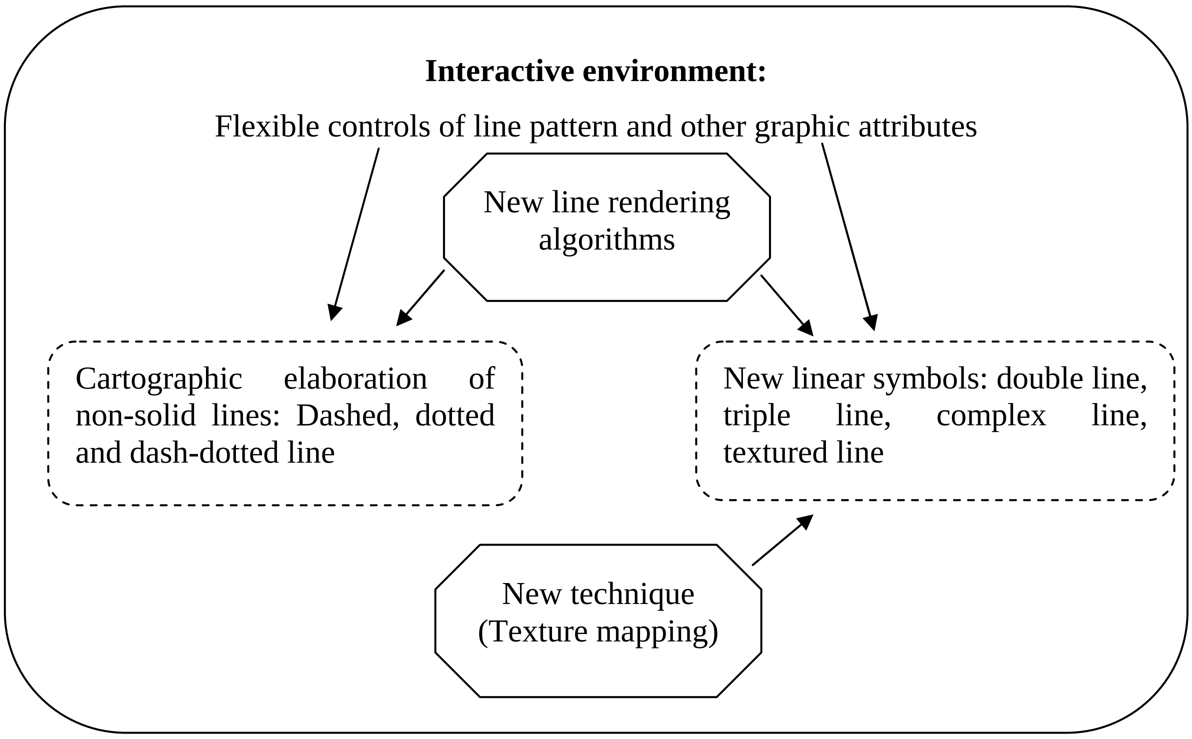

Interactive environment:
Flexible controls of line pattern and other graphic attributes
New line rendering algorithms
Cartographic elaboration of non-solid lines: Dashed, dotted and dash-dotted line
New linear symbols: double line, triple line, complex line, textured line
New technique (Texture mapping)

Fig. 1.7 The objectives and approaches of this work

Dashed frames: objectives; Solid frames: approaches.

1.3 Structure of this thesis

A description of the scientific background is made in Chapter 2, including both aspects from cartography and from computer graphics. Some inappropriate linear symbols that should be avoided are summarized in Chapter 3. In Chapter 4, the details of the new drawing methods are described. In Chapter 5, the applications of new the program are demonstrated, including the application of texture mapping in line symbolizations. A discussion and outlook is presented in Chapter 6.

Chapter 2 The basis of cartography and computer graphics for generating 2D linear map symbols

2.1 Cartographic representation of linear symbols

Cartography is a subject to study the strategies of scaling the real world onto a smaller, two-dimensional plane, i.e. the maps. Compared with aerial photos or satellite images, maps are used to express the real world in such a way that any selected geo-spatial phenomenon is represented by an abstract map symbol, which is usually made up of several basic graphic elements, including points, lines, areas (see Fig. 2.1) [MacEachren, 1995].

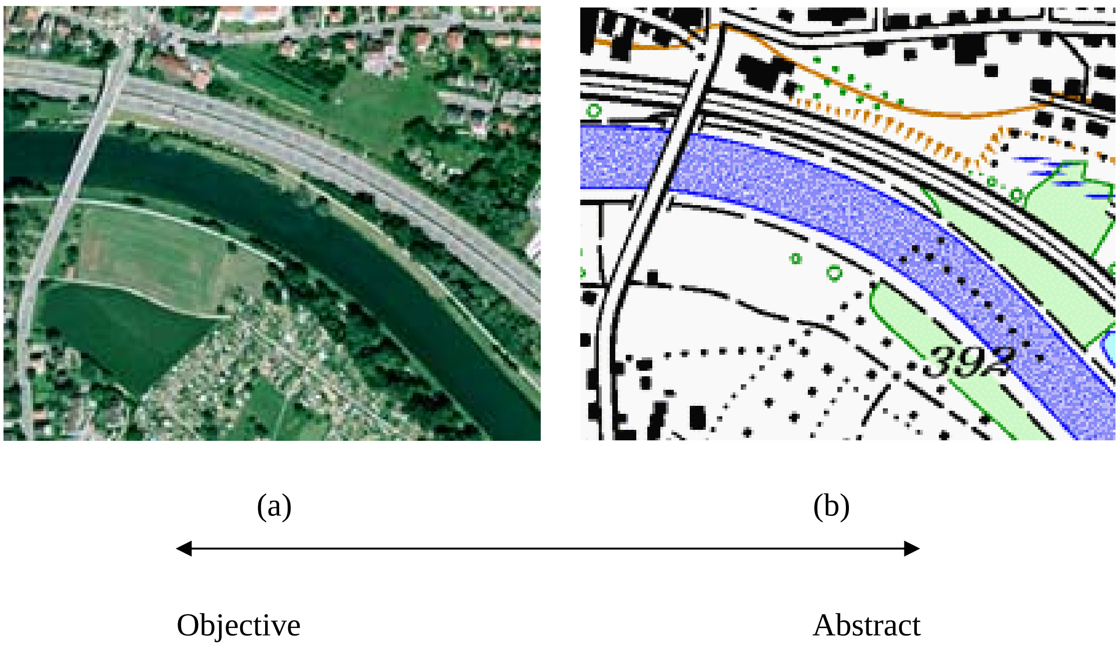


Fig. 2.1 Objective vs. abstract symbolization

(a) Swissimage © Swiss Federal Office of Topography, Wabern
(b) Pixelkarte PK25 © Swiss Federal Office of Topography, Wabern

An important characteristic of the cartographic representation is the fact that map producers can control the methods of symbolization. Mostly, map symbols are defined by assigning them with appropriate visual variables. Jacques Bertin presented his pioneering work on visual representations in his book, Semiology of Graphics. He classified the visual variables into several classes, including position, size, shape, value, color, orientation and texture [Bertin, 1983]. These factors can be applied individually or in combination to build all kinds of graphic element. Based on the properties of represented data, cartographers adopt different strategies to apply these visual variables. To design linear symbols, the following visual variables are frequently used.

1. Width (or Size)

Theoretically, a line has length but no width. To make it visible, it is provided with thickness (Fig. 2.2). The size of a line is represented by the line width. Adding different values of width is a simple way to differentiate linear map symbols. As a general rule, thick lines are normally associated with larger values or greater importance than thin lines.

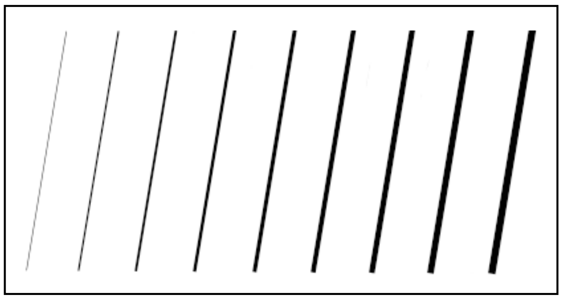


Fig. 2.2 Lines with different width

2. Color

Color is applied to show the qualitative differences of line symbols. Moreover, some geographic bodies are often labelled with their natural colors. For example, traditionally, we associate blue to water bodies and green to vegetation.

3. Style or shape

Lines can be roughly classified into two main styles, solid lines and non-solid lines. Non-solid lines contain constitutive gaps. A non-solid line can be drawn as dashed, dotted or dash-dotted shape (Fig. 2.3).

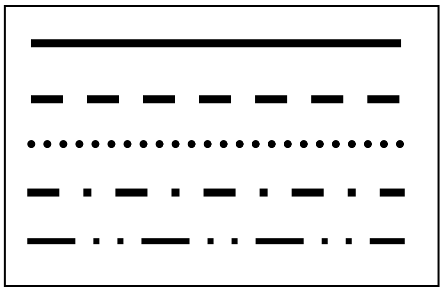


Fig. 2.3 Different line styles

4. Pattern or texture

This visual variable provides lines with sophisticated looks. The categories of possible line patterns are theoretically unlimited. Fig. 2.4 shows some lines with complex patterns that have been used as linear map symbols.

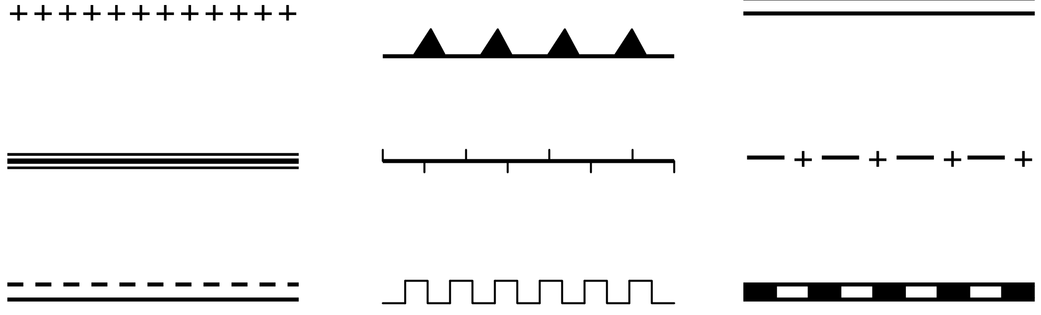


Fig. 2.4 Some line patterns

5. Orientation

This graphic variable can be used in two ways. One is to point out line directions with arrows or other markers. The other is to specify the polarity of textures (Fig. 2.5).

Through the appropriate combination of these strategies, a great variety of linear symbols with diversified effects can be produced. Fig. 2.6 shows examples of linear map symbols from four sources representing physical, dynamic and abstract subjects.

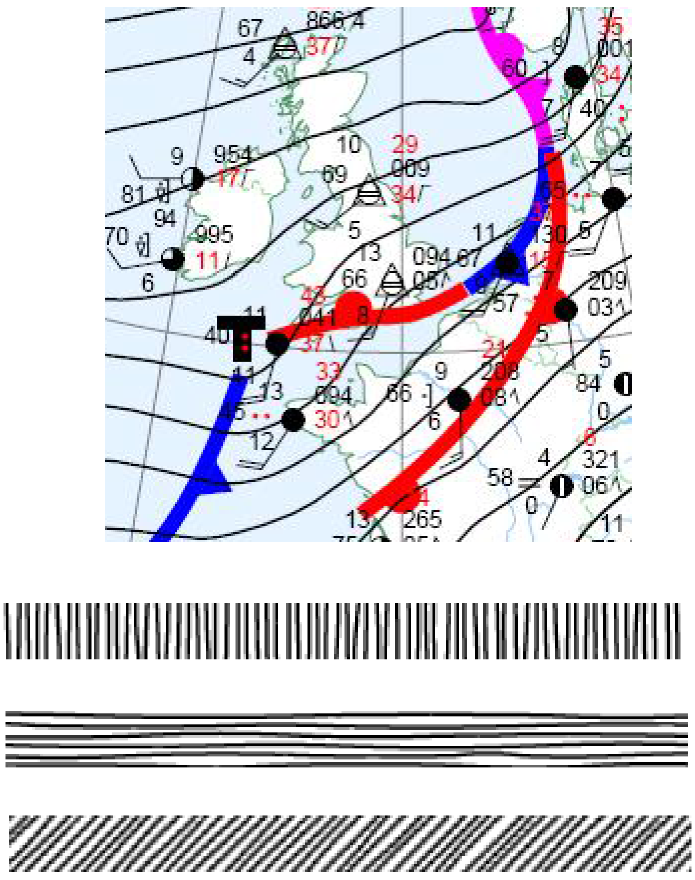


Fig. 2.5 Orientation

Upper figure: Wind direction, from ground weather chart on Dec. 30, 2006 © Federal Office of Meteorology and Climatology MeteoSwiss

Below: different orientations of textures

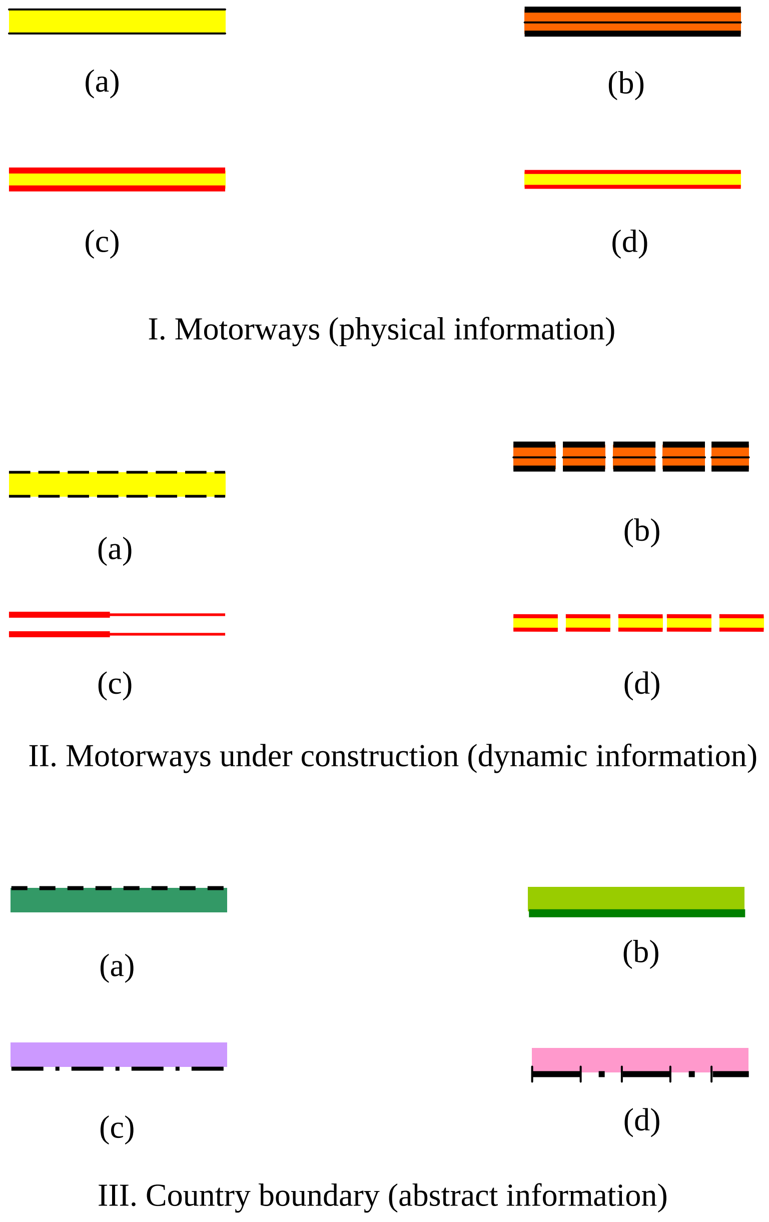

(a)
(b)
(c)
(d)
I. Motorways (physical information)
(a)
(b)
(c)
(d)
II. Motorways under construction (dynamic information)
(a)
(b)
(c)
(d)
III. Country boundary (abstract information)

Fig. 2.6 Comparison of different legends from four sources

(a) Landeskarte der Schweiz mit umliegenden Ländern, 1: 1 000 000

© Swiss Federal Office of Topography, Wabern

(b) Generalkarte der Schweiz, 1: 300 000 © Swiss Federal Office of Topography, Wabern

(c) Strassenkarte der Schweiz, 1: 350 000 © Orell Füssli, Kartographie AG

(d) Chinese atlas, 1: 4 000 000 © Xi'an map printer

2.2 Computer graphics background of drawing 2D lines

This section cover some computer graphics backgrounds of this work, including the basic theories, concepts and methods.

2.2.1 Vector and raster graphics

To display an image on an output device (monitor, printer and so on), the image data must be first converted into a format that can be recognized by the machine. There are two major types of image data, vector data and raster data. Vector data is stored as 2D coordinates that represent points, lines and polygons. Vector images can be easily scaled up or down, or moved around without any loss of image details.

Raster images are also called bitmap images, or digital images. They are made up of a huge number of square picture elements (pixels) aligned on a 2D raster grid. The principle of making a raster image is to specify all the constitutive pixels. Due to the manner of the alignment of the pixels, a raster image can only reflect the image edge in an approximate way, an undesired phenomenon of aliasing, or “jaggies” or “staircasing” exists (Fig. 2.7) [Foley, 1996].

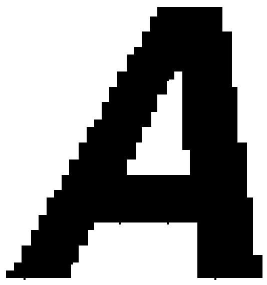


Fig. 2.7 Aliasing in a raster image

Geo-referenced data can be obtained and stored in both vector and raster format. The vector format is more suitable to describe discrete objects that have distinct outlines, such as roads, rivers and political boundaries. In contrast, raster data is not very well suited to represent linear features. It is better to describe objects with continuous attributes; including area symbols like elevation grids or land cover mosaics [Bernhardsen, 1996; Chrisman, 1997; deMers, 1997].

Since the dominant output devices for digital graphics are raster engines, there is an indispensable step to convert the vector data into raster format. In this work, the methodology of how to produce 2D raster lines for cartographic applications. First, the basic methods of the line rendering process are summarized.

Straight line segment

The line rendering process aims to construct a line based on its geometric description. It contains two steps, the first is to deliver a description of a line's geometry; the second is to convert the geometric data into raster representation.

As a graphic primitive, the geometry of a line can be precisely described by mathematical equations. There are several basic line styles that are widely used in map designs, including solid line, dashed line, dotted line and dash-dotted line. The geometric description varies for different line styles. Among them, a solid line is the most fundamental form. The drawing method of solid line is the basis for all other line styles. We start with a simple solid line to introduce the strategies of drawing a line in a 2D coordinate system.

Theoretically, in a Cartesian coordinate system, a straight solid line is described by the following formula:

$$y = m \cdot x + b \qquad [2\text{-}1]$$

where m is the slope and b is y-intercept. A unique linear path is specified given the values of m and b. If the line shown in Fig.2.8 is given by a starting point $V_1(x_1, y_1)$ and an ending point $V_2(x_2, y_2)$, the formula changes to:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \qquad [2\text{-}2]$$

One thing to note is the case when the line is parallel to the y-axis, the problem of a zero denominator should be avoided.

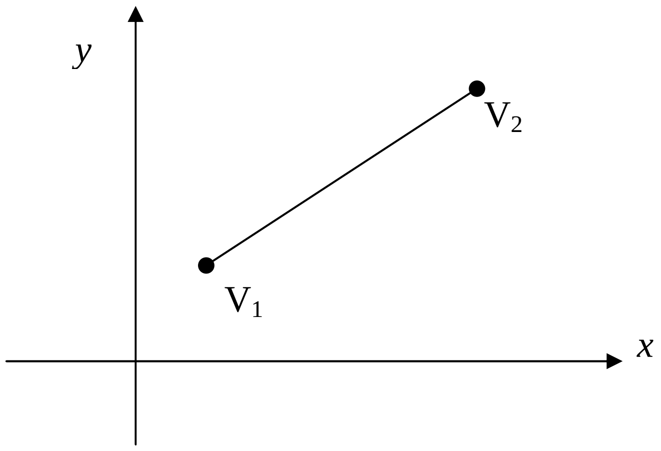
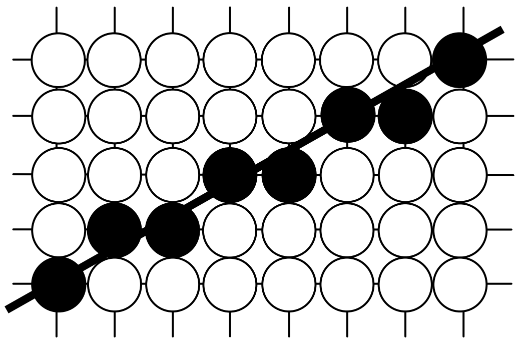
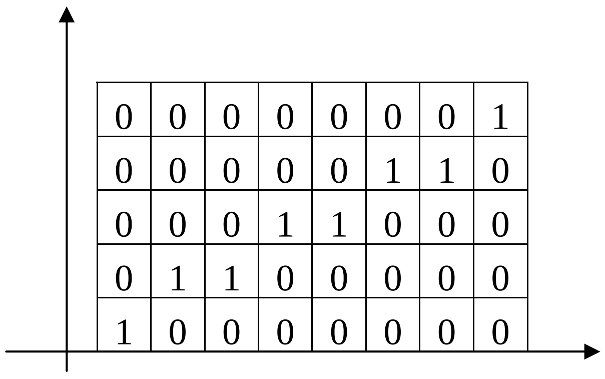


Fig. 2.8 Two end points determine a line



(a)



0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	0
0	1	1	0	0	0	0	0
1	0	0	0	0	0	0	0

(b)

Fig. 2.9 Magnified view of a raster line (a) and corresponding memory buffer (b).

This is the most frequently used mathematical description of a line in computer graphics. In theory, this line is continuous and infinitely thin. But on the raster screen, which is not a continuous space, the line can only be approximately displayed as a collection of discrete pixels along the ideal line. It is rather trivial to specify the "correct" pixels of vertical, horizontal and diagonal lines if they are aligned exactly on the pixel grid. But in other cases, we can only draw a line more or less close to but never exactly along the "true line" (Fig. 2.9a). The question is how to choose the pixels that are closest to the real line. There are some well-known line rendering algorithms that can be used to perform this task. In general, a good line rendering algorithm should produce raster lines satisfying the following criteria [Newman, 1979]:

- lines should be straight
- lines should start and terminate accurately
- lines should have constant width
- lines should be created efficiently

2.2.2 Basic line drawing algorithms

There are three basic line drawing algorithms, Digital Differential Analyzer (DDA), Bresenham's Algorithm and Midpoint Algorithm [Sprould,1979; Hearn, 1986; Newman, 1981; Krishnamurthy, 2002]. They are used to determine which pixel around the "real" location should be taken to best approximate a raster line. For all figures in section 2.2, we assume a pixel to be centered on the integer coordinates.

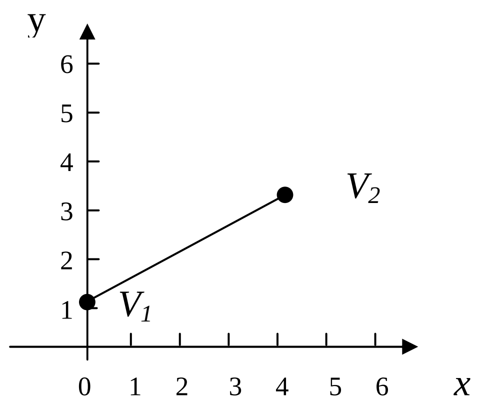


Fig. 2.10 A line with equation of $y = 0.5*x + 1.2$

DDA (Digital Differential Analyzer)

In Fig. 2.10, line V_1V_2 is specified by two end points: V_1 (0, 1.2) and V_2 (4.8, 3.6). Then the mathematical description of this line according to equation [2-2] is:

$$y = 0.5 * x + 1.2 \qquad [2\text{-}3]$$

From this quation, we can get the location of every point on the line between the two end points. Because of the constraints of the 2D raster grid, x and y must be integers. If we look at x_i, which is an integer, we get

$$y_i : y(x_i) = round(0.5 * x_i + 1.2) \qquad [2\text{-}4]$$

After rounding the result of the expression. For the next pixel, the value of x_{i+1} is increased by 1 and we get the next y_{i+1} accordingly (see Fig. 2.11).

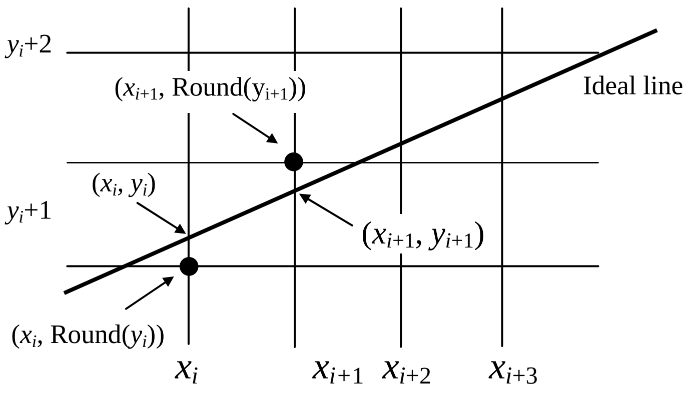


Fig. 2.11 The principle of DDA

Taking the line above as an example, then

for x = 2; we get : y = round (0.5 * 2 + 1.2) = round (2.2) = 2

x = 3,   y = 3

x = 4,   y = 3

….

A main drawback of DDA is the relatively slow speed. The round-off process requires the slow floating-point computation. In addition, the round-off error may accumulate. Two more efficient algorithms are Bresenham's Algorithm and Midpoint Algorithm.

Bresenham's algorithm and midpoint algorithm

Compared with DDA, Bresenham's algorithm and the Midpoint algorithm are two faster line algorithms. They are used to drawn any lines with only integer arithmetic. The principles of the two methods are essentially the same, which share one idea named after Jack E. Bresenham [Bresenham, 1985; Foley, 1996].

Bresenham's Algorithm introduces a parameter of "error value" as a ruler. The "error value" *d* are the vertical distances between the real position on the line (x_i , y_i) and two surrounding pixels: (x_n , y_n) and (x_n , y_n+1) (Fig. 2.12).

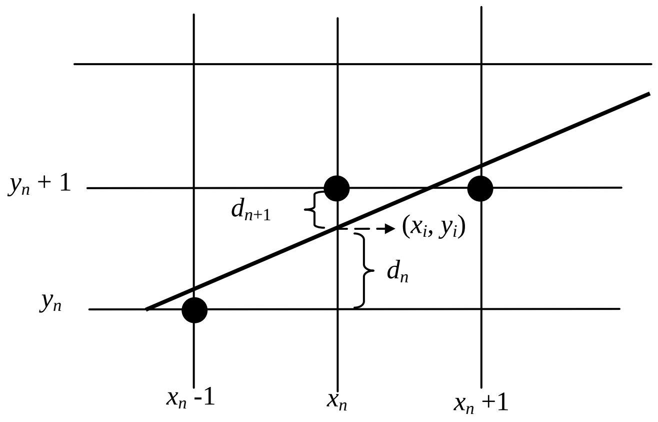


Fig. 2.12 The principle of Bresenham's Algorithm

Here:

$$\begin{cases} d_n = |y_n - y_i| \\ d_{n+1} = |y_n + 1 - y_i| \end{cases} \qquad [2\text{-}5]$$

In case: $d_n < d_{n+1}$; we choose the pixel (x_n, y_n);

$d_n > d_{n+1}$; we choose the pixel (x_n, y_n +1)

Midpoint Algorithm is very similar with Bresenham's Algorithm to draw lines, except the Midpoint Algorithm is better to draw arbitrary conics [Foley, 1996]. Given a point (x_i, y_i) on the line, there are two pixels (x_n, y_n) and (x_n, y_n +1) that can be chosen. The midpoint (x_n, y_m) is in the middle between them,

where $y_m = \frac{y_n + y_{n+1}}{2}$. [2-6]

If $y_i - y_m$ <0, then choose the pixel of (x_n, y_n), otherwise, if $y_i - y_m$ >0, choose (x_n, y_n +1) (see Fig. 2.13).

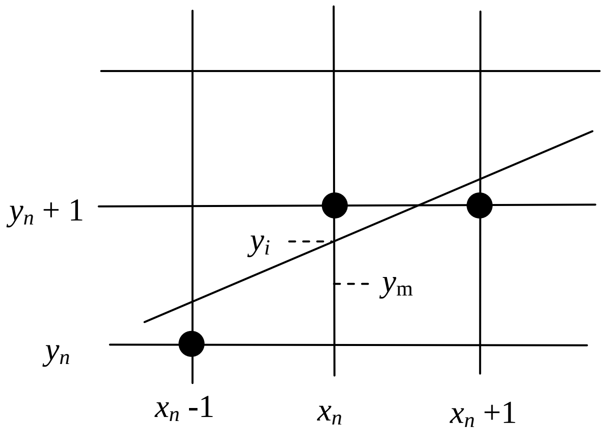


Fig. 2.13 The principle of Midpoint algorithm

In the above introduction, the principles of the three commonly used algorithms are explained based on the assumption that: $x_0 < x_1$, $y_0 < y_1$, $0<m<1$, $b>0$. For all other cases, there will be small differences of the functions. In this work, the Bresenham's Algorithm to render lines is adopted. It satisfies very well the criteria of drawing lines that have been mentioned in section 2.2.1.

The methods introduced above only determine which pixels to use along the ideal line path. To get good visual results, we also need additional treatments. For linear symbols, we can modify the appearance by adding caps at line ends, or joints at line turns.

2.2.3 Caps and joints

Due to the manner of pixel alignment, a line might have either horizontal end or vertical end (Fig. 2.14) [Foley, 1996].

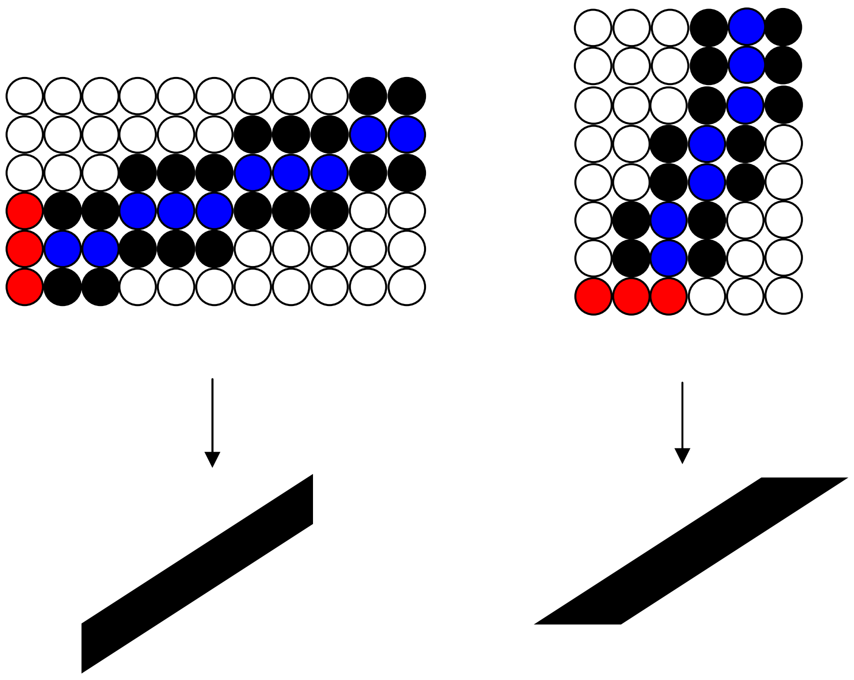


Fig. 2.14 Horizontal and vertical ends

To compensate such effects, a "cap" can be added to the line ends to obtain a uniform appearance. There are three frequently used caps, flat cap, round cap and square cap. A flat cap is obtained by adding an edge that is perpendicular to the line geometry and with the length the same as the line width (Fig. 2.15). With flat caps, the line length has no change. Compared with flat cap, an additional half square or circle is added to get a square cap or round cap (Fig.

2.16). There is a slight change of line length by adding square caps or round caps.

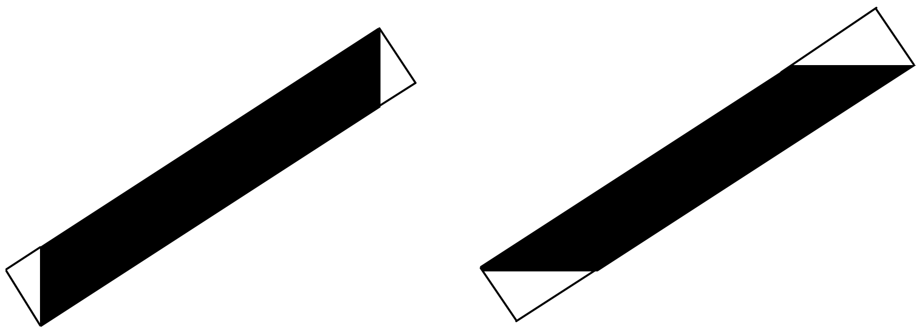


Fig. 2.15 Flat caps

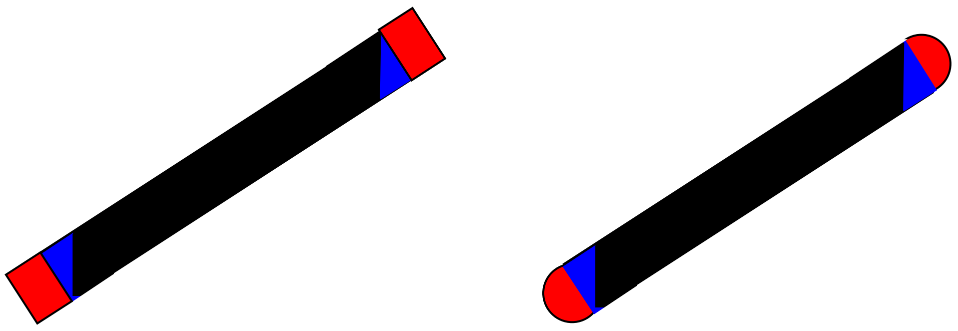


Fig. 2.16 Square caps and round caps

When two thick line segments in rectangular shapes meet to form a turn, they will leave a gap and the line appears broken (Fig. 2.17). To make up smooth turns, a special feature of joint can be added. Three kinds of joints are frequently used, including miter joint, bevel joint and round joint, as shown in Fig. 2.18.

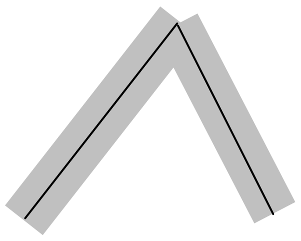


Fig. 2.17 A gap exists at the place where two line segments meet

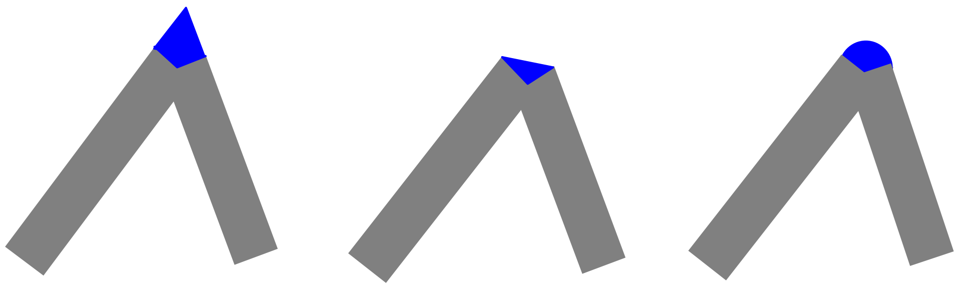
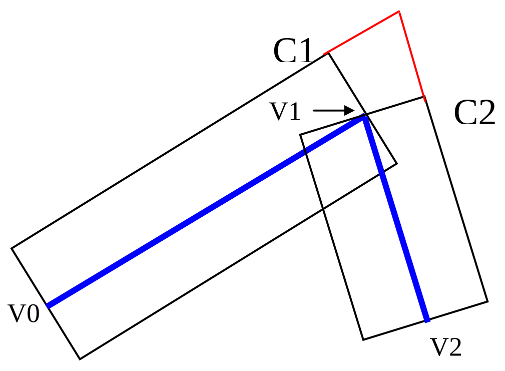
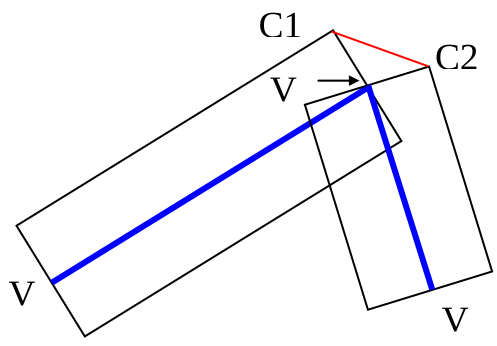


Fig. 2.18 Three kinds of line joints: miter, bevel, and round joint

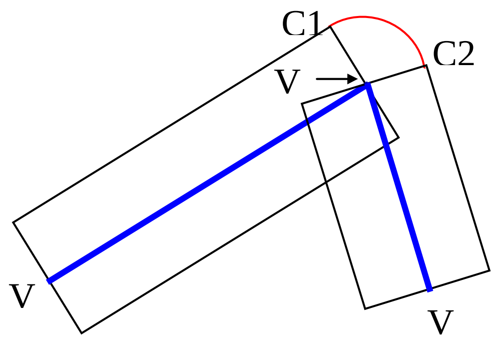
As shown in Fig. 2.19, the line joint is outlined by connecting the two points, C1 and C2. With different kind of connections, we can get miter joint (a), bevel joint (b) and round joint (c).



a. Miter joint



b. Bevel joint



c. Round joint

Fig. 2.19 The drawing of three kinds of line joints: miter, bevel, and round joint

2.2.4 Dashed line

The style of a dashed line is specified by two parameters, dash array and dash phase. The dash array defines the dash length and the gap length. The dash phase specifies the start point within a dash array.

We can look a dashed lines as a series of discrete (interspaced) solid lines (dashes): $\overline{V_0P_0}$, $\overline{P_1P_2}$ …, in Fig. 2.20. The positions of $V_0(x_0, y_0)$ and $V_1(x_1, y_1)$ are two end points of a straight line. To draw a dash line between V_0 and V_1, we need to calculate the positions at all dash ends $\{P_i(x_i, y_i) \mid x_0 < x_i < x_1; y_0 < y_i < y_1; i = 0,1,2...\}$, which can be described as:

$$\begin{cases} x_i = x_1 - (x_1 - x_0) * \overline{P_iV_1} / \overline{V_0V_1} \\ y_i = y_1 - (y_1 - y_0) * \overline{P_iV_1} / \overline{V_0V_1} \end{cases} \qquad [2\text{-}7]$$

Where

$\overline{P_iV_1}$ is the distance from P_i to V_1, and $\overline{V_0V_1}$ is the distance from V_0 to V_1.

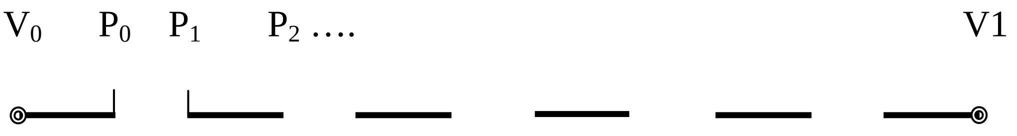


Fig. 2.20 Diagram of a dashed line

Then we get,

$$\overline{P_0V_1} = \overline{V_0V_1} - \overline{V_0P_0} = \overline{V_0V_1} - 1 * dash\_length \qquad [2\text{-}8]$$

$$\overline{P_1V_1} = \overline{V_0V_1} - \overline{V_0P_1} = \overline{V_0V_1} - 1 * dash\_length - 1 * gap\_length \qquad [2\text{-}9]$$

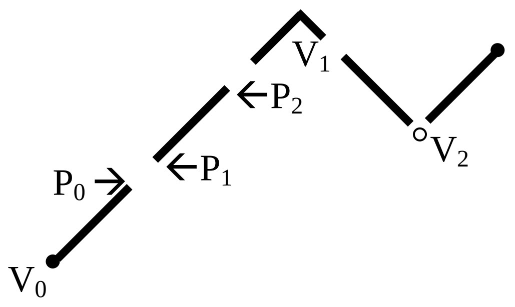


Fig. 2.21 A dashed polyline is drawn continuously

Many current available tools create dashed polylines in a continuous way. If the turn falls within a dash, the remains will be drawn at the beginning of the next section; otherwise, the turn is occupied by a gap (Fig. 2.21).

Another parameter of dash phase can also be adjusted. In Fig. 2.21, the dash phase is set as 0, which means it starts with a full dash. It can also be set to other values, and then the above equations need to be modified.

Suppose *S* represents the dash phase and it should be within the range between 0 to the sum of a dash length and a gap length logically.

If $0 < S \le dash\_length$, then:

$$\overline{P_0V_1} = \overline{V_0V_1} - (\overline{V_0P_0} - S) = \overline{V_0V_1} - (dash\_length - S) \qquad [2\text{-}10]$$

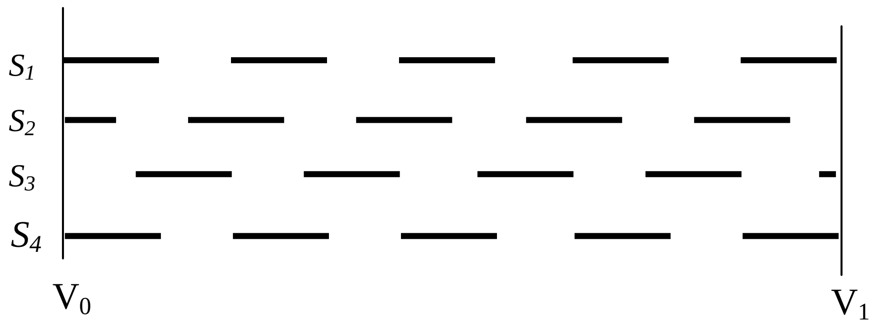


Fig. 2.22 Lines with different dash phase :S_1 = 0; S_2 = 0.5*dash_length; S_3 = dash_length; S_4 = 2*(dash_length + gap_length)

Otherwise if $dash\_length < S < (dash\_length + gap\_length)$, the dash will start from $\overline{P_1P_2}$ instead of $\overline{V_0P_0}$, we get:

$$\overline{P_1V_1} = \overline{V_0V_1} - (\overline{V_0P_1} - S) = \overline{V_0V_1} - (dash\_length + gap\_length - S) \qquad [2\text{-}11]$$

When the dash phase *S* is bigger than the sum of one dash length and one gap length, it will be decreased by the sum until it falls into the range from 0 to the sum of a dash length and a gap length. For example, in Fig. 2.22, S_4 has the same effect as S_1 after the decrease. For many dash line drawing methods, the default setting of dash phase is at 0, which ensures a dash line starts with a full dash.

2.2.5 Dotted line

A Dotted line is specified by two parameters, line width and gap length. The line width is determined by the dot size, which is normally drawn as filled circle or square. The gap length is the distance between two adjacent dots. The principle of drawing a dotted line is to specify the position of each dot.

The equation [2-7] is also used to draw a dotted line between $V_0(x_0, y_0)$ and $V_1(x_1, y_1)$ (Fig. 2.23),

Here we get

$$\overline{P_iV_1} = \overline{V_0V_1} - \overline{V_0P_i} = \overline{V_0V_1} - i * gap\_length \qquad [2\text{-}12]$$

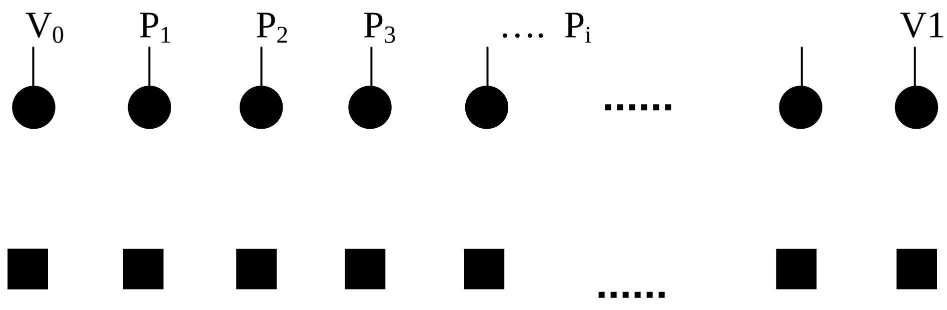


Fig. 2.23 Dotted lines

For a dotted polyline, if the corner falls in a gap, the remaining part will be the start of next segment (Fig. 2.24).

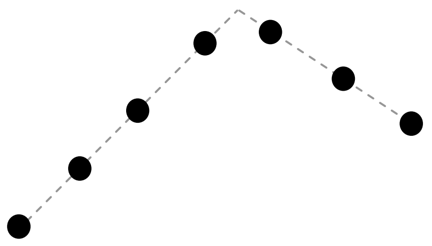


Fig. 2.24 A dotted polyline is drawn continuously.

2.2.6 Dash-dotted line

A dash-dotted line can be regarded as a special case of a dashed line. If every other dash is drawn as a square instead of a rectangle, it will create a dash-dotted style (Fig. 2.25).

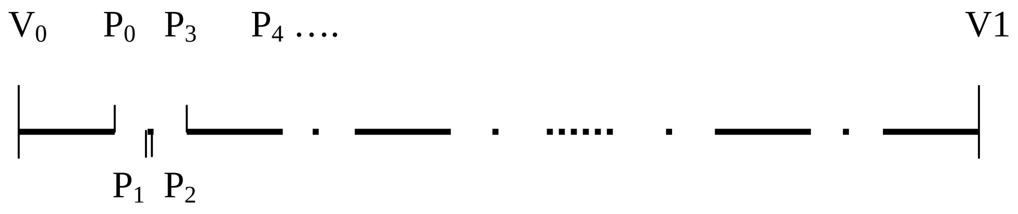


Fig. 2.25 A dash-dotted line

A dash-dotted line is specified by four parameters: dash length, gap length, square edge length and dash phase. To draw a dash-dotted line between the vertices $V_0(x_0, y_0)$ and $V_1(x_1, y_1)$, every dash end is calculated by the equation [2-7] and if the value of $\overline{V_0P_0}$ is the dash length, $\overline{P_1P_2}$ is the gap length, dot size is the edge length of the small square, we get:

$$\overline{P_0V_1} = \overline{V_0V_1} - \overline{V_0P_0} = \overline{V_0V_1} - 1 * dash\_length \qquad [2\text{-}13]$$

$$\overline{P_1V_1} = \overline{V_0V_1} - \overline{V_0P_1} = \overline{V_0V_1} - 1 * dash\_length - 1 * gap\_length \qquad [2\text{-}14]$$

$$\overline{P_2V_1} = \overline{V_0V_1} - \overline{V_0P_2} = \overline{V_0V_1} - 1 * dash\_length - 1 * gap\_length - 1 * dot\_size \qquad [2\text{-}15]$$

2.2.7 Anti-aliasing with lines and curves

Anti-aliasing is a software technique that helps to reduce the poor visual effect of aliasing without changing the output device [Chryssafis, 1986; Foley, 1996]. The principle of anti-aliasing is to add some pixels around the image edges. These extra pixels have intermediate colors between the background and the signal. Fig. 2.26 illustrates the principle of this technique. The left circle is drawn without anti-aliasing. A magnified view below shows the jagged edge. The right circle is a result using anti-aliasing. The border looks smoother than that of the left image. In the magnified view, it is clear that some extra pixels with intermediate values are added.

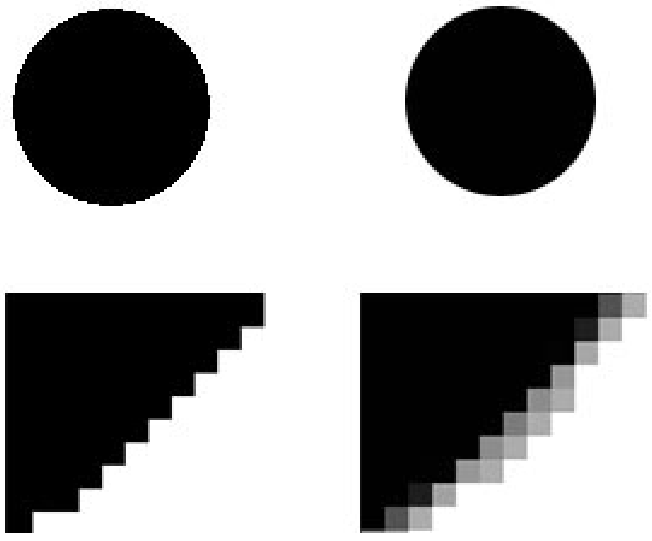


Fig. 2.26 The principle of anti-aliasing

In this work, the prefiltering method is adopted to create anti-aliasing lines and curves. This method treats each pixel as a small area. The color of each pixel is computed based on how much the overlap between the pixel area (rectangle) and the image area is [Grimsdale, 1991]. With the anti-aliasing treatment, the edges of raster lines and curves have smooth appearances at the cost of being slightly blurred.

2.2.8 Bézier curve

Besides the polyline introduced above, we often need smooth curves in cartographic symbolization. Although we can use polylines to approximate curves, this method is impractical since a large numbers of endpoints are needed to specify a curve. Like straight lines, curves can also be described by mathematical equations, which require much less control vertices. One efficient formulation is the Bézier curve, named after a French engineer, Pierre Bézier (Fig. 2.27) [Bézier, 1974; Krishnamurthy, 2002].

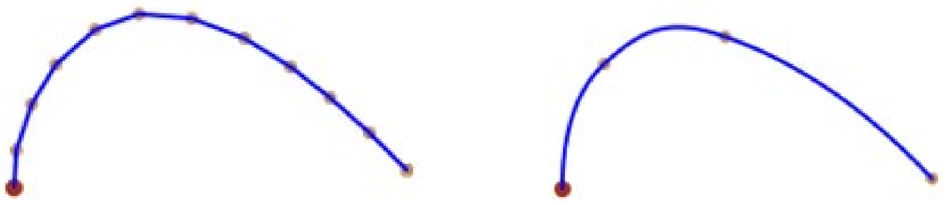


Fig. 2.27 Comparison of two curves made up of a polyline (left) and cubic Bézier curve (right). The dots are depicting the vertices.

A popular form of Bezier curve used in computer graphics is the cubic Bezier curve, which can be described in a parametric form:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3,\ for 0 \le t \le 1 \qquad [2\text{-}16]$$

where the polynomials are known as Bernstein basis polynomials of degree 3.

$$b_{i,3} = \binom{3}{i} t^i (1-t)^{3-i}, i = 0,...,3 \qquad [2\text{-}17]$$

To generate a cubic Bézier curve between two endpoints P_0 and P_3, we need to fix two additional control points (P1 and P2 in Fig. 2.28). Then a simple geometric approach, the de Casteljau algorithm, can be applied. The de Casteljau algorithm describes the curve as a recursive series of linear interpolations. It is fast to subdivide a Bézier curve into two curve segments at an arbitrary parametric location. Using this method, three intermediate points are established (Q_0, Q_1 and Q_2). The positions of the three points depend on the coefficient *t*, which varies from 0 to 1.

Point $\mathbf{Q}_0$ varies from $\mathbf{P}_0$ to $\mathbf{P}_1$ and describes a linear Bézier curve.
Point $\mathbf{Q}_1$ varies from $\mathbf{P}_1$ to $\mathbf{P}_2$ and describes a linear Bézier curve.
Point $\mathbf{Q}_2$ varies from $\mathbf{P}_2$ to $\mathbf{P}_3$ and describes a linear Bézier curve.
Point $\mathbf{R}_0$ varies from $\mathbf{Q}_0$ to $\mathbf{Q}_1$ and describes a quadratic Bézier curve.
Point $\mathbf{R}_1$ varies from $\mathbf{Q}_1$ to $\mathbf{Q}_2$ and describes a quadratic Bézier curve.
Point $\mathbf{B}(t)$ varies from $\mathbf{R}_0$ to $\mathbf{R}_1$ and describes a cubic Bézier curve.

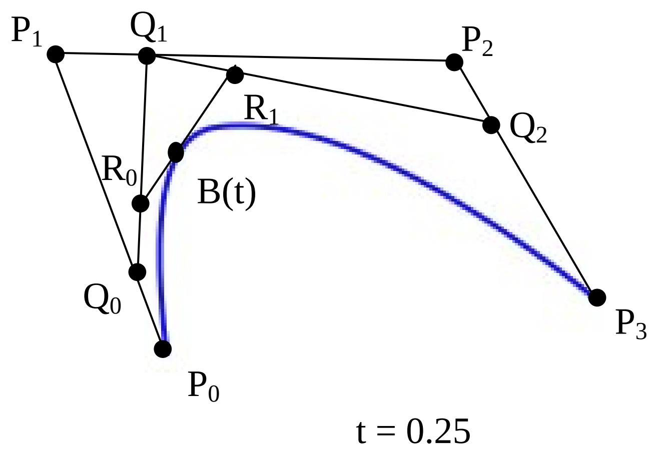


Fig. 2.28 Demonstration of the cubic Bézier curve drawing process

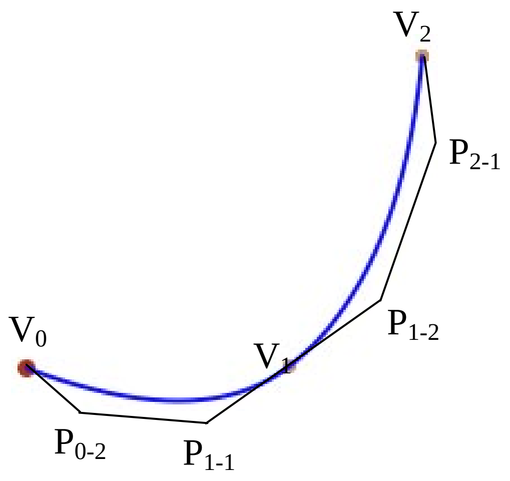


Fig. 2.29 Drawing a cubic Bézier curve

For a cubic Bézier curve as shown above, the position of any point on the curve is determined by two end points and two control points. In Fig. 2.29, V_0 and V_1 is a pair of end points; P_{0-2} and P_{1-1} are the two control points. Similarly, P_{1-2} and P_{2-1} are the control points of the next curve segment between V_1 and V_2.

2.2.9 Geometrical transformations

2.2.9.1 Geometrical 2D transformation

In computer graphics, a geometric transformation is an operation to map a position (x, y) to a new position (x', y'). The purpose of using geometric transformation is to move an image by changing the spatial locations of its vertices. The movement can happen between two different coordinates or on the same coordinate. During map construction, we perform this operation when the spatial data are transformed to a 2D map, and when we apply graphics functions such as zooming and panning.

Transformation is executed by simply computing the new positions for all original vertices of an object. There are three basic 2D transformation functions, including translation, scale, and rotation. When applying transformations, we are handling a group of points, which can be described in matrix form. Thus, the computation of transformation functions is usually done with matrices. Beside the computation of transformation, pixel interpolation is often needed when doing scaling or to get smooth image boundaries.

Translation:

This is a function to move a position along the *x* or/and *y* axis.

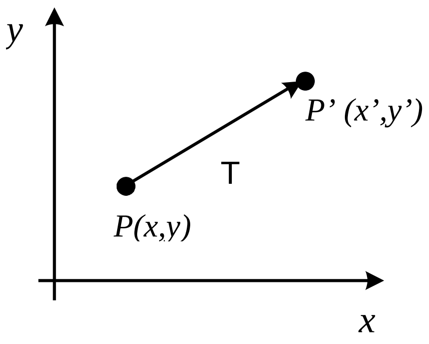


Fig. 2.30 Translation

Original equation:

$$\begin{cases} x' = x + k_x \\ y' = y + k_y \end{cases} \qquad [2\text{-}18]$$

Translations can be written in matrix form as follows:

$$P = \begin{bmatrix} x \\ y \end{bmatrix},\ \mathrm{P'} = \begin{bmatrix} x' \\ y' \end{bmatrix},\ \mathrm{T} = \begin{bmatrix} k_x \\ k_y \end{bmatrix} \qquad [2\text{-}19]$$

then, $P' = P + T$. [2-20]

We have the matrix description of the new position:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} k_x \\ k_y \end{bmatrix} = \begin{bmatrix} x + k_x \\ y + k_y \end{bmatrix} \qquad [2\text{-}21]$$

Scale:

Scale is a function that is used to change the size of an object. When doing a scaling transformation, the extra pixels have to be added by either pixel duplication or interpolation.

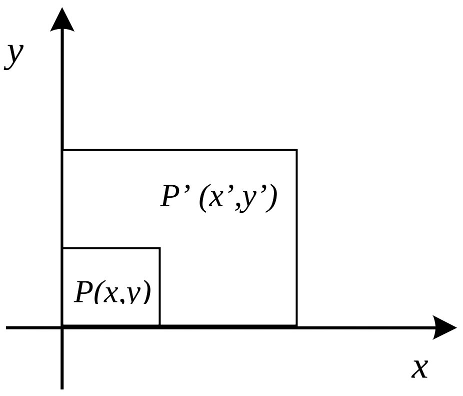


Fig. 2.31 Scale

Scale equations:

$$\begin{cases} x' = S_x * x \\ y' = S_y * y \end{cases} \quad [2\text{-}22]$$

in which s_x and s_y are the scaling factors.

The matrix representation is

$$P = \begin{bmatrix} x \\ y \end{bmatrix},\ \text{P'}=\begin{bmatrix} x' \\ y' \end{bmatrix},\ \text{S} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \quad [2\text{-}23]$$

Then $P' = S \bullet P$, [2-24]

We have the matrix description of the new position:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} \quad [2\text{-}25]$$

Rotation:

By applying a rotation function, the orientation of an object can be changed.

The rotation equations is:

$$\begin{cases} x' = \cos\theta * x - \sin\theta * y \\ y' = \sin\theta * x + \cos\theta * y \end{cases} \qquad [2\text{-}26]$$

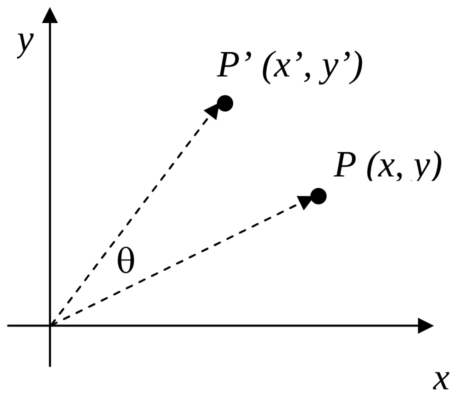


Fig. 2.32 Rotation

The matrix representation is:

$$P = \begin{bmatrix} x \\ y \end{bmatrix},\ P' = \begin{bmatrix} x' \\ y' \end{bmatrix},\ R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad [2\text{-}27]$$

Then $P' = R \bullet P$, [2-28]

we get matrix description of the new position:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} \qquad [2\text{-}29]$$

2.2.9.2 Composition of geometrical 2D transformation

In practice, moving an image often needs different transformations. Instead of applying one after the other, an efficient way is to express them by one matrix operation. This can be done by applying composition of 2D transformation, in which several transformation functions are multiplied together, resulting in one matrix. To do this, a homogeneous coordinate is introduced. This is done by adding a third non-zero coordinate, W, to the 2D coordinates (x, y). Thus for 2D transformations, we can define a point in homogeneous coordinates as

(*x/w*, *y/w*, 1).

The basic transformation functions can be written in 3X3 matrices.

Translation: $$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & k_x \\ 0 & 1 & k_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad [2\text{-}30]$$

Scale: $$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad [2\text{-}31]$$

Rotation: $$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad [2\text{-}32]$$

For example, to rotate a rectangle around a specific point, the composed transformation includes three steps (Fig. 2.33):

1. Translate the pivot point of the rectangle to origin,
2. Rotate the rectangle,
3. Translate it back to the original position

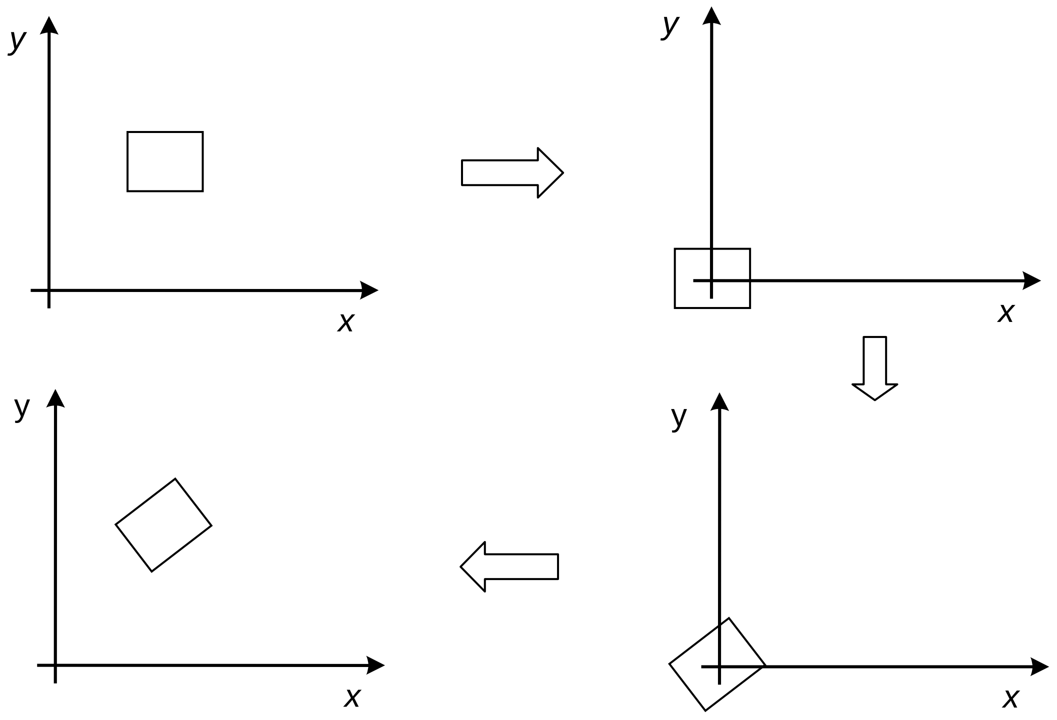


Fig. 2.33 An example of a composed transformation

Accordingly, we multiply three matrices to get the composite transformation:

$$\begin{bmatrix} 1 & 0 & S_x \\ 0 & 1 & S_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -S_x \\ 0 & 1 & -S_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

[2-33]

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x\cdot(1-\cos\theta)+y\cdot\sin\theta \\ \sin\theta & \cos\theta & y\cdot(1-\cos\theta)-x\cdot\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

2.2.10 Path-based drawing

We have introduced methods to draw polylines and curves. In many cases, we often need to handle more than one shape with complex geometries that are composed of some frequently used primitives, such as arcs, lines, curves and polygons. Such an image can be drawn efficiently by using the path-based drawing method [Apple, 2007; MSDN, 2006].

Paths are often used to define multi-part shapes or holes. One path may contain many subpaths. Each subpath can specify more than one different primitive, such as arcs, lines and curves. All the shapes defined in one path can be rendered in one step by calling the path function. For example, for a complex geometry with irregular outline, we can store the vertices in one path and render them in one step.

To define a path, we need to specify a current point and an end point. The path function will draw one subpath after the other till the end point. The two points can overlap (closed path) or do not (open path). Line can be drawn as an open path, in which the current point is always set as the start point (Fig. 2.34).

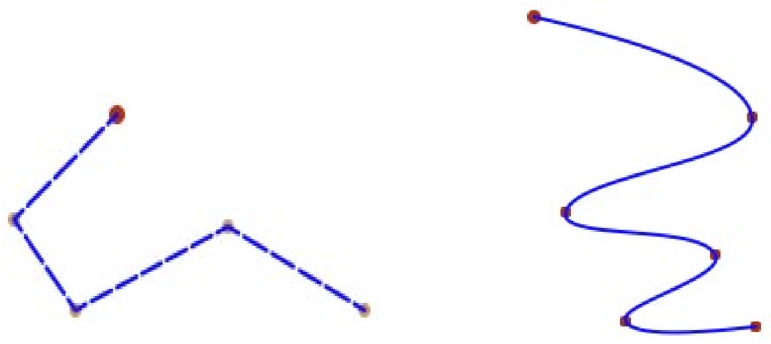


Fig. 2.34 Path-based drawings

2.2.11 Texture mapping

Lines generated using traditional drawing methods typically lack color variations, which lead to poor realistic details. In contrast, texture mapping is a technique to add realism by painting targets with color variations that can be obtained from the real world. The procedure of texture mapping can be divided into three parts: choosing a texture source; mapping it onto the object polygon; filtering the mapped texture image to remove aliasing.

In this work, a texture source is a 2D texture image that is stored as an array of color values (texels). It can be any image, either a photo or a painting obtained from real world; or images constructed by computer programs.

Mapping is the process of substituting the corresponding pixels of an object on the screen with texels of a texture imageThe object polygon on the screen and the texture image are based on different coordinates: the first defining the object surface is based on device coordinates (x, y), and the second is on a different texture coordinates (u, v) (Fig. 2.35). We need to set up the relations between a pixel and the corresponding texel. This is done by applying the 2D geometric transformations that project the texture space onto the object space. In this work, we used an inverse mapping method. That means the object space is first specified as a 2-D polygon on the screen. Then, for each pixel within this polygon, we compute the positions of the corresponding texels of the texture image and take the weighted color values to substitute the target pixel.

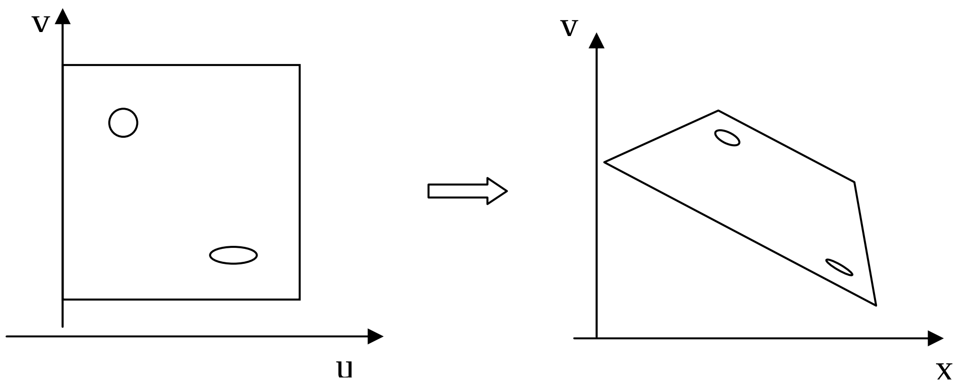


Fig. 2.35 The mapping process

Filtering is a function that is necessary to display the mapping results on the screen grid by maintaining a suitable appearance. During the mapping process, the original texture image will mostly be scaled and distorted. This may lead to unwanted effects, such as blurring (original resolution is too low) or aliasing (original resolution is too high). Filtering is applied to compensate this result and is a process similar with anti-aliasing. The mipmapping method is used in this work to save the filtering process. With this method, the resolution of the original texture image is reduced before filtering. A series of pre-calculated, anti-aliasinged texture images are first set up (Fig. 2.36). The one that best matches the target size will be taken for filtering.

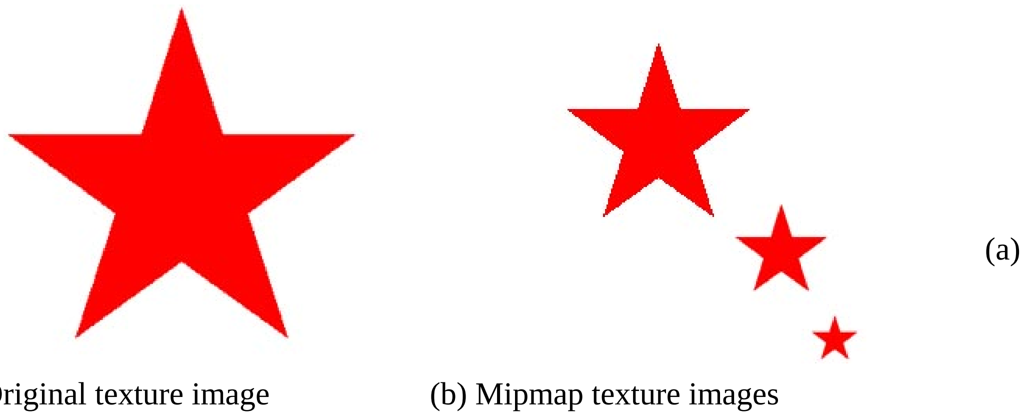


Fig. 2.36 Examples of mipmap images

2.2.12 Graphics libraries

A graphic library is a package of drawing functions to facilitate the drawing process. There are many kinds of graphic libraries that are suited for different purposes, like 2D or 3D image creations, animations, game development and so on. Computer operating systems have already included graphic libraries to support the drawing functions, like Graphics Device Interface (GDI) and GDI+ for Windows, Quickdraw and Quartz 2D for Macintosh, and X11 for Unix-based systems. These graphics libraries are closed systems and do not offer much flexibility for experimentation. Besides, they are platform-specific.

To generate complex graphics with higher quality, we need to use more specialised graphic libraries. In this work we use a graphic library called AGG (anti-grain Geometry) as a working platform. AGG is a free, open source 2D graphic library developed by Maxim Shemanarev (http://www.antigrain.com/). It is written in standard C++ and is compatible with any of the current main operating systems. AGG can render vector data into pixel images in memory with anti-aliasing and subpixel accuracy, which is very useful for map design. Other important features include the high flexibility and reliability. It is especially suited for line rendering due to functions like:

- Different types of line joints and line caps.
- Dashed line generator.
- Clipping rectangle and arbitrary shapes.
- Using images as line patterns.

Rendering in separate color channels.

Chapter 3 Cartographic considerations of linear symbols

3.1 Introduction

Compared with other graphical products, special considerations are necessary when creating cartographic line symbols. Good line symbolization not only requires that the line paths follow the geo-data to be represented as closely as possible, but also consider the optimal readability of the features. Getting a balance between these two factors remains a major concern of cartographic line symbolization.

Maps generally contain a wide range of different line symbols. Lines may also interact with each other as well as other symbols. To properly design lines on maps, we should pay attention specifically at line turns and intersections. Inappropriate representations at such places may disrupt linear symbols, which may confuse or even misinform the reader.

In this chapter, we have collected a number of unsuitable line symbols with problems concentrated at turns and intersections. As a comparison, optimal results are also presented. The inappropriate linear symbols shown in this chapter can be divided into two classes: turns within a single line and interactions between different lines.

3.2 Within a single line

Usually there are no problems with single solid lines. For other lines with gaps, we should specifically watch turns and resulting patterns with inappropriate representation.

3.2.1 Turns should be solid

Turns are important positions within a line path, especially turns showing sharp angles. Such information should be clearly presented. Since non-solid lines are composed of both solid parts and gaps, a general rule to design such lines is that gaps should not occur at turns.

In Fig.3.1a, the two dashed shapes show several sharp turns, which are obscured by gaps. Map users are generally interested in knowing the exact positions. A good symbolization preferably shows such locations with roughly equal dashes (Fig. 3.1b). Although in Fig. 3.1a, the three dotted outlines are indicating the same shapes, a more accurate representation of the same geo-data should accentuate the corner positions (Fig. 3.1b). If the crucial vertices at the corners are missing as in Fig. 3.1a, wrong information is provided.

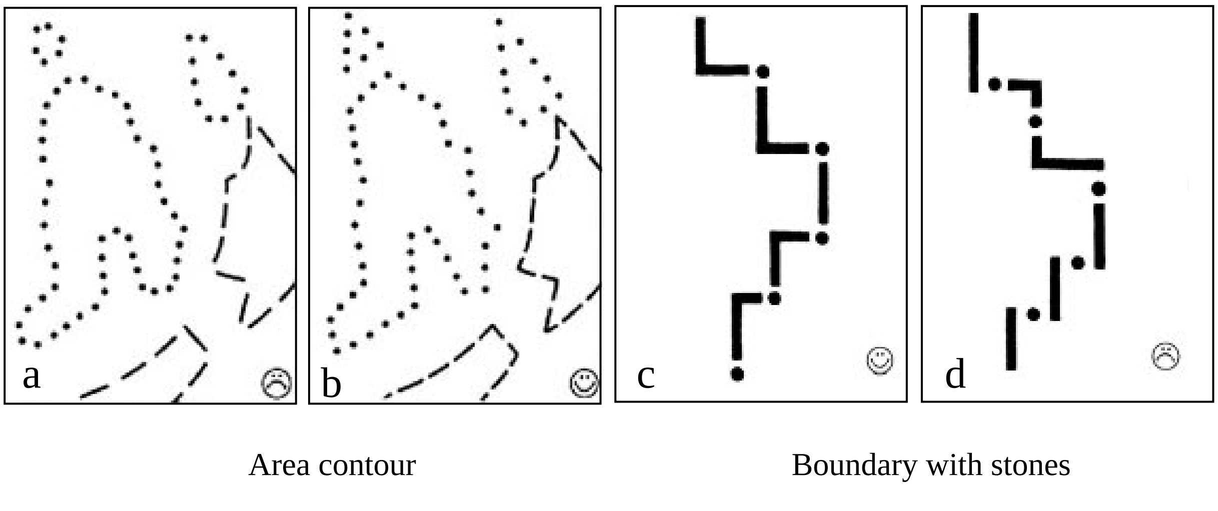


Fig. 3.1 Turns should be drawn with solids © [Spiess, 1995a]

This rule does not only apply to dotted lines and dashed lines, but also to dash-dotted lines, where turns should be drawn as either dashes or dots. For artistic purposes, the two arms at a turn should be of the same type (either dashes or gaps) and roughly the same length. In Fig. 3.1d, the turns are drawn as solids (dashes or dots), but all of them have unbalanced arms (one dash plus one gap). A better solution is shown in Fig. 3.1c, where turns drawn as dots are flanked by two gaps; turns drawn as dashes have two solid arms.

3.2.2 Line pattern should not be interrupted

Non-solid lines are made up of alternating solid parts and gaps. There are several factors that could interfere the display of line patterns.

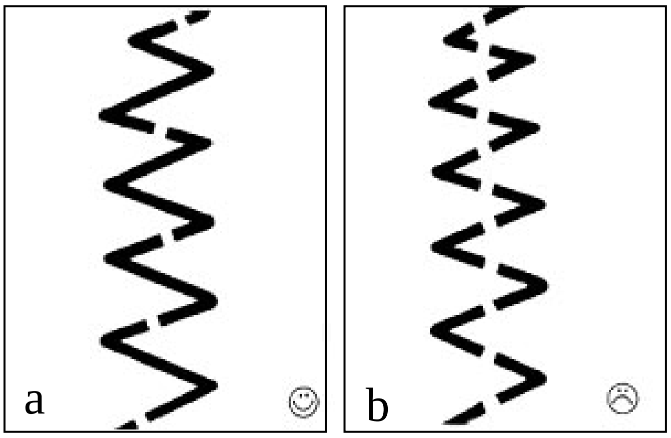


Fig. 3.2 Alignment of gaps © [Spiess, 1995a]

a. Alignment of gaps

Improper alignment of gaps may interrupt the flow of lines. In Fig. 3.2b, the dashed line satisfies the rule of solid turns with balanced arms, but the gaps are forming a regular pattern along a line that breaks the path into two halves. This representation can be improved by accidentally distributing or removing some of the gaps (Fig. 3.2a).

b. Density of gaps

It is mainly the solid parts that indicate the lines. Gaps should not occupy a significant portion of a line. Large gaps risk obscuring the line shape. In Fig. 3.3a, the geometry of the dotted line is blurred due to the large gaps. It failed to adequately present the geo-spatial information. A better solution is shown in Fig. 3.3b.

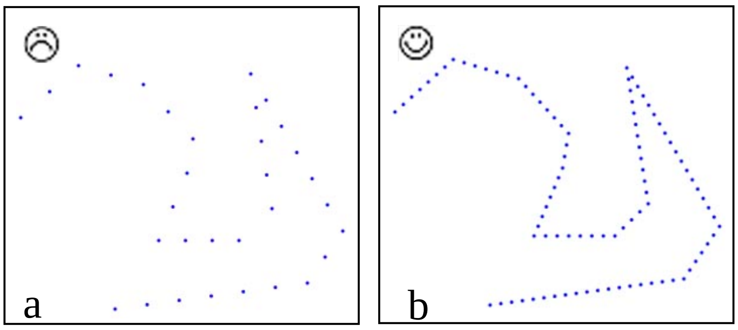


Fig. 3.3 Gap density is too high

c. Turns with densely packed vertices

The patterns of non-solid lines are easy to be interrupted at turns composed of densely packed vertices. The problem of Fig.3.4b is the paths show too many details, so that the dots are packed together; and the dotted line is twisted. The line path can be better demonstrated by slightly simplify the line geometry (Fig. 3.4a). This small modification will not lose any important geo-information due to the high density of vertices.

In Fig. 3.2d, the crowded vertices are on a sequence of abrupt turns; this kind of geo-information cannot be clearly displayed if the straight line segments are too short. This problem can be solved by using long dashes at these small regions. This treatment satisfies the cartographic requirement of accuracy and does not disturb the dashed line pattern (Fig. 3.4c).

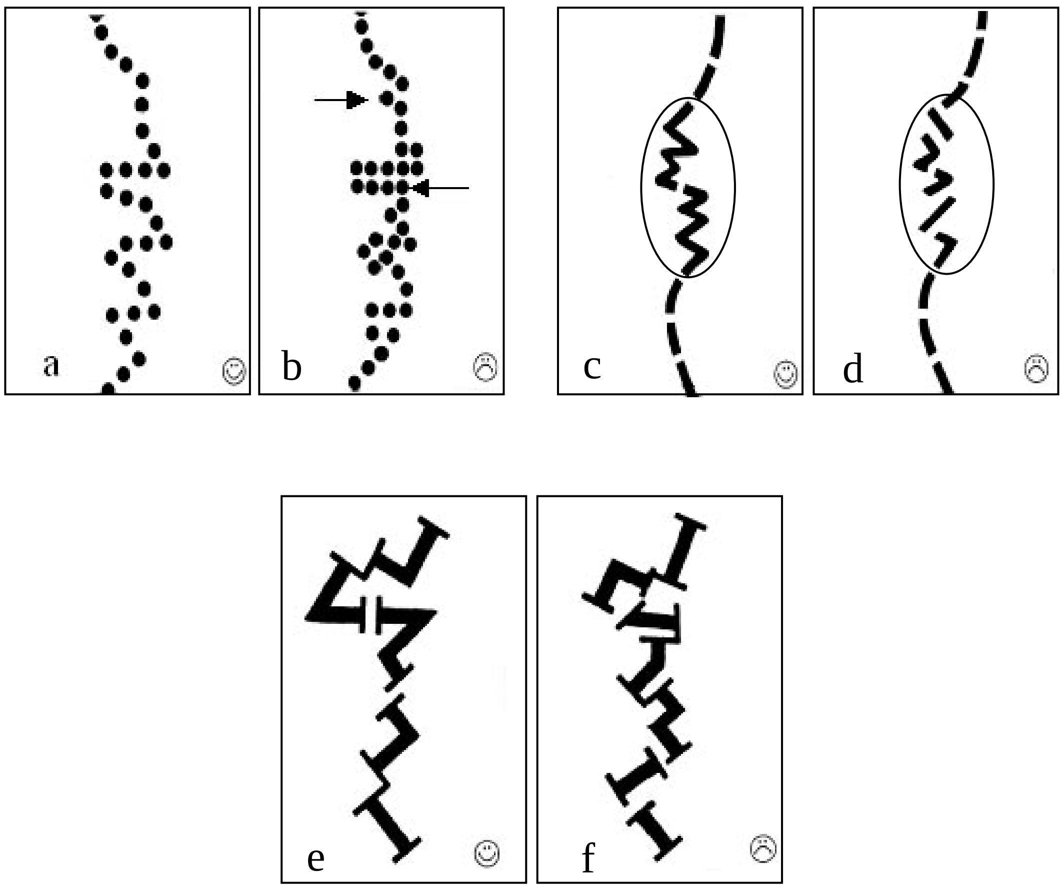


Fig. 3.4 Turns with crowded vertices © [Spiess, 1995a]

Some lines are made up of complex drawing units (e.g. Fig. 3.4 e and f). If these big components pack at the turns containing crowded vertices, it could make a mess and confuse the readers (Fig. 3.4f). We can adjust the gap length or the size of the drawing unit and also moderate the geometries slightly (Fig. 3.4e).

3.3 Intersections and junctions

Intersections between two different lines are important positions that need to be clearly shown. Considering the variety of line styles, there are many possibilities to graphically solve intersections. To reflect the real situation, we may also want to use specific features like bridges, tunnels and so on.

3.3.1 Between two single lines

When two simple solid lines meet, this will normally not cause problems. If one or both of them are non-solid lines, the intersection should fall on the solid part (of the dashed or dotted line). The situations and general rules can be summarized as follows:

a. Between two dashed lines: the dashes should be centered around the point of intersection (Fig. 3.5)

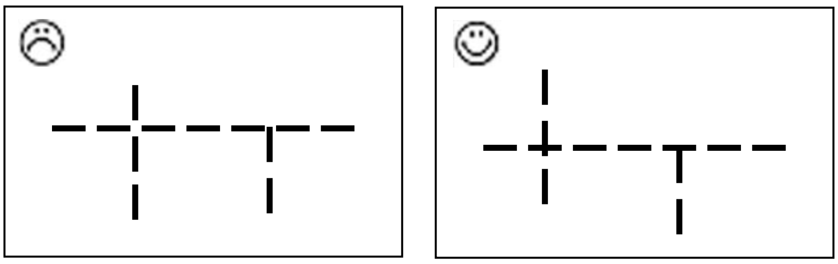


Fig. 3.5 Two dashed lines should meet with dashes

b. Between two dotted lines: the dot should lie on the intersection (Fig. 3.6).

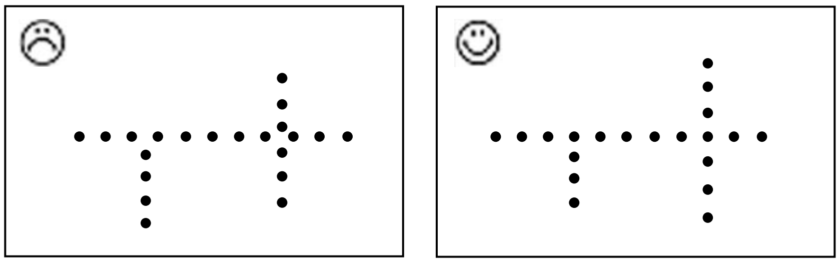


Fig. 3.6 Two dotted lines should meet with a dot

c. Between two dash-dotted lines: the intersection should lie on a dot (Fig. 3.7).

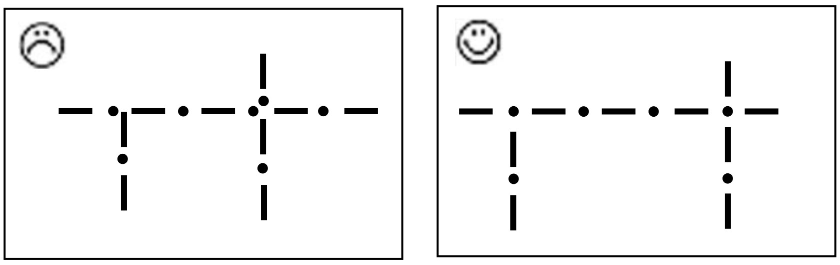


Fig. 3.7 Two dash-dotted lines should meet with a dot

d. Between a dashed line and a solid line: the dash should be centered on the intersection point. In case of a junction: the line should start with the solid part (Fig. 3.8).

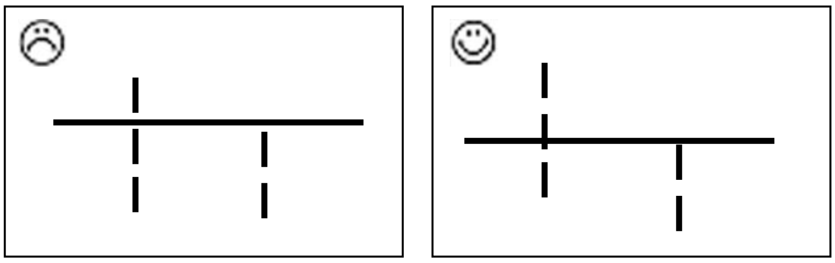


Fig. 3.8 A dashed line should meet a solid line with a dash

e. Between a dashed line and a dotted line: a dot should lie on the intersection (Fig. 3.9).

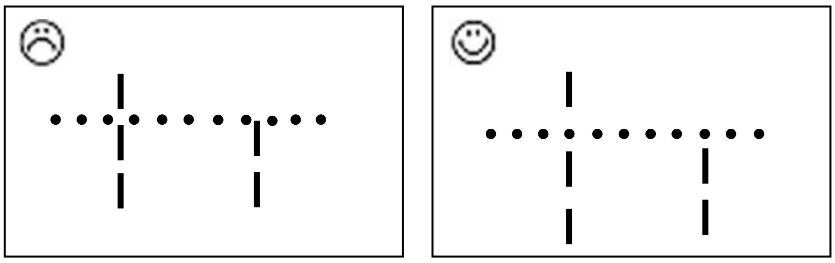


Fig. 3.9 A dashed line should meet a dotted line with a dot

f. Between a dashed line and a dash-dotted line: a dot should lie on the intersection (Fig. 3.10).

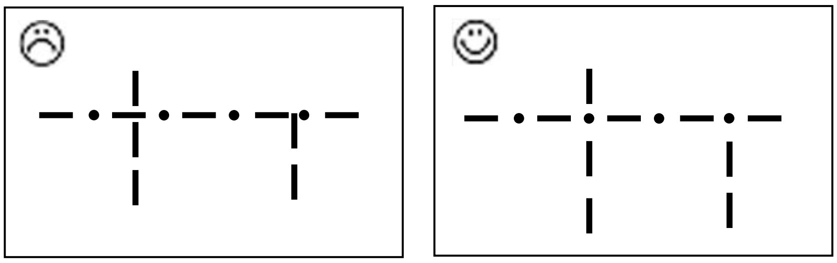


Fig. 3.10 A dashed line should meet a dash-dotted line with a dot

g. Between a dotted line and a solid line: the intersection should lie on a dot (Fig. 3.11).

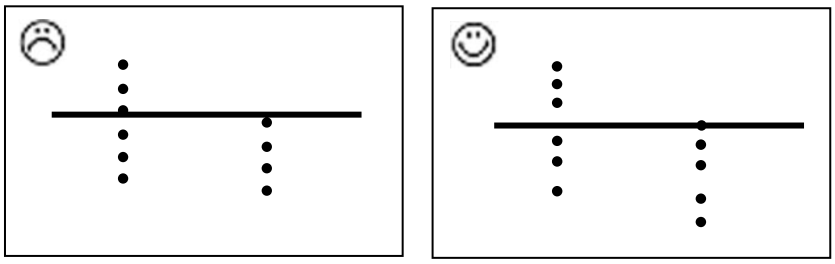


Fig. 3.11 A dotted line should meet a solid line with a dot

h. Between a dotted line and a dash-dotted line: the intersection should lie on a dot (Fig. 3.12).

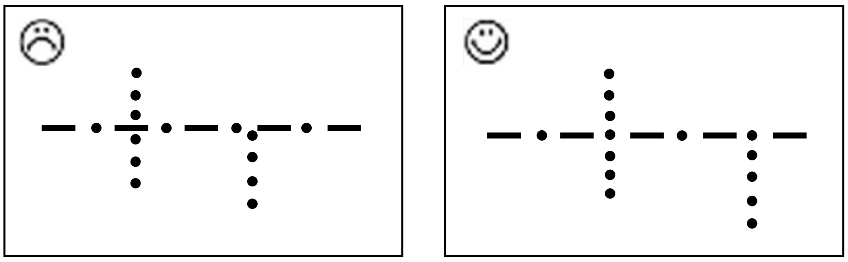


Fig. 3.12 A dotted line should meet a dash-dotted line with a dot

i. Between a dash-dotted line with a solid line: the intersection should lie on a dash (Fig. 3.13).

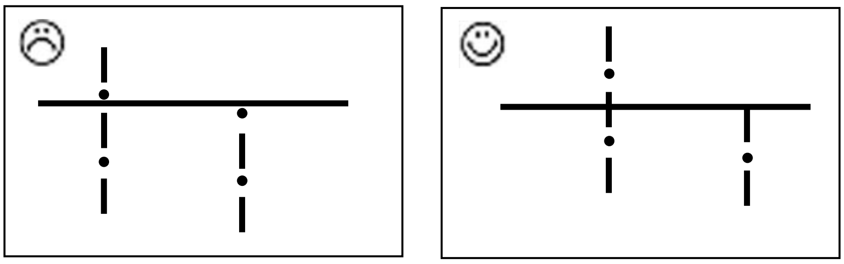


Fig. 3.13 A dotted line should meet a solid line with a dash

3.3.2 With complex lines

There are various treatments of intersections with complex lines involved. The following are a few examples of some considerations.

When two multiple lines meet, the intersection should not touch the interior parts of the lines (compare Fig. 3.14a and b; c and d).

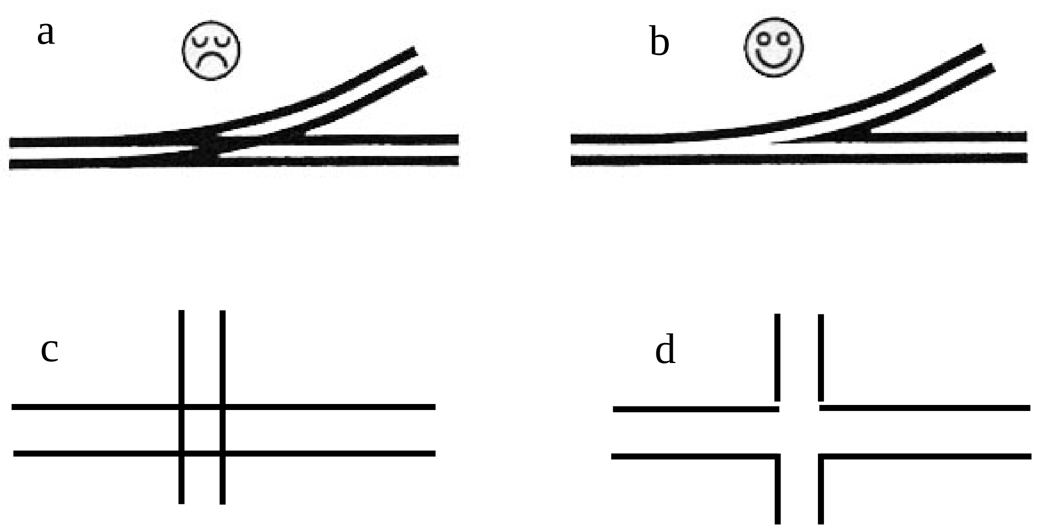


Fig. 3.14 Intersection of two double lines:

a, b © [TopoKarto, 1996] Darstellung der Elemente, Bahnen, p. 11

Lines composed of symbols should start and end with a complete symbol (compare Fig. 3.15a and b).

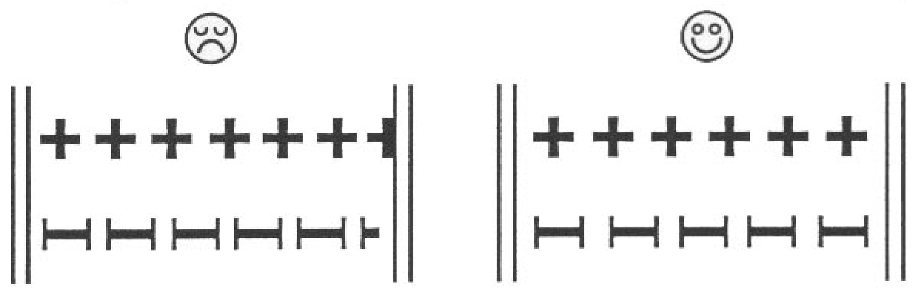


Fig. 3.15 Complex lines should terminate with complete symbol.

© [TopoKarto, 1996] Darstellung der Elemente, Grenzen, p. 2

3.4 Introducing cartographic considerations of linear symbols into computer programs

For a cartographer who is familiar with cartographic principles, inappropriate line symbolization as shown in this chapter should never occur if a map was carefully drawn. But until now, many of these skills have not been grafted into computer programs yet. So when we use common graphics software to draw digital maps, the preliminary results might contain unsatisfied effects.

This work aims to introduce the general cartographic principles of line symbolization into computer programs. Based on the above-introduced examples, there are two approaches to achieve this goal. Either we search for new drawing methods, or we adopt different line styles to circumvent the problem. For the first approach, new algorithms for line drawing need to be developed. For the second one, we need to construct a variety of new line styles. In the next chapter, we will present solutions using both approaches.

Chapter 4 Principles of the novel line drawing algorithms

4.1 Overview

This work aims to develop new methods to improve cartographic line symbolization. As a result, a new line drawing program is developed, which is called “SmartLine”. The development of this program focuses on two aspects. The first is to introduce cartographic principles into the line drawing process, so that the results either don’t need further modifications or can be adjusted easily to meet cartographic requirements. The second is to enrich line symbolization covering both styles and visual qualities.

A major achievement of SmartLine is the development of new line drawing algorithms for cartographic applications. Present line drawing algorithms typically lack some cartographic principles, which leads to the failure to demonstrate some important geographic information clearly and unambiguously. This phenomenon is especially noticeable when drawing non-solid lines, where gaps should not be presented at key positions, such as turns, intersections and line ends. In contrast, the new line drawing algorithms adopt new methods to control line strokes. They allow the automatic generation of lines with solid symbols (dashes or dots) at the key positions along the line paths.

To realize the new line drawing algorithms, many new parameters controlling line strokes are introduced, like dash length, gap length, dot size and so on. A new computer language of “Line Feature Description” is developed to describe these new settings. It is used to describe all the line features recognized by SmartLine, including both the new parameters and the common visual variables (line width, color and pattern). Together with another file specifying line geometries, map users can control the line appearance and line contents (geometries) separately.

In this chapter, the principles of the innovations made in the new line drawing algorithms are described, including the formalization of input data and the principles of the new line drawing algorithms.

4.2 Flow diagram of SmartLine

The procedure of SmartLine is illustrated in Fig. 4.1. Basically, the process can be grouped into three major parts; input, processing, and output.

4.2.1 Formalization of input data

The task of the input data section is to organize the information to be processed in the following drawing step, including line geometry and visual variables. For map designers, this is the place where to participate in the design process. By managing the input information, they determine both, the contents and appearance of a map.

There are two methods to manipulate the input data, batch mode and interactive mode. For the batch mode, the input information is stored in two XML files. One XML file, “line_info.xml”, is constructed to provide general line information, including layer and locations of geometry file. The other XML file, which is written in a new language of “Line Feature Description”, contains definitions of most visual variables of the line. Both XML files are open to the users; users can decide on the map contents and appearance by just editing these two files.

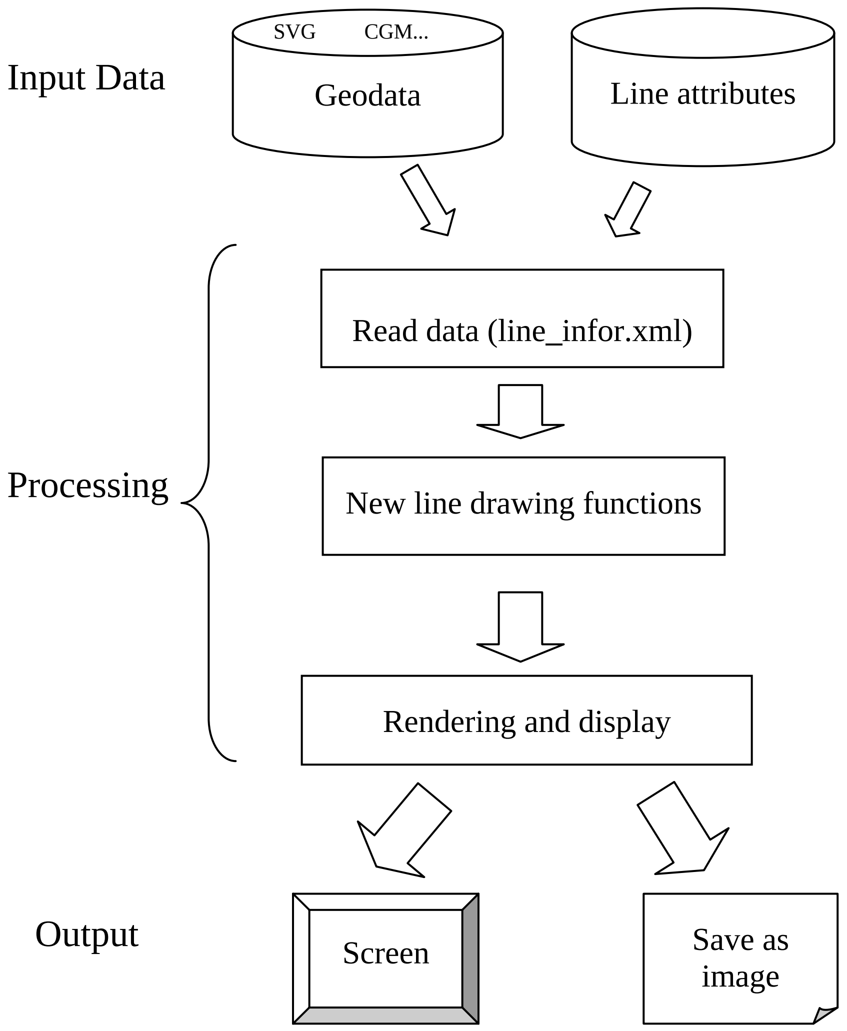


Fig. 4.1 The procedure of SmartLine

4.2.1.1 The data structure of geometric information

Most of the geometric paths originate from a GIS database; this data has to be first converted to a format known by the SmartLine program. Accepted formats are SVG, CGM, and a self-defined format, which simply consists of Cartesian coordinate pairs (Fig. 4.2).

Line's geometry by SVG path

```
<path id="922" d="M 665440.9375 213835.4843 L 665644.9375 214039.2968
                   L  665728.5625 214108.1718 L 665765.3750 214135.8281
                   L  665855.6250 214196.8281 L 665898.1875 214220.3437
                   L  666096.4375 214339.4843 "
/>
```

Line's geometry by CGM

```
LINE   2913,3219 2915,3205 2916,3202 2921,3185 2921,3184
       2923,3174 2927,3137 2928,3134 2929,3121 2929,3104
       2926,3090 2923,3080 2918,3067 2913,3057 2903,3040
       2848,2952 2834,2930 2800,2879 2786,2856 2776,2840
       2770,2831 2758,2813 2758,2812 2750,2797 2740,2779;
```

Line's geometry by coordinates pairs

```
2
   000 450,    050 464.65,    100 500,    150 535.35,    200 550,    250 535.35
   300 500,    350 464.65,    400 450,    450 464.65,    500 500,    550 535.65
   600 550,    650 535.65,    700 500,    750 464.65,    800 450
```

Fig. 4.2 The three formats recognized by the SmartLine program

4.2.1.2 Line feature description

Theory

For many present graphics software, the graphics information and the target objects are integrated together into the rendering process. A disadvantage of this method is that we need to add the graphics information repeatedly if the same settings are applied to different targets. A different method is to separate the graphics information and the target objects. An example is the application of Cascading Style Sheets (CSS). CSS is a kind of computer language to describe the document styles (including documents of HTML, XHTML, SVG and XML). It works as a reference containing a list of the visual variables. To generate objects with the same visual representations, we only need to provide a single CSS file, which is easy to maintain and edit. Since the graphic contents and their appearances are supplied separately, we can get access to the designing platform without opening the application program.

There are many different lines using the same legend in one map, the same legend may also be used in different maps. To facilitate the definition of line features, the same idea of CSS is applied to this work for drawing line symbols. A basis of the new line drawing algorithms is the introduction of many new parameters specifying the line styles, including dash length, gap length, dash start and so on; many of them can not be described by CSS. Thus, a new language of “Line Feature Description” is developed in this work. It is used to describe the line features that can be recognized by SmartLine. The description of line features is stored in a file of "line_properties.xml”. During line rendering process, line features are added by providing the location and name of this file.

This definition approach has two advantages over the traditional ways to draw line symbols. First, it improves the efficiency of symbolization. If a map contains several lines with the same properties, or if the same line symbols are drawn in different maps, we just need to construct one file of "line_properties.xml”; provide the name of the line symbol as a key and the location of this file during the rendering process. Second, it provides a flexible way to describe line features. The construction of "line_properties.xml” is application independent, we can either use an interactive platform to design the line appearance and save the settings in the file, or we can edit the file directly without opening the application program.

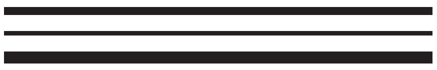
Application

In SmartLine, “Line Feature Description” defines 11 parameters specifying the line appearance, which are listed in Table 4.1.

Fig. 4.3 shows an application of “Line Feature Description” describing a symbol of highway used by Swisstopo. The seven specifying parameters are stored in the file "line_properties.xml”. All of them are enclosed within the “road” element, a child element of “line”, which enables to distinguish from other geometric primitives such as “point” or “area” for future applications.

Name	Description
Color	Line color in RGB mode
line-width	Line width
line-style	Solid, dashed, dotted or dash-dotted
dash_solid_length	The length of a dash
dash_gap_length	The length of a gap
dash_start_position	Dash phase
dash_number	The number of repetitions for each dash pattern (including one dash _solid_lengh and one dash_gap_length)
dot_length	Dot size of dotted line
parallel-number	Single, double or triple line
symmetrical	If the parallel lines have the same appearance (for double and triple line)
distance-between-line	Distance between two parallel lines (for double and triple line)

Table 4.1 The parameters specifying line appearance defined in “Line Feature Description”



(a). Highway from the legend of the Swiss National Map Services.

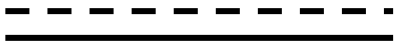
```
<line>
        <road
                name="highway" ………………..(1)
                parallel-number="3" ……………(2)
                symmetrical="yes" ……………...(3)
                distance-between-line="3" ……..(4)
                line-width="3,1,3"……………….(5)
                line-style="solid" ………………..(6)
                color="#FF0F0FFF"…………….(7)        />
</line>
```

(b) Features of the above-mentioned example as described in "line_properties.xml”.

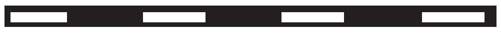
Fig. 4.3 An application of “Line Feature Description”

We can also store the visual representations of all linear symbols on a map in one file "line_properties.xml”. For example, it can contain ten elements if there are ten different line symbols on a map. The definitions for all line features are stored together with the name of that line symbol as the identifier of the element. During the rendering process, the same line symbols will be drawn by giving the path of "line_properties.xml” and the name of the line symbol. Fig. 4.4 shows another example that describes the attribute information of two more complex linear symbols as used by the Swiss National Map Services.

The application of “Line Feature Description” may be extended to draw other kinds of symbols. For example, we can create an XML file “symbol_property.xml”; different kinds of symbols are grouped by elements such as lines, points, areas etc. The visual representations of each symbol should be described in the same way as introduced above.



(a) 3rd class road



(b) cable railway: single track

```
<line>
        <road
                name="3class"
                parallel-number="2"
                symmetrical="no"
                distance-between-line="3"
                line-width="1.2"
                line-style="dash,solid"
                dash_solid_length ="2"
                dash_space_length ="1"
                dash_start_position ="0"
                color="#FF000000"
        />
        <railway
                name="narrow-single"
                parallel-number="3"
                symmetrical="no"
                distance-between-line="3"
                line-width="1.5,3,1.5"
                line-style="solid,dash,solid"
                dash_solid_length ="2"
                dash_space_length ="3"
                dash_start_position ="1.8"
                color="#FF000000"
        />
</line>
```

(c) Description in “Line Feature Description”.

Fig. 4.4 A “Line Feature Description” file containing two line symbols

4.2.1.3 Storage of general line information

The “Line Feature Description” is used to describe the line appearance; other general information is given by another file named “line_info.xml”. It acts as a reference to provide the following information, which will be processed by the drawing functions (Fig. 4.1):

1. Define the location of the Line’s Feature Library
2. Define the layer
3. Define the name of the line symbol used as the identifier in Line Feature Description
4. Define the location of the geometry file for each line symbol
5. Define the visibility of each layer and each line

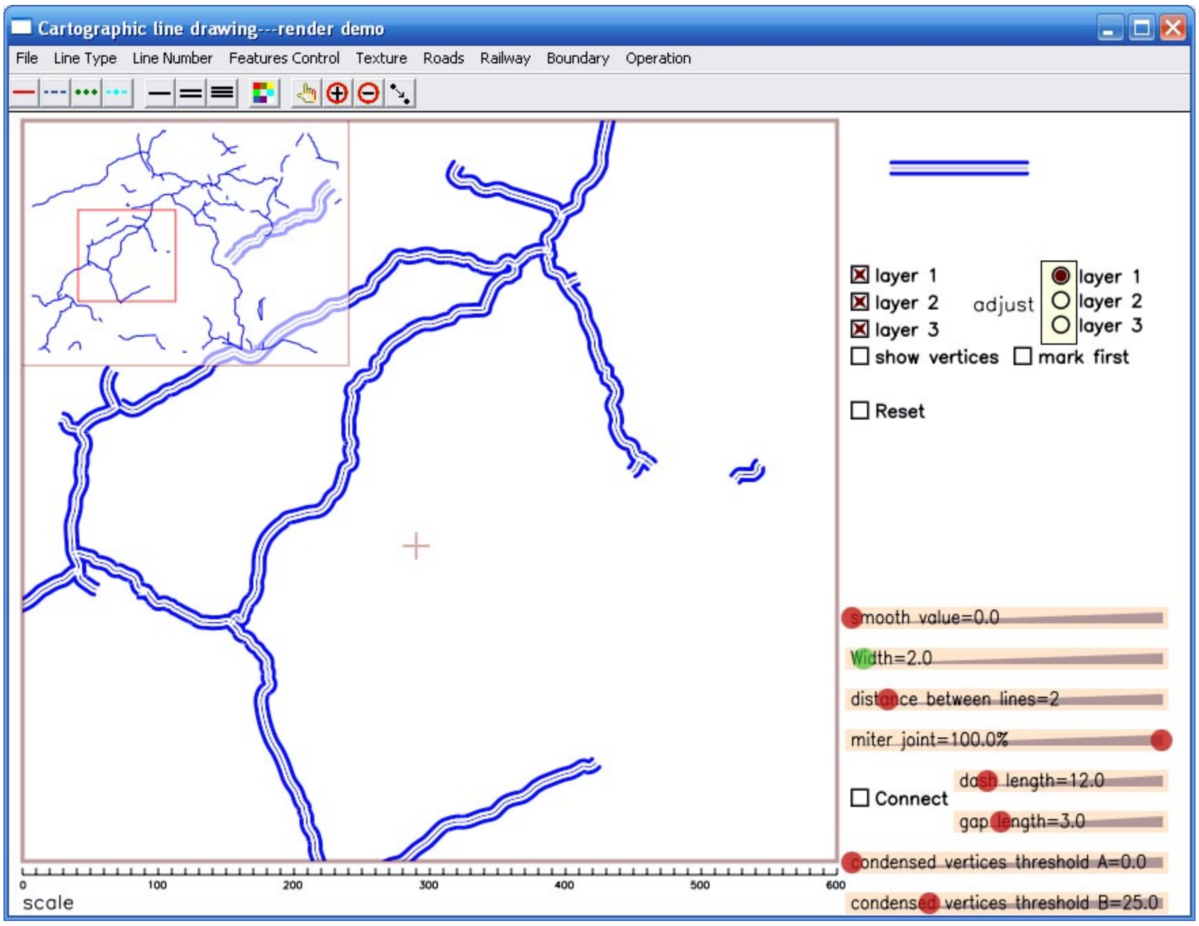
Fig. 4.5 is an application of “line_info.xml”. Within the line element <line descriptor=”xml/line_properties.xml”>, the “descriptor” attribute specifies the location of the “Line Feature Description” file, which is “xml/line_properties.xml”. In the child element <layer order=”2” object_number=”3” visible=”no”>, “order” defines in which layer the following line is contained (layer 2 in this case); “object_number” indicates the number of lines within this layer (3 lines) and “visible” specifies the visibility of this layer. Each of the following three child elements defines a line symbol. For example, the first child element of **<**road id=”201” name=” highway “ visible=”yes”>gis_data/dat/test01.dat </road**>** contains three attributes: “id” is the code of the road; the “name” attribute points to the highway in the “Line Feature Description”; “visible” controls the visibility of this line. The “road” element, **“**gis_data/dat/test01.dat**”**, specifies the location of the geometry file.

```
<line descriptor="xml/line_propeties.xml">
            <layer order="2" object_number="3" visible="no">
                <road id="201" name="highway" visible="yes">
                        gis_data/dat/test01.dat</road>
                <road id="202" name="5class" visible="yes">
                        gis_data/dat/test02.dat</road>
                <road id="203" name="6class" visible="yes">
                        gis_data/dat/test03.dat</road>
            </layer>
  </line>
```

Fig. 4.5 The “line_info.xml” file with the general line information

4.2.1.4 A user-friendly interface managing the input data

For convenience, a user-friendly working environment is constructed to manipulate the input information. It contains tools including menus, sliders, check boxes, and combo boxes to control the appearance of the line symbols (see Fig. 4.6). They enable users to judge and see the results immediately. After the design process, the information can be stored in the two XML files, “line_info.xml” and "line_properties.xml”. The drawing engine will create line symbols based on the information specified by these two files. If we need to add new line symbols or change the visual variables, we can edit these two files directly. For maps using the same legend, we just need to create new file similar as “line_info.xml” to specify map contents and layout, and provide the path to the same "line_properties.xml”.



☒ layer 1
☒ layer 2
☒ layer 3
☐ show vertices
☐ mark first

● layer 1
○ layer 2
○ layer 3

☐ Reset

smooth value=0.0
Width=2.0
distance between lines=2
miter joint=100.0%
☐ Connect
dash length=12.0
gap length=3.0
condensed vertices threshold A=0.0
condensed vertices threshold B=25.0

Fig. 4.6 The user-friendly interface of the program

4.2.2 Processing

The input information specified by the two XML files will be first parsed by a program that is built on top of the Expat XML parser [Cooper, 2000]. Then, the information is stored in two corresponding structures. One structure specifies the geometries and the other specifies the visual variables.

The first structure is called “line_source”, which contains the paths and names of the “Line Feature Description”, the geometries, names of the linear symbol, and other attributes like visibility, layer order and so on (Fig. 4.7).

```
struct line_source
{
        //layer infomation
        unsigned*    layer_num;
        unsigned*    layer;
        bool*        visible;

        //road infomation
        unsigned*    id;
        char**       style;
        char**       name;
        char**       data_source;

        //general infomation
        char*        descriptor;
        unsigned     total_line_num;
        int          curr_line;
};
```

Fig. 4.7 Structure of “line_source”

The second structure is called “line_properties”, which includes information about visual variables like color, width, cap style, joint style, line style, textured line or plain line, the distance between parallel lines (for double line and triple line) and so on (Fig. 4.8).

```
struct line_properties
{
          char*               style;
          char*               name;
          unsigned            layer;
          bool                visible;

          int                 para_number;      //1,2 or 3
          bool                symmetrical;
          double              distance;
          double              width[3];
          ……
}
```

Fig. 4.8 Structure of “line_properties”

The drawing functions in the following rendering process are the core of the whole program, including drawing algorithms for solid lines, dashed lines, dotted lines, dash-dotted lines, double lines and triple lines. The principles of these algorithms are described in the remaining chapter. The whole program is developed in C++ and compiled under both the Windows and Linux platform.

4.3 Control of miter joint

4.3.1 Theory

The drawing method for solid lines is the basis for all other line drawing. Normally, there are fewer problems with solid lines compared with other lines. The classic algorithms to render solid lines have been introduced in chapter 2 and are adopted for this program. But one property of a solid line that should draw our attention to is the miter joint.

Using miter joints is one way to connect straight line segments. In Fig. 4.9, the size of the joint, which is the length of $|V_1 J_{10}|$, is determined by the angle $\angle V_0 V_1 V_2$.

$$|V_1 J_{10}| = \frac{0.5 * line\_width}{\sin(0.5 * \angle V_0 V_1 V_2)} \qquad [4\text{-}1]$$

Theoretically, if the angle $\angle V_0 V_1 V_2$ tends towards zero, the length of $|V_1 J_{10}|$ grows indefinitely long (see Fig. 4.9). Obviously, such large peaks are not necessary and will interfere with the display of other information. Usually, a miter limit is specified to cut off large peaks, if the length of $|V_1 J_{10}|$ exceeds the miter limit.

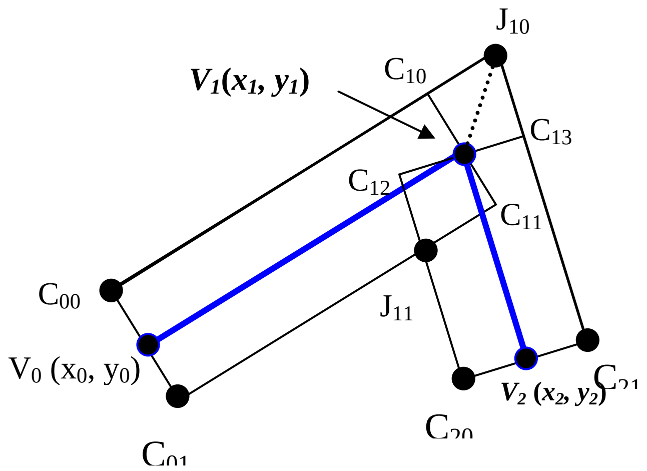


Fig. 4.9 Miter joint built by line segments $\overline{V_0 V_1}$ and $\overline{V_1 V_2}$

In this work, two parameters controlling the peak length were introduced. One is a length threshold $|V_1 J_{10}|$ and the other is a cutting percentage of the miter length. The length threshold is set internally in the program as two times the line width. In case the length of $|V_1 J_{10}|$ is larger than the threshold the angle will be cut off to an extent that is a user-controlled parameter between 0 and 100%. When it is set to 0.0%, the miter joint is cut off completely and become a bevel joint (Fig. 4.10). A slider is provided by the SmartLine application, allowing users to control this cutting percentage (Fig. 4.11).

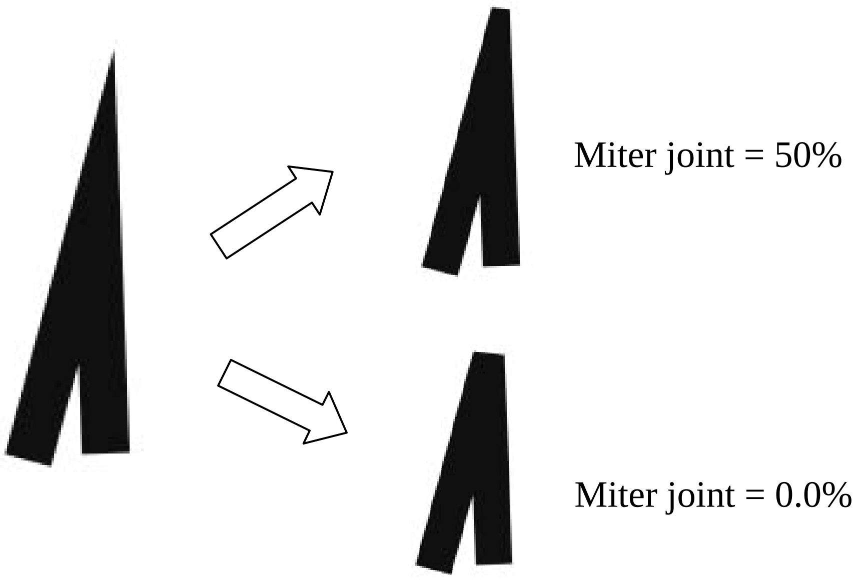


Fig. 4.10 Miter joint with sharp angle and cutting results

Fig. 4.11 Controller for the miter joint

4.3.2 Computations

In Fig. 4.9, to draw a miter joint at $\angle V_0V_1V_2$, we need to compute the positions of J_{10} and J_{11}. First, we can get the positions of (C_{00}, C_{01}, C_{10}, C11) and (C_{20}, C_{21}, C_{12}, C_{13}) based on the positions of V_0, V_1, V_2 and line width.

Suppose C_{00} = (ax, ay), C_{10} =(bx, by), C_{21} = (cx, cy), C_{13} = (dx, dy), the intersection J_{10} = (x, y), then

$$\begin{cases} \dfrac{y-ay}{by-ay} = \dfrac{x-ax}{bx-ax} \\ \dfrac{y-cy}{dy-cy} = \dfrac{x-cx}{dx-cx} \end{cases} \qquad [4\text{-}2]$$

By solving this quadric equation group, we get

$$\begin{cases} x = ax + \dfrac{(ay-cy)*(dx-cx)-(ax-cx)*(dy-cy)}{(bx-ax)*(dy-cy)-(by-ay)*(dx-cx)}*(bx-ax) \\ y = ay + \dfrac{(ay-cy)*(dx-cx)-(ax-cx)*(dy-cy)}{(bx-ax)*(dy-cy)-(by-ay)*(dx-cx)}*(by-ay) \end{cases} \qquad [4\text{-}3]$$

If we set

$$r = \frac{(ay-cy)*(dx-cx)-(ax-cx)*(dy-cy)}{(bx-ax)*(dy-cy)-(by-ay)*(dx-cx)} \qquad [4\text{-}4]$$

the solution can be simplified to

$$\begin{cases} x = ax + r*(bx-ax) \\ y = ay + r*(by-ay) \end{cases} \qquad [4\text{-}5]$$

With the new settings, we get modified equations as follows.

$$\begin{cases} x_1 = ax + r*(bx-ax)*miter\_limit \\ y_1 = ay + r*(by-ay)*\ miter\_limit \end{cases} \qquad [4\text{-}6]$$

$$\begin{cases} x_2 = ax - r*(bx-ax)*\ miter\_limit \\ y_2 = ay - r*(by-ay)*\ miter\_limit \end{cases} \qquad [4\text{-}7]$$

The coefficient of "*miter_limit*" refers the cutting percentage, which is between 0 to 100%.

Similarly, we can also get J_{11} = (x1, y1). Suppose $C_{01} = (a'x, a'y)$, $C_{11} = (b'x, b'y)$, $C_{20} = (c'x, c'y)$, $C_{12} = (d'x, d'y)$, then

$$\begin{cases} x1 = a'x + r*(b'x - a'x) \\ y1 = a'y + r*(b'y - a'y) \end{cases} \qquad [4\text{-}8]$$

where

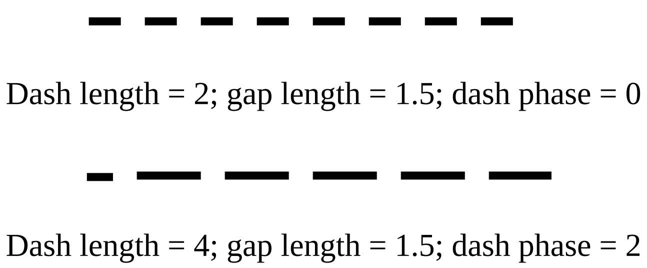
$$r = \frac{(a'y - c'y)*(d'x - c'x) - (a'x - c'x)*(d'y - c'y)}{(b'x - a'x)*(d'y - c'y) - (b'y - a'y)*(d'x - c'x)} \qquad [4\text{-}9]$$

4.4 Dashed line

Compared with solid lines, it may arise more problems when drawing dashed lines, especially at line ends, turns, and intersections. Fig. 4.12 shows three examples of inappropriate dashed lines such as gaps at key positions and unbalanced arms. A focus of the new drawing algorithms is to provide automatic solutions of these problems.

4.4.1 Stroke adjustment

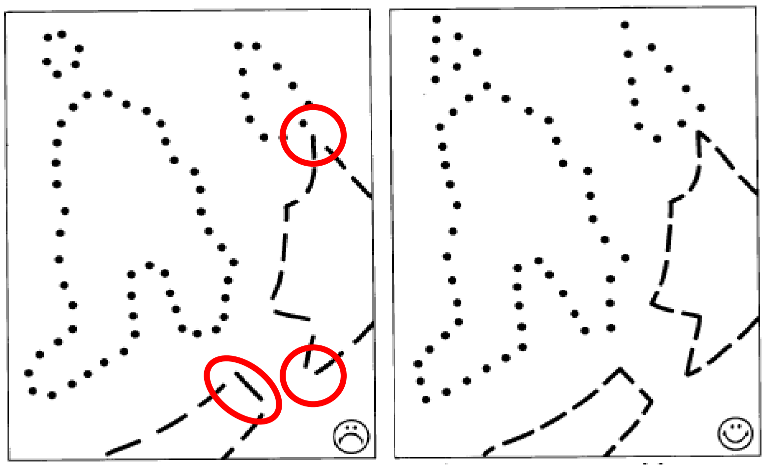
The problems shown in Fig. 4.13 are caused by the nature of dashed line patterns. The pattern of a dashed line is commonly specified by a dash array and the dash phase. The dash array determines a sequence of dash length and gap length that is repeatedly aligned. The dash phase determines the start position of the first sequence (Fig. 4.12).



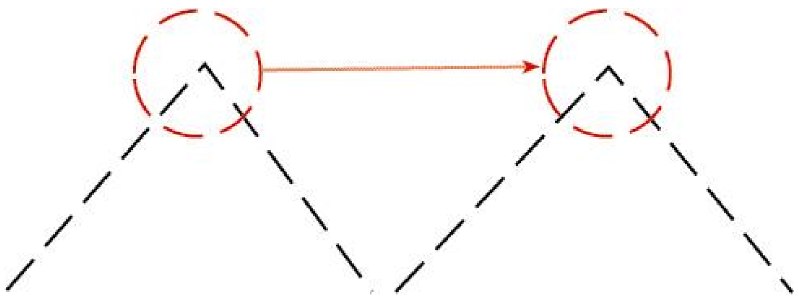
Dash length = 2; gap length = 1.5; dash phase = 0
Dash length = 4; gap length = 1.5; dash phase = 2

Fig. 4.12 Two dashed line patterns

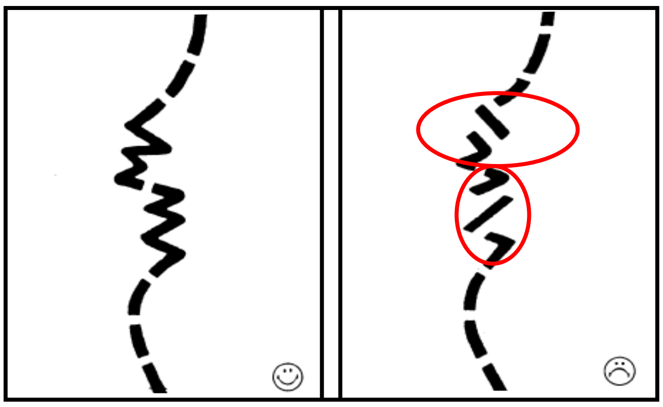
With traditional drawing methods, once a dashed line pattern is chosen, a computer can “faithfully” produce a continuous dashed line. Thus, it is possible to get the following results (Fig. 4.14): either it ends with an incomplete solid part (Fig. 4.13a) or a gap (Fig. 4.13b).



a. Turns with inappropriate gaps (left) and optimized results (right).



b. Turns should better have balanced arms



c. Large gaps at key positions

Fig. 4.13 Some inappropriate results of dashed lines ©[Spiess, 1995a]

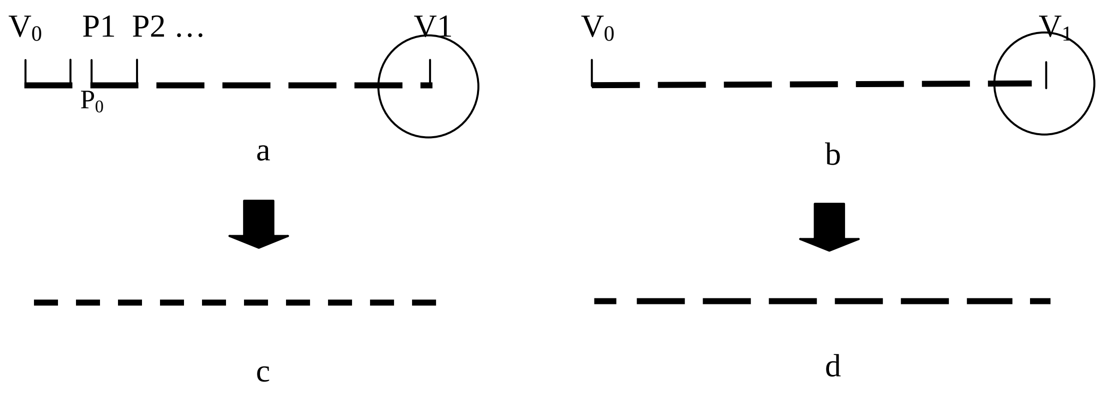


Fig. 4.14 Adjustment of dash pattern

In theory, if the strokes could be adjusted, we can control the appearance of dashed lines to avoid such problems. For example, we can adjust dash lengths and/or the gap lengths, so that we can define any position on the line to be a gap or a solid (Fig. 4.14c). Or we can adjust the dash phase to move the positions of all dashes and gaps (Fig. 4.14d). We can also coordinately adjust these three parameters at the same time.

Fig. 4.15 is a demonstration of the method to draw a dashed line with a corner using present graphics software: suppose V_0, V_1, … V_n, are the vertices of a dashed line.

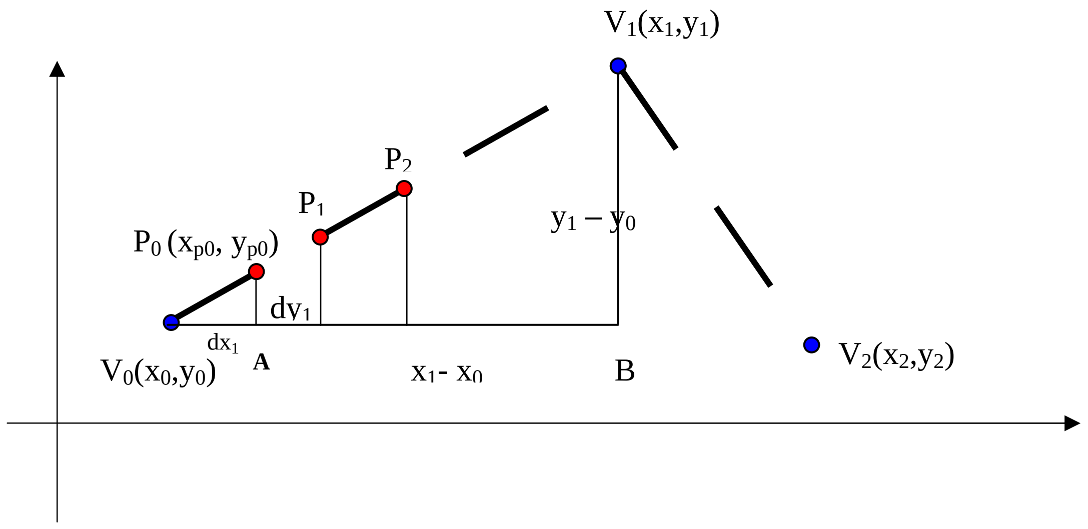


Fig. 4.15 The principle to draw a dashed line using traditional method

To calculate the positions of the dash ends, *P₀, P₁, ... , Pₙ* (in Fig. 4.15), since:

$$\frac{V_0P_0}{V_0V_1} = \frac{V_0A}{V_0B} = \frac{P_0A}{V_1B} \qquad [4\text{-}10]$$

we can get:

$$\text{P}_0\text{:} \begin{cases} x_{p0} = x_1 - (x_1 - x_0) * \overline{P_0V_1} / \overline{V_0V_1} \\ \\ y_{p0} = y_1 - (y_1 - y_0) * \overline{P_0V_1} / \overline{V_0V_1} \end{cases} \qquad [4\text{-}11]$$

$$\text{P}_1\text{:} \begin{cases} x_{p1} = x_1 - (x_1 - x_0) * \overline{P_1V_1} / \overline{V_0V_1} \\ \\ y_{p1} = y_1 - (y_1 - y_0) * \overline{P_1V_1} / \overline{V_0V_1} \end{cases} \qquad [4\text{-}12]$$

where

$$\overline{P_0V_1} = \overline{V_0V_1} - \overline{V_0P_0} = \overline{V_0V_1} - dash\_length \qquad [4\text{-}13]$$

$$\overline{P_1V_1} = \overline{V_0V_1} - \overline{V_0P_1} = \overline{V_0V_1} - \overline{V_0P_0} - gap\_length\,. \qquad [4\text{-}14]$$

By connecting V_0 and P_0, we get the first dash. For the next dash, we need to subtract the length of the first dash and gap from the whole segment V_1V_2. By repeating this process, we can get the positions of all dashes.

But most line geometries have irregular paths; there could be many corners that need to be considered simultaneously. An adjustment of a single parameter at a time may not be sufficient to overcome all problems. Especially, most maps have many line symbols. Map designers have to search extensively for good solutions to cover all the problems. And all the work has to be done manually. Therefore, an automatic solution needs to be developed. Ideally, we hope that the drawing engine could recognize turns and crosses of dashed lines. Then, it should automatically adjust the strokes to make balanced dashes at these places.

4.4.2 Principle of the new drawing method

To solve the problems with dashed lines mentioned above, we developed a new drawing algorithm. In the following, we will present an example showing the two major problems with dashed lines to explain the new method: gaps at corners ($\angle V_0V_1V_2$ in Fig. 4.16) and unbalanced arms ($\angle V_1V_2V_3$ in Fig. 4.16).

With present dashed line drawing method, a rigid pattern of dashes and gaps is produced between start and end point. While in SmartLine, the dashed line is drawn as a series of consecutive short dashed lines, each pair of consecutive vertices is considered as the end points of an individual segment, which will be drawn as an individual short dashed line.

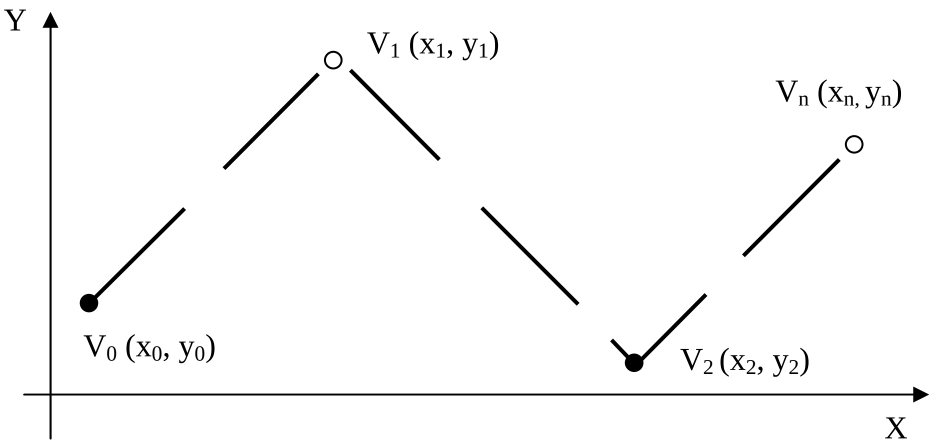


Fig. 4.16 Example of dashed line with a blank turn and a turn flanked with unbalanced arms

The new method can be explained in three steps. First, we consider a dash plus a gap as a drawing unit. Each small dashed line segment contains an integer number of drawing units. Taking the pair of vertices (V_1, V_2) in Fig. 4.16 as an example, the length of $\overline{V_1V_2}$ is 14. If we consider a dash length of 4 plus a gap length of 2 as one unit, $\overline{V_1V_2}$ actually contains 2.3 of such units (14/6). This number should be adjusted to an integer number, either 2 or 3. This can be done either by a cut-off or a round-off process. In SmartLine, we took the round-off process. In this case, 2.3 is rounded to be 2. Then, there is a round-off error, the extra length of *l* (Fig. 4.17). Because the locations of vertices are fixed, the extra length must be evenly allocated within $\overline{V_1V_2}$. That's why the dash length and gap length have to be re-calculated. In SmartLine, the gap length is kept untouched and only the dash length is adjusted. Accordingly, the new dash length should be 4+*l*/2. After the adjustment, the gap length is kept to be 2, but the dash length will be 5 (instead of 4). Now the segment $\overline{V_1V_2}$ starts with a full dash and ends with a full gap.

In the second step, the dash phase of each short line segment is reset. In most present graphics applications, the dash phase is set to 0, which allows the line start with a full dash (Fig. 4.18). In SmartLine, we set the phase to half the dash length. Then, each line segment will start with and end in a half dash. This ensures every corner to be solid since they consist of two half dashes from the previous and following line segment (Fig. 4.19). The whole dashed line also has one half dash at the beginning and one half dash at the end.

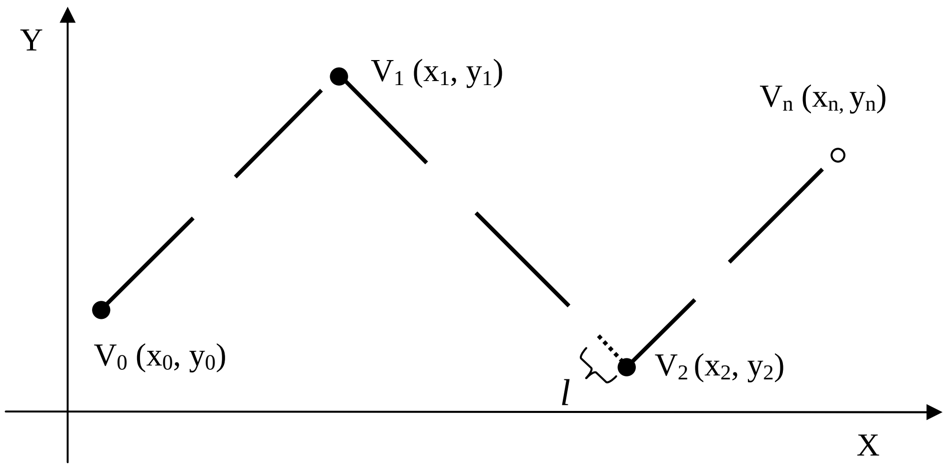


Fig. 4.17 Recount the number of drawing unit

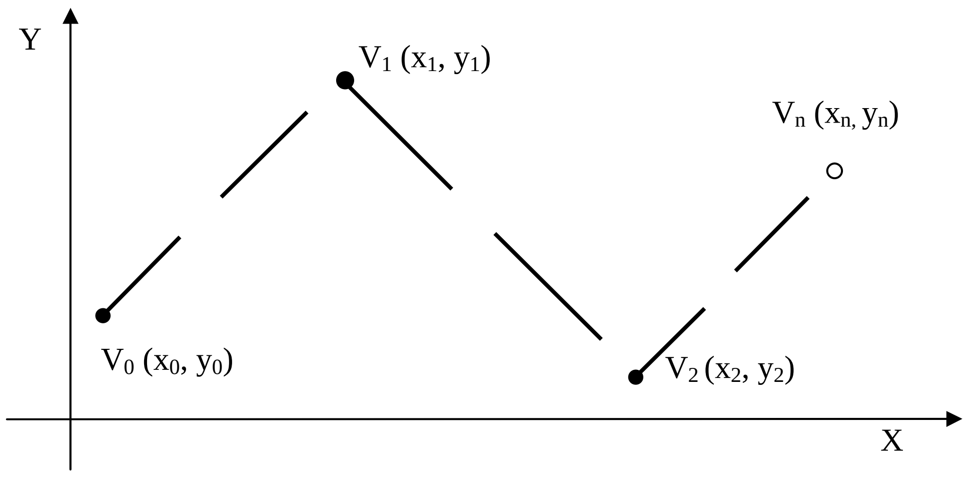


Fig. 4.18 After adjustment of dash length

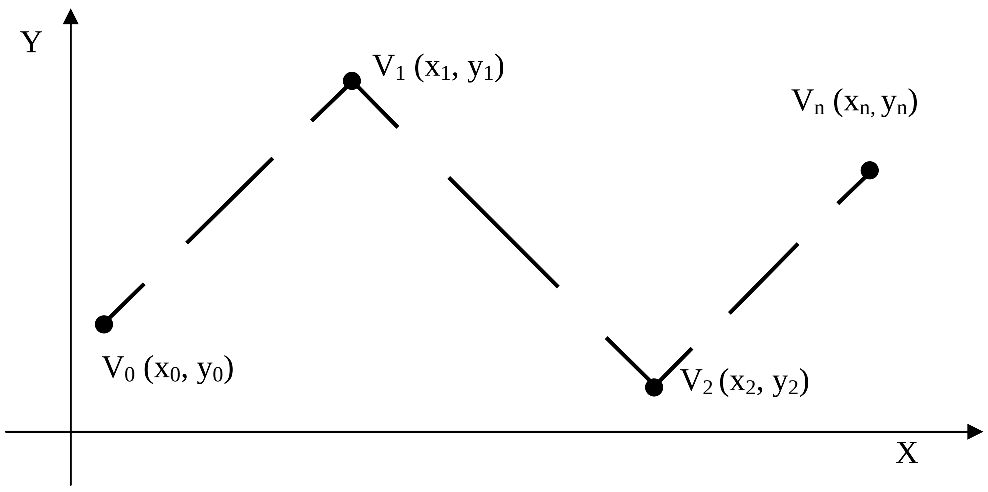
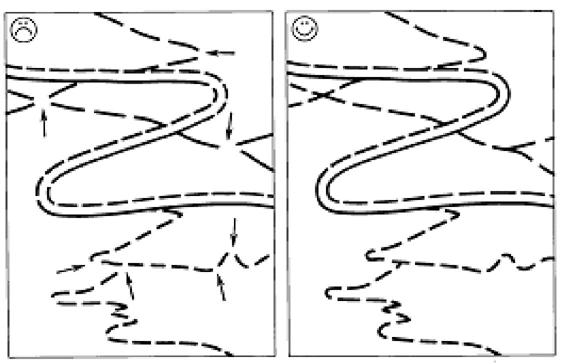
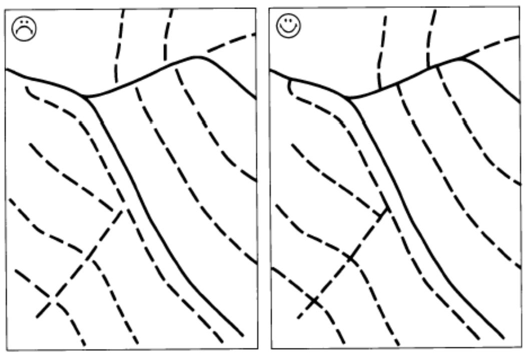


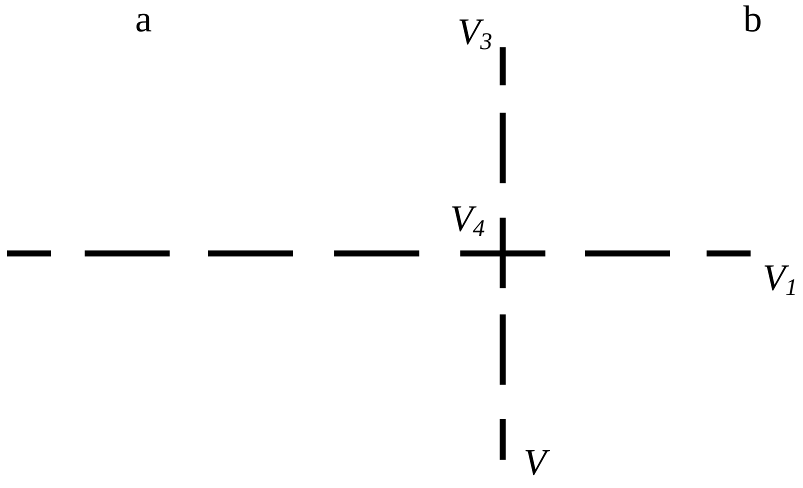
Fig. 4.19 Reset dash phase



a



b



c. Drawing a cross of two dashed line using the new method

Fig. 4.20 Crosses of dashed lines: a and b ©[Spiess, 1995a]; c, using the new method.

This method ensures corners to be solid, but it does not exactly produce equally balanced arms. This is because the length of adjacent arms is determined by the dash length of different line segments and these dash lengths may not be the same after adjustment. In the worst case, a ratio between the lengths of two arms might be nearly 2:1. But in most situations, the difference is so small (for most applications, there are only a few pixels) that the human eye may not perceive it.

This method also provides a solution to the problems with crosses of dashed lines. Fig. 4.20a and b are two examples showing the problems. We want the crosses to be drawn as solid parts. Because crosses are important positions, most of them will be sampled during data collection. As far as there exists a vertex at the cross point, the new method will automatically create solid intersections. Instead of drawing two dashed lines with one cross, it actually produces four dashed line segments with four balanced arms at the cross point (Fig. 4.20c).

4.4.3 Short distances between adjacent vertices

In addition to line patterns, the arrangement of line drawing components is also an important consideration during line symbolization. For example, if some vertices are closely packed at turns, it may cause unpleasant effects with either the traditional method or the new method. In Fig. 4.21a, the problem is that the distances between adjacent vertices are too short. With traditional method, the important geographic information at the sharp turns is missing. Fig. 4.21b is the result using the new method. Although the turns are clearly drawn as solid dashes, the continuous line path is disrupted by too many gaps. One solution is to switch to solid line styles within this small region. A few of short solid line segments will not interrupt the overall dashed line pattern (Fig. 4.21c).

To activate this drawing function, a threshold of “Condensed vertices threshold A” is set up. Initially, the value is set to be zero to turn off this function. When we find problems similar to those shown in Fig. 4.21a and want to connect all the condensed vertices with solid lines, we can adjust this setting. In theory, this threshold should be smaller than the length of one dash plus one gap. For example, if one dash plus one gap is 17 pixels, we might set this threshold for instance to 15. Wherever the distance between two adjacent vertices is less than 15 pixels, the computer will draw a solid line between the two points. The rest of the line is still drawn using the new dashed line method.

This solution works well for short regions containing sharp turns. But if a long solid line is created, it may disturb the whole dashed line pattern. In case where it is not necessary to show all of the crowded vertices, for example at smooth turns, we might use the traditional method. Fig. 4.22a shows a line with smooth turns whose vertices are closely packed. Fig. 4.22b shows the result after automatic adjustment of dash length using the new dashed line drawing method. In the new method, to ensure the generation of solid turns, the two half dashes at line ends are first to be drawn. Thus, when some distances between two vertices are smaller than one dash plus one gap, they are drawn as solid lines. In this situation, using the traditional drawing method is a better choice. Fig. 4.22c is the result using the traditional method. Even though it has gaps at turns, no important geographic information is missing due to the smooth geometries, and the dashed line pattern is still kept constant.

Another threshold was introduced in the program to switch between the new and traditional method, which is called “Condensed vertices threshold B”. At least, this threshold should be 1.5 times of one dash plus one gap. If the distance between two adjacent vertices is smaller than the threshold, it uses the traditional drawing method; otherwise it uses the new drawing method.

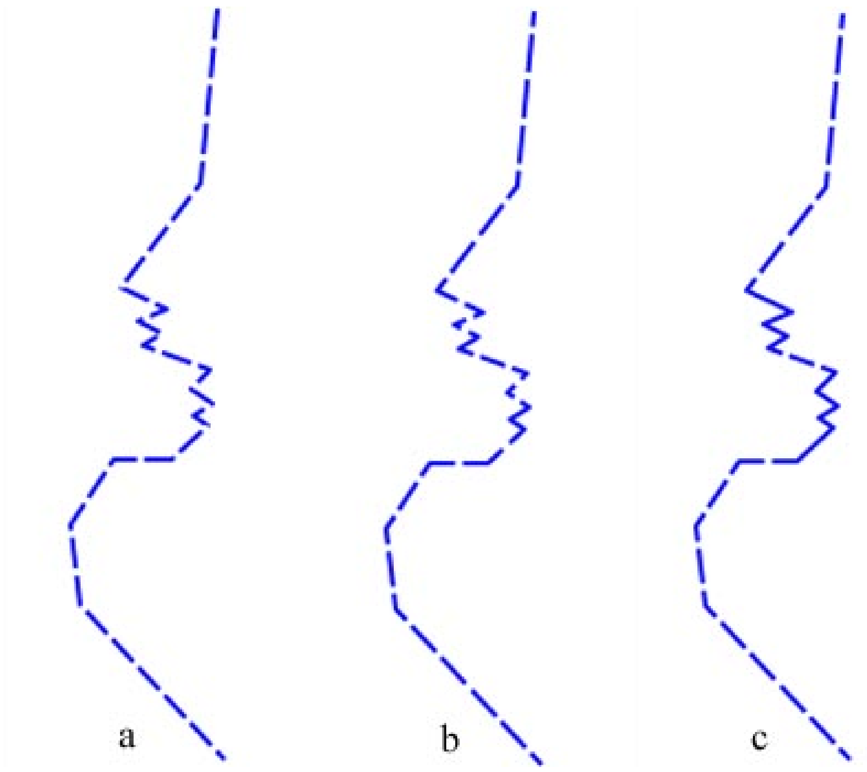


Fig. 4.21 Results at crowded vertices with different methods

a. Traditional method; b. New method; c. New method with modifications

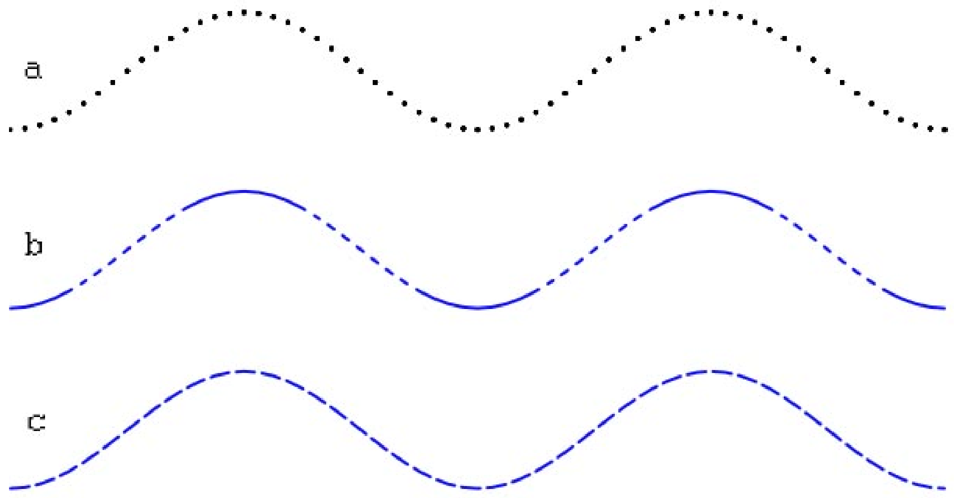


Fig. 4.22 Dashed lines with closely packed vertices

a: geometric vertices; b: using new method; c: using traditional method

4.5 Dotted line

The new method of drawing dotted lines is similar to draw dashed lines. A dotted line can be looked as a dashed line with short dashes and long gaps. In SmartLine, each dot is drawn as a filled circle (Fig. 4.23); the dot size (determined by line width) and gap length can be adjusted by users.

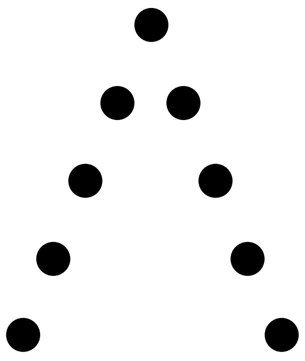


Fig. 4.23 The way taken by SmartLine to show dots of dotted lines

Like drawing a dashed line, there are also considerations at corners, crosses and line ends when drawing a dotted line. Generally, we want to have dots instead of gaps at these places. Fig. 4.24 presents some examples of appropriate dotted lines. The corners and crosses are not properly shown. These problems could also be solved by adjusting the line strokes, including dot size and gap length. But if a dotted line has many turns, this simple treatment is not enough to cover all the problematic places. It can be an expensive work since many maps contain many line symbols. A better way relies on the improvements of the dotted line drawing functions for an automatic solution.

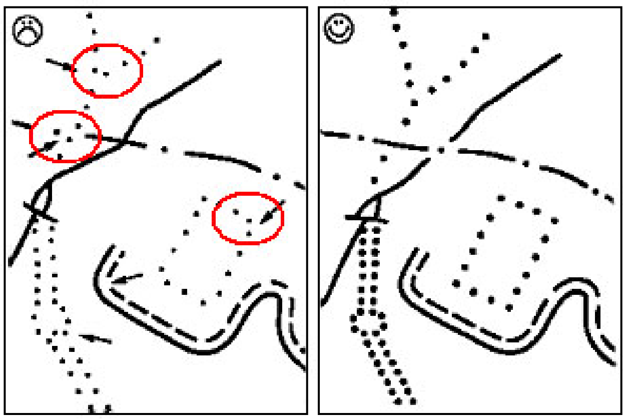


Fig. 4.24 Turns of dotted line should be filled with dots

A new method to draw dotted lines is developed in SmartLine. The idea is similar to the new method of drawing dashed lines. A dotted line is looked as a series of strait line segments with two adjacent vertices as the end points. Each line segment is rendered individually. The gap length is automatically adjusted so that each segment only contains integral number of gaps. For example in Fig. 4.25, $\overline{V_0V_1}$ contains 3.5 gaps, this number is rounded off to be an integer of 4. The error will be evenly distributed within four gaps. Meanwhile, one more dot is added to the end of this segment (Fig. 4.25). As a result, five dots will be drawn along $\overline{V_0V_1}$. Using this method ensures that there is always a dot at a turn or an intersection and each line ends precisely in a dot.

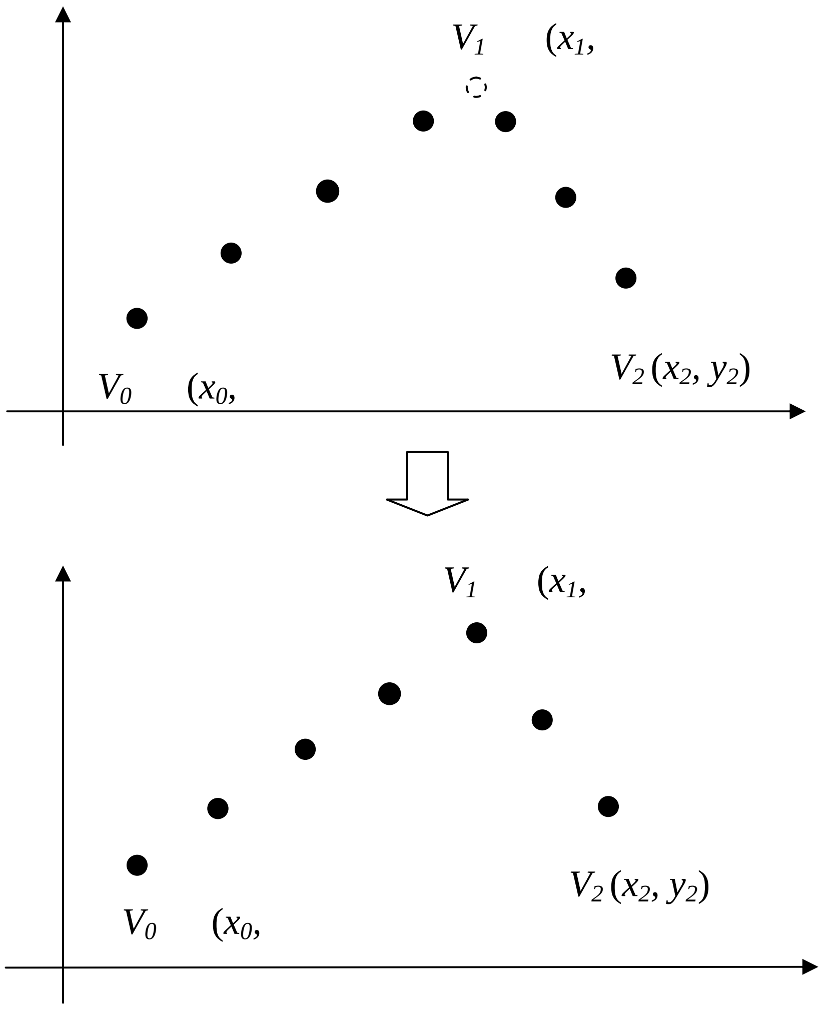


Fig. 4.25 Diagram of the new method to draw a dotted line

4.6 Dash-dotted line

If we consider a dot in a dash-dotted line as a short dash, the stroke of dash dotted line can be looked as a special case of dashed line: each drawing unit is composed of a long dash, a gap, a short dash, and a second gap instead of a dash and a gap as for a dashed line.

During map design, we should also pay attention to corners and crosses of dash-dotted lines. Generally, we want to have dots at these places; each corner has balanced arms followed by two gaps. But the traditional method does not consider these corner positions; there might be gaps or unbalanced arms (Fig. 4.26). To improve the drawing process of dash-dotted lines, I made the following modifications of the drawing method (the solution is similar to what I did with dashed lines):

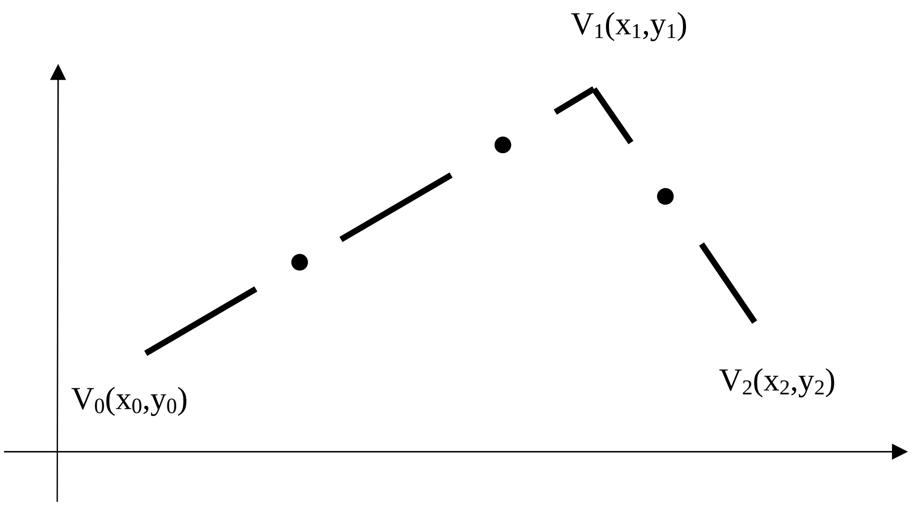
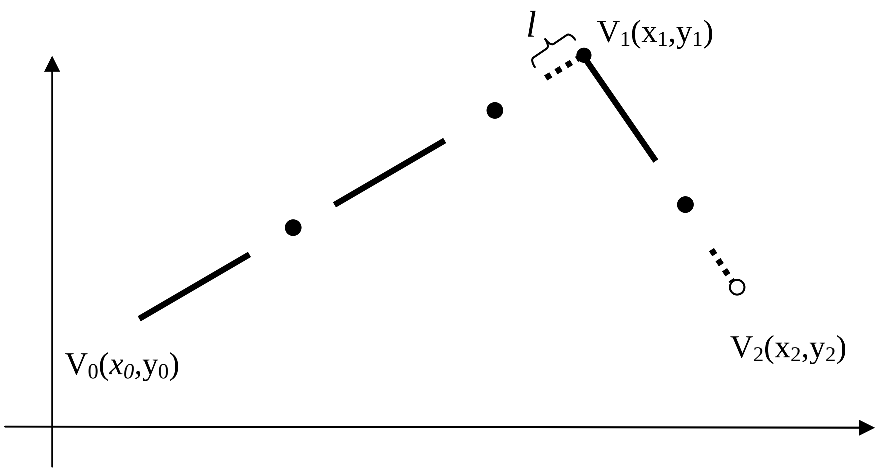
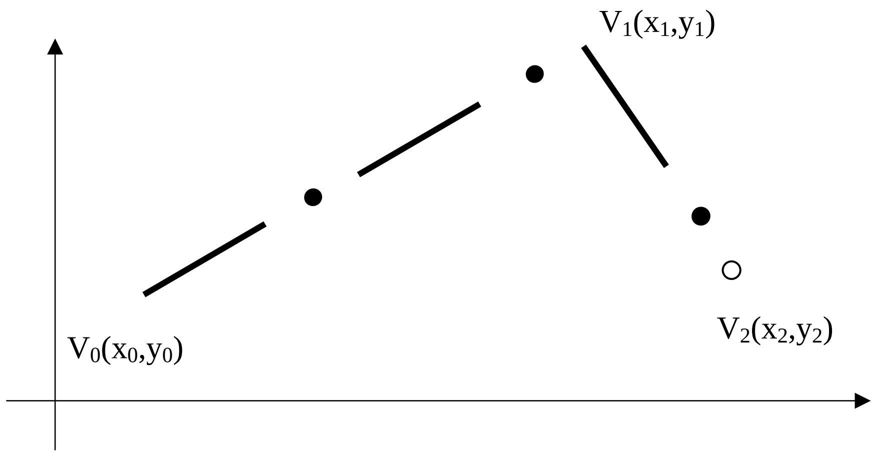


Fig. 4.26 A dash-dotted line drawn with traditional method

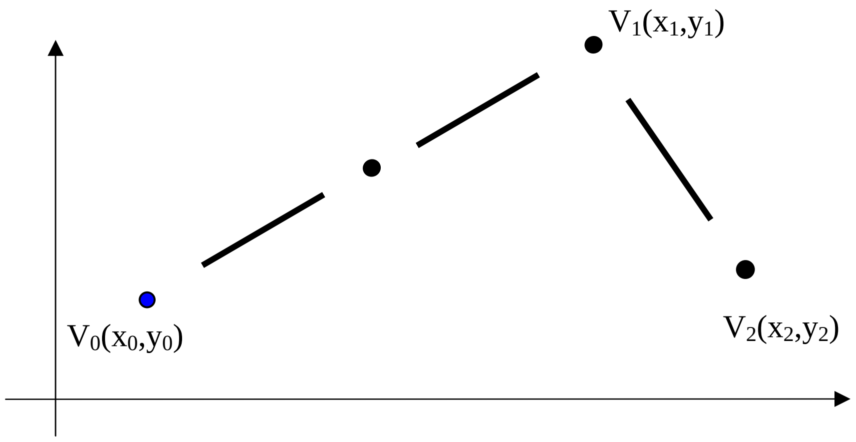
First, the number of drawing units between two adjacent vertices recounted. This number is rounded to an integral round-value. Second, the error is evenly distributed between the dashes. Third, the dash phase is set to start with a dot, and at the end point, a dot is added (Fig. 4.27).



Step 1. Adjust the number of drawing units between two adjacent vertices



Step2. Adjust the dash length after the recalculation of the number of the drawing units



Step 3. Reset start point and end point to dots

Fig. 4.27 Principle of new drawing method of dash-dotted lines

If adjacent vertices are too closely packed, there will be the same problems as with dashed lines when using the new method. Accordingly, the solution is similar to the one before. If the position of the vertex does not necessarily have to be marked, the traditional method is applied; if we need to mark out all the crowded vertices, we just connect them by a solid line segment. The two control thresholds, “Condensed vertices threshold A” and “Condensed vertices threshold B” that I mentioned in the dashed line section before are also applicable for dash-dotted lines.

4.7 Double line and triple line

Double lines and triple lines are two kinds of multiple lines that are used for the symbolization of roads, railways, boundaries and so on. Each constitutive single line can have the same or different appearances (symmetrical, Fig. 4.28a, b; asymmetrical, Fig. 4.28c, d).

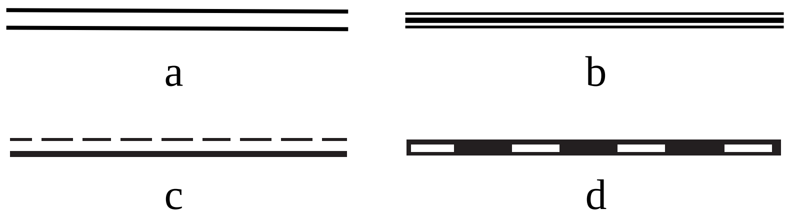


Fig. 4.28 Double lines and triple lines used as map symbols

At present, there is no efficient way to produce multiple lines in commonly used graphics software. With such tools, multiple lines are mostly created by cloning single lines. For example, to draw a double line as in Fig. 4.28a, a single line is first duplicated and shifted so that it is parallel to the original one. For an asymmetrical double line, after duplication, one line is given a different pattern (Fig. 4.29a). To draw a line as shown in Fig 4.28d, two individual lines between the same vertices are drawn. Each one has different attributes; one is above the other (Fig. 4.29b).

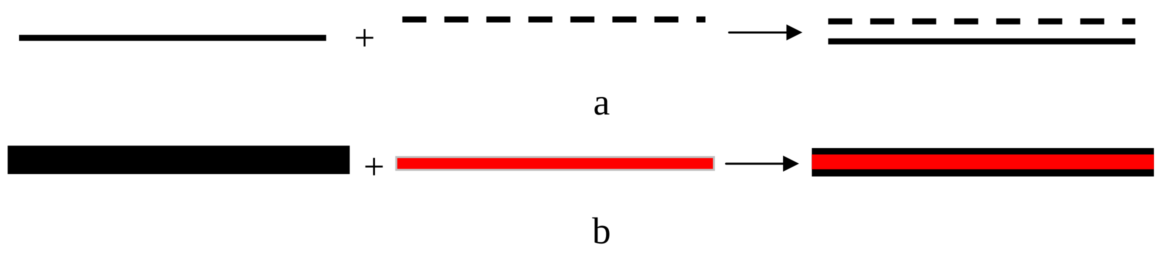
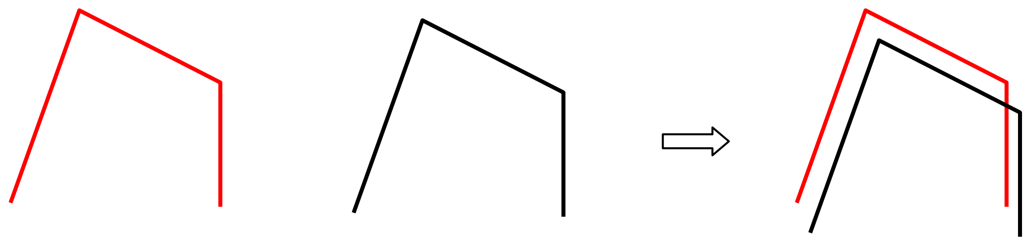
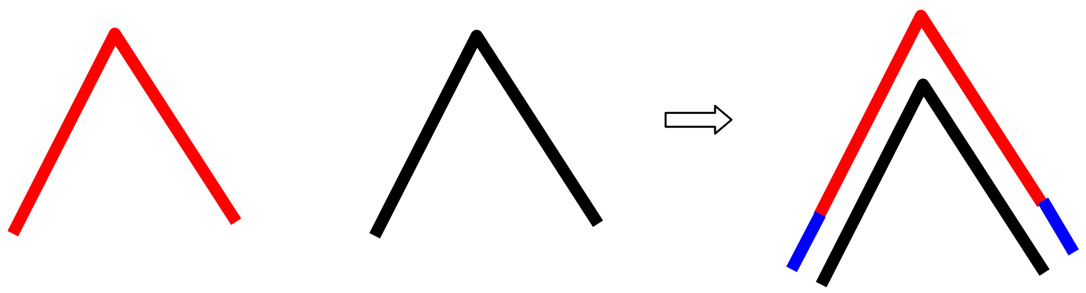


Fig. 4.29 Traditional strategies to draw multiple lines

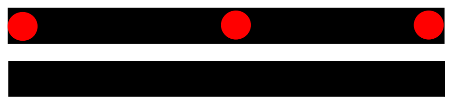
But these methods have some intrinsic problems. For example, most cartographic lines have irregular shapes; it is not possible to draw a double line by means of a simple duplication (Fig. 4.30a). In most cases, the line ends need to be modified (Fig. 4.30b). For multiple lines, only the original line follows the exact positions, the composed line may not reflect the real positions precisely (Fig. 4.30c). Most of these operations, including duplication and applying patterns, have to be done manually; it is difficult to correct all the problems. These problems are even more severe when using curves; untill now, there is no method to draw parallel Bézier curves. Therefore, more efficient automatic approaches should be developed.



a. Curved double lines cannot be created by simple duplication



b. Line ends need to be modified



c. Duplicated lines may deviate from the original path (red dots represent the original vertices)

Fig. 4.30 Some present methods to draw multiple lines

In this work, a new, efficient method is developed to draw multiple lines. The principle is to compute the correct paths of the constitutive parallel lines and render them one after the other. For example, to draw a double line along the three vertices of V_0, V_1, V_2 (Fig. 4.31), four single lines are drawn that are parallel to the path given by $V_0V_1V_2$. The distance between each of the four parallel single lines to the central line (V_0V_1 or V_1V_2) remains constant. At line ends, $V_{1\text{-}0}V_{2\text{-}0}$ is perpendicular to V_0V_1; $V_{1\text{-}2}V_{2\text{-}2}$ is perpendicular to V_1V_2. Note that the length of $V_{1\text{-}0}V_{1\text{-}1}$ is not the same as $V_{2\text{-}0}V_{2\text{-}1}$ and $V_{2\text{-}1}V_{2\text{-}2}$ is different from V_1V_2. At corners, $V_{1\text{-}1}$ is the cross point of two arms ($V_{1\text{-}0}V_{1\text{-}0'}$ and $V_{1\text{-}2}V_{1\text{-}2'}$); $V_{2\text{-}1}$

is the cross point of two arms ($V_{2\text{-}2}V_{2\text{-}2'}$ and $V_{1\text{-}2}V_{1\text{-}2'}$). The crucial point is to compute positions of the 6 vertices from V_0, V_1 and V_2.

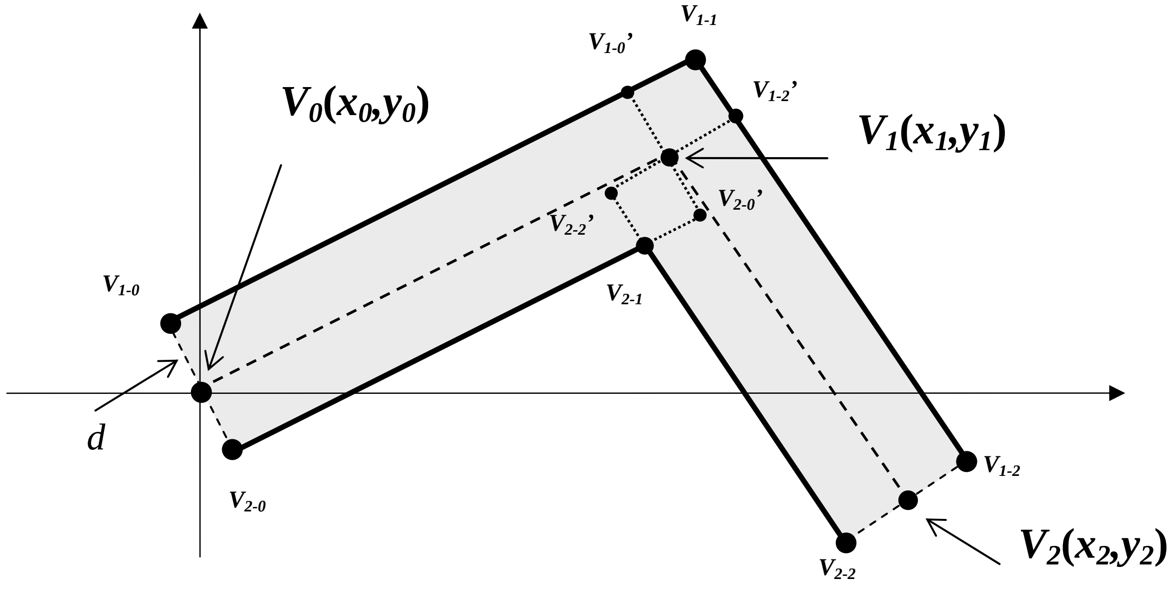


Fig. 4.31 The principle to compute a double line's paths: The thick black lines are to be drawn on a map, The light gray region represents the white line between the two parallel lines

Suppose three vertices: $V_0 = (x_0, y_0)$, $V_1 = (x_1, y_1)$, $V_2 = (x_2, y_2)$; the distance between two parallel lines is $2d$. We can get the positions as:

$$\begin{cases} V_{1-0} = (x_0 - dx_1, y_0 + dy_1) \\ V_{1-2} = (x_2 - dx_2, y_0 - dy_2) \end{cases} \qquad [4\text{-}15]$$

$$\begin{cases} V_{2-0} = (x_0 + dx_1, y_0 - dy_1) \\ V_{2-2} = (x_2 + dx_2, y_0 + dy_2) \end{cases} \qquad [4\text{-}16]$$

Where

$$\begin{cases} x_1 = \dfrac{y_1 - y_0}{\overline{\overline{V_0 V_1}}} \\ y_1 = \dfrac{x_1 - x_0}{\overline{\overline{V_0 V_1}}} \end{cases} \qquad [4\text{-}17]$$

$$\begin{cases} x_2 = \dfrac{y_2 - y_1}{\overline{V_1V_2}} \\ y_2 = \dfrac{x_2 - x_1}{\overline{V_1V_2}} \end{cases} \qquad [4\text{-}18]$$

For the cross point of $V_{1\text{-}1}$, if we can get $V_{1\text{-}0} = (ax,ay)$, $V_{1\text{-}0}' = (bx,by)$, $V_{1\text{-}2} = (cx, cy)$, $V_{1\text{-}2}' = (dx, dy)$,

then $V_{1\text{-}1} = (x, y)$, where

$$\begin{cases} x = ax + \dfrac{(ay-cy)*(dx-cx)-(ax-cx)*(dy-cy)}{(bx-ax)*(dy-cy)-(by-ay)*(dx-cx)}*(bx-ax) \\ \\ y = ay + \dfrac{(ay-cy)*(dx-cx)-(ax-cx)*(dy-cy)}{(bx-ax)*(dy-cy)-(by-ay)*(dx-cx)}*(by-ay) \end{cases} \qquad [4\text{-}19]$$

In this program, the distance between two parallel lines is defined as the distance between their inner edges. If the line width is changed, the paths of the two parallel lines will be recalculated to retain the width of the empty stripe between the two single lines (Fig. 4.32).

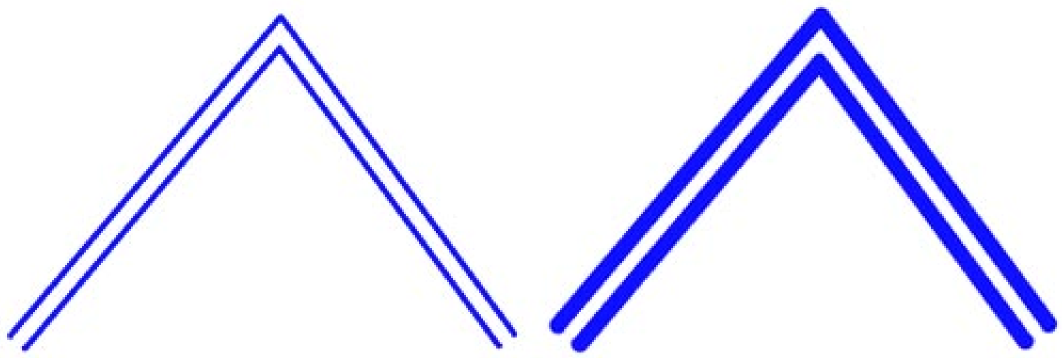


Fig. 4.32 Adjusting the line width of a double line (The distance between two parallel lines is kept constant when the line width is changed.)

If the region between parallel lines is transparent, other symbols underneath will show up. At crosses, one multiple line will cover the other; then we get the following effects (Fig. 4.33).

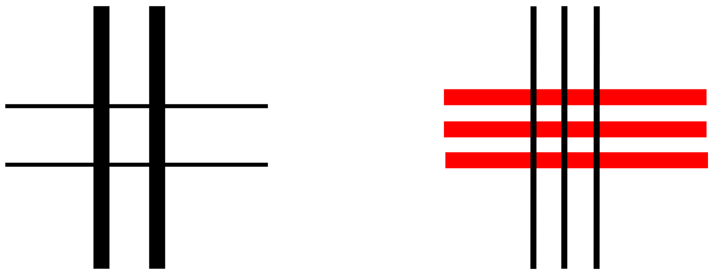


Fig. 4.33 Overlaps at crosses of multiple lines

To get clear display the crosses, additional parallel white lines are added, which are not transparent and will fill the empty region and cover the overlaps at crosses. For example, to draw a triple line along $V_0V_1V_2$, in addition to the three parallel visible single lines, two additional white lines in between will be rendered (gray rectangles shown in Fig. 4.31). The width of each white line is the same as the distance between two parallel lines, which is defined as the distance between inner edges. For a double line, we only need to add one thick white line between two parallel lines. The computation of the paths of the covering white lines is the same as that of the colored parallel lines (Fig. 4.34).

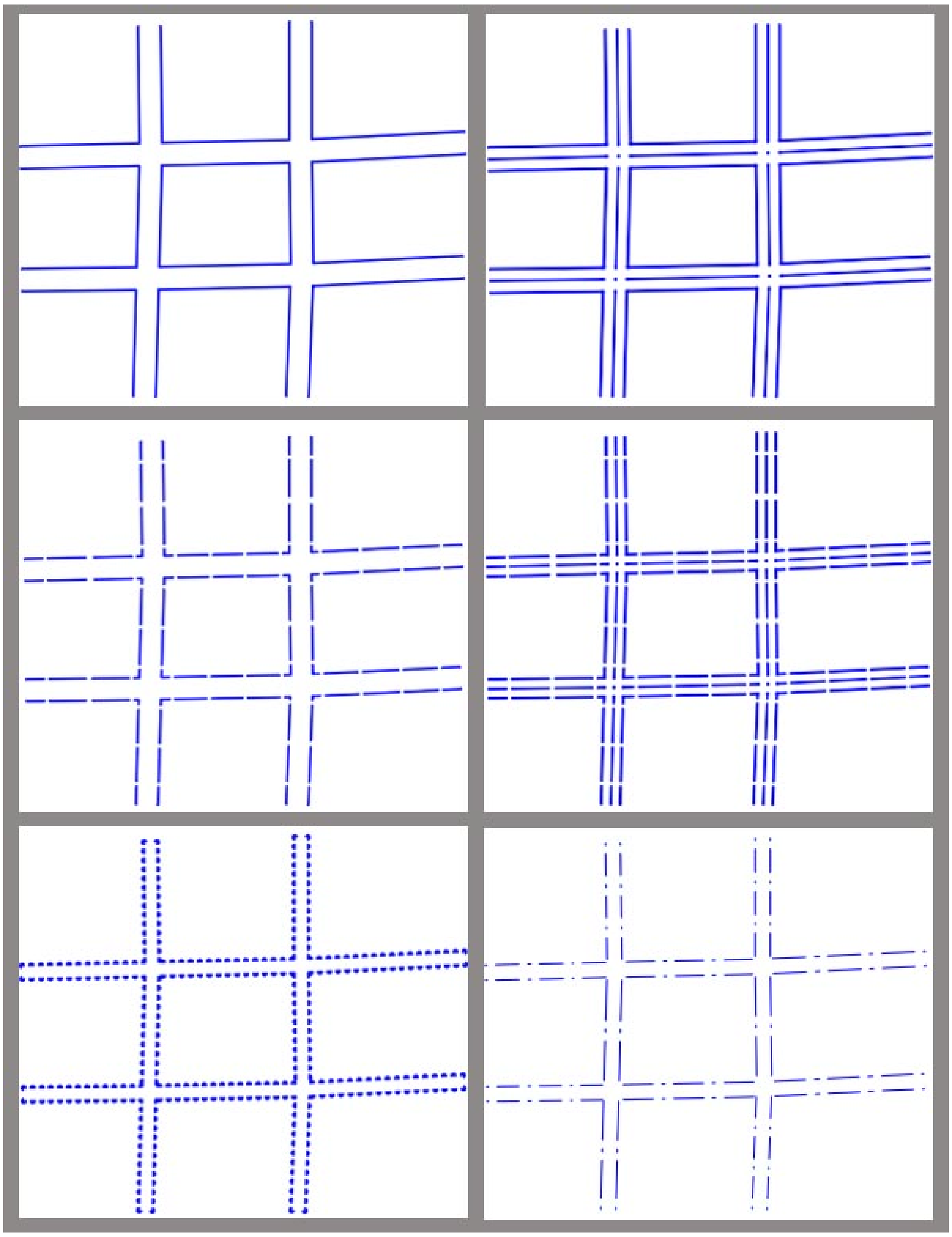


Fig. 4.34 Crosses of multiple lines using the new method

Now we get the paths of all single parallel lines. This method is more efficient and simplifies the traditional approaches. In addition, the paths are computed in such a way that the problems shown in Fig. 4.30 do not turn up.

This method also works to generate double and triple curves. By making some modifications, it can be applied to produce other multiple lines, including dashed lines, dotted lines and dash-dotted lines. For these lines, the program first gets positions of dashes (or dots) along the original vertices and then computes the corresponding positions on the parallel lines. In this way, each constitutive line will have the same patterns except at line turns.

Fig. 4.35 highlights the principle to draw the turn of a double dashed line. The red dashed lines display the real path of a single dashed line along the geometry ($V_0V_1V_2$). First, the strokes of the middle red dashed lines are obtained using the new dashed line drawing method. The strokes of the two parallel lines are computed in a way that they have exactly the same pattern as the middle line except the last dash at turn. As shown in Fig. 4.35, the black and red fragments demonstrate the two parallel dashed lines to be drawn on a map. The outer line

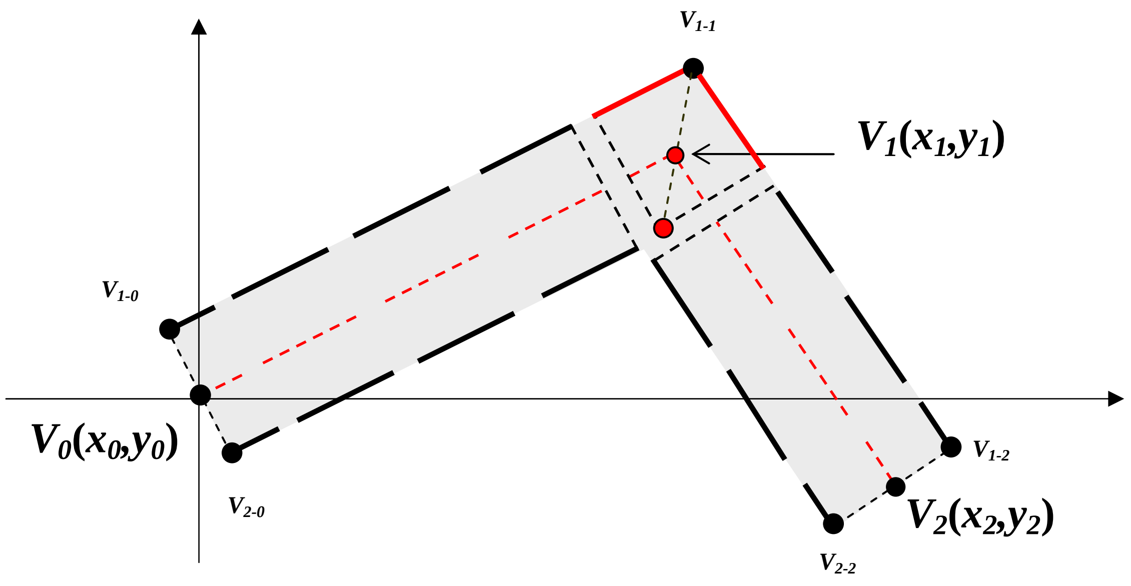


Fig. 4.35 The principle of drawing a double dashed line: V_0, V_1 and V_2 are the three vertices. The black and red lines are to be drawn on the map; the red dashed lines in the middle will not be drawn.

always ends in a fragment longer than a half dash at the turn. For the inner line, it could either end in a gap (Fig. 4.35) or a fragment shorter than a half dash depending on the dash length, gap length and/or the angle of the turn (Fig. 4.36).

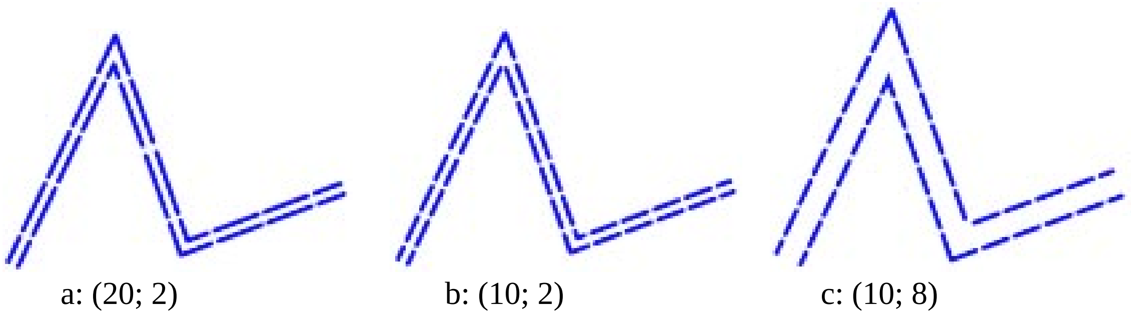


Fig. 4.36 Examples of different results at turns with different dash strokes: The outer line around a turn always ends in a fragment longer than a half dash. The numbers in the parentheses represent the dash length and distance between parallel lines respectively.

Fig. 4.37 shows the principle to draw a double dotted line. The two black dotted lines are parallel to the middle red dotted line, which is obtained using the new dotted line drawing method. At the turn, the outer dotted line always ends in a gap longer than the settings.

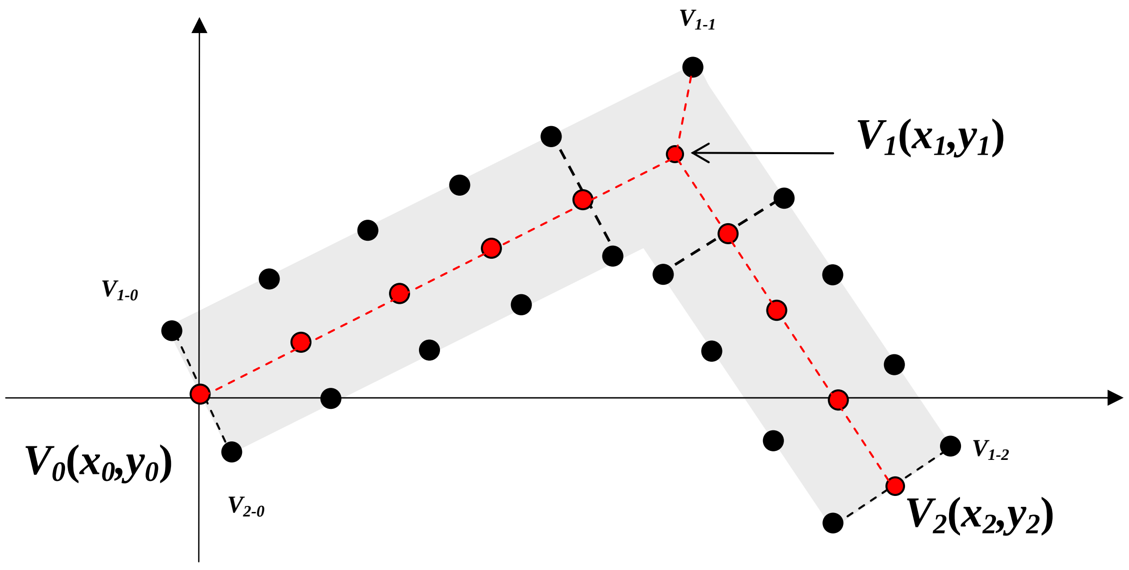


Fig. 4.37 The principle of drawing a double dotted line: The black dots are to be drawn on the map; the red dots in the middle will not be drawn.

Fig. 4.38 shows the strategy to draw a double dash-dotted line. At the turn, the outer line ends in a dash instead of a gap. The inner line may end in a gap, a dot or a dash (like the case in Fig. 4.38). To draw a triple line, the middle red lines shown in Fig. 4.35, Fig. 4.37 and Fig. 4.38 will be shown, their strokes are computed using the new non-solid line drawing method developed in this work. Fig. 4.39 shows some examples of multiple lines with gaps using the new methods. The brown dots represent the vertices.

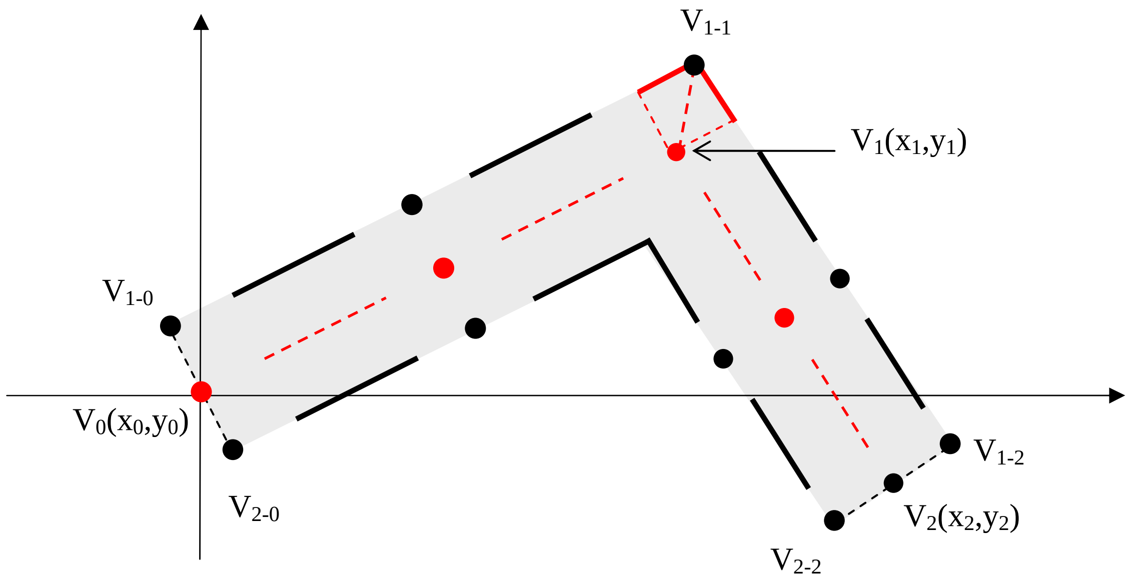


Fig. 4.38 The principle of drawing a double dash-dotted line

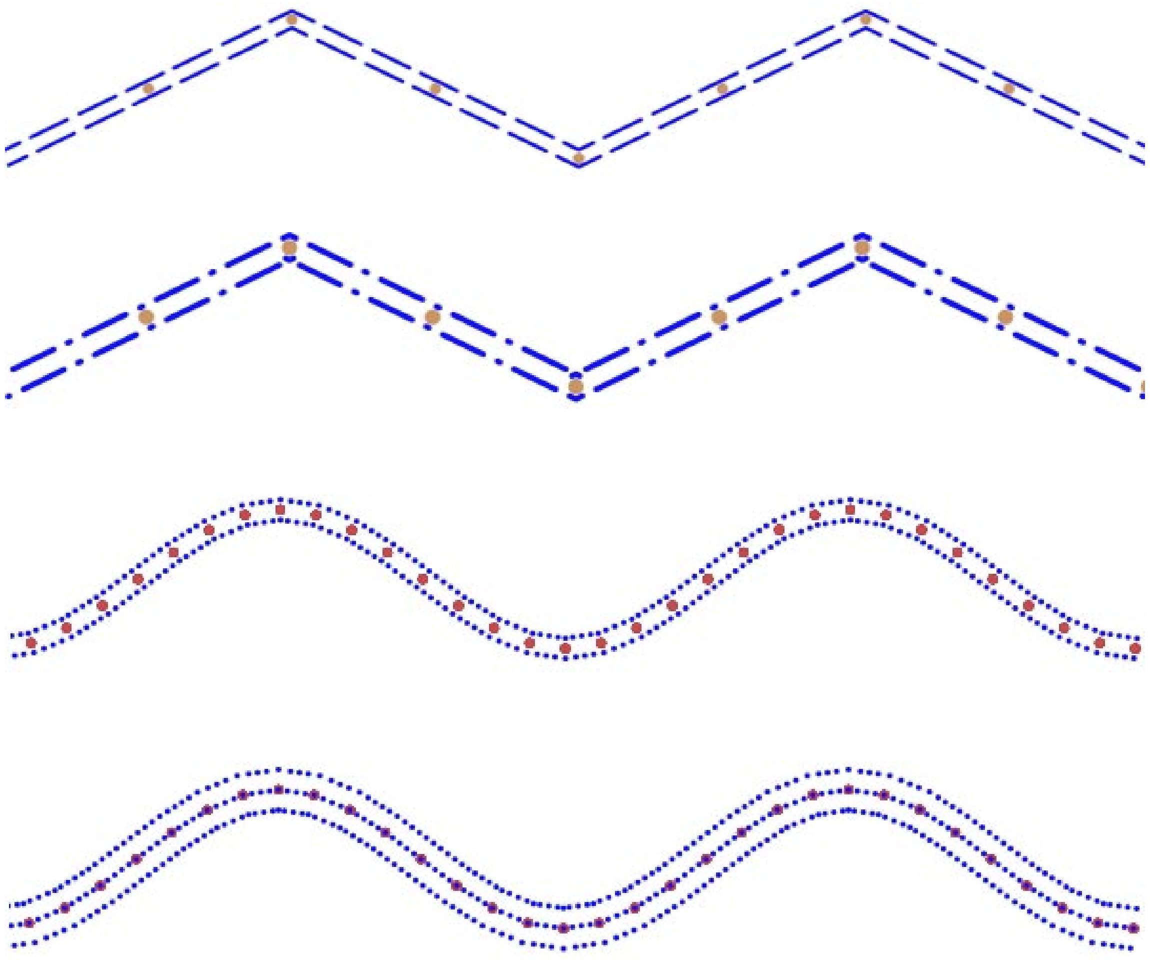


Fig. 4.39 Multiple lines created using new methods

Brown dots show the vertices

Chapter 5 Applications of SmartLine

5.1 Overview of the editorial functions provided by SmartLine

In this chapter, the application of SmartLine is introduced. For each line style, the applicable editorial functions are explained and examples are presented.

One of the powerful tools provided by SmartLine is the flexible editorial platform to control the line appearance. Because it is impossible to meet all cartographic principles with a single automatic solution, the ability to make flexible controls is indispensable for users to optimize the results. Compared with present graphics programs, many new parameters specifying line styles are introduced in SmartLine, such as dash length, gap length, dot size and so on. Combined with the new line drawing algorithms, it is now much easier to obtain line symbols satisfying cartographic rules. In addition, by applying appropriate operations on the versatile editorial functions, we can generate numerous kinds of line patterns. An overview of these editorial parameters is summarized in table 5.1.

	Solid line	Dashed line	Dotted line	Dash-dotted line	Double/Tripe line	Some complex lines
Width	+	+	+	+	+	+
Colour	+	+	+	+	+	+
Cap	+	+	+	+	+	-
Joint	+	+	-	-	+	-
Miter joint limit	+	+	-	-	+	-
Dash length	-	+	-	+	+	+
Gap length	-	+	+	+	+	+
Condense vertices	-	+	-	-	+	+
Smooth value (curve)	+	+	+	+	+	+
Distance between parallel lines	-	-	-	-	+	-
Symmetry of parallel lines	-	-	-	-	+	-
Texture mapping	+	-	+	-	-	-
Pan	+	+	+	+	+	+
Zoom	+	+	+	+	+	+
Edit geometry	+	+	+	+	+	+
Multiple layers	+	+	+	+	+	+

Table 5.1 Editorial parameters provided by SmartLine. “+”: Applicable; “-“: not applicable

5.2 Solid line

Color and width

In Smartline, line color and line width can be adjusted freely through sliders. The line color is controlled in RGB mode. The interface is shown in Fig. 5.1.

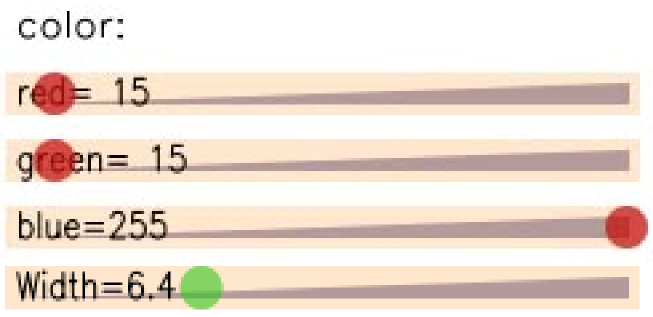


Fig. 5.1 Interface for line color and line width

Line cap

Line ends can be modified by adding one of three kinds of line caps, round, flat and square, which can be selected from the “Feature Control” menu (Fig. 5.2). This setting can also be applied to dashed lines and dash-dotted lines, where caps are added at the ends of each dash.

Line joint

Line joints can either be of miter, round, or bevel type. The control interface and the resulting effects are shown in Fig. 5.3.

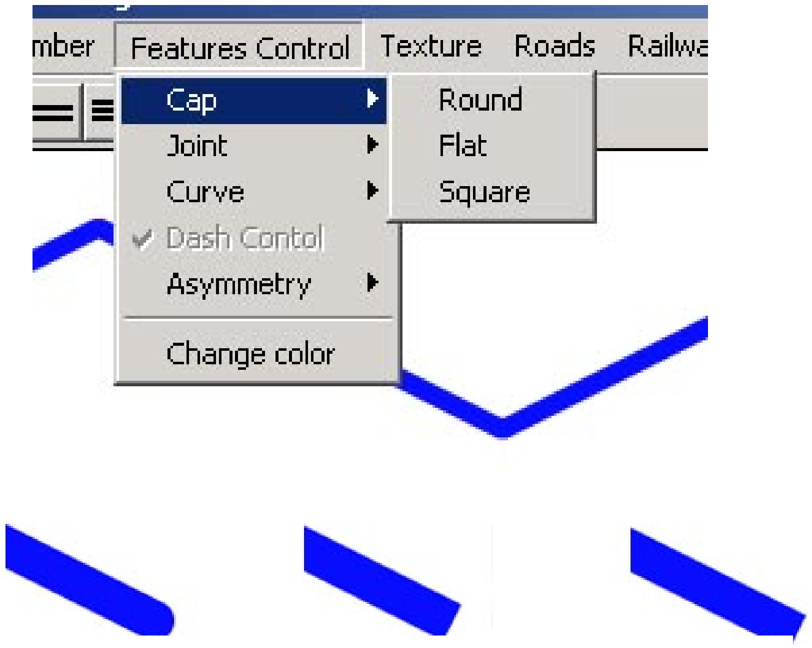


Fig. 5.2 Three different kinds of line caps from the "Features Control" menu

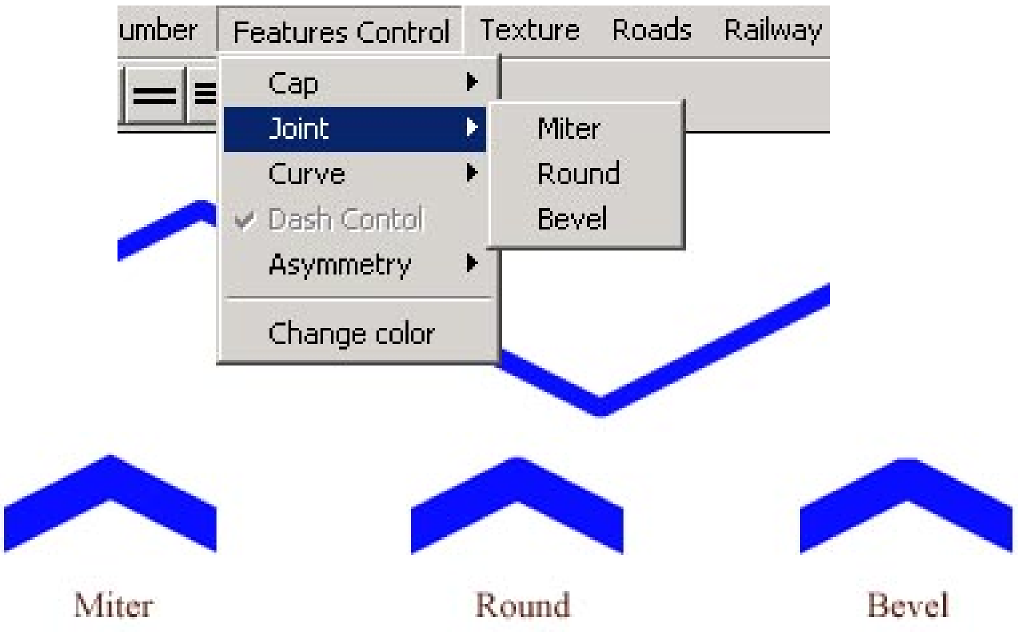


Fig. 5.3 Three different line joints and the control interface

Multiple solid lines

SmartLine provides efficient methods to draw multiple lines. Three icons are used to indicate single lines, double lines, and triple lines (Fig. 5.4). Several functions are available to control the appearance of multiple lines.

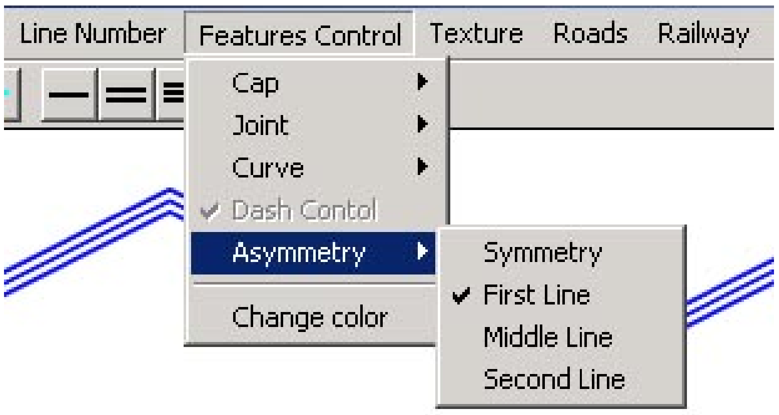
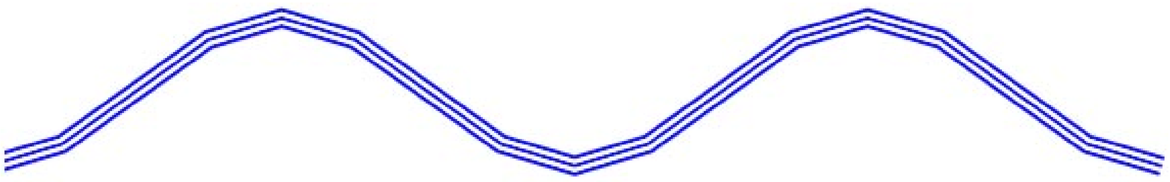
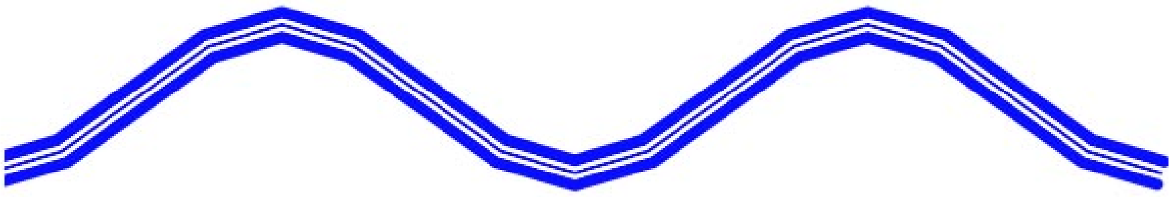


Fig. 5.4 Symmetry control for multiple lines

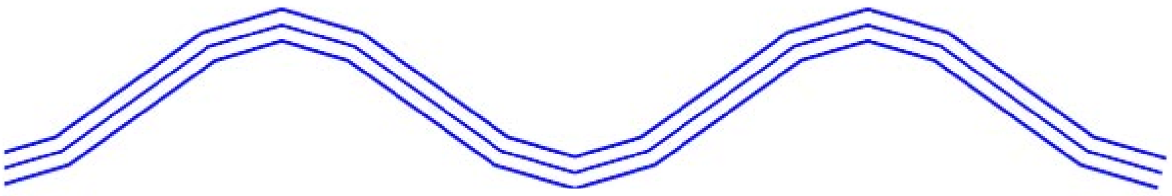
With traditional methods, the graphics information of a multiple line is defined for all the constitutive parallel single lines. There is one single setting that affects all the parallel lines. SmartLine allows us to modify the visual representations of each single line individually. The control menu is shown in Fig. 5.4. If it is set to “Symmetry”, we change the features for all parallel lines at once. Or we can make changes of line features individually by selecting a single line that we want to modify. Another control parameter for multiple lines is the distance between parallel lines, which can be adjusted by a slider control. These control functions are also applicable to multiple dashed, dotted or dash-dotted lines.



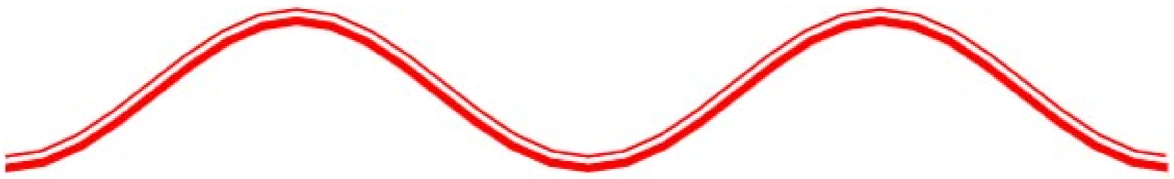
Width=2.0; Distance between lines=2



Width=6.0 (first and second line) and 2.0 (middle line); Distance between lines=2



Width=2.0; Distance between lines=6



First line width=2.0; Second line width=4.7; Distance between lines=2

Fig. 5.5 Examples of multiple solid lines using the new drawing method

Such a flexible editorial environment makes it possible to design many different line styles. Fig. 5.5 shows some examples of multiple solid lines with different settings. The jogging paths around ETH Hoenggerberg are symbolized using a double solid line (Fig. 5.6).

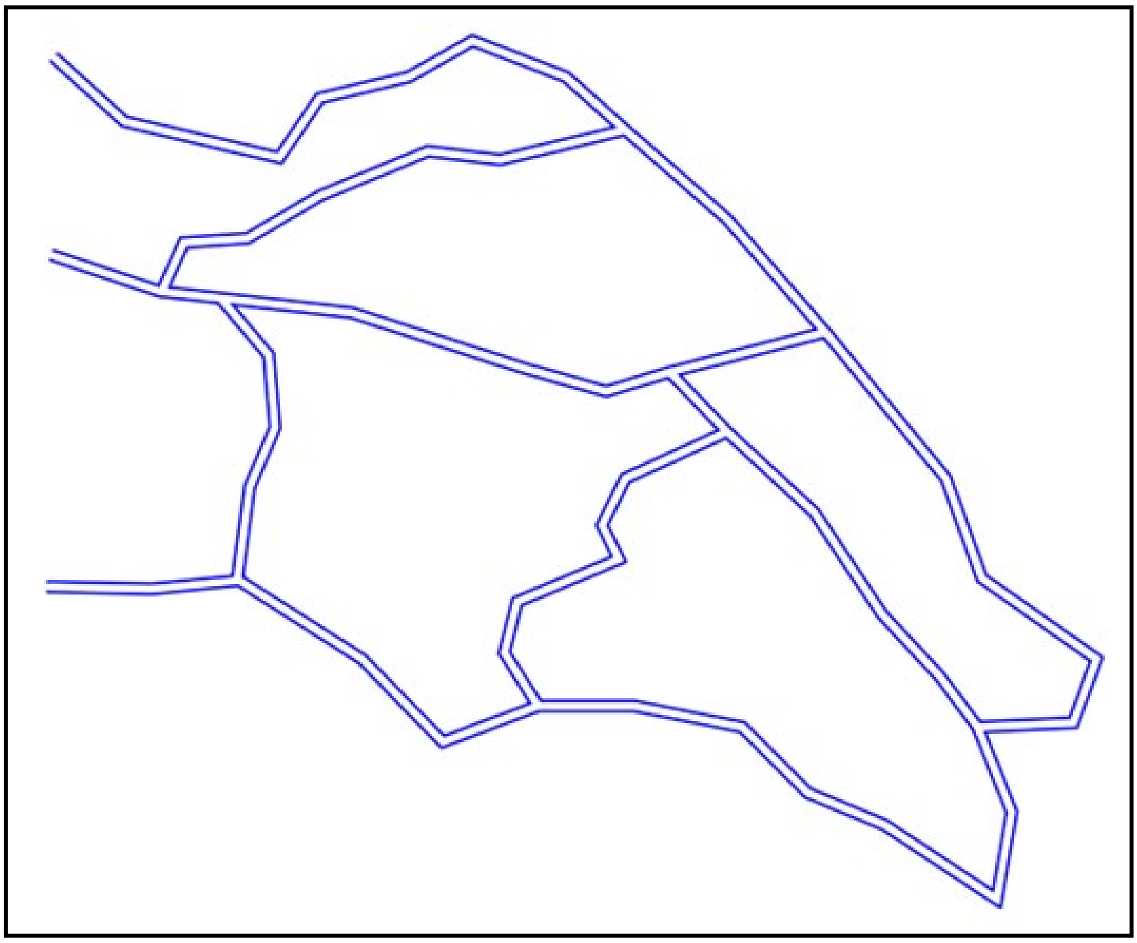


Fig. 5.6 Jogging paths around ETH Hoenggerberg using double solid line style

5.3 Dashed line

In SmartLine, the width of a dashed line is defined as the height of a dash (*d*); the gap length (*l*) is defined as the distance between the edges of two adjacent rectangles (Fig. 5.7). We can add caps at the ends of each dash. When we adjust the line width, the cap size will be changed simultaneously. For round and square caps, the gap length will be reduced with thicker lines. In such cases, we need to increase the gap length to keep the line pattern visually constant.

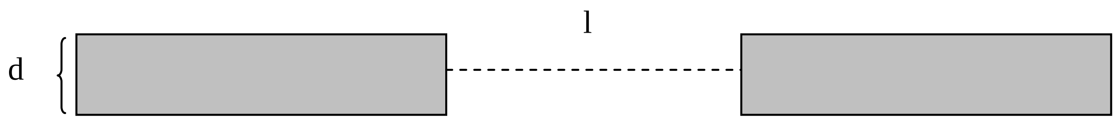


Fig. 5.7 Definition of line width and gap length of a dashed lines in this work

Many graphics programs only provide limited dashed line patterns; the dash array and dash phase cannot be easily adjusted by users. Therefore, it is difficult to make delicate adjustments of the line pattern. SmartLine allows map designers to give a flexible control over dashed line patterns.

The dash phase is set internally to half a dash; the dash array (dash lengths and gap lengths) can be controlled separately by users (Fig. 5.8). If the gap length is set to be 0, we obtain a similar effect with solid lines. For multiple dashed lines, we can control the line features for each single line individually. The advantage of such flexible controls is that we can perform a coordinate adjustment, which is especially important when there are intersections between different lines and other symbols. It also enables us to create many different patterns. Fig. 5.9 shows some examples of dashed lines using the new dashed line drawing method with different settings; each line has solids at turns with balanced dashes.

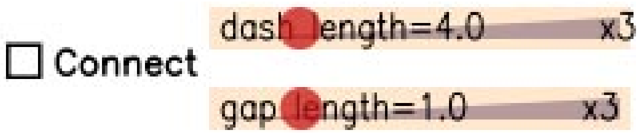


Fig. 5.8 Two control sliders for dashed lines

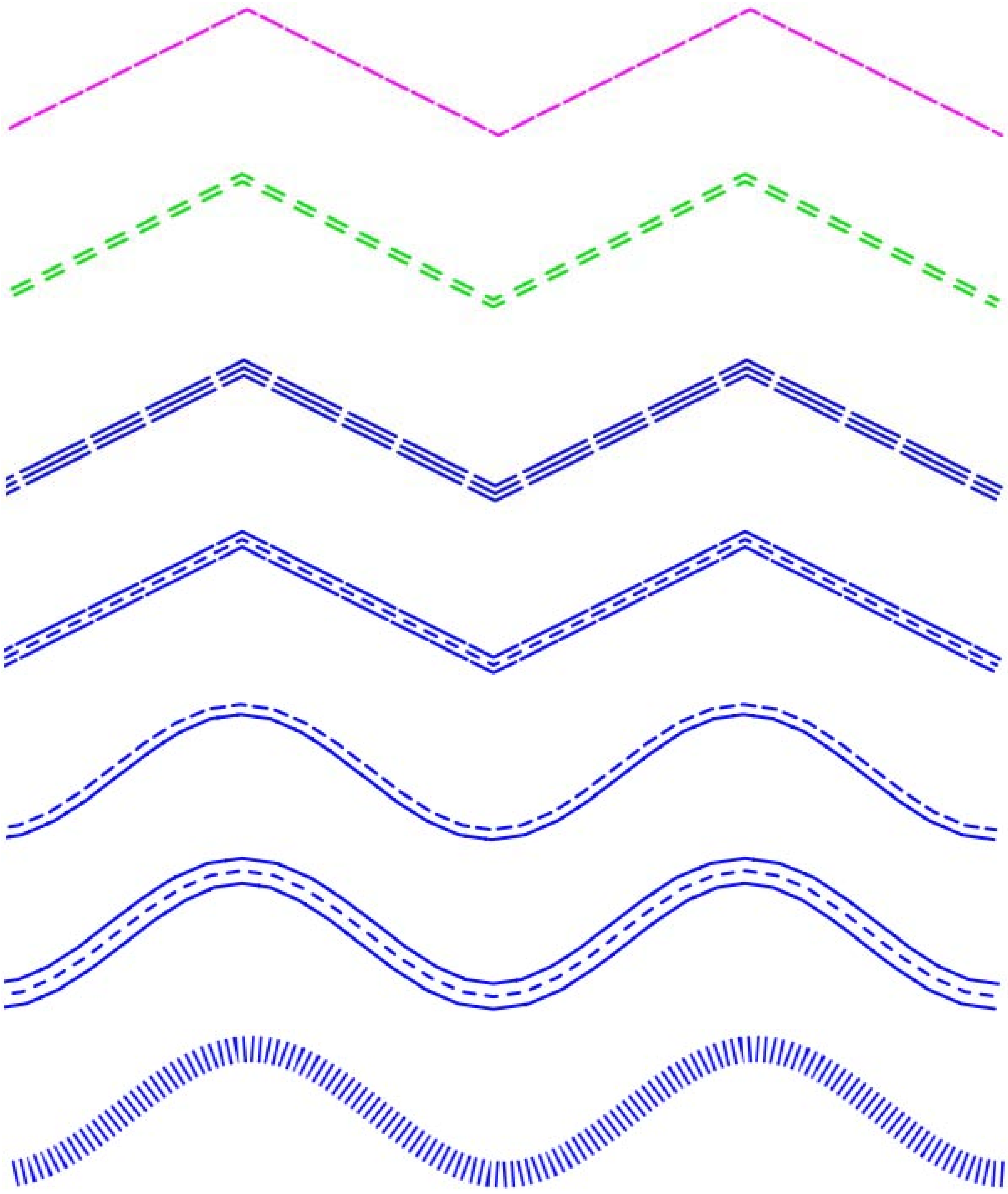


Fig. 5.9 Some dashed lines with different settings

An application of the new dashed line drawing method is shown in Fig. 5.10, which represents the jogging path around ETH Hoenggerberg. It is created automatically using SmartLine without any manual modification (Note the solids at corners and intersections).

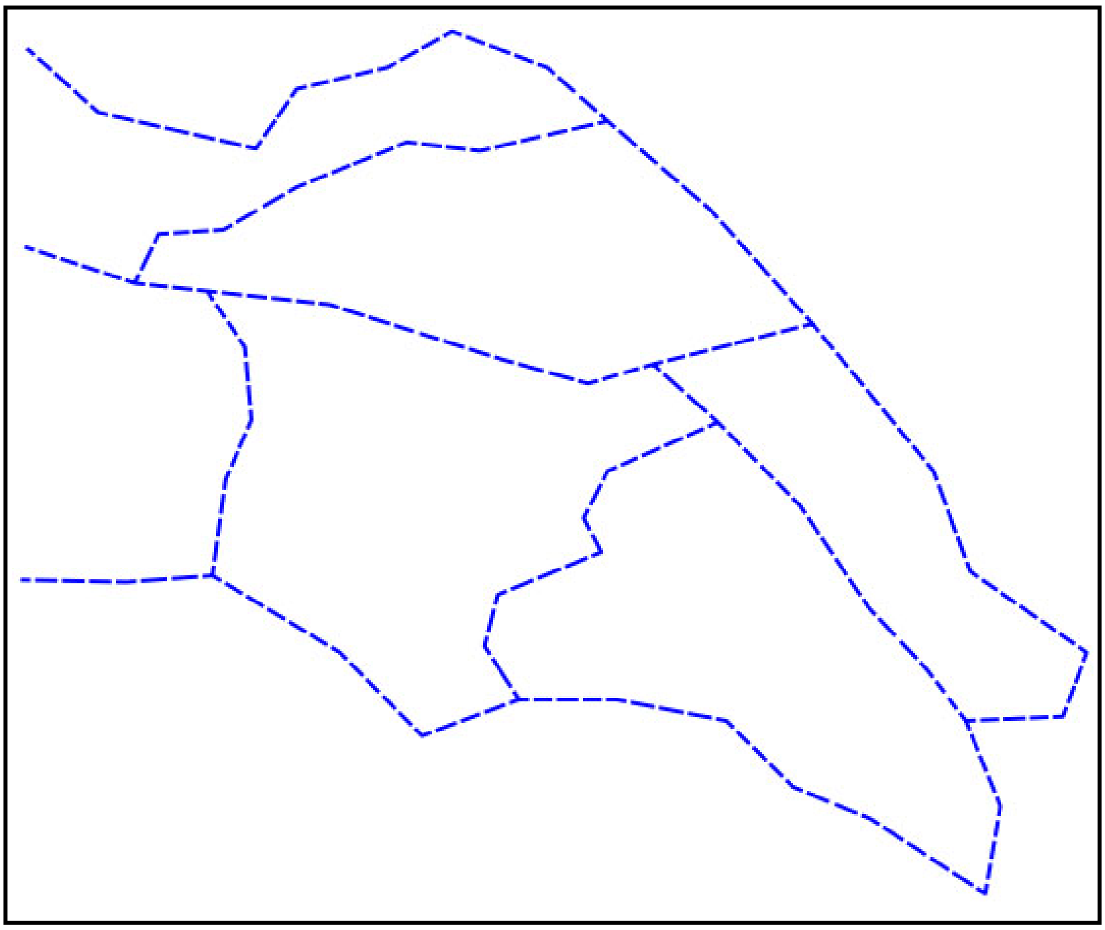


Fig. 5.10 Dashed Jogging paths around ETH Hoenggerberg using SmartLine

5.4 Dotted line and dash-dotted line

A dotted line is drawn as a sequence of circles along a line. The line width is defined as the diameter of each circle (*d*); the gap length (*l*) is the distance between the centers of two adjacent circles (Fig. 5.11). The new method creates dotted lines with dots at turns flanked by two gaps.

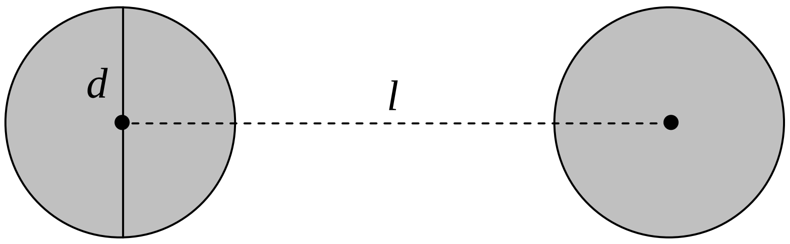


Fig. 5.11 Definition of line width and gap length of a dotted lines in this work

With the new dash-dotted line drawing method, each dot is constructed by a thin rectangle (1.5 pixels in length as default setting) plus two round caps. The line width is represented by the dash height (*d*); the gap length (*l*) is the distance between the edges of a long dash and the following dot (Fig. 5.12). When we adjust the line width, the cap size will be adjusted proportionally. A dash-dotted line drawn with the new method has dots at turns flanked by two roughly equally-sized gaps.

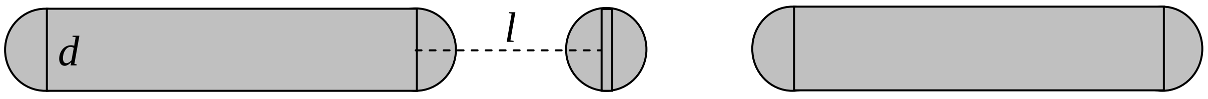


Fig. 5.12 Definition of line width and gap length of a dash-dotted line in this work

The control parameters of the dotted and the dash-dotted line can also be adjusted individually. Then, it is possible to design many different patterns (Fig. 5.13). Fig. 5.14 shows the jogging path around ETH Hoenggerberg as dotted and dash-dotted lines. Note that the turns and intersections are drawn as dots with roughly balanced gaps at these key positions.

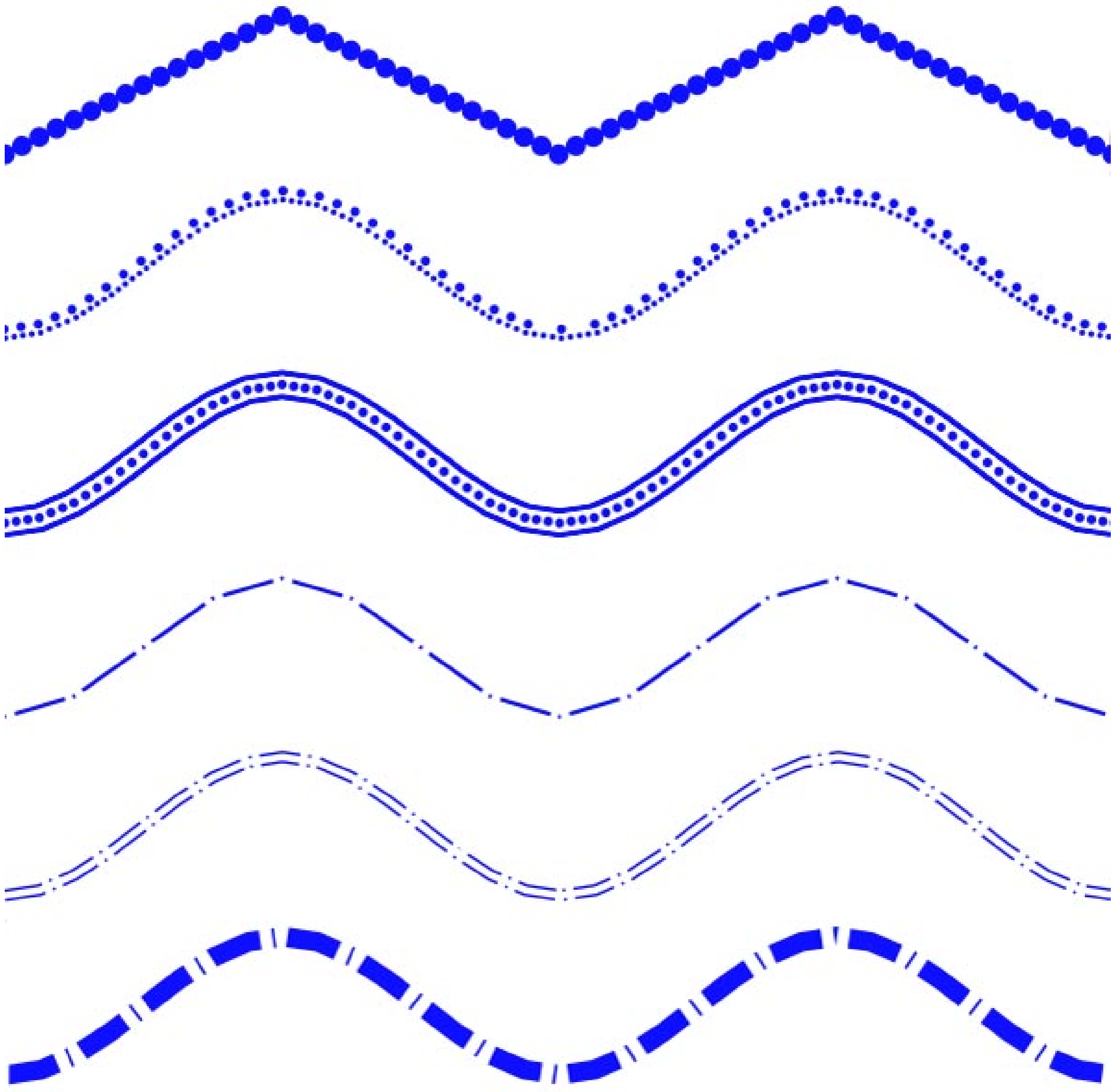


Fig. 5.13 Examples using the new methods for dotted and dash-dotted lines

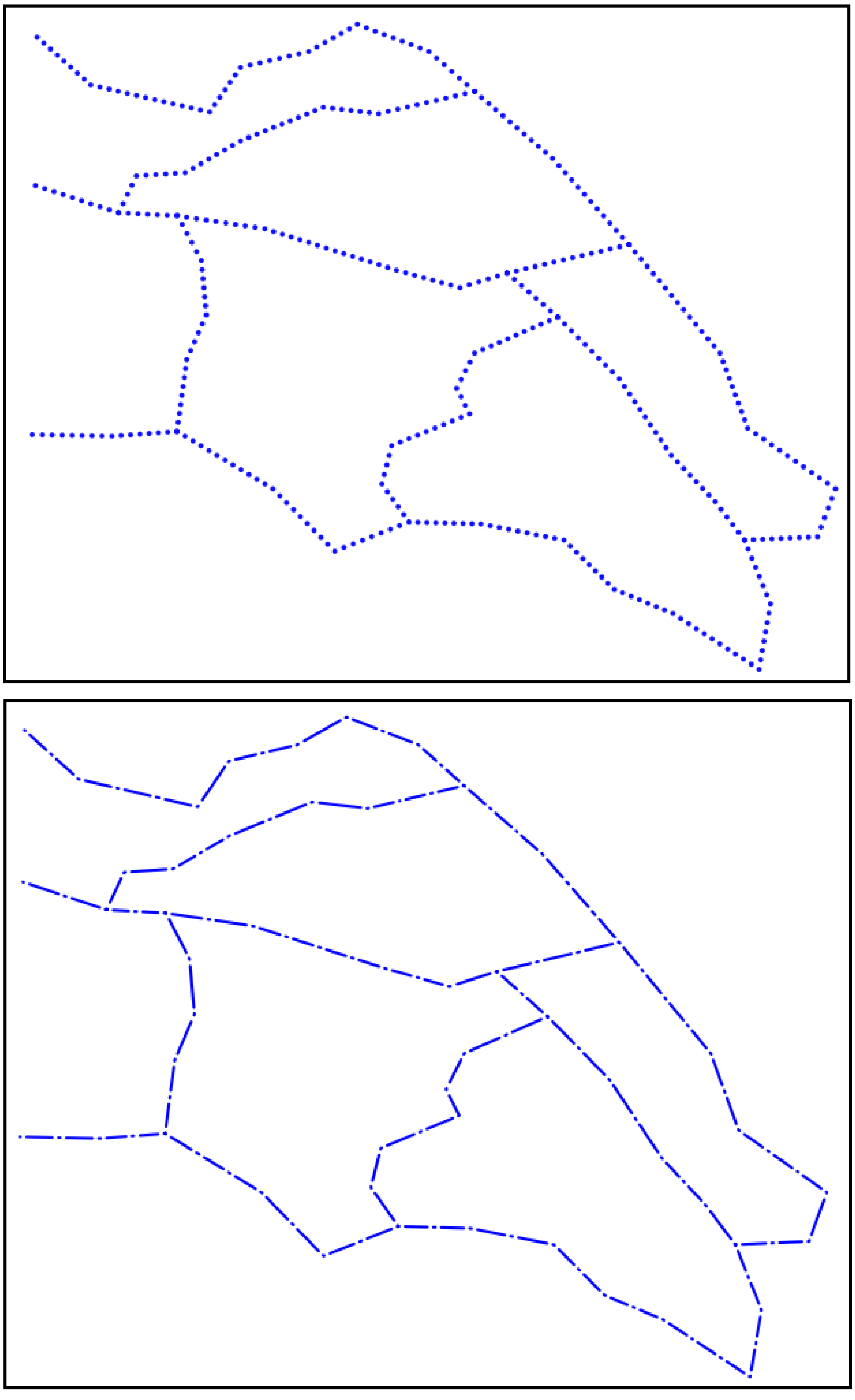


Fig. 5.14 Examples using new drawing methods of dotted line and dash-dotted line

5.5 Using new line drawing methods to generate some commonly used complex lines

We can produce some complex lines that are commonly used as map symbols by introducing some modifications to the new drawing methods. In the following, some line symbols from the Swiss National Map Series created with SmartLine are presented.

5.5.1 Based on the dashed line drawing method

We can generate the following complex line symbols efficiently by adding some modifications to the new dashed line drawing method (Fig. 5.15).

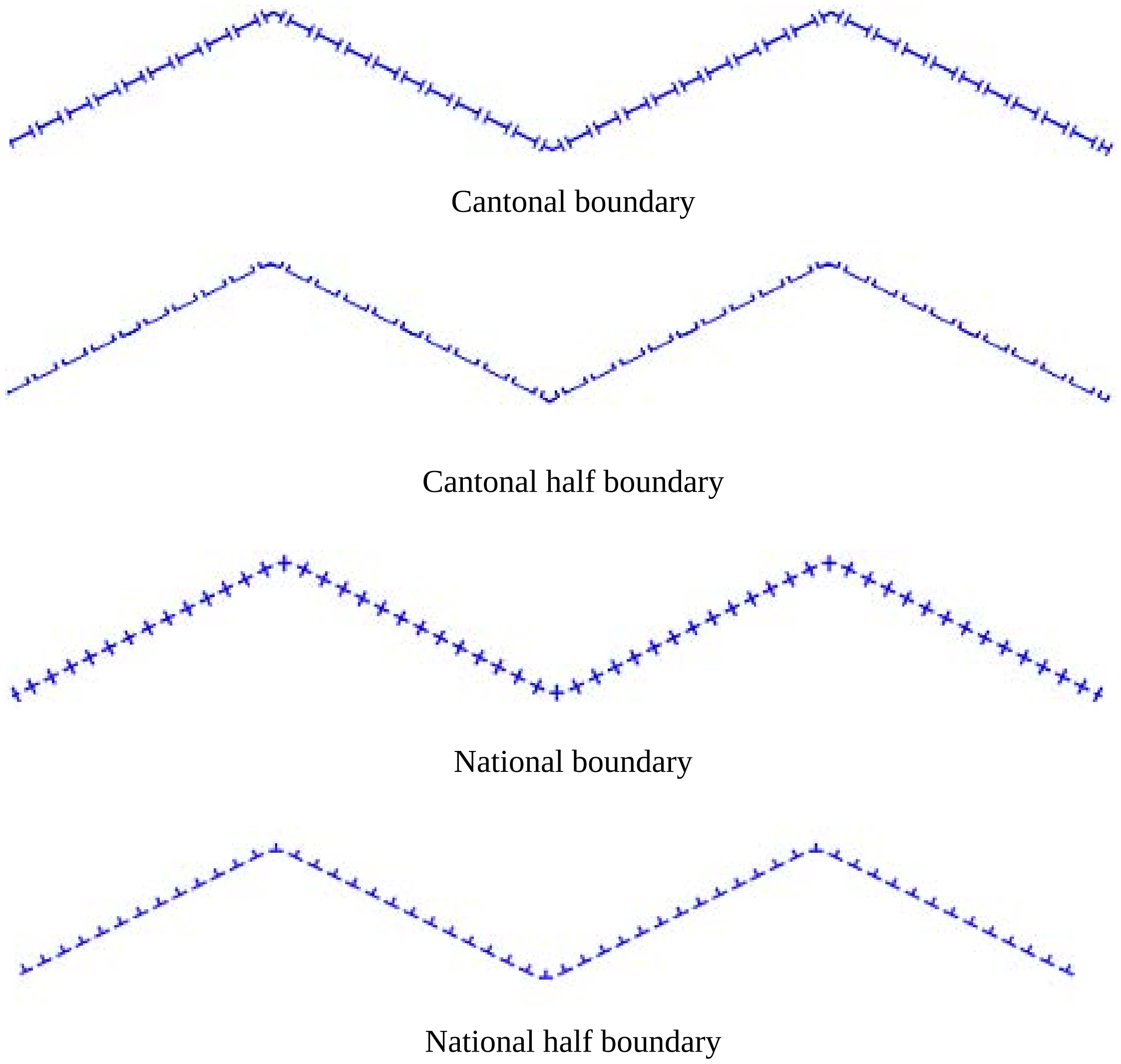


Fig. 5.15 Some boundaries used in the Swiss National Map Services

The method to draw cantonal boundaries or cantonal half boundaries is a modified version of drawing square caps. Fig. 5.16 shows the diagram to draw a square cap. V_0 and V_1 is a pair of vertices at dash ends. The rectangles of $C_{00}C_{00}'C_{01}C_{01}'$ and $C_{10}C_{10}'C_{11}C_{11}'$ make up two square caps that we can add at each dash ends. The width of each rectangle is the same as the dash width. Based on the positions of V_0, V_1 and the line width, we can get the positions of the two square caps. For flat caps, we only need to compute the positions of the rectangle given by $C_{00}C_{01}C_{11}C_{10}$. For round caps, we need to add half a circle at each dash end.

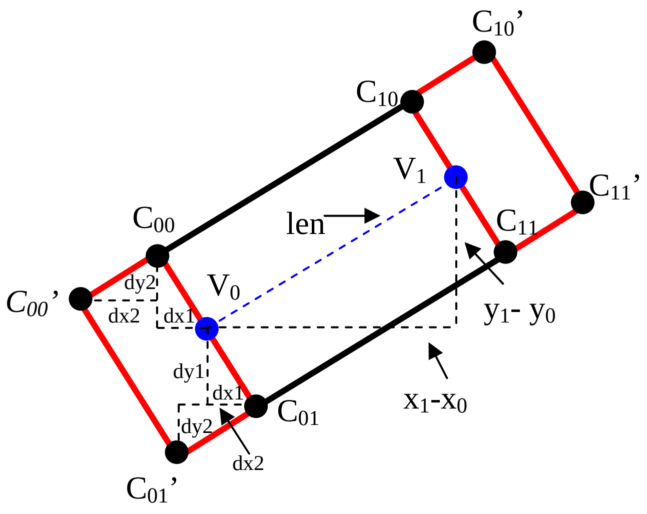


Fig. 5.16 Schematic representation of a square cap

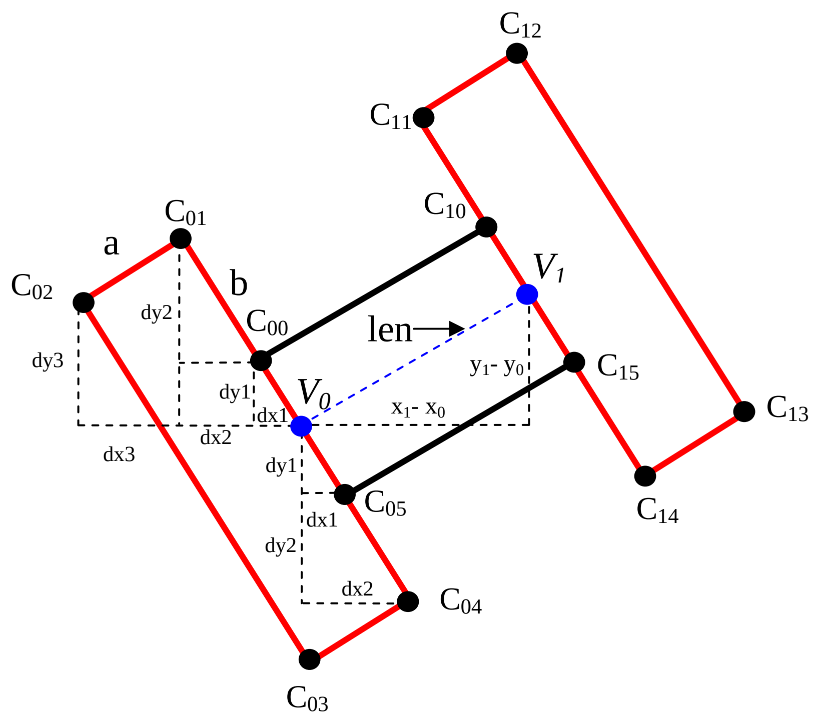


Fig. 5.17 Schematic representation of a component of a cantonal boundary

To draw a dashed symbol of cantonal boundary, we can add a special square cap, in which the width of a square cap is larger than the width of the dash (Fig. 5.17)

The positions of the larger square cap are calculated as follows:

If we set

$$\begin{cases} dx_2 = a * dy_1 \\ dy_2 = a * dx_1 \end{cases} \qquad [5\text{-}1]$$

$$\begin{cases} dx_3 = (line\_width + b) * (y_1 - y_0) / len \\ dy_3 = (line\_width + b) * (x_1 - x_0) / len \end{cases} \qquad [5\text{-}2]$$

Here, *a* and *b* specify the size of the square, where *a* determines the length between C_{01} and C_{02}, and *b* is the length between C_{00} and C_{01} (Fig.5.17).

Then we get

$$\begin{cases} C_{00} = (x_0 - dx_1 + dx_2, y_0 + dy_1 + dy_2) \\ C_{01} = (x_0 - dx_3 + dx_2, y_0 + dy_3 + dy_2) \\ C_{02} = (x_0 - dx_3, y_0 + dy_3) \\ C_{03} = (x_0 + dx_3, y_0 - dy_3) \\ C_{04} = (x_0 + dx_3 + dx_2, y_0 - dy_3 + dy_2) \\ C_{05} = (x_0 + dx_1 + dx_2, y_0 - dy_1 + dy_2) \end{cases} \quad [5\text{-}3]$$

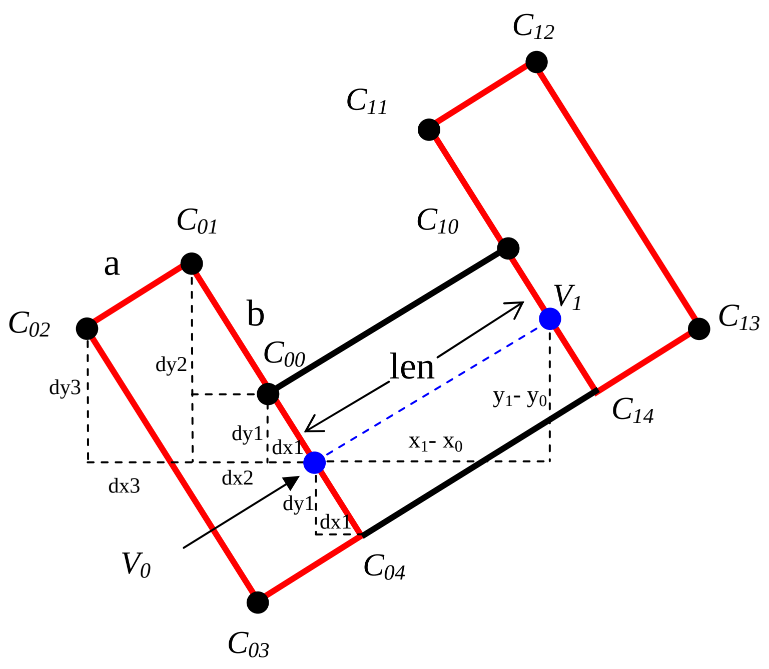


Fig. 5.18 Schematic representation of a component to draw a cantonal half boundary

The half cantonal boundary is drawn as a half square (Fig. 5.18). We can get:

$$\begin{cases} C_{00} = (x_0 - dx_1 + dx_2, y_0 + dy_1 + dy_2) \\ C_{01} = (x_0 - dx_3 + dx_2, y_0 + dy_3 + dy_2) \\ C_{02} = (x_0 - dx_3, y_0 + dy_3) \\ C_{05} = (x_0 + dx_1 - dy_2, y_0 - dy_1 - dx_1) \end{cases} \quad [5\text{-}4]$$

Fig. 5.19 shows the symbols of the cantonal boundary using both the traditional method and the new method. Compared with the traditional method, there are several places that are better displayed using the new method: 1. It starts and ends with the same unit which is half a dash; 2. It has roughly balanced arms at turns; 3. The geometries of sharp turns are clearly shown.

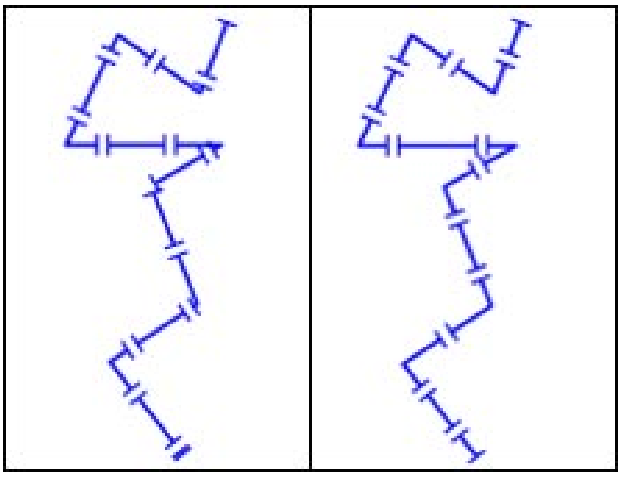


Fig. 5.19 Comparison of the traditional method with new method to draw a complex line

The method to draw a national boundary is also derived from the new dashed line drawing method. The drawing unit is a cross. To draw a cross, we need to compute the positions of twelve points (Fig. 5.20).

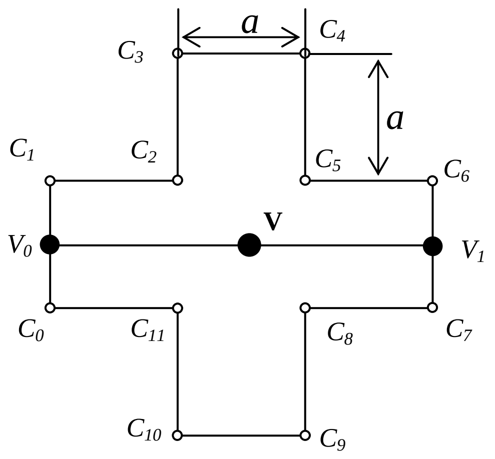


Fig. 5.20 Schematic representation to draw a component of a national boundary

$\overline{V_0V_1}$ is one dash, if $V(x,y)$ is the center point between V_0 and V_1, the position is:

$$\begin{cases} x = 0.5*(x_1 + x_0) \\ y = 0.5*(y_1 + y_0) \end{cases} \qquad [5\text{-}5]$$

The positions of the twelve points describing a cross are:

$$\begin{cases} C_0 = (x_0 + dx_1, y_0 - dy_1); & C_1 = (x_0 - dx_1, y_0 + dy_1) \\ C_2 = (x - dx_1 - dx_2, y + dy_1 - dy_2); & C_3 = (x - dx_3 - dx_2, y + dy_3 - dy_2) \\ C_4 = (x - dx_3 + dx_2, y + dy_3 + dy_2); & C_5 = (x - dx_1 + dx_2, y + dy_1 + dy_2) \\ C_6 = (x - dx_1, y + dy_1); & C_7 = (x + dx_1, y - dy_1) \\ C_8 = (x + dx_1 + dx_2, y - dy_1 + dy_2); & C_9 = (x + dx_3 + dx_2, y - dy_3 + dy_2) \\ C_{10} = (x + dx_3 - dx_2, y - dy_3 - dy_2); & C_{11} = \end{cases}$$

$$(x + dx_1 - dx_2, y - dy_1 - dy_2)$$

Where

$$dx_1 = line\_width*(y_1 - y_0)/\overline{V_0V_1}$$
$$dy_1 = line\_width*(x_1 - x_0)/\overline{V_0V_1}$$

$$dx_2 = dy_1$$
$$dy_2 = dx_1 \qquad [5\text{-}6]$$

$$dx_3 = a*line\_width*(y_1 - y_0)/\overline{V_0V_1}$$
$$dy_3 = a*line\_width*(x_1 - x_0)/\overline{V_0V_1}$$

The coefficient *A* determines the height of the cross and the "*width*" is determined by the dash length.

Similarly, we can obtain the symbol of the national half boundary by computing the positions of eight points (Fig. 5.21)

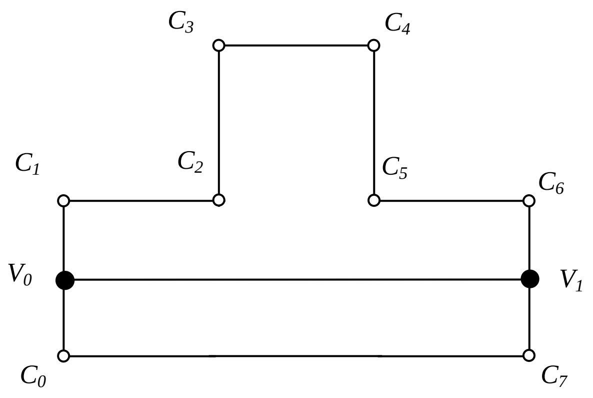


Fig. 5.21 Schematic representation to draw a component of a national half boundary

5.5.2 Based on the multiple line drawing method

Since the features of each parallel line can be adjusted separately, we can create many different line styles. Fig. 5.22 shows some multiple linear symbols used in the Swiss National Map Series drawn by SmartLine (Fig. 5.22).

With appropriate settings, we can also generate some complex lines that are commonly used as map symbols. For example, the narrow single track railroad shown in Fig. 5.23 is drawn as a triple dashed line. The first and second parallel lines have no gaps (gap length = 0); the middle line is thick and has long gap lengths (Fig. 5.23). The narrow double track railroad is drawn as a triple dash-dotted line; the first and second line has no gaps. The freight railroad is also drawn as a triple dashed line. Compared with narrow single track, the middle line has long gaps and short dashes. Similarly, we can generate the tramway by reducing both the gap length and dash length of the middle line.

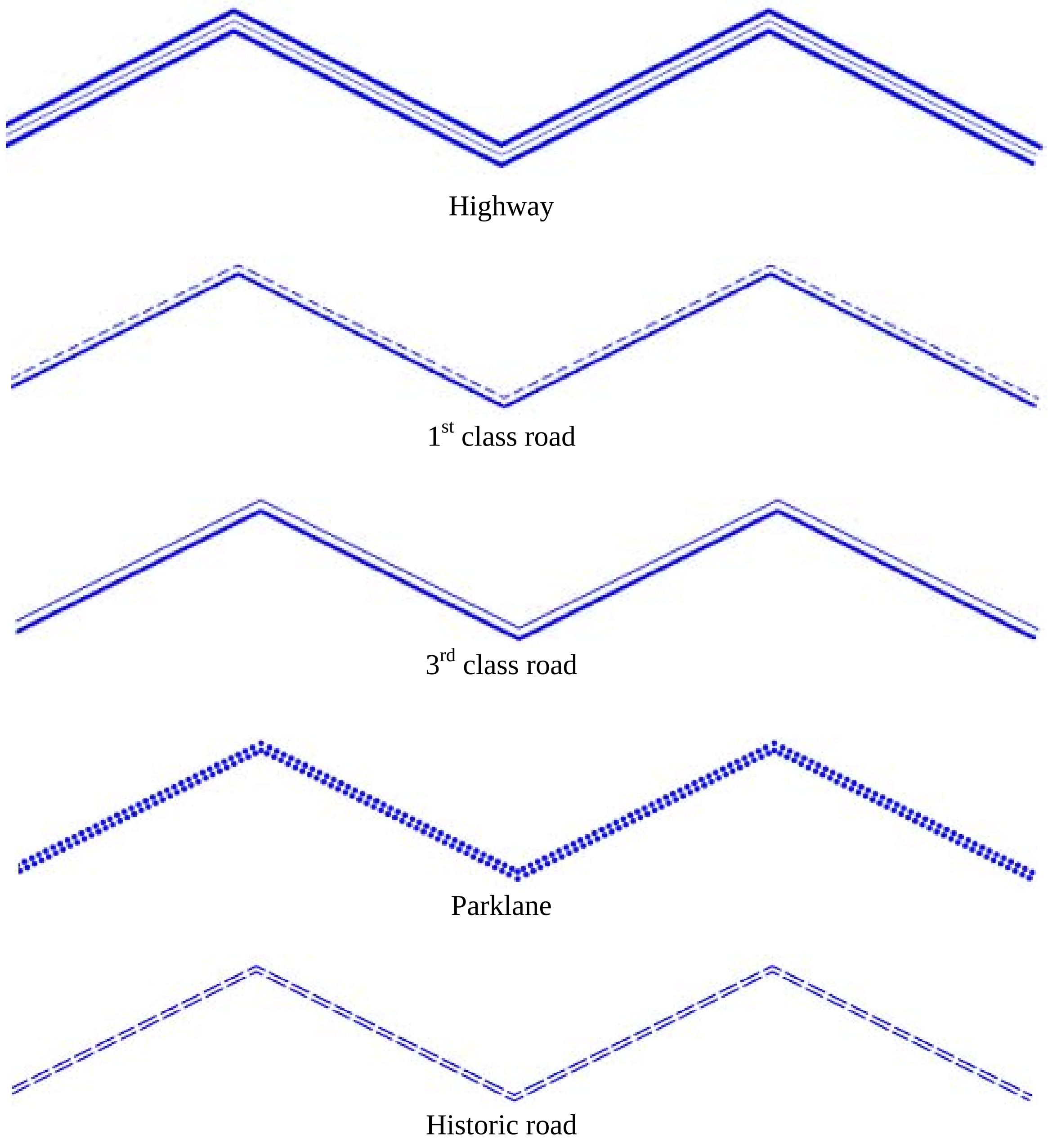


Fig. 5.22 Some complex linear symbols used in the Swiss National Map Series created by the new drawing methods

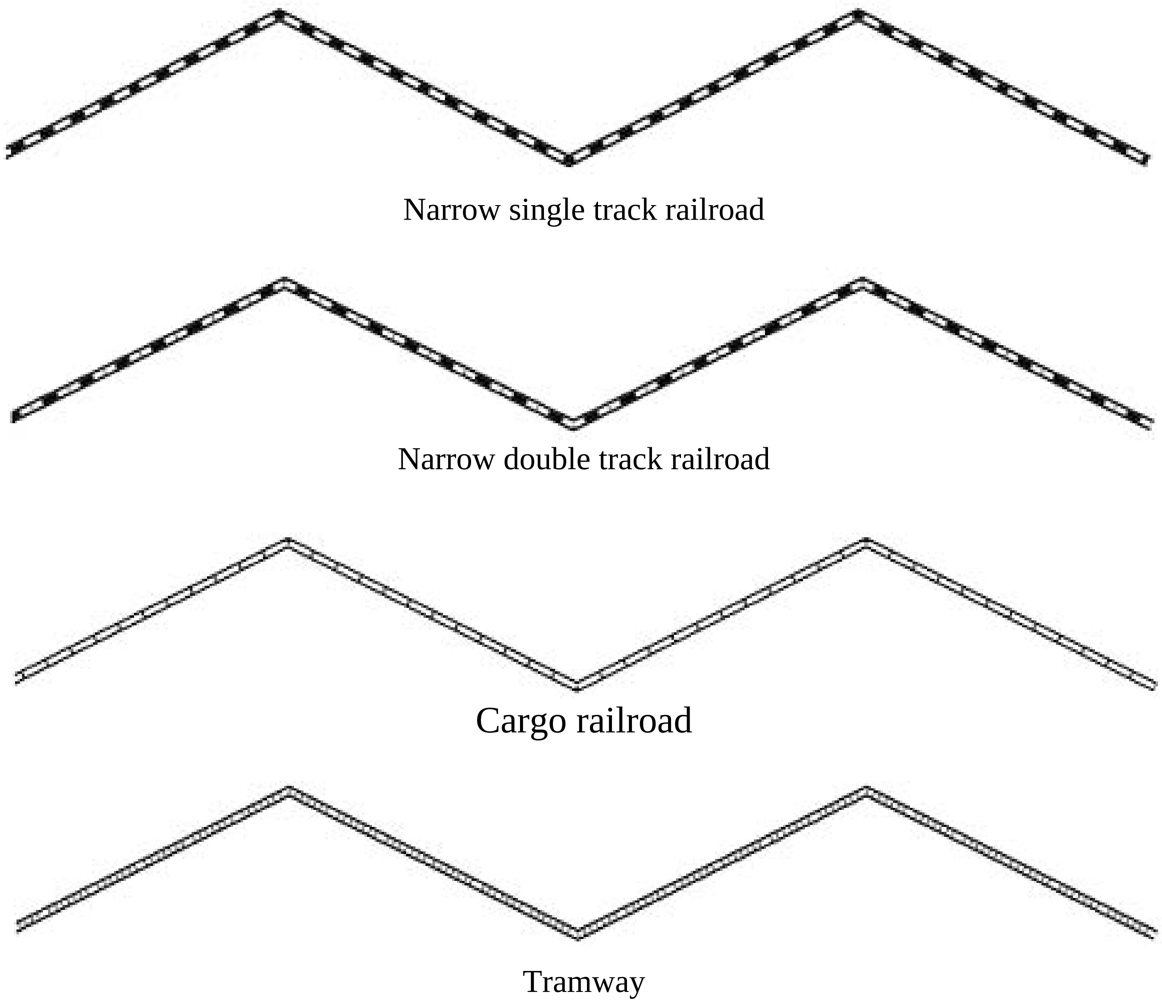


Fig. 5.23 Some special linear symbols from Swisstopo created by the new drawing methods

5.6 Curve

Cartography takes an approximate or even abstracted way to reflect the physical world. Although polyline can be used to mimic smooth paths, it generally requires many control points. In contrast, curves provide efficient ways to generate smooth views of polylines, including sharp angles, with a relatively small number of vertices. When comparing Fig. 5.24b and c, only half the number of control vertices is needed to draw a smooth curve.

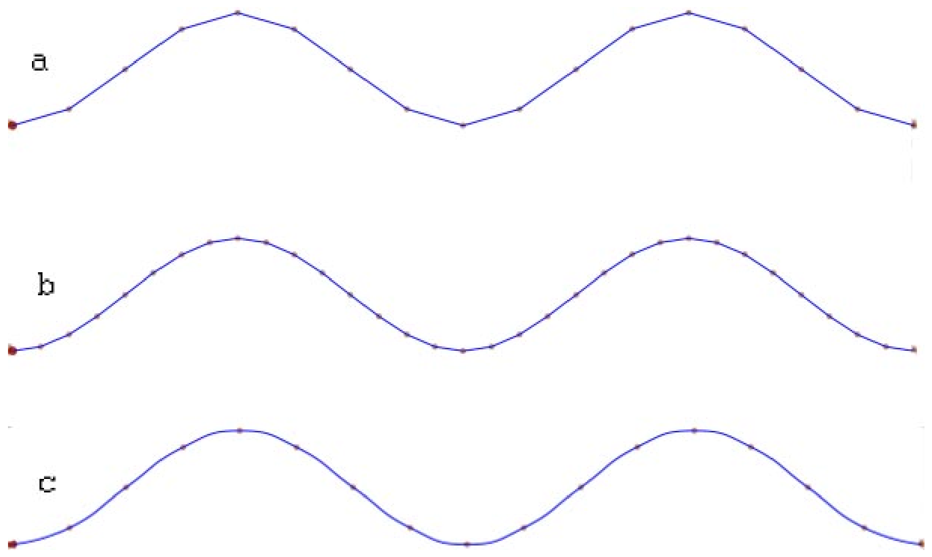


Fig. 5.24 Comparison of polylines (a, b) and the cubic Bézier curve (c)

SmartLine can only produce cubic Bezier curves using the traditional method, which means that they are drawn in a rigid way from the start to the end at once. This is due to the fact that for the new line drawing method developed in this work, a line is drawn as a series of straight line segments, each of which is specified by two vertices; while to create a Bezier curve, there are at least three vertices needed that do not lie on one straight line. Using the traditional method may produce gaps at turns or intersections of non-solid curves. This problem can be solved by adjusting the gap length or the dash length. Fig. 5.25 shows some curves drawing with the new method developed in this work. The reason of the shortcoming of the dotted line is described in chapter 6.

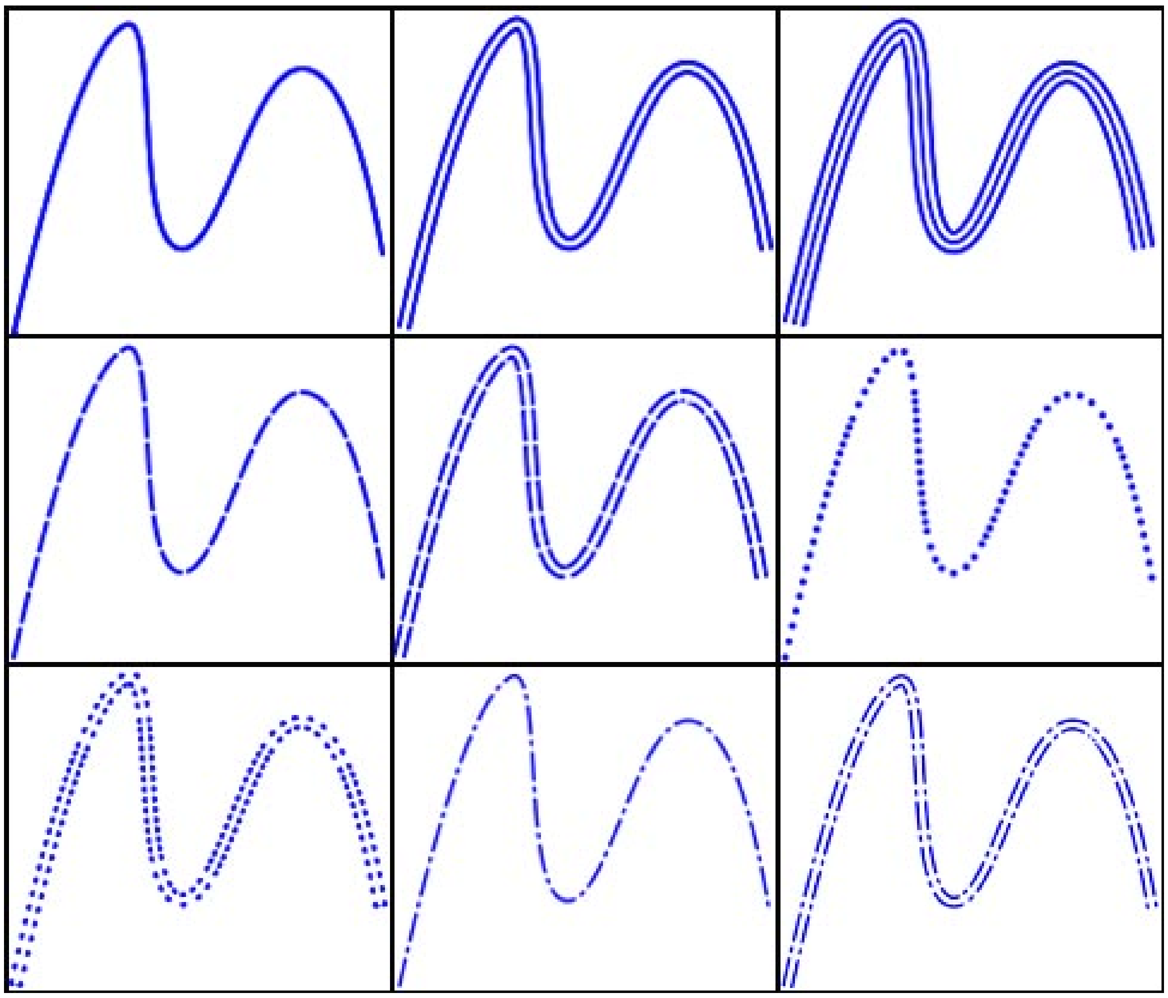
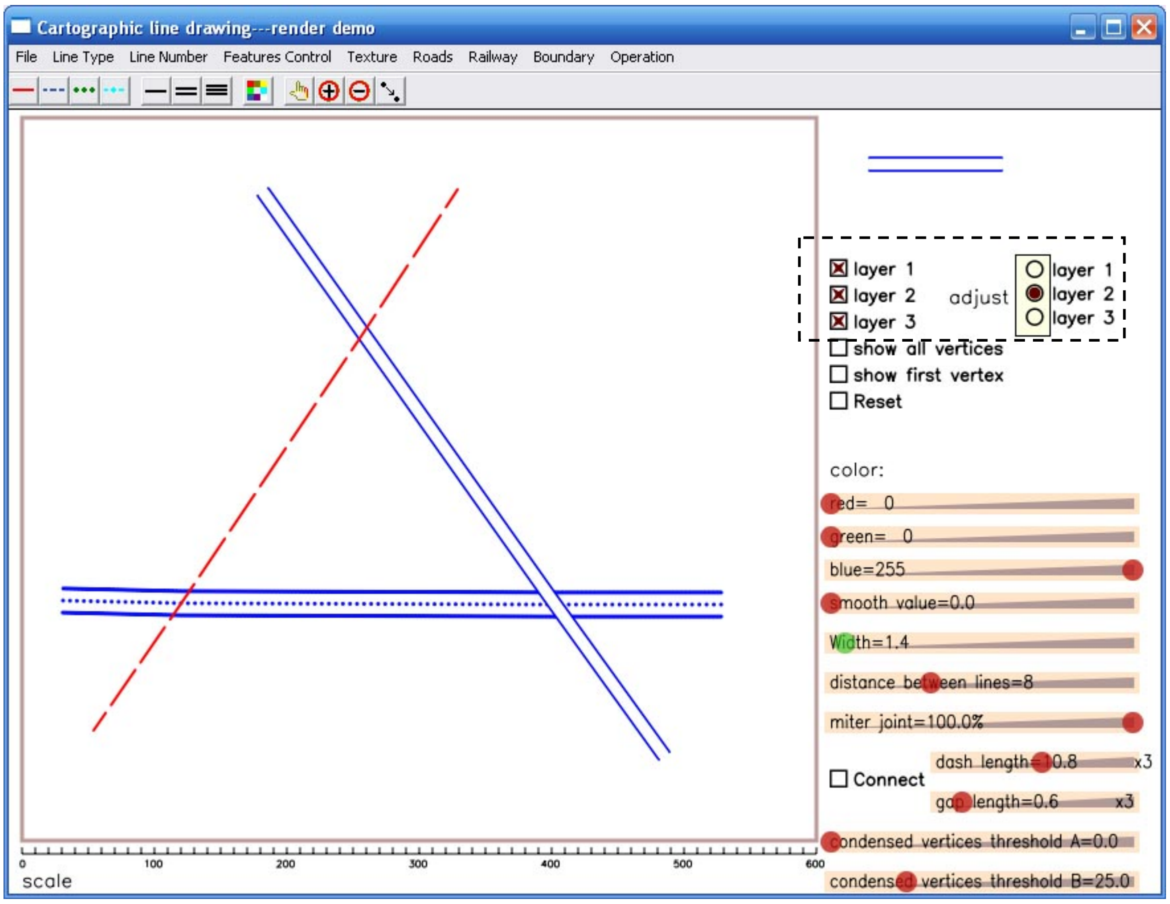


Fig. 5.25 Curves in different styles using the new drawing method

5.7 Manipulate linear symbols on different layers

In order to obtain efficient control over the same kind of line symbols, a layer function is introduced in SmartLine. Lines having the same visual representations can be loaded on one layer. Once a layer is selected, the adjustments will be applied to all lines of this layer without intervening symbols from other layers (dashed frame in Fig. 5.26). At present, SmartLine allows building three layers, but it is possible to add more by adjusting the program. Fig. 5.26 shows one example drawn with three different linear symbols loaded from three layers.

Intersections of lines are important places that need to be carefully designed. Depending on the real situations and the map designer’s personal preference, there could be many solutions to show intersections. SmartLine can produce two kinds of intersections: fusions and overlaps. Lines with the same properties will be loaded on the same layer and they fuse at the intersections with roughly balanced arms (either dashes or gaps); if lines are loaded on different layers, lines on layer 1 cover lines on layer 2, and layer 3 is at the bottom (Fig. 5.26).



☒ layer 1
☒ layer 2
☒ layer 3
adjust: ○ layer 1 ● layer 2 ○ layer 3
☐ show all vertices
☐ show first vertex
☐ Reset
color:
red= 0
green= 0
blue=255
smooth value=0.0
Width=1.4
distance between lines=8
miter joint=100.0%
dash length=10.8 x3
gap length=0.6 x3
☐ Connect
condensed vertices threshold A=0.0
condensed vertices threshold B=25.0

Fig. 5.26 An example of three linear symbols loaded from three layers

5.8 Other editorial functions

Sometimes, cartographers want to slightly modify the line geometries in order to show the paths more clearly. For example, in places where the vertex density is too high, we can remove some excessive vertices. If the vertices density is too low, we might want to add some intermediate vertices to smooth the path. Such kind of moderate adjustment should not change the accuracy of the original geo-information.

SmartLine provides a function to adjust geometries, which must be activated by users (arrow in Fig. 5.27). To add a vertex, we move the mouse to the destination and click on the left button; to delete a vertex, we select it and click on the right button (Fig. 5.28). To change the position of a vertex, we select it first, hold down the left mouse button, and drag it to the destination.

SmartLine also allows us to change the viewport, which can be done by three functions: pan, zoom-in and zoom-out. (Fig. 5.27, from left to right).

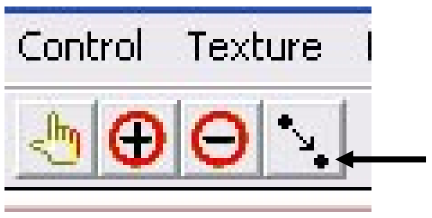


Fig. 5.27 Tool bar for some control functions

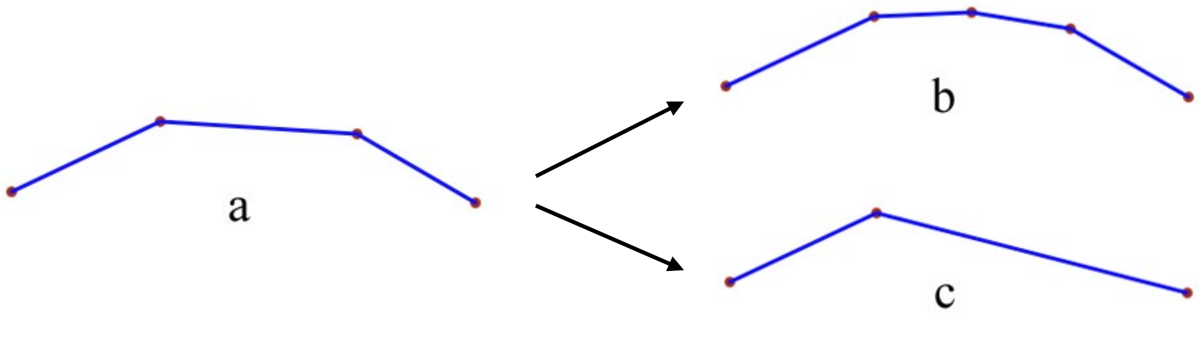


Fig. 5.28 Adding (b) or removing (c) a vertex from a line (a)

5.9 Application of texture mapping in line symbolization

Map designers are familiar with graphic primitives of simple geometrical forms, like points, lines, and areas. In fact, more complex forms like texture images may also be used as graphics primitives. Although this idea has been proposed more than a decade ago [Haeberli, 1993], it has never been studied carefully in cartographic applications. In this work, the application of using texture images to generate line symbols was studied. It turns out that the performance of line symbolizations could be greatly improved by using texture images as a graphic primitive. Fig. 5.29 shows some examples of line symbols created with texture images.

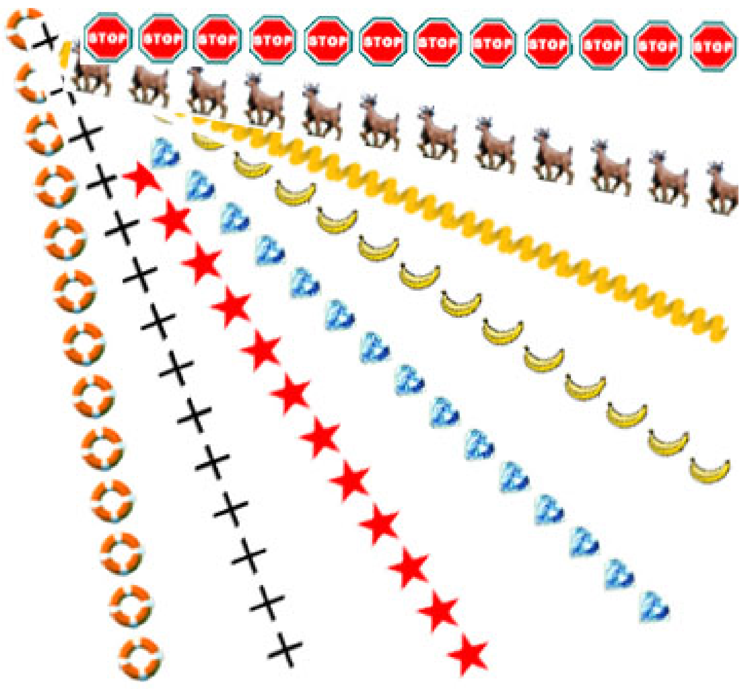


Fig. 5.29 Examples of line symbols created from texture images

5.9.1 Texture source

An appropriate texture source is critical to produce a high quality textured line. A map is used to scale the physical world onto a limited area. On a digital map product, most line symbols are only a few pixels in width. When mapping an image to such a thin area, the size of the mapped image should be relatively small and it is preferable to use images with clear color contrasts.

Generally, in order to express abstract information, it is better to use images with a realistic view. For example, we can use national flags to label national boundaries. To express natural geographic phenomena like paths and rivers, we can try images with different degrees of realism (Fig. 5.30). In any case, it is important to carefully select and test the use of a texture image.

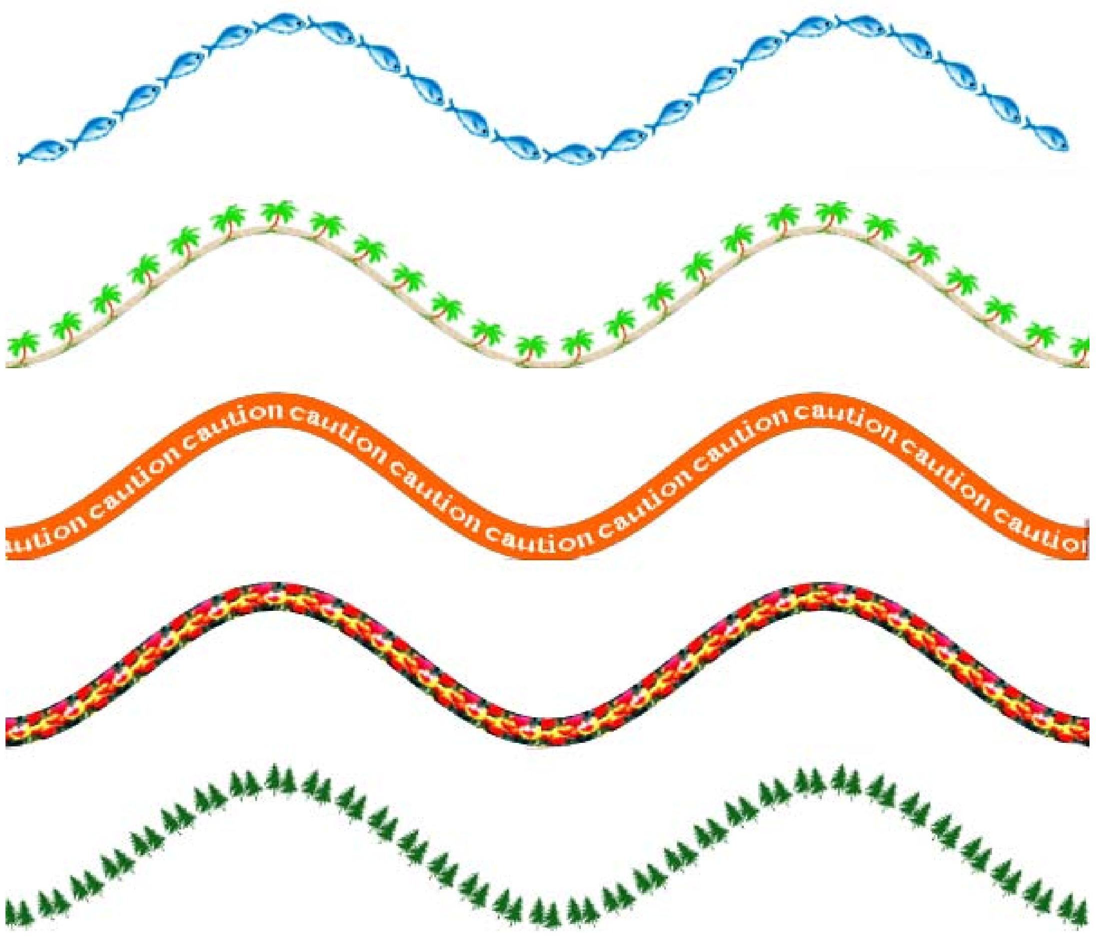

caution caution caution caution caution caution caution caution

Fig. 5.30 Examples of different textured lines

From top to bottom: a caravan of fishes riding on sea waves; a land border with sandy beach and palm trees; a restriction line; a flower path; a forest belt.

A collection of texture images has been constructed; all have been tested to generate line symbols. Each of them is a 32 by 32 pixel bmp file. This collection is classified in 9 categories, including flowers, signs, water, geometry, animals, flags, trees, paths, and others (Fig. 5.31). It can be expanded easily; users can also design new images themselves to further extend this collection.

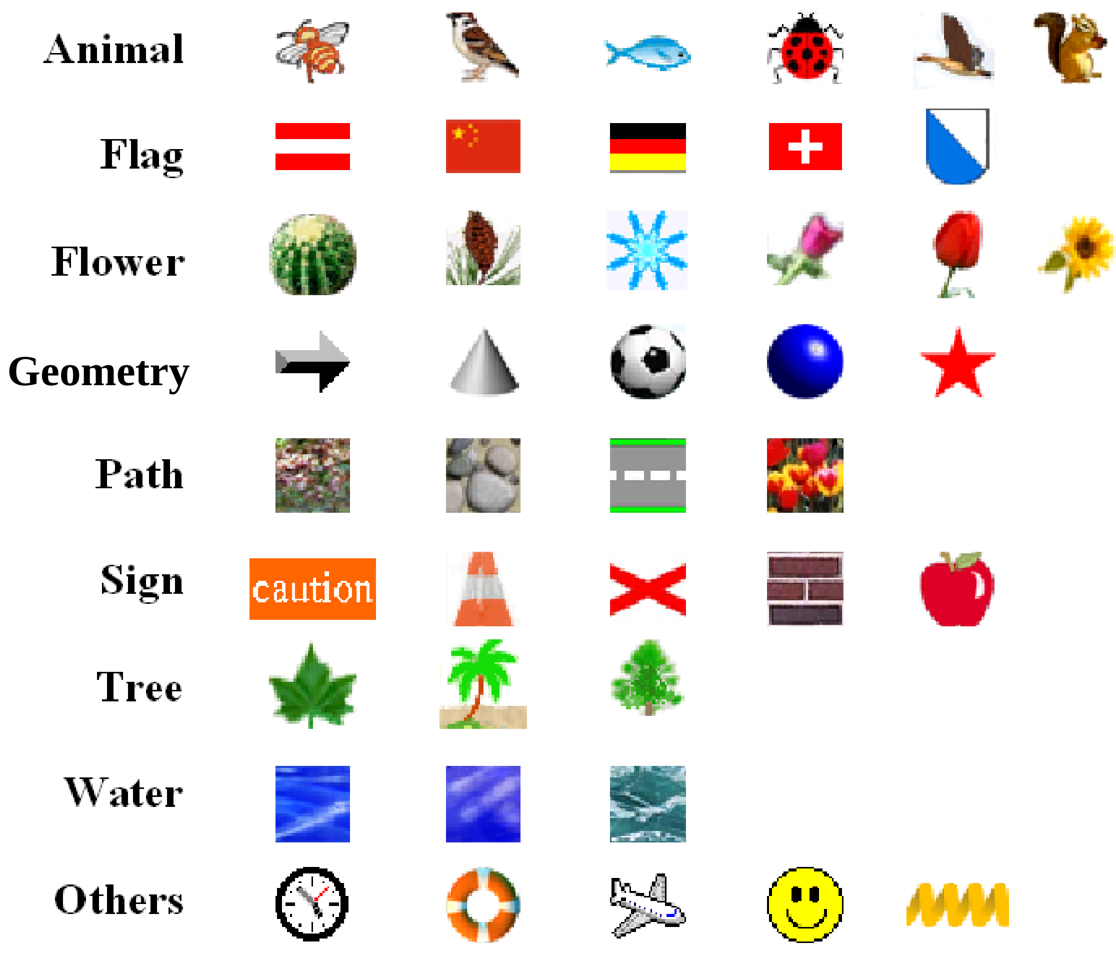


Fig. 5.31 Some texture images available in SmartLine

5.9.2 Mapping

The procedure of mapping contains two steps: the first is to specify the target space; the second is to fill the target space with the texture image (Fig. 5.32). The second function is provided by the graphics library of AGG. To apply this technique, the main work is to specify the target area along a line path. This process is similar to specify the outline of a thick plain line with miter joints, but the fillings are different.

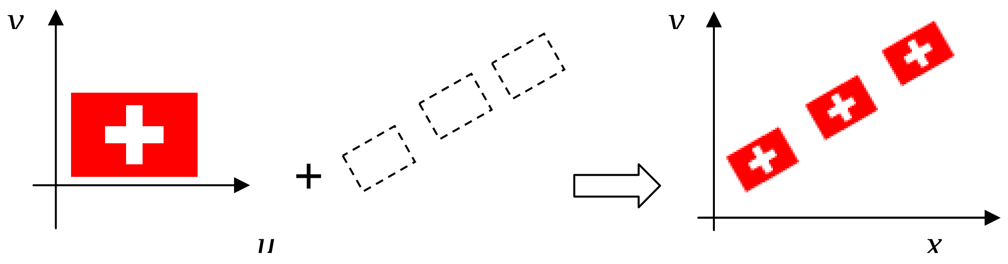
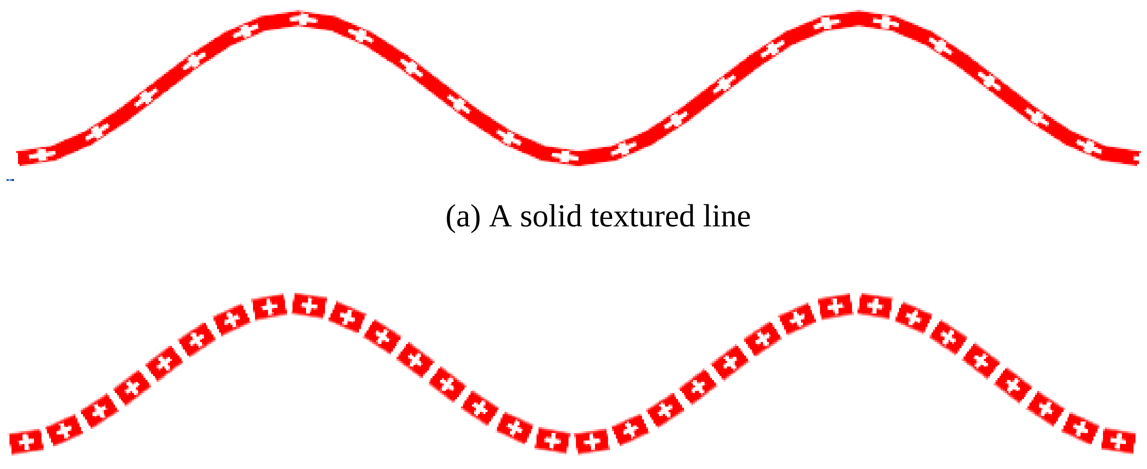


Fig. 5.32 Generation of a texture line involves 2D transformations

For general applications of texture mapping, an important step is to set up spatial relations between the texture images in one coordinate system with target registration points in a different coordinate. Occasionally, it requires complex computations of distortion if the target area has irregular outlines. In contrast, since we are pasting images onto regularly shaped areas (Fig. 5.32), there are mostly no complex distortions involved to generate textured lines.

There are only two kinds of textured lines, solid textured lines and dotted textured lines. The solid textured lines have no gaps compared with the dotted texture lines (Fig. 5.33).


(a) A solid textured line

(b) A dotted textured line

Fig. 5.33 Examples of solid and dotted texture lines using the same texture source

Mapping for solid textured lines

Fig. 5.34 demonstrates the target spaces for a solid line. The texture image is a square with different colors at the left and right edges (shown on the right side), which can clearly show how the target space is specified. A chain of rectangles (or squares) is aligned continuously from line start to line end. The image could be bended or split at joints. The line could ends in a partial image.

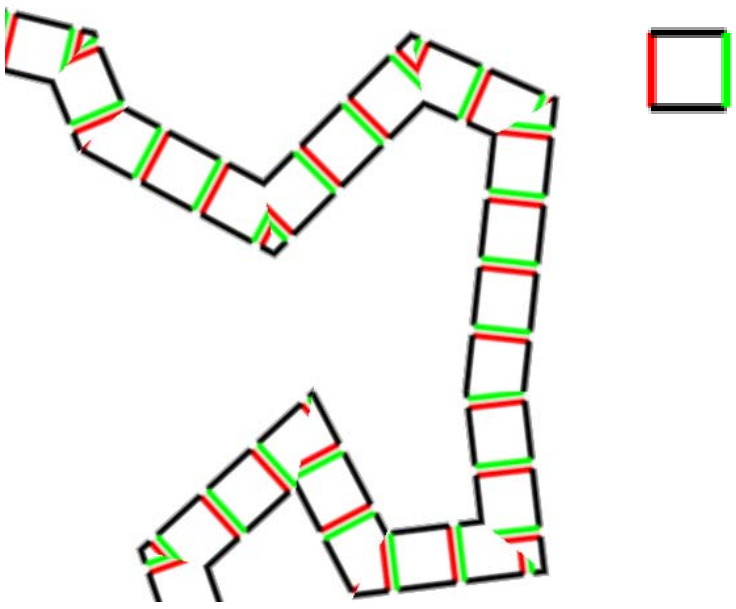
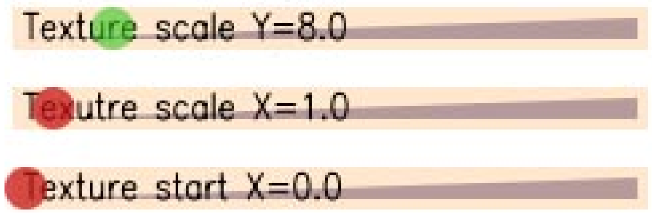


Fig. 5.34 The specification of target area of a solid line
The texture image is shown on the right.

In SmartLine, the original square image can be stretched. The default settings of a textured line are: x=1.0; y=8.0. The setting of "y" determines the width of that line; the setting of "x" decides on the scaling of the texture image along the path. The start position of the line texture is set to "0" (Fig. 5.35).

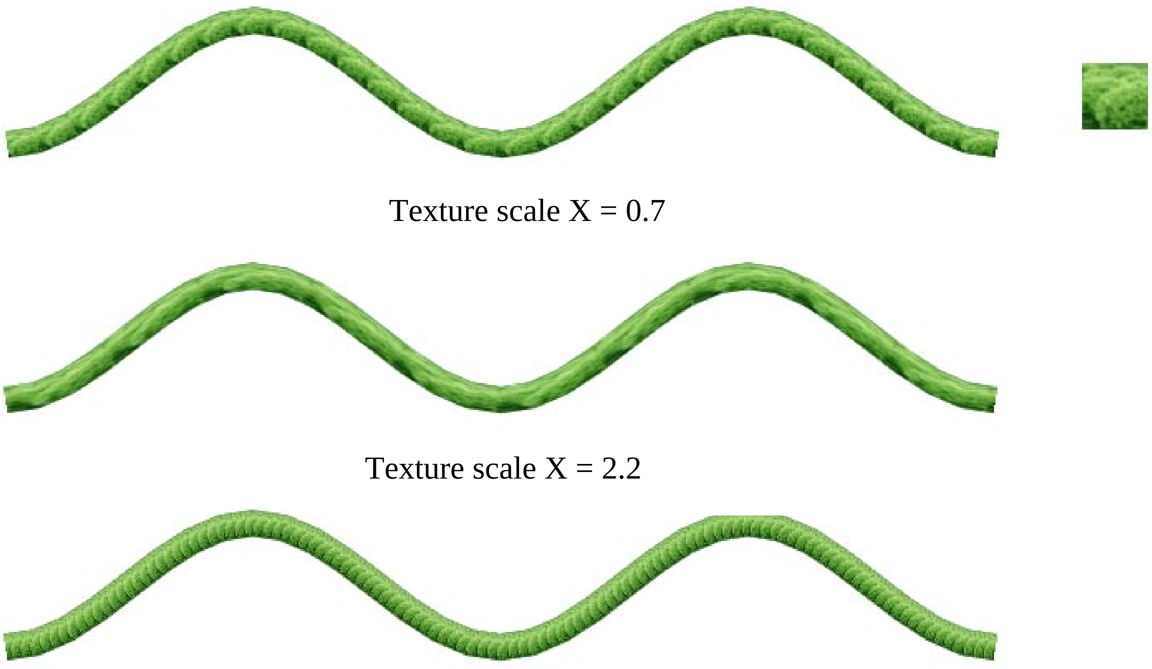

Texture scale Y=8.0

Texutre scale X=1.0

Texture start X=0.0

Fig. 5.35 Controllers for texture size

Fig. 5.36 shows an example of how to get different results by stretching the texture images. With different settings of the “Texture scale X”, we can generate objects such as “holly fence”, “grass path” or “pipeline”. The texture image is shown on the right side.



Texture scale X = 0.2

Fig. 5.36 Various effects with the same texture image but different scaling along the x-axis

As a general rule, it is better to use solid line styles to show continuous phenomena. The texture image should better have no polarity or only have left and right polarity. As a result, the textured line looks continuous and there will be little unpleasant effects brought by split or partial images. Fig. 5.37 shows an example of a river and a pedestrian path. The texture images are two 32x32 bitmaps shown at the right. Both textured lines are created using solid line styles in curve mode.

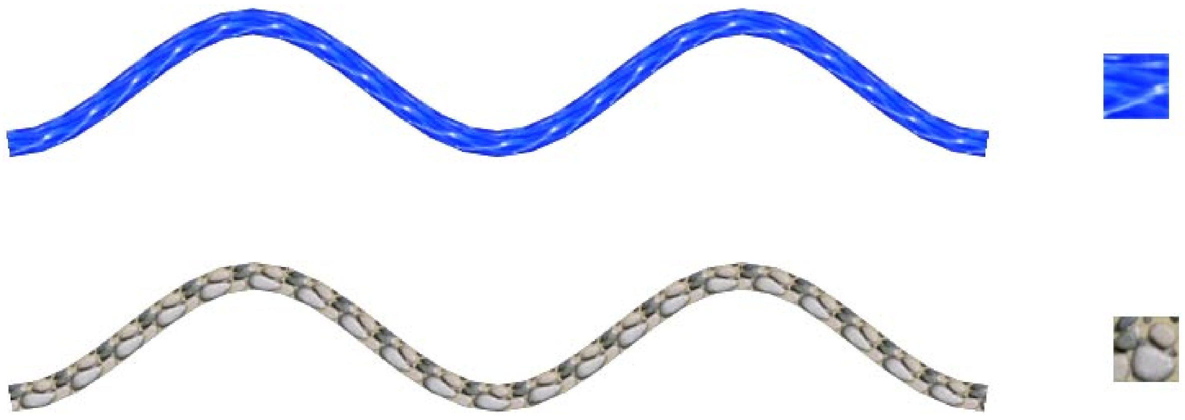


Fig. 5.37 Examples of textured solid lines

If the texture image has a realistic view, using the solid line style may bring inappropriate effects at turns and line ends (Fig. 5.38). Fig. 5.37 is an example using a solid line style; the star is chopped at turns and the line end. For this kind of texture sources, it is better to use the dotted line style.

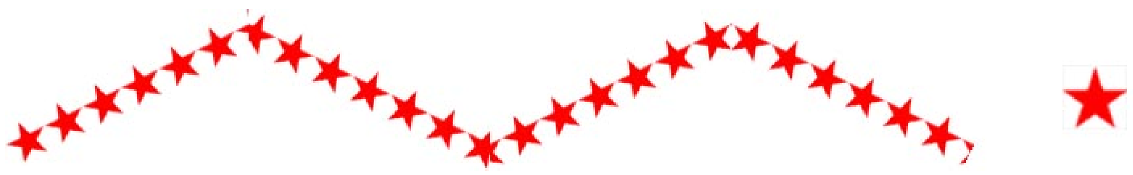


Fig. 5.38 Inappropriate textured line using a solid line style

Mapping for dotted textured lines

Fig. 5.39 shows the specification of target areas of a dotted line. Each dot is the center of a square; the texture image cannot be stretched. At places where there are no spaces to put in a complete image; some images could be covered by adjacent ones.

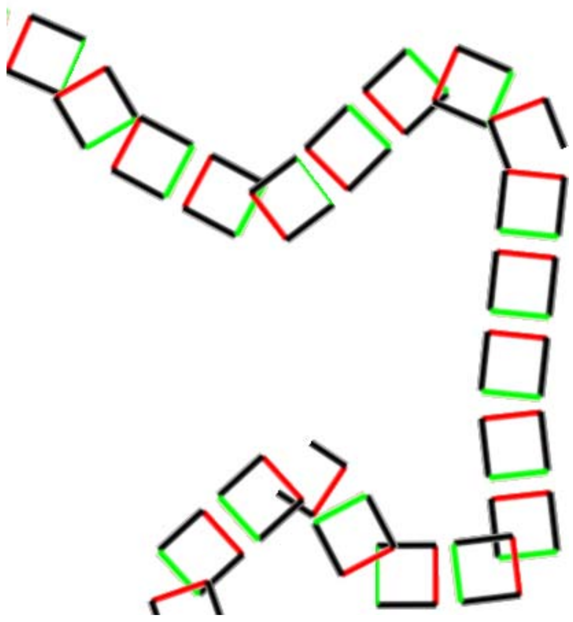


Fig. 5.39 The specification of target areas of a dotted line

Dotted textured lines are suitable to display abstract information, which is often symbolized by images with realistic views. Fig. 5.40 shows two more examples of textured lines using dotted line styles.

There are two parameters to control dotted textured lines: line width and gap length. Since each target space of a texture image is a square, the line width (or texture size) can be adjusted by a single slider of “Texture scale Y”. The gap length between two texture images can be controlled by a slider of “Gap length”. The texture gap can be removed by setting the “Gap length” to 0, and then we can obtain similar results as when using a solid line style (Fig. 5.41). Sometimes, this is even better than using the solid line style because there will be less split of textures turns and no partial images at line ends.

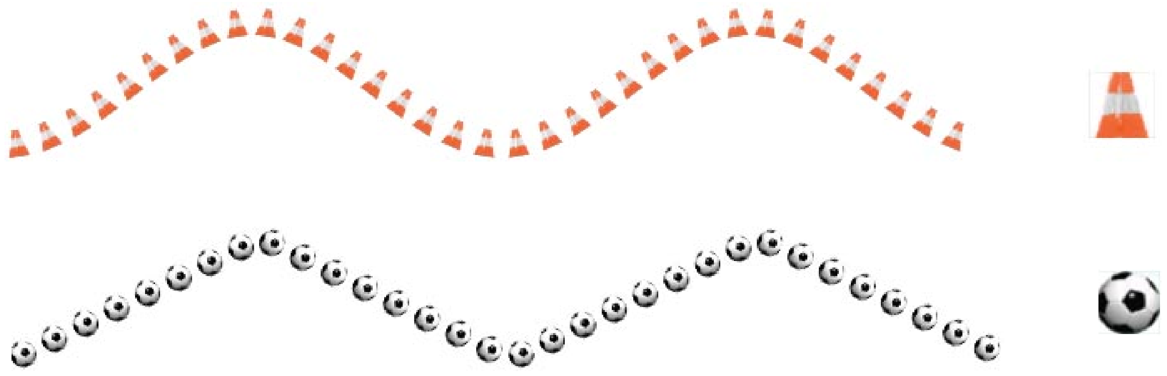
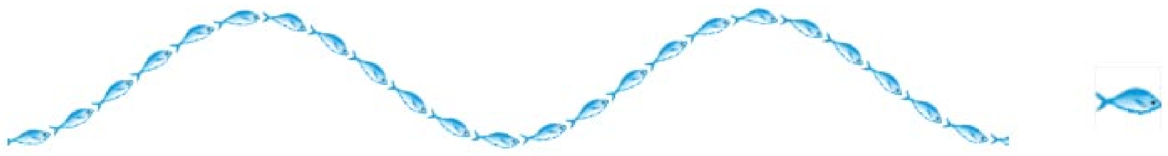
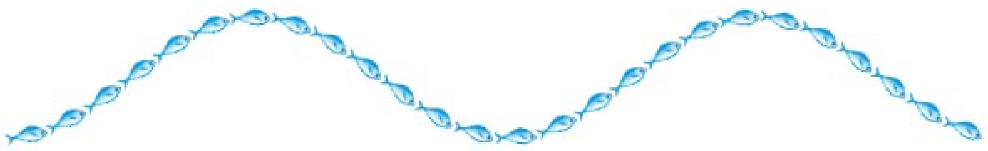


Fig. 5.40 textured lines using dotted line style to show abstract information



Solid line style, texture scale Y = 25.8; X = 0.9



Dotted line style, texture scale X = 1.0; gap length = 0.0

Fig. 5.41 Various effects using dotted line styles with different settings

5.9.3 Examples of textured lines

In this section, examples of line symbolization using textured images are presented. In Fig. 5.42, the Swiss territory boundary is outlined with Swiss national flags. It uses a dotted line style in curve mode in order to obtain smooth turns. Fig. 5.43 shows the confluence of several rivers near Zurich city using a solid line style. In Fig. 5.44, two texture images are used to represent the jogging paths near ETH Hoenggerberg. In Fig. 5.45, the three paths are symbolized by different texture images loaded from three layers.

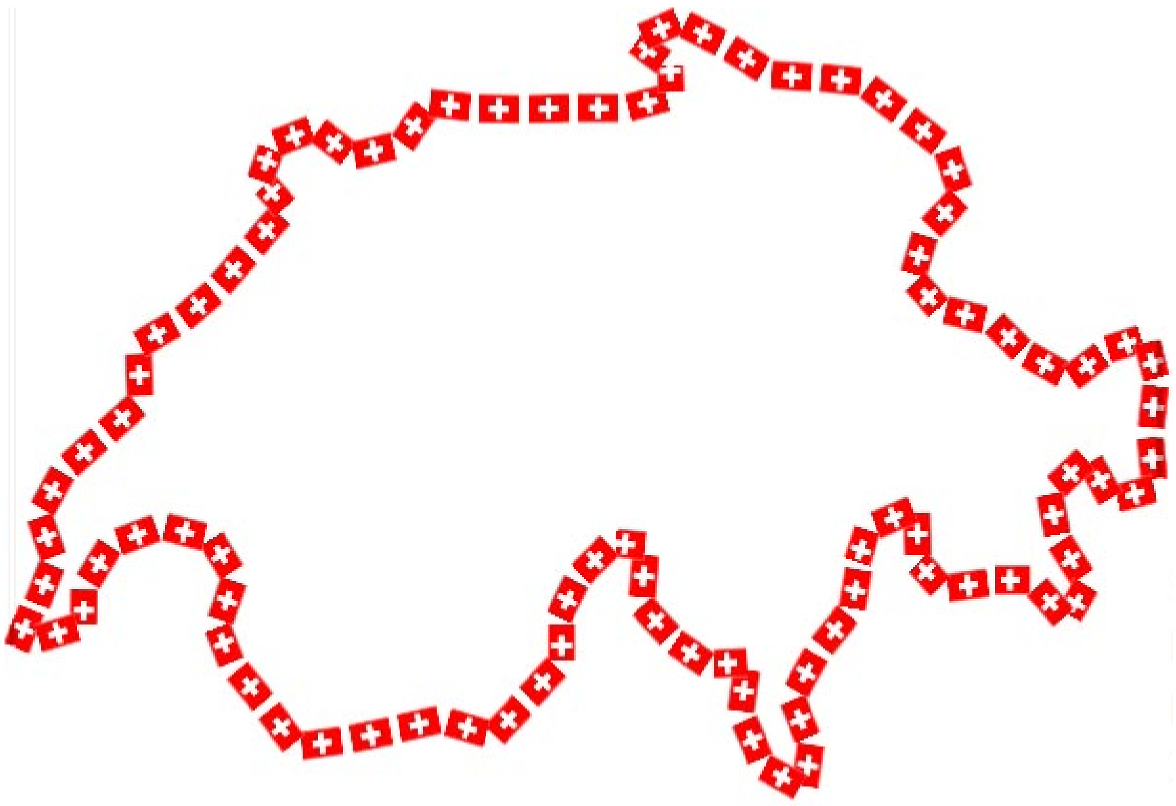


Fig. 5.42 A textured Swiss border

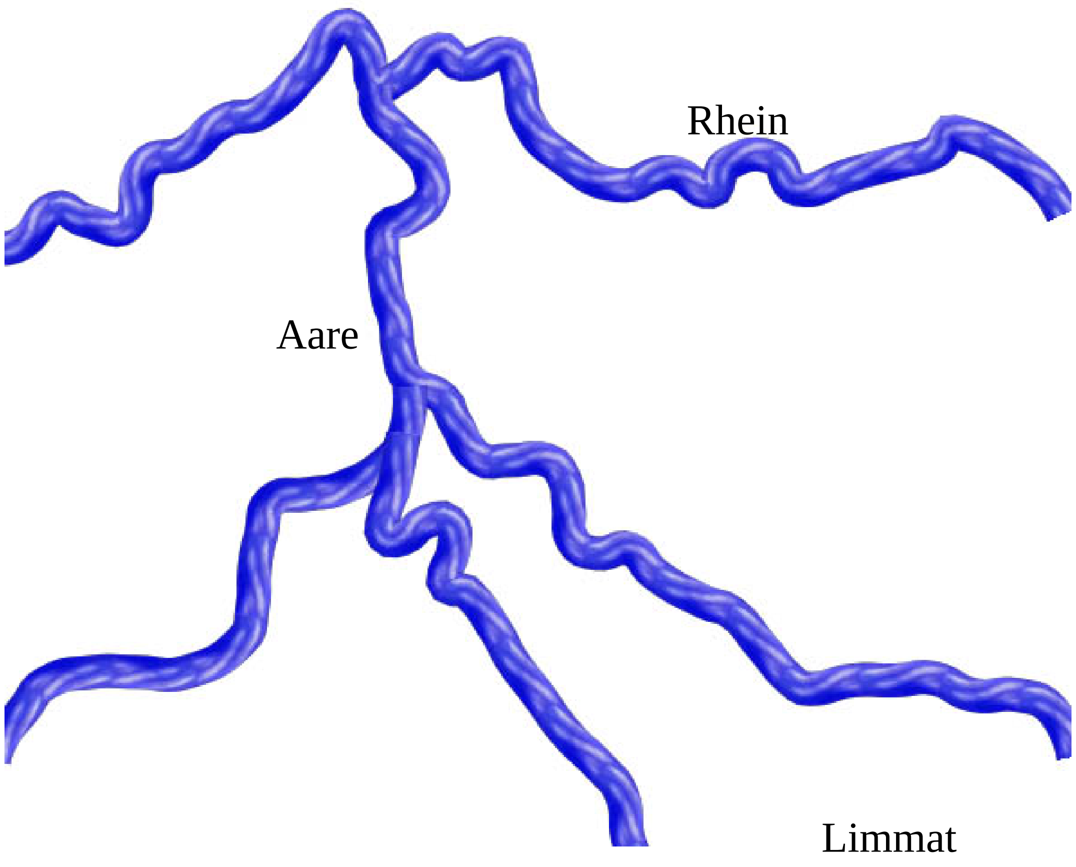


Fig. 5.43 Confluence of Aare, Reuss, Limmat, and Rhine river near Zurich

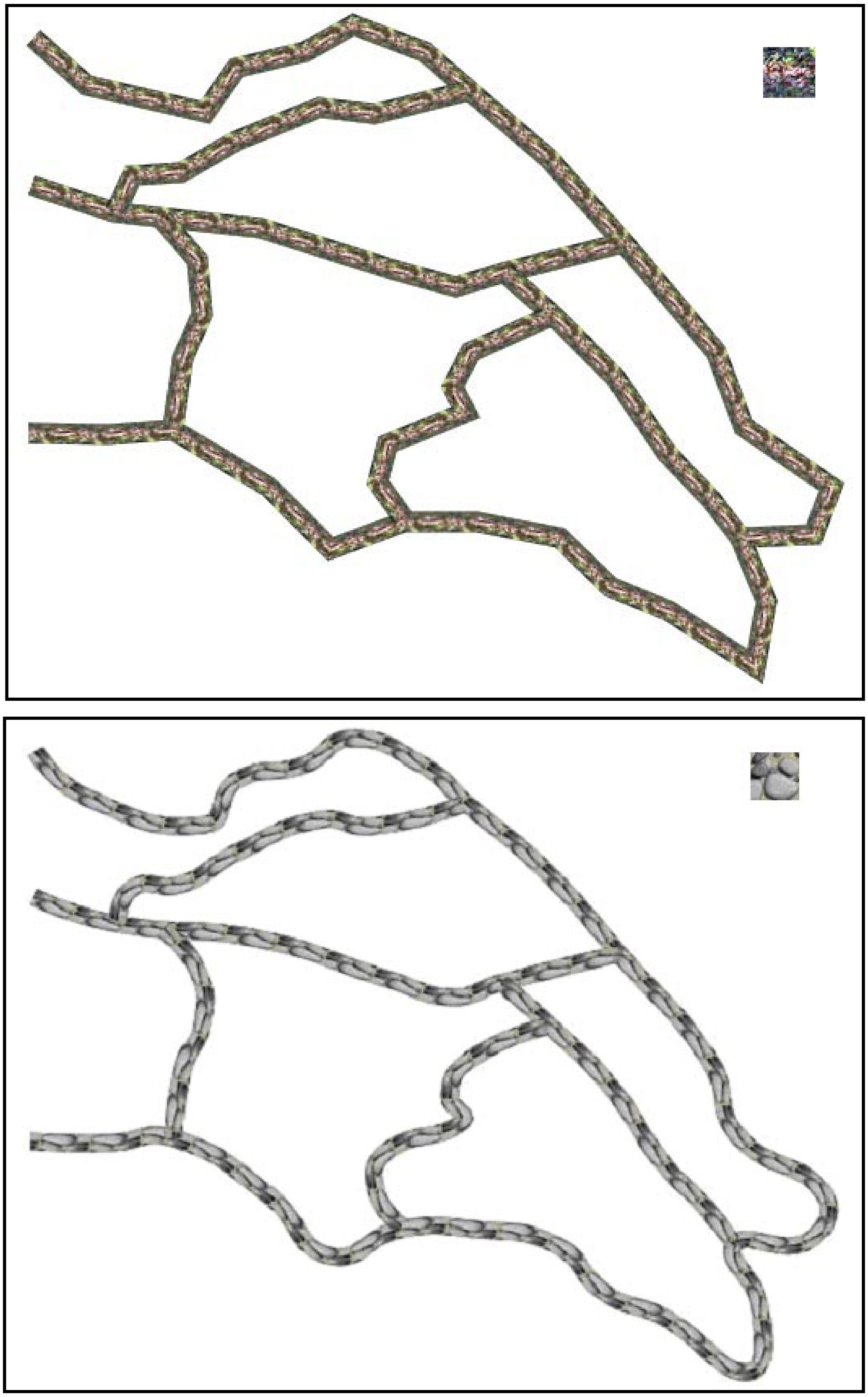


Fig. 5.44 Jogging paths around ETH Hoenggerberg using different texture images

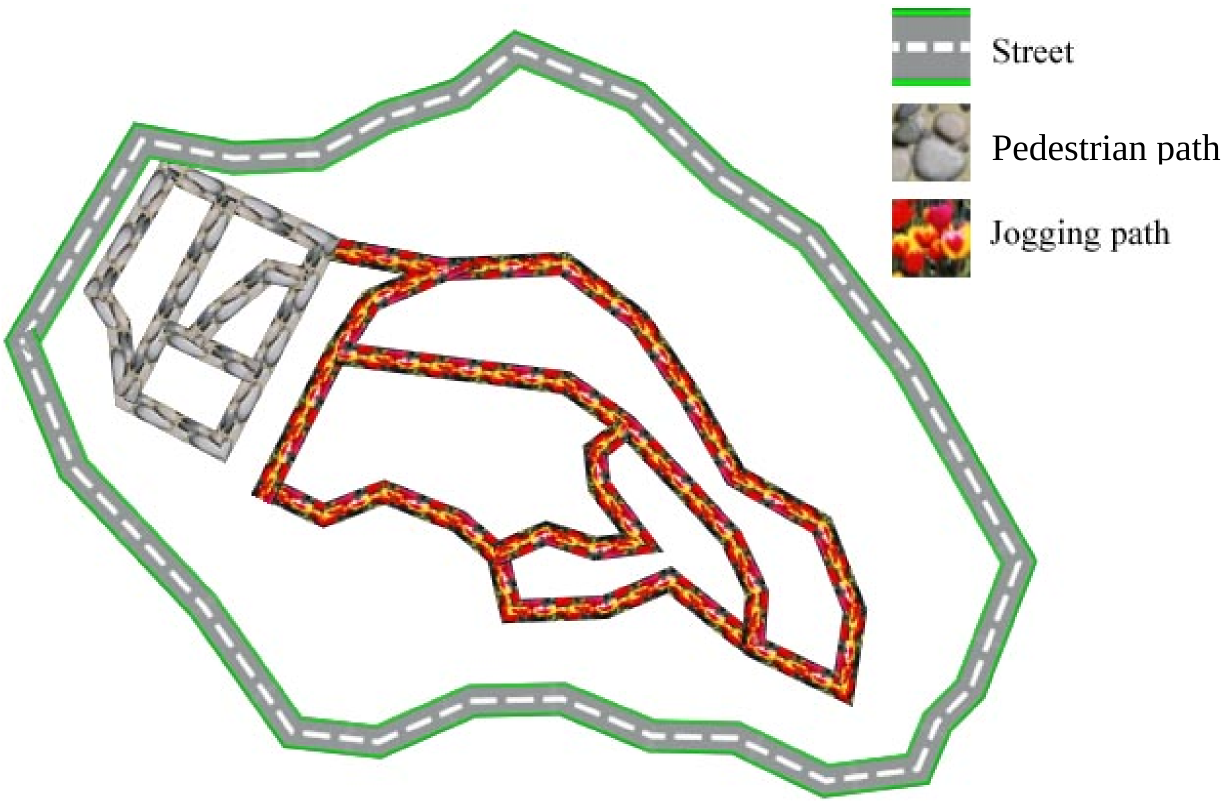


Fig. 5.45 Three paths around ETH Hoenggerberg loaded from three different layers

Chapter 6 Discussion and conclusions

This work aims to improve the cartographic line symbolization by focusing on two aspects. The first is to introduce cartographic rules into line drawing processes in a way that the presentation of line paths could better meet cartographic requirements. The second is to enrich line symbols with respect to both category and visual quality. Two strategies were studied to achieve these goals, including the development of new line drawing algorithms and the application of texture mapping techniques to produce textured lines. As a result, a new line drawing program, SmartLine, was developed to demonstrate these progresses aiming to generate line symbols for cartographic applications.

6.1 New algorithms to draw non-solid lines

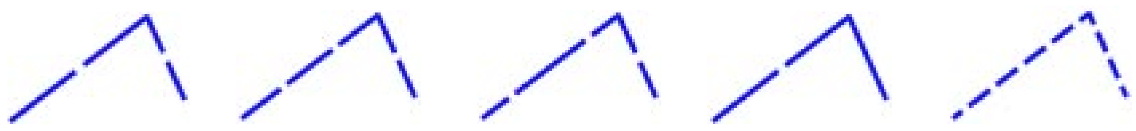
As a general cartographic rule, there are some positions on a line path that should be clearly displayed as solid symbols, including turns, intersections, and line ends. Otherwise, some important geographic information will be missing if gaps are generated at such places. While many present graphics programs pay little attention to this aspect, extensive modulations are generally required to modify the gap positions. A main achievement of the new algorithms is the ability to draw non-solid lines, which includes dashed lines, dotted lines, and dash-dotted lines, with solid turns, intersections, and line ends.

6.1.1 Generating solid turns

The primary concern of the new line drawing algorithms is to avoid gaps at turns, intersections and line ends. To draw a dashed line, the basic idea is to treat each pair of adjacent vertices as the two end points of a straight-line segment, which starts with or ends in a half dash. A round-off function is performed to enable each line segment containing only an integer number of dashes and gaps. Because gaps are normally much shorter than dashes, there is less chance for readers to notice different dash lengths than different gap lengths. Thus, it is the dash length to be adjusted first. As a result, each turn is made up of two half dashes from two adjacent line segments. In addition, this method also allows the whole dashed line end exactly in a half dash.

Similar ideas were also used to develop new methods to draw dotted lines and dash-dotted lines. For dotted lines, it is the gap length to be automatically adjusted. For dash-dotted lines, line starts, line ends, turns and intersections are drawn as dots instead of dashes.

A main shortcoming of this method is that after automatic adjustment, the dash lengths for dashed line or the gap length for dotted line may vary along a dashed line. In the worst case, the largest dashes may be nearly two times as long as the short ones. See the examples of dashed line in Fig. 6.1a, each of the two line segments contains one dash plus one gap after the round-off process. The dash



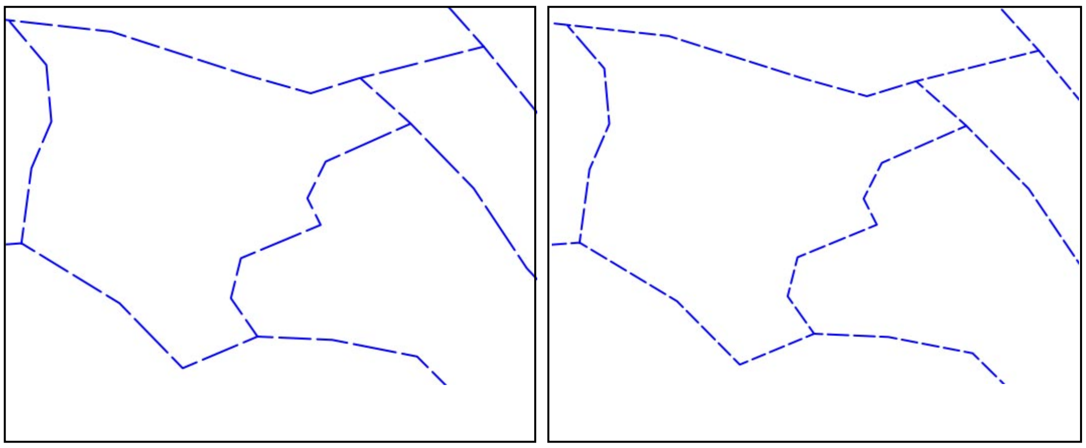
a (37.3; 3.0) b (36.8; 3.0) c (25.2; 3.0) d (60.0; 3.0) e (9.0; 3.0)

Fig. 6.1 Different dash lengths using the new dashed line drawing method (Figures in parentheses indicate dash length and gap length respectively, in units of pixel).

length of the left arm is clearly longer than the right one. This difference can be reduced by decreasing the dash length (Fig. 6.1b and c). An example of this problem on dotted line can be seen in Fig. 5.25.

Although both the left line segments in Fig. 6.1b and c contain two dashes and two gaps, only the ones in Fig. 6.1c are drawn as two half dashes and one full dash; the one in Fig. 6.1b contains two half dashes and one partial dash. This is to ensure the generation of solid turns, which is a drawing priority: the two half dashes at line ends are to be drawn first, then the gaps, and finally the other dashes. If there is not enough space to put two half dashes and a gap within one line segment, it will be drawn as a solid line (right arm of Fig. 6.1d; Fig. 4.22); if there is not enough space to contain two full dashes and two gaps, the dash in the middle can only partially be displayed (Fig. 6.1b). Generally, the more the dashes (or gaps) a strait line segment contains, the less differences will result because the error is distributed to more dashes (Fig. 6.1e and Fig. 6.2).

In contrast, traditional dashed line drawing method will generate lines with uniformed sized dashes and gaps. In some cases, if the position of gaps does not interfere with the display of important geometric information, we can switch to the traditional method. For example, at soft turns containing crowded vertices, the new method will produce some solid line segments (Fig. 4.22). If we want to keep a regular line pattern, we can draw these regions using the traditional method. Although it may leave some gaps at turns, no important information will be lost by a few of gaps at smooth turns with crowded vertices. In case of sharp turns with crowded vertices, we could treat them as solid line segments. The dashed line pattern will not be interrupted by small pieces of solid lines (Fig. 4.21). For this purpose, two parameters are introduced to switch between the new line drawing methods and these two modifying methods (Section 4.3.3). Users can adjust these two parameters depending on the map scale.



Dash length = 30; Gap length = 6.4   Dash length = 17; Gap length = 5.1

Fig. 6.2 Diminishing differences of the dash lengths using the new dashed line drawing method by adjusting dash length and/or gap length

Left: some dashes are obviously much longer than others;

Right: the difference is reduced by decreasing the dash length and gap length

6.1.2 Generating solid intersections

Intersections of two lines are important positions that need to be clearly displayed. Like turns, intersections should also be displayed as solid symbols. When two lines meet, they either cross, or one is above the other. We can distinguish two different types of intersection.

1. Between two lines with the same properties
2. Between two lines with different properties

With the new dashed line drawing method, SmartLine provides a simple solution to the first type of intersections. Since lines with the same properties will be contained in one layer, each intersection is drawn as four half dashes (or gaps for dotted line and dash-dotted line) if the point of intersection is captured on both lines. In this case, lines are assumed to cross each other at the intersections.

For the second type of intersections, lines with different properties will be drawn on different layers. They either cross each other, or one line is placed above the other. SmartLine adopts a simple solution: lines on layer 1 will cover lines on layer 2, layer 2 covers layer 3, and so on. This solution relies on the assumption that the positions of intersection have been captured during data acquisition. Unfortunately the vertices at intersections cannot be distinguished from other vertices. If it is required to display more complex information at intersections, like ramps, bridges, tunnels etc, we must find a way to recognize the meeting points and add special symbols, like pictorial symbols with realistic views.

6.1.3 Generating multiple lines

Double lines and triple lines are two kinds of multiple lines that are frequently used in cartographic symbolization. However, up to now there is no efficient method to draw these kinds of lines. People have to use some indirect drawing methods and it often requires complicated modifications of the preliminary results (Fig. 4.30).

In this work, new algorithms were developed to draw double lines and triple lines efficiently. The basic idea is to compute the geometry of each consecutive single line individually. Since the cartographic principles have been integrated into the drawing algorithms, some unintended effects of the indirect methods could be avoided (Fig. 4.30). The new single line drawing methods are also used to draw the multiple lines. Therefore, the double or triple dashed lines (or dotted lines, dash-dotted lines) also have solid turns and intersections. For each consecutive single line, the line features can be controlled separately. This can help us to easily create many complex line symbols (Fig. 5.9 and Fig. 5.13).

For double or triple lines, the presentation of intersections uses the same solutions as for drawing single lines: if two double lines or triple lines have the same properties and are loaded on one layer, they will cross each other at intersections; otherwise, the line on the upper will cover the line on the lower layer (Fig. 4.35).

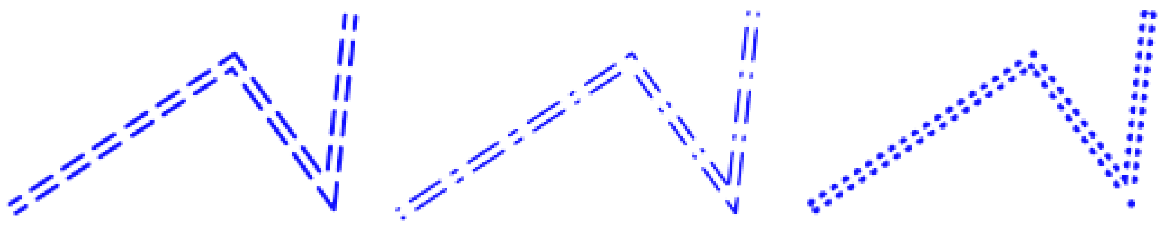


Fig. 6.3 Turns of multiple lines using the new methods (The outer line always ends in solid parts, while the inner line may end in gaps.)

However, there are still some items that need to be improved, especially the display of turns (Fig. 6.3). For dashed lines and dash dotted lines, the outer line will end in a dash, while the inner line may end in a gap. For dotted lines, although a dot is drawn at the turn of the outer line, this turn is flanked with two gaps that are larger than the adjacent gaps. The difference becomes evident if the turn has a sharp angle. The reason is that the primary concern of the new multiple line drawing method is to keep the parallel single lines the same pattern, which is determined by the center line drawn with the new method. Since the outer lines have different lengths from the center line, the differences will be allocated at the turns. As a result, the parallel lines may have different arms flanking a turn. However, these shortcomings can be overcome by adjusting the control parameters, including dash length, gap length, and the distance between parallel lines.

6.1.4 Generating textured lines

Texture mapping turns out to be an interesting tool to greatly enrich the variety and improve the visual qualities of line symbols. A crucial factor of a successfully application is an appropriate texture image. Some general rules should be considered when designing texture images for line symbols. Unlike the common texture mapping, the target area of a textured line is very thin. The mapped images often need to be shrunk. Therefore, it is recommended to use textures with high contrast and color variation. If the texture images are attached side by side with no gaps, they should better have no symmetry or have only left and right symmetric texture to get a continuous picture.

A collection of more than one hundred texture images has been created. All of them have been tested to generate textured lines. People can design new texture images to enlarge this collection. At the time, Smartline works well with 32x32 bmp images.

There are only two kinds of textured lines, solid and dotted lines. Solid textured lines are more appropriate to display phenomena with continuous character like river, road, and so on. With solid line style, the width and height of each texture image can be freely adjusted. If the texture image has margins, we can get "gaps" of dotted lines. In contrast, dotted textured lines are suitable to show discrete objects with realistic views. The dot is the center of a square image. With dotted line style, the texture image cannot be stretched. If the gap length is set to be 0, we can get similar results with solid line styles.

In order to show the texture, the textured lines are generally thicker than the plain lines. The thick texture images may occupy more spaces and may interrupt the display of other symbols. Thus, one should consider avoiding too many textured lines on a small-scale map. For example, we can use the textured lines to label boundaries, or some important objects that require realistic views. Because the textured line is relatively thick, map users may not be able to precisely follow a line path with crowded vertices. In some cases, we can only use textured lines to approximately show the geometries (Fig. 5.42).

Texture mapping may not only be used for line symbolizations, it might also be useful for other graphic primitives such as points and areas. Since points and areas can be drawn much thicker than lines, some complex 3D features can also be displayed, like reflection, bump, and other effects containing much color variations.

6.1.5 Data formalization

A line symbol is generated by the combination of line geometry (geo-data) and the visual representations (specified by line features). In SmartLine, these two factors are separated. Since the bases of the new line drawing algorithm depends on the introduction of many new parameters specifying line features (Table 4.1), a new language called “Line Feature Description” is created to describe these settings.

This method of “Line Feature Description” helps to improve the efficiency of the line drawing process. A map often contains many lines with the same features. With this method, line features can be applied in one step by applying a single file describing line features. We can even use the same file to apply the same line symbols on different maps easily. This file can be opened and edited directly or by saving the modifications even when running the program.

6.1.6 An interactive editorial environment

As shown in the previous discussions, although the new line drawing algorithms have integrated many cartographic rules, they need still further improvements. Because there could occur many complicated situations in a map, it is quite difficult to develop a single solution to cover all the problematic cases. As a supplement to the new line drawing methods, the versatile control functions are important to make delicate adjustments for a convincing symbolization.

Compared with general graphic software, SmartLine offers many flexible control functions (Table 5.1). It provides an interactive editor allowing mapmakers to carry out coordinate adjustments to modify the line paths. Many inappropriate effects caused by the new line drawing methods can be easily optimized through the interactive control functions (Fig. 6.1 and 6.2, e.g.). In addition, users can also create numerous complex line patterns by simply adjusting some attributes, including line width, dash length, gap length and so on (Fig. 5.9 and 5.13).

6.2 Outlook

The solutions developed in SmartLine can be integrated into any map-drawing program to improve line symbolization. In the future, the following points may need further investigations.

- Versatile input capability: One important feature of a high standing graphic program is to be able to support the input file in different formats as many as possible. SmartLine only supports two popular data formats (svg and cgm) and one self-defined data format. Improved input/output capability would therefore be a concentration in the further studies.
- Some results generated by SmartLine are still not graphically perfect and need further improvements. For example, at present, the multiple lines only have miter joints. An extra long joint could be produced at a turn with very sharp angle; this problem is especially noticeable for multiple dotted lines. This situation could be improved by doing the similar miter joint controls of single lines.
- Animation and more interactive functions: An interesting application would be the creation of animated and interactive map products. The ability of flexible controls and animated information over line features is not only a supplement to the automatic drawing algorithms, but also provides an environment for users to actively participate in the map drawing process. Since many cartographic rules have been integrated into the solutions, people with little map-making trainings could also be able to generate convincing cartographic products.
- The introduction of texture mapping into cartography opens new way to enrich the whole symbolization process, including dots and areas. As a different kind of texture source, procedural functions allow the image to be scaled without losing details. Its application in map symbolization is worth to be further investigated. In the long run, more complex graphics

information, like reflections, bumps, 3D textures, etc. could be added to map symbols.

List of figures

References

[Apple, 2007] Apple Inc. (2007): *Introduction to Quartz 2D Programming Guide*, Apple Developer Connection. http://developer.apple.com/documentation/GraphicsImaging/Concep-tual/drawingwithquartz2d/dq_paths/chapter_4_section_1.html#//apple_ref/doc/uid/TP30001066-CH211-TPXREF101

[Baella, 1994] Baella B.; Colomer J. L.; Pla M.; *Map Generalizer: A Practical Experience*. ICA Working Group on Map Generalization, Edinburgh, September 1994.

[Bartels, 1987] Bartels, R.H.; Beatty, J.C.; Barsky B.A. (1987): *An Introduction to Splines for Use in Computer Graphics*. Morgan Kaufmann Publishers. Inc. pp211~245

[Berhardsen, 1992] Berhardsen, T. (1992): *Geographic Information Systems*. Viak IT/Norwegian Mapping Authority, Arendal, Norway.

[Berhardsen, 1996] Berhardsen, T. (1996): *Geographic Information Systems*. Halsted Press.

[Bernhardsen, 1999] Bernhardsen, T. (1999): *Geographic Information Systems: an Introduction*. Wiley, New York.

[Bertin, 1983] Bertin, J. (1983): *Semiology of Graphics: Diagrams, Networks, Maps.* Madison, WI, University of Wisconsin Press, (French edition, 1967).

[Bézier, 1974] Bézier, P.: *Mathematical and Practical Possibilities of UNISURF.* In Barnhill, R.E.; Riesenfeld R.F., eds: *Computer Aided Geometric Design*, Academic Press, New York

[Blinn, 1989] Blinn, J.F. (1989): *Dirty Pixels*. IEEE Computer Graphics and Applications, 100~105, July 1989, Jim Blinn's Corner

[Bresenham, 1965] Bresenham, J.E. (1985): *Algorithm for Computer Control of a Digital Plotter*. IBM Systems Journal, 4(1), pp25~30

[Bresenham, 1985] Bresenham, J.E. (1985): *Run Length Slice Algorithm for Incremental Lines*. In EarnShaw, R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp59~104

[Bresenham, 1988] Bresenham, J.E. (1988): *Ambiguities in Incremental Line Rastering.* In Earnshaw, R.A.: *Theoretical Foundations of Computer Graphics and CAD*. Springer-Verlag Berlin Heidelberg, pp329~359

[Brons, 1985] Brons, R. (1985): *Theoretical and Linguistic Methods for Describing Straight Lines.* In EarnShaw, R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp19~58

[Brodlie, 1985] Brodlie, K.W. (1985): *Methods for Drawing Curves*. In EarnShaw R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp303~324

[Chrisman, 1997] Chrisman, N.R. (1997): *Exploring Geographic Information Systems*. John Wiley and Sons.

[Chryssafis, 1986] Chryssafis, A. (1986): *Anti-aliasing of Computer Generated Images: A Picture Independent Approach*. Computer Graphics Forum; 5: 125~129, June 1986

[Castle, 1985] Castle, C.M.A.; Pitteway, M.L.V. (1985): *An Application of Euclid's Algorithm to Drawing Straight Line.* In EarnShaw, R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp135~140

[Cooper, 2000] Cooper, C.: *Expat XML Parser Toolkit (Version 1.2)*.
http://www.jclark.com/xml/expat.html

[Cromley, 1992] Cromley, R. G., *Digital Cartography*, Prentice Hall, Englewood Cliffs, New Jersey.

[Davis, 1980] Davis L. S.. (1980); *Representation and Recognition of Cartographic Data.* In Freeman H.; Pieroni G. G. (1980); *Map Data Processing*. Academic press

[deMers, 1997] deMers, M.N. (1997): *Fundamentals of Geographic Information Systems*. John Wiley and Sons.

[Dorst, 1985] Dorst, L. (1985): *The Accuracy of the Digital Representaion of a Straight Line.* In EarnShaw R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp141~152

[Douglas, 1973] Douglas, D.H.; Peucker, T.K. (1973): *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricatures*. The Canadian Cartographer, 10(2):112~122, December 1973

[Earnshaw, 1985] Earnshaw, R.A. (1985): *A Reviews of Curve Drawing Algorithms.* In EarnShaw R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp289~302

[Eicher, 2005] Eicher, C.; Briat M.O. (2005): *Supporting Interactive Editing of Cartographic Representations in GIS Software,* 22nd International Cartographic Conference, A Coruña

[Enderle, 1984] Enderle G.; Kansy K.; Pfaff G. (1984): *Computer Graphics Programming,* Springer-Verlag, pp149~167

[Field, 1984] Field, D. (1984): *Two Algorithms for Drawing Anti-Aliased Lines*. Graphics Interface, pp89~95

[Foley, 1996] Foley, J.D.; van Dam, A.; Feiner, S.K.; Hughes, J.F. (1996): *Computer Graphics: Principles and Practice*. 2. Edition, Addison-Wesley

[Forrest, 1985] Forrest, A.R. (1985): *Antialiasing in Practice.* In EarnShaw R.A.: *Fundamental Algorithms for Computer Graphics,* Springer-Verlag, pp113~134

[Galanda, 2003] Galanda M. (2003): *Automated Polygon Generalization in a Multi Agent System.* Dissertation. Faculty of Mathematical-Nature Science, University of Zurich

[Gupta, 1981] Gupta, S.; Sproull, R.F. (1981): *Filtering Edges for Grey-scale Displays*. Computer Graphics, 15(3):pp1~5, August 1981

[Fiume, 1989] Fiume, E.L. (1989), *The Mathematical Structure of Raster Graphics*. Academic Press. Inc. pp138~157

[Foley, 1982] Foley, J.D.; Dam A.V. (1982): *Fundamental of Interactive Computer Graphics*. Addison-Wesley Publishing Company Inc.

[Franklin, 1986] Franklin, W. R. (1986): *Problems with Raster Graphics Algorithms.* In Kessener L.R.A.; Peters F.J.; vanLierrop M.L.P.: *Data Structures for Raster Graphics*. (Ed) Springer-Verlag, pp1~8

[Freeman, 1980] Freeman H. (1980); *Analysis and Manipulation of Linear Map Data.* In Freeman H.; Pieroni G. G. (1980); *Map Data Processing*. Academic press

[Giloi, 1978] Giloi, W.K. (1978), *Interactive Computer Graphics, Data Structure, Algorithms, Languages*. Prentice-Hall Inc.

[Haeberli, 1993] Haeberli, P.; Segal M. (1993): *Texture Mapping as a Fundamental Drawing Primitive*. In Cohen, M.F.; Puech, C.; Sillion F.: *Fourth Eurographics Workshop on Rendering*, pp 259–266.

[Hake, 2002] Hake, G.; Grünreich, D.; Meng, L. (2002): *Kartographie*. 8. Edition, Verlag Walter de Gruyter, Berlin.

[Hardy, 2005] Lee, D.; Hardy, P. (2005); *Automatic Generalization – Tools and Models.* International Cartographic Conference, International Cartographic Association, A Coruna, Spain, July, 2005.

[Hardy, 2006] Monnot, J.L.; Hardy, P.; Lee D. (2006); *An Optimization Approach to Constraint-Based Generalization in a Commodity GIS Framework.* ICA Workshop on Generalization and Multiple Representation, Vancouver, United States, 25 June 2006.

[Hearn, 1986] Hearn, D.; Baker M. P. (1986): *Computer Graphics*. Prentice-Hall International Editions

[Heckbert, 1989] Heckbert, P.S. (1989): *Fundamentals of Texture Mapping and Image Warping*. Mater’s Thesis. Dept. of Electrical Engieering and Computer Science. University of California, Berkley, CA, USA

[Hewitt, 1991] Hewitt, W.T.; Grave, M. (1991); Roch, M.: *Advances in Computer Graphics IV*, Springer-Verlag. pp41~43.

[Hurni, 1995a] Hurni, L.; Leuzinger, H. (1995). *Principles of Cartographic Design and Their Impact on Digital Production Methods*. Proceedings of the 17th ICA International Cartographic Conference, Barcelona, pp1553-1563.

[Hurni, 1995b] Hurni, L. (1995): *Modellhafte Arbeitsabläufe zur digitalen Erstellung von topographischen und geologischen Karten und dreidimensionalen Visualisierungen*. Dissertation of Institute of Cartography, ETH Zurich.

[Huxhold, 1991] Huxhold, W.E. (1991): *An Introduction to Urban Information Systems*. New York, OUP.

[Imhof, 1972] Imhof, E. (1972): *Thematische Kartographie*. Lehrbuch der Allgemeinen Geographie, Verlag Walter de Gruyter, Berlin.

[Krishnamurthy, 2002] Krishnamurthy, N. (2002): *Introduction to Computer Graphics*. Tata McGraw-Hill Publishing Company Limited

[Jäger, 1990] Jäger, E. (1990): *Untersuchungen zur kartographischen Symbolisierung und Verdrängung im Rasterdatenformat*, Ph.D. thesis, Fachrichtung Vermessungswesen, Universität Hannover.

[Jiang, 1996] Jiang, B. (1996): *Cartographic Visualization: Analytical and Communication Tools*. In: Cartography, Band 25, Nr. 2.

[Laurini, 1992] Laurini, R.; Thompson, D. (1992): *Fundamentals of Spatial Information Systems*. London, Academy Press.

[MacEachren, 1995] MacEachren, A.M. (1995). *How Maps Work Representation, Visualization and Design*. The Guilford Press, New York/London.

[Maguire, 1991] Maguire, D.J.; Goodchild, M.F.; Rhind, D.W. (1991): *Geographical Information Systems: Principles and Applications*. Avon, Longman Scientific and Technical.

[Martin, 1991] Martin, D. (1991): *Geographical Information Systems and Their Socioeconomic Applications*. London, Routledge.

[McMaster, 1992] McMaster R. B.; Shea K. S. (1992): *Generalization in Digital Cartography Association of American Geographers*. Washington D.C.

[MSDN, 2006] MSDN Library (2006): http://msdn2.microsoft.com/en-us/library/ms536370.aspx. Microsoft Corporation.

[Newman, 1981] Newman, W.M., Sproull, R.F. (1981): *Principles of Interactive Computer Graphics*, Second Edition, McGraw-Hill, Inc.

[Ketcham, 1985] Ketcham, R.L. (1985): *A High-Speed Algorithm for Generating Anti-Aliased Lines*. Proceedings of the SID, 26(4): pp329~336

[Kraak, 1996] Kraak, M.J.; Ormeling, F. (1996): *Cartography: Visualization of Geospatial Data*. Addison Wesley Longman Limited, Essex.

[MacEachren, 1994] MacEachren, A.M. (1994): *Some Truth with Maps: A Primer on Symbolization and Design*. Association of American Geographers, Washington.

[MacEachren, 1994] MacEachren, A.M.; Taylor, D.R.F. (Hrsg.) (1994): *Visualization in Modern Cartography*. Band 2, Pergamon; Elsevier Science Ltd., Oxford.

[Madej, 2001] Madej, Ed; *Cartographic Design Using ArcView GIS*. OnWord Press

[Mairson, 1988] Mairson, H.G., Stolfi J. (1988): *Reporting and Counting Intersections between Two Sets of Line Segments* In Earnshaw R.A.: *Theoretical Foundations of Computer Graphics and CAD*. Springer-Verlag Berlin Heidelberg, pp307~326

[Murray, 1994] Murray, J.D.; van Ryper, M.(1994): *Encyclopedia of Graphics File Formats*. O'Reilly & Associates, Inc. pp57~88.

[Neudeck, 2001] Neudeck, S. (2001): *Zur Gestaltung topographischer Karten für die Bildschirmvisualisierung*. Dissertation im Fachbereich Bauingenieur- und Vermessungswesen, Universität der Bundeswehr München, München.

[Peuquet, 1990] Peuquet, D.J. and Marble, D.F. (1990): *Introductory Readings in Geographic Information Systems*. London, Taylor and Francis.

[Pitteway, 1988] Pitteway, M. L.V.; Banissi, E. (1988): *Grid Geometries which Preserve Properties of Euclidean Geometry: A Study of Graphics Line Drawing Algorithms*. In Earnshaw R.A.: *Theoretical Foundations of Computer Graphics and CAD*. Springer-Verlag Berlin Heidelberg, Pp359~387

[Rawshaw, 1988] Rawshaw, L.H. (1988): *Béziers and B-Splines as Multiaffine Maps*. In Earnshaw R.A.: *Theoretical Foundations of Computer Graphics and CAD*. Springer-Verlag Berlin Heidelberg, pp757~776

[Robinson, 1995] Robinson, A.H.; Morrison, J.L.; Muehrcke, P.C.; Kimerling, A.Jon.; Guptill, S.C. (1995): *Elements of Cartography*. 6. Edition, John Wiley & Sons, Chichester.

[Romanova, 1991] Romanova, C.; Wagner, U. (1991): *A VLSI Architecture for Anti-Aliasing*, In *Advances in* Grimsdale, R.L.; Strasser, W.: *Computer Graphics Hardware IV*. Springer-Verlag, pp75~90

[Saux, 2003] Saux E. (2003): *B-Spline Functions and Wavelets for Cartographic Line Generalization*. Cartography and Geographic Information Science, 30(1):33~50

[Slocum, 2005] Slocum, T.A; McMaster, R.B.; Kessler, F.C.; Howard, H.H. (2005): *Thematic Cartography and Geographic Visualization*. 2. Edition, Prentice Hall

[Smith, 1995] Smith, C.; Blachman, N. (1995): *The Mathematia Graphics Guidebook*, Addison Wesley Publishing Company

[Skala, 1985] Skala, V. (1985): *An Interesting Modification to the Bresenham Algorithm for Hidden Line Solution.* In EarnShaw R.A.: *Fundamental Algorithms for Computer Graphics*, Springer-Verlag, pp593~602

[Spiess, 1983] Spiess, E. (1983): *Thematische Kartographie*. Lecture script from Institute of Cartography, ETH Zurich, unpublished.

[Spiess, 1995a] Spiess, E. (1995), *Skript of Kartographie Grundzüge: Kartengestaltung*. Lecture script from Institute of Cartography, ETH Zurich, unpublished

[Spiess, 1995b] Spiess, E. (1995): *Some Problems with the Use of Electronic Atlases*. 9. Konferenz der LIBER Gruppe der Kartenbibliothekare, Workshop "Digitale Karten", The LIBER Quarterly, Vol. 5, Nr 3.

[Star, 1990] Star, J.; Estes, J. (1990): *Geographical Information Systems: An Introduction*. Englewoods Cliffs, New Jersey, Prentice Hall.

[Staufenbiel, 1973] Staufenbiel W. (1973): *Zur Automation der Generalisierung Topographischer Karten mit besonderer Berücksichtigung grossmassstäbiger Gebäudedarstellungen.* Dissertation an der Fakultät für Bauwesen der Universität Hannover.

[SWISSTOPO] Federal Office of Topography swisstopo.
http://www.swisstopo.ch/.

[Tyner, 1992] Tyner, J. (1992): *Introduction to Thematic Cartography*. Prentice Hall, Upper Saddle River; New Jersey.

[Thalman, 1987] Thalman, N. M.; Thalmann D. (1987): *Image Synthesis, Theory and Practice*, Springer-Verlag

[TopoKarto, 1996] TopoKarto, *Richtlinien für die kartographische Bearbeitung von Landeskarten*. Wabern.

[Veregin, 1999] Veregin, H. (1999): *Line Simplification, Geometric Distortion, and Positional Error*. Cartographica, 36(1):25~39

[Weszka, 1980] Weszka J. S. (1980); *A Comparative Texture Classification Experiment.* In Freeman H.; Pieroni G. G. (1980); *Map Data Processing*. Academic Press

[Wise, 2002] Wise, S. (2002), *GIS Basics*, London: Taylor & Francis, 2002

[XML, 2004] XML (2004): *Extensible Markup Language (XML) 1.1*. World Wide Web Consortium (Hrsg.). http://www.w3.org/TR/2004/REC-xml11-20040204/ (version: 05/2006).

[Yamaguchi, 1988] Yamaguchi, F. (1988): *Curves and Surfaces in Computer Aided Geometric Design*, Springer-Verlag. pp169~216

Curriculum Vitae

Personal Data

Name:	Fei He
Date of Birth:	June 12, 1975
Place of Birth:	Yunnan, China
Nationality:	P. R. China
Marital status	Married

Education

1993-1997	B.A. at Xidian University, Xi’an, China
1997-1998	Master courses training at Northwest Polytechnic University, China
1997-2000	M.S. at Xi'an Institute of Optics and Precision Mechanics, The Chinese Academy of Sciences
2000-2001	Pre-doctorate course training at Xidian University, China
2002–2007	Doctoral studies at Institute of Cartography, ETH Zurich