# A customizable method for handling line joins for map representations based on shader language

**Article** *in* Annals of GIS · June 2016

DOI: 10.1080/19475683.2016.1191546

CITATION
1

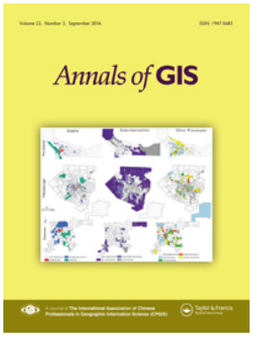
READS
501

**4 authors**, including:

Guoqiang Peng
Nanjing Normal University
**9** PUBLICATIONS **64** CITATIONS
SEE PROFILE

Songshan Yue
Nanjing Normal University
**39** PUBLICATIONS **495** CITATIONS
SEE PROFILE

Guonian Lü
Key Laboratory of Virtual Geographic Environment, Ministry of Education of PRC, …
**179** PUBLICATIONS **1,859** CITATIONS
SEE PROFILE



Annals of GIS

ISSN: 1947-5683 (Print) 1947-5691 (Online) Journal homepage: http://www.tandfonline.com/loi/tagi20

# A customizable method for handling line joins for map representations based on shader language

Peibei Zheng, Guoqiang Peng, Songshan Yue & Guonian Lu

**To cite this article:** Peibei Zheng, Guoqiang Peng, Songshan Yue & Guonian Lu (2016) A customizable method for handling line joins for map representations based on shader language, Annals of GIS, 22:3, 215-233, DOI: 10.1080/19475683.2016.1191546

**To link to this article:** https://doi.org/10.1080/19475683.2016.1191546

Published online: 30 Jun 2016.

ANNALS OF GIS, 2016
VOL. 22, NO. 3, 215–233
http://dx.doi.org/10.1080/19475683.2016.1191546

Taylor & Francis
Taylor & Francis Group

# A customizable method for handling line joins for map representations based on shader language

Peibei Zheng[a,b,c], Guoqiang Peng[a,b,c], Songshan Yue[a,b,c] and Guonian Lu[a,b,c]

[a]Key Lab of Virtual Geographic Environment, Ministry of Education, Nanjing Normal University, Nanjing, China; [b]Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing, China; [c]State Key Laboratory Cultivation Base of Geographical Environment Evolution (Jiangsu Province), Nanjing, China

**ABSTRACT**

The rendering of line joins is important for representing linearly changing information and for allowing map readers to develop a more comprehensive understanding of the real world from maps. Although a range of software-based and graphic processing unit (GPU)-accelerated rendering methods have been developed to render wide lines, linear map symbols are much more complicated compared to commonly used line types in drawing geometric lines, and the rendering of lines joins should consider a filling process that is continuous, rational and efficient. This paper proposes an improved method to generate and adjust the *U*–*V* parameters (which are used to control the rendering process of linear map elements within the GPU shader language) of the vertices in line joins to provide a customizable approach for map designers to implement different drawing effects of line joins for drawing different line elements. Based on the customizability, the continual and rational rendering of line joins are studied. Two experiments are conducted to demonstrate the method's feasibility and efficiency; the results demonstrate that line joins can be effectively and rationally drawn with a variety of linear map symbols. The efficiency of rendering linear map elements is also improved with this method.

**ARTICLE HISTORY**
Received 26 March 2016
Accepted 15 May 2016

**KEYWORDS**
Line joins; linear map elements; shader language; GPU acceleration

## 1. Introduction

The symbolization of various map elements is important in cartography research (MacEachren 2004; Goodchild, Yuan, and Cova 2007; Robinson et al. 2012). With the development of research on geographic information system (GIS) and computer visualization technologies, the representation of maps largely depends on the rendering of symbolized geospatial objects (Hearnshaw and Unwin 1994; Dymon 2003; Peng 2014). Vector data, raster data, point cloud data and other types of data are commonly used to depict the real world for map representations (Goodchild, Haining, and Wise 1992; Neteler et al. 2012; Virtanen et al. 2015). Among these data types, the rendering of vector data is an important research topic (Yuan 2001; Schneider, Guthe, and Klein 2005; Haunert and Sering 2011; Chen, Wen, and Yue 2015; Lin et al. 2014; Guo, Wu, and Xie 2015; Voženílek 2005). According to the OpenGIS Simple Feature Access standards (also called ISO 19152), a range of geometry types (such as points, line strings, polygons, multi-points, multi-polygons etc.) can be implemented for simple geometry feature (Herring 2011). Symbolization methods are necessary to represent these features in a map; the Styled Layer Descriptor (SLD) specification and the Symbology Encoding specification (both proposed by the Open Geospatial Consortium [OGC]) have outlined the basic symbolization attributes of features, which are called Symbolizers (OGC 2015a, 2015b). Three basic Symbolizers can be employed to symbolize different geometry feature types: the LineSymbolizer (linear map symbols), the PolygonSymbolizer (area map symbols) and the PointSymbolizer (point map symbols). These standards and specifications mainly focus on the description of the symbolization rules for map elements, and their implementation largely depends on the detailed rendering platform and techniques (Dietze and Zipf 2007).

From the perspective of rendering techniques on GIS and computer visualization, the basic process for rendering map elements can be summarized as *geometry assembly*, *coordinate transformation* and *pixel filling* (MacEachren and Taylor 1994; Stoll, Gumhold, and Seidel 2005; Wen et al. 2013; Batra et al. 2005). In the *geometry assembly* process, the original coordinates or coordinate list should be converted to the new drawing geometries to represent them in a map (e.g. the

**CONTACT** Guoqiang Peng geop1987@gmail.com

coordinates of a point must be assembled as a rectangle, a circle or other geometry types). In the *coordinate transformation* process, user-view-related transformation parameters should be applied for each vertex of a drawing geometry (such as using the world-view-projection transformation matrix to obtain the coordinates onscreen). In the *pixel filling* process, the final representation effect of a geometry feature is generated by using different symbolization rules (e.g. using a solid colour or a texture image to fill the composed rectangle of a point). According to this basic rendering process and compared to point and area map elements, the drawing of a linear map element should satisfy the requirements that the drawing geometry is computed as a buffer area based on a specific line width (in the *geometry assembly* process) and that the linear map symbol should be repeatedly filled along the line direction (in the *pixel filling* process).

Therefore, the difficulty of rendering linear map elements is mainly the result of the demand of drawing linear symbols along the line direction. Changes in the line direction at the position of line joins increases this difficulty. A mathematical line consists of successive coordinates and contains no 'visual' line joins. The 'Gap' and 'Overlap' drawing effects can appear when applying a specific line width for map representation aims. Line joins must be handled properly to avoid these incorrect and unaesthetic drawing effects. In the SLD specification, the Stroke element of the LineSymbolizer is used to define the graphical Symbolization parameters (which are mainly derived from SVG/CSS2 standards) for linear map elements (Zipf 2005; Bos et al. 2015). The handling of line joins in the Stroke element is demonstrated as 'LineSymbolizer/Stroke/CssParameter[@name = "stroke-linejoin"]'; and the 'stroke-linejoin' SvgParameter elements are used to enumerate the permitted values for line joins, namely, 'mitre', 'bevel' and 'round' (according to the World Wide Web Consortium recommendation) (SVG 2015). These norms provide clues to implement these line join types, but the methods that are used to render them are not described and are system-dependent (Deoliveira 2008; Iosifescu, Hugentobler, and Hurni 2010; Steiniger and Hunter 2012).

The rendering of line joins with certain linear symbols during the representation process of maps is tightly related to user cognition regarding the spatial information: line joins reflect both gradually changing (i.e. a driving route that consists of a list of GPS coordinates can be presented by using gradually changing colours to represent different driving speeds, and the line joins' colour should also reflect this gradually changing information) and suddenly changing information (i.e. the line joins in a human-assigned boundary line would be better handled as sharp corners to reflect this absolute, suddenly changing information) (Garlandini and Fabrikant 2009; Dong, Ran, and Wang 2012; Harrie, Stigmar, and Djordjevic 2015; Iosifescu Enescu et al. 2015). Traditional geometric drawing methods have mainly focussed on the generation of different shapes for line joins (or continual rendering); the rational rendering of different spatial objects has not been considered as frequently. In this paper, continual rendering means that a line join should not have any gaps or overlaps between two line segments; rational rendering means that the rendering of a linear map element should consider the specific characteristics of the ready-to-be-presented spatial objects. Because SLD specification does not bind with any specific rendering technologies, the rendering of line joins depends on the same rendering techniques as drawing linear map elements.

In the GIS and computer visualization field, a range of software-based geometric drawing approaches have been studied for rendering linear map elements, such as the Graphic Drawing Interface Plus (GDI+), which is used in ShapeMap, DotSpatial and System for Automated Geoscientific Analyses (Steenbeek et al. 2013; Aydinoglu and Bilgin, 2015; GDI 2015); the Anti-Grain Geometry (AGG) library, which has been imported into Mapnik, MapServer and MapGuide (Lienert et al. 2012; AGG 2015); the Cairo library, which has been imported into Mapnik and GRASS (Cairo 2015); and other rendering methods such as Skia (Yoo et al. 2014) and Qt Painter (Qt, 2015). As shown in Figure 1, the GDI+ method defines a *Pen* object and sets the line join type to draw line strings. The AGG library first defines a path, which is then converted to stroke (including the setting of line join). The Cairo library also defines a path by a specific function. When drawing solid lines with these methods, line joins can be correctly handled; however, when drawing dashed lines, the flexibility of drawing line joins is limited. As shown on the right-hand side of Figure 1, some incorrect filling effects can occur in a line join. And for some commercial GIS software, the rendering methods they employed in their own programs are also GDI+, AGG, Cairo and some other rendering methods. Amending these odd polygons in a line join usually requires a significant amount of computation to adjust the values of the dash-offset parameter, which would significantly influence the rendering efficiency.

Graphic processing unit (GPU) rendering methods, which are characterized by hardware acceleration, have been studied as promising approaches to improve the drawing efficiency and to achieve more dynamic representation effects (Rost et al. 2009; Deng, Deng, and Xu 2013). The original line drawing methods from

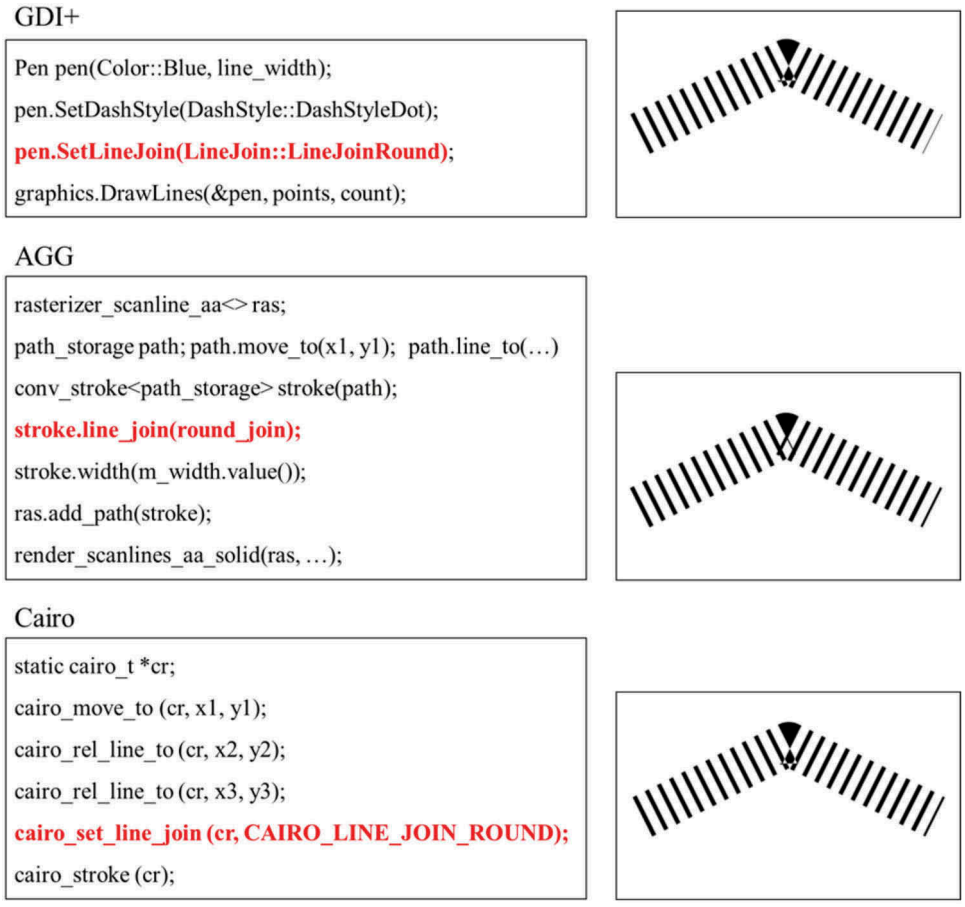



**Figure 1.** Addressing line joins with software-based rendering methods.

the Open Graphics Library (OpenGL) or DirectX 3D (D3D) do not handle the 'Gap' and 'Overlap' problems in line joins (Woo et al. 1999; Luna 2008). To address this issue, shader technology was employed to help render line strings of different line widths. The project gl-agg was developed to draw anti-aliasing lines (gl-gl-agg 2015), the Cairo library includes OpenGL to support GPU rendering (Nilsson and Reveman 2004), and the dashed-line rendering method has been studied to draw dashed lines based on the shader language (Rougier 2013).

However, when rendering linear map elements with different symbols, these methods are usually integrated with a method that combines different lines (with different line widths, in different fill colours and using different dash offsets and intervals) to process the symbolization (Schnabel 2005; Mei and Li 2007). The requirement of drawing multiple geometry primitives in this method still limits the drawing efficiency. Recently, a method that uses a symbol structure-related 'function' in shader program to build and render linear map symbols has been designed and developed to relieve this problem (Yue et al. 2016). Based on this shader function-based rendering method, the *geometry assembly* process can be implemented by using triangulation (also called tessellation), and the *pixel filling* process is implemented by using GPU computation. As shown in Figure 2(b), the 'Gap' and 'Overlap' problem can be solved by using different triangulation methods to generate different 'shapes' (the line join types, as described in the SLD specification); the mitre type is presented in Figure 2(b). Although the handling method for line joins and line caps has already been introduced in the shader function-based method, the rendering process still does not rationally handle line joins. As introduced in Figure 2(a and b), the continual rendering issue has been addressed within previous shader function-based methods, but an irrational rendering problem may still exist, as presented in Figure 2 (c). A rational rendering result is presented in Figure 2(d).

Based on the above analysis, this paper focuses on the rendering of line joins and considers both the drawing effect (continual and rational rendering) and

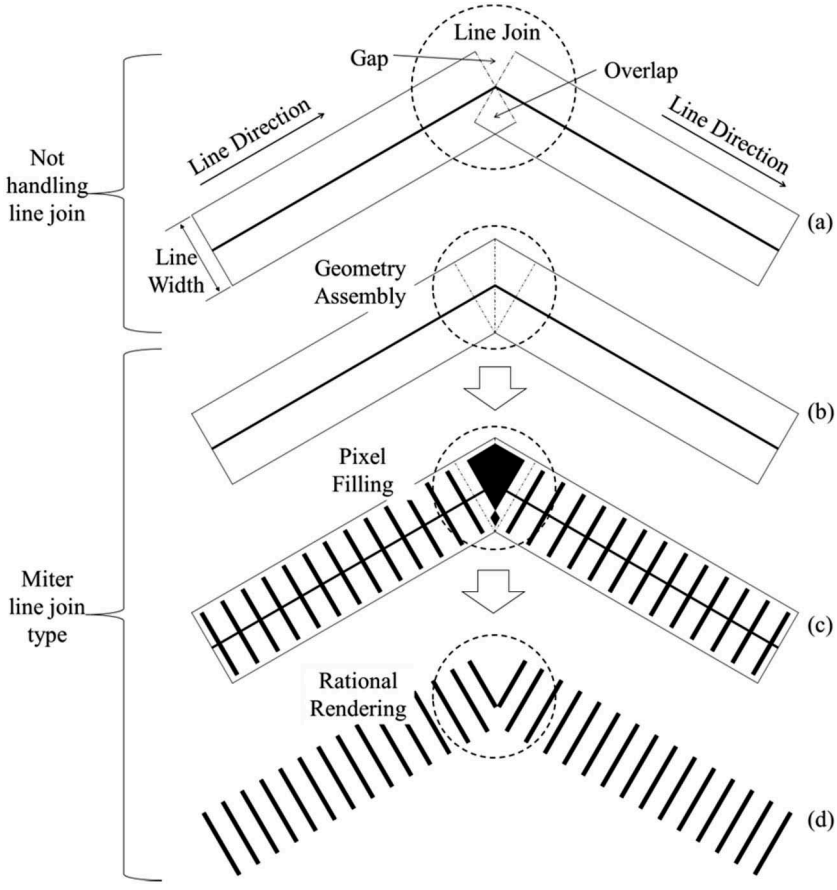



**Figure 2.** Example of the handling process for line joins.

the drawing efficiency. By improving the previous shader function-based linear map symbol rendering method, the proposed method provides a customizable way for map designers to build different drawing effects of line joins. The shape of a line join can be customized by using different tessellation methods, and the filling of a line join can be customized by generating and adjusting the shader computation-related parameters of each vertex in a line join. The rational rendering effect can be achieved when combined with the structural information of different linear map symbols. In addition, this customizable solution employs the GPU shader language to fill line joins, which is based on GPU computation in a per-pixel rendering mode, to satisfy the drawing efficiency demands.

The remainder of this article is organized as follows. Section 2 discusses related work on rendering line joins. Section 3 introduces the basic methodology of the proposed method for rendering line joins, which is customizable and extendable so that users can implement different line join types. In Section 4, the method for addressing the continual and rational rendering of line joins is presented; moreover, four types of commonly used linear map symbols are explained in detail. Section 5 introduces and demonstrates the capability of the proposed method in practical map rendering applications. Finally, conclusions and a discussion are presented in Section 6.

## 2. Introduction of the shader function-based rendering method

The GPU-accelerated rendering method is widely used to render 2D maps and 3D scenes. The programmable rendering pipeline term is widely used to describe the method that employs shader programs to control the rendering process in a per-pixel mode. The shader function-based method for building and rendering linear map symbols was designed to help draw linear map elements based on this basic characteristic. Both the central processing unit (CPU) and GPU are involved in this method: the CPU is mainly in charge of the geometry building process, and the GPU mainly computes the pixel's position on the screen and the pixel's colour. As shown in Figure 3(a), the shader function-based method employs a vertex buffer and an index buffer

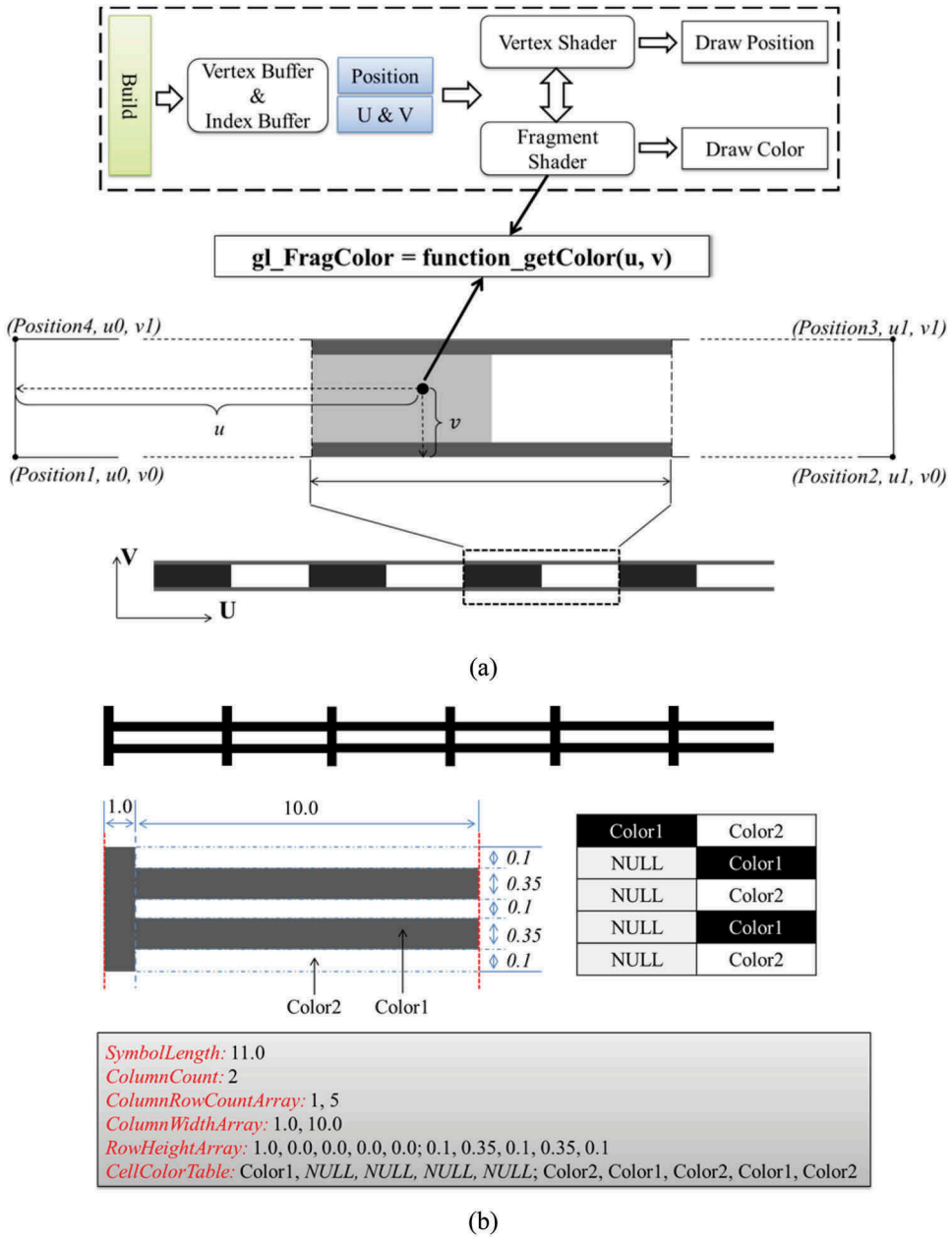



**Figure 3.** The shader function-based linear map symbol rendering method.

to hold the information of each vertex (after tessellating lines according to the line width), and the draw position and draw colour are addressed by the shader program. A set of parameters (*U* and *V*) are proposed to assist in the assignment of the colour of each pixel. *U* indicates the distance from the current pixel to the start of a line, and *V* indicates the distance from the current pixel to the vertical bottom of a line. Thus, the *pixel filling* process in this method can be addressed with the related *U*–*V* function, or the 'function_getColor(u, v)', as shown in the centre of Figure 3(a).

The rendering of line joins is handled alongside the filling process of a line. The symbol structure information should be transmitted into the fragment shader program and thus support the computation of 'function_getColor(u, v)'. The basic method is presented in Figure 3(b). In total, 6 main parameters should be input into the fragment shader program: *SymbolLength*, which indicates the total length of a linear map symbol (the length of the example symbol is 11.0); *ColumnCount*, which indicates the number of columns (the number of columns of the example symbol is 2); *ColumnRowCountArray*, which indicates how many rows are in each column (the first column has one rows, and the second column has five rows); *ColumnWidthArray*, which contains the width value of each column (the

widths of the two columns are 1.0 and 10.0); *RowHeightArray*, which indicates the height value of each row (the first column has only one row, with a height of 1.0, and the second column has five rows, with heights of 0.1, 0.35, 0.1, 0.35 and 0.1); and *CellColorTable*, which contains the colour value of each column–row cell (Color1 and Color2 are assigned to each cell).

Using shader function-based rendering methods to draw linear map elements can significantly improve the drawing efficiency because the *pixel filling* process is conducted with the shader program and computed by the GPU. However, the handling of line joins in this method continues to suffer from the problem of drawing effects. From the *pixel filling* aspect, line joins should be continually drawn, and from the rational drawing aspect, line joins should be drawn based on the symbol structure to retain the semantic information of a line string.

## 3. Methodology

The basic methodology of the proposed method is presented in Figure 4. Different types of line joins are first built with the corresponding triangulation method to solve the 'Gap' and 'Overlap' that are formed by two line segments in a line string. As shown in Figure 4(a), the 'none', 'mitre', 'round' and 'bevel' types of line joins can be customized by the users.

The rendering of line joins can also be specified by the users based on different shapes of line joins. Figure 4(b) shows that the rendering of linear map elements can be regarded as the repeated filling of linear map symbols. The symbol structure can be described by a colour table, which organizes the colour information of a symbol in a column–row array, and the colour of a pixel is selected from this colour table based on the parameters *U* and *V*, which are introduced in Section 2. The *U*–*V* values for each pixel within a

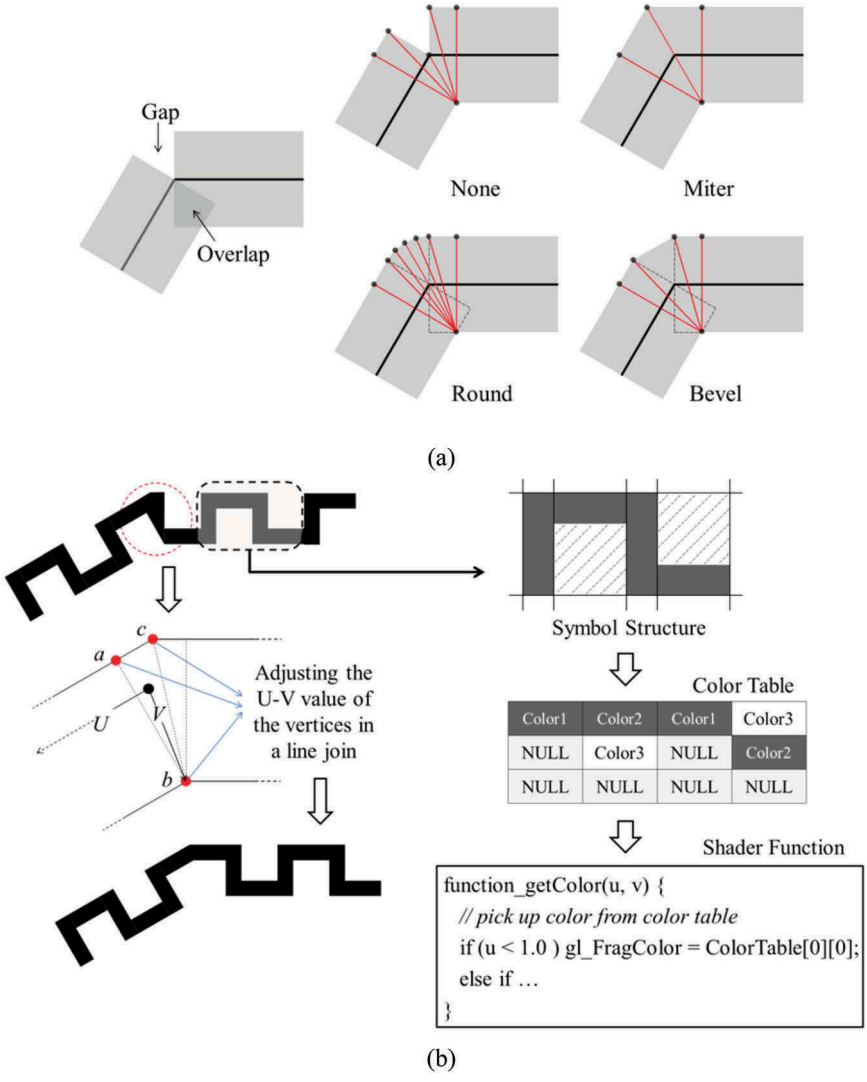



**Figure 4.** Basic methodology for handling line joins based on the shader language.

triangle are determined by the corresponding *U–V* values of the three vertices (interpolated by the GPU computation). Therefore, the colour of the pixels in the line join can be customized by the users by adjusting the *U–V* value of the vertices in a line join. In Figure 4 (b), the line join is constructed as two triangles (the left triangle is *a–b–c*). If the *U–V* value of *a* and *b* makes the filling of this triangle by selecting colour from the first or third column in the corresponding colour table, the line join would be filled as a black triangle, which is not in accordance with the symbol's structure. The drawing effect can be changed by adjusting the original *U–V* value. As shown in the lower left of Figure 4(b), one possible modified drawing effect can be achieved by adjusting the *U–V* value in the second column. This adjustment process can be customized by the user according to different linear map symbols and drawing demands.

Using this customizable method, the drawing of line joins can be addressed on a per-pixel basis. Generating different rendering effects of a line join mainly depends on the combination of CPU and GPU computation. The structural information (which mainly contains the colour table information) of a symbol can be built during CPU computation, and the *U–V* values of a line join can be assigned and adjusted based on the judgment of whether the line join is in the appropriate cells of the colour table. Different types of linear map symbols should be separately handled to make the entire line drawing rational.

## 4. Line join handling method based on shader language

### 4.1. Customizable handling process

To ensure that the consistency of drawing linear map symbols in the vertical direction is maintained, line joins are composed of a range of two-segments. As shown on the left-hand side of Figure 5(a), the original line string *Pb–P–Pa* is divided into three parts: *Pb–P1*, *P2–Pa* and *P1–P–P2*. *Pb–P1* and *P2–Pa* can be regarded as rectangles along the direction of the line, and the region formed by *P1–P–P2* is the line join (using the mitre type). To continually fill a line string, the *P1–P–P2* region takes on a conjunction role to inherit the drawing of *Pb–P1* and to facilitate the drawing of *P2–Pa*. The basic triangulation method for the mitre line join type is presented on the right-hand side of Figure 5(a). Points *bR* and *bL* are introduced to explain the expansion from *Pb* with a specific line width. *bR1* and *bL1* are generated based on the line *Pb–P*. Similarly, *aR, aL, aR1* and *aL1*

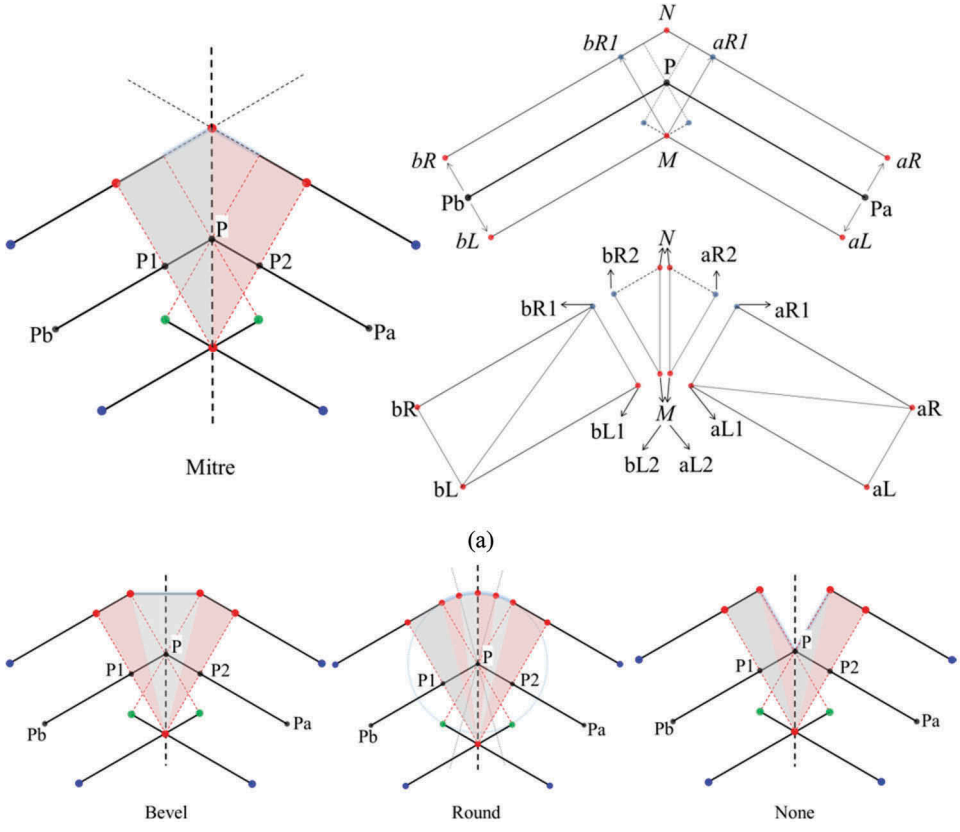



**Figure 5.** Basic rendering process for line joins.

can be computed for the line *P–Pa*. To eliminate the 'Gap' and 'Overlap', point *N* is computed by intersecting line *bR–bR1* and line *aR–aR1*, and point *M* is computed by intersecting line *bL–bL1* with line *aL–aL1*. Therefore, the final path of the line string is *bR–N–aR–aL–M–bL*. The final triangulation of this line join can be determined by the triangles that are shown in the right-bottom portion of Figure 5(a). In addition, the tessellation methods for the other line join types (bevel, round and none) are presented in Figure 5(b).

In the process of generating triangles, the parameter *U* and *V* should be assigned to each vertex for all triangles. According to the above analysis, the *U* parameter indicates the length along the line direction. Therefore, when attempting to smoothly draw the entire line string, the *U* value of *P1* and *P2* should be the same or match the structure information of the filled symbol. In addition, the filling of the line join should also consider the symbol structure information. As introduced in Section 3, the symbol structure information is mainly described by using a colour table and transmitted into the fragment shader program. In the shader program, the colour of each pixel in the line is picked based on the *U* and *V* values (interpolated based on the corresponding vertices' *U–V* values).

When applying this colour-picking scheme to draw line joins, the *U* and *V* parameters should be assigned based on the combination of line length information and symbol structure. Vertices in the triangles of line join areas should assigned varying *U* and *V* values according to the requirements of the rendering effect. Figure 6 shows that *U* is assigned based on the symbol length, the *U* value of the previous segment's end point (i.e. point *P1* in Figure 5) and the next segment's starting point (i.e. point *P2* in Figure 5); *V* is generally set to 0 or 1 based on the line width. This process can be conducted within the CPU computation along with triangulation processing.

In Figure 6(a), the left starting point of the line join (*bR2* and *bL2*) is assigned a *U* value that is equal to the last segment's end point (*bR1* and *bL1*), and the right end point of the line join (*aR2* and *aL2*) is assigned a *U* value that is closest to an integral multiple of the symbol length. In Figure 6(b), the *U* value of each vertex in the line join is assigned the same value as the last segment's end point (*bR1* and *bL1*), and the start of the next segment (*aR1* and *aL1*) is also assigned the same *U* value. In Figure 6(c), the left starting point of the line join (*bR2* and *bL2*) is assigned a *U* value that is closest to an integral multiple of the symbol length, and the *U* value of *N* is assigned by adding the length of *bR2–N*. For the right-hand part of the line join (i.e. triangle *aL2–N–aR2*), the *U* value of each vertex is assigned as the mirror of the left triangle (*bL2–N–bR2*). With different user-defined *U* and *V* parameters, the colour-picking function in the shader language can fill line joins to achieve various effects.

### 4.2. Continual and rational rendering

The continual rendering of a line join mainly focuses on building a shape (via the triangulation of different methods) and filling this shape based on the two connected line segments (by assigning *U* and *V* values). Based on this method, the rational drawing of a line join focuses on how to adjust the *U* and *V* values of the vertices in a line join such that the entire line string can be more clearly rendered and the geospatial information can be rationally depicted. A typical example is the black-and-white dashed symbol in Figure 7. When symbolizing a line string with this symbol (given that the background of the map application is white), if a line join is filled with the white colour, map readers can be easily lead to think that there are two lines here and that this white region belongs to neither of these lines. Moreover, map readers may find it difficult to determine where the turning point is because the line join area (in white) is somewhat long.

Therefore, the filling of line joins should consider the symbol structure and adjustment of the *U* and *V* values of vertices in line joins based on the rational drawing rules. These rules are defined by map designers, and they can essentially be implemented by determining which columns of the symbol structure are the 'background' columns; moreover, they should not be used to fill line joins. The rational drawing method is explained in Figure 7 by using an example symbol. In the process of assigning *U* and *V* values for vertices in a line join, the original *U* value is computed based on the distance from the current vertex to the start vertex; if the original *U* value belongs to the white cell, it should be adjusted to the nearest black cell. As shown in the bottom of Figure 7, if the *U* value of *bR2* and *bL2* is closer to the black cell in the left segment, the line join should be connected with the left segment and filled with a black colour; otherwise, the line join should be connected to the right segment. Thus, the line join can be filled with black colour, and the entire line string can be better represented.

A simple method to determine the side on which the line join is to be adjusted is to compare the *U* value of *bR2* and *bL2* with the half value of the width of the white cell. The width of the white cell can be easily obtained from the symbol's structure information. The *U* value is divided by the symbol's total length to obtain the round number of symbol count:

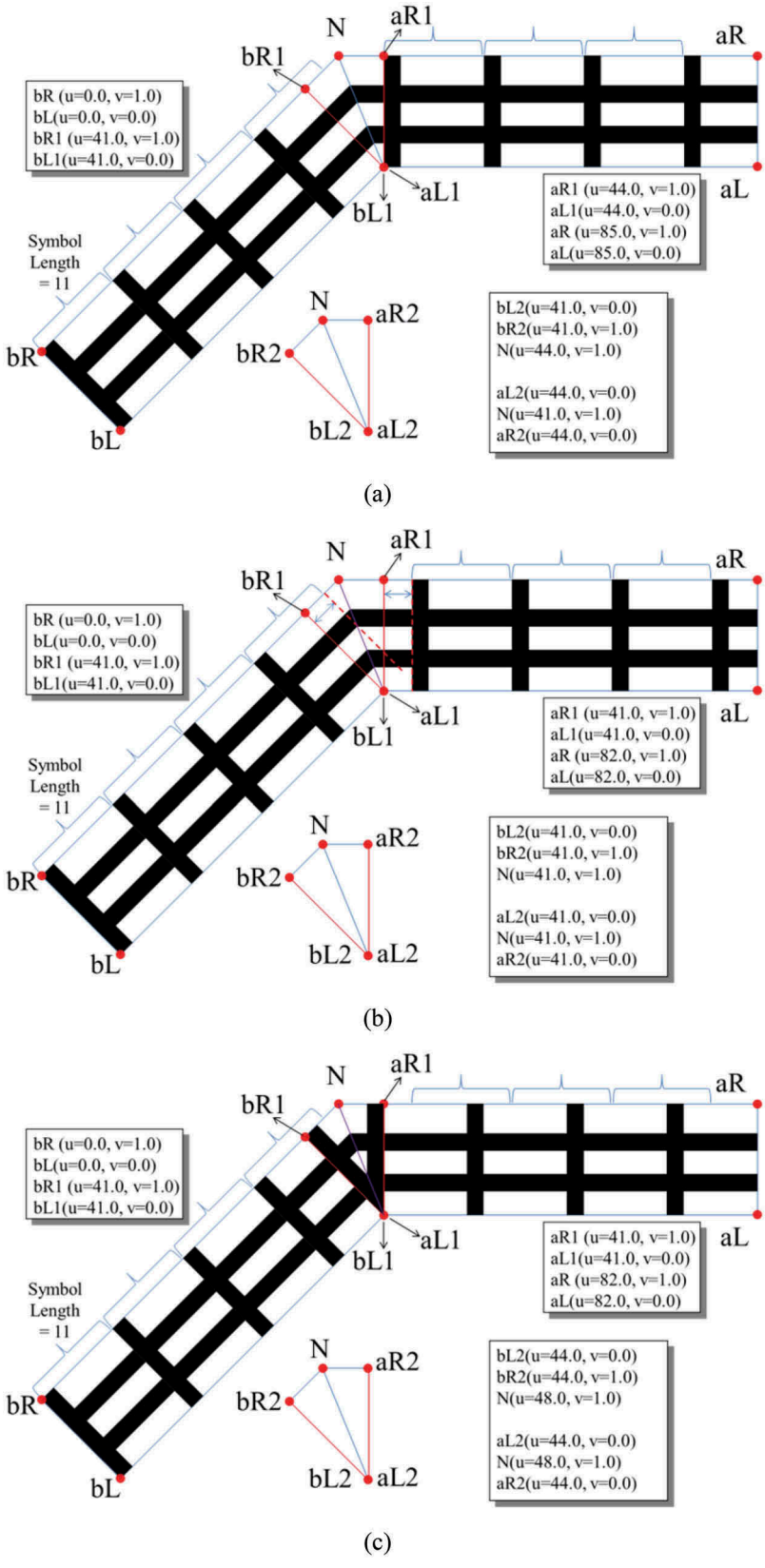



**Figure 6.** Three different drawing effects customized using user-defined parameters.

symbolCount = (int) (*U*/*SymbolLength*). The new *U*′ value can be computed by subtracting the length symbolCount × *SymbolLength*, which indicates the length between the start of a single symbol and current pixel, from the original *U* value. The assessment of whether the *U* value of *bR2* and *bL2* is greater than

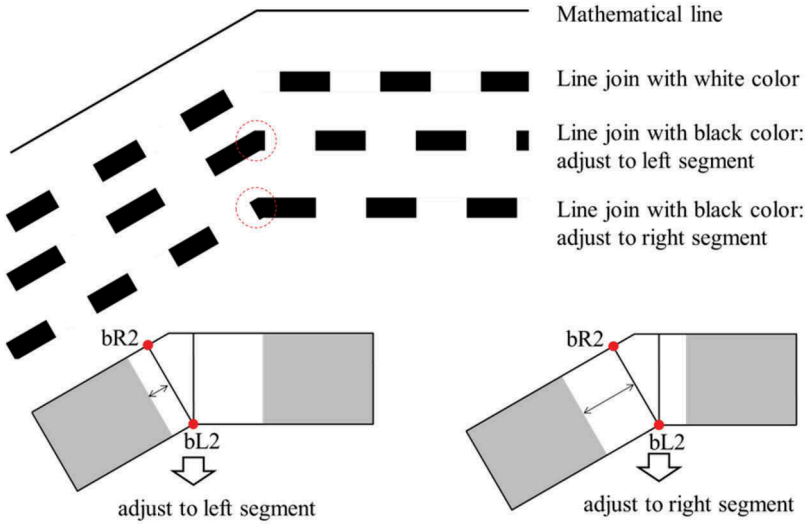



**Figure 7.** Adjusting the *U* and *V* value to draw line strings rationally.

half of the white cell can then be accomplished by assessing whether the *U′* value is larger than the half width of the white cell.

### 4.3. Applying to different line types

According to the SLD specification and other commonly used GIS map applications, the customizable handling method for line joins is implemented for four basic line types: solid lines, dashed lines, gradient colour lines and transition lines (i.e. width-changed lines).

#### 4.3.1. Solid lines

Solid lines are repeated with the same colours along the line direction; however, the filling colour varies in the vertical direction. Therefore, the *pixel filling* process for a solid line is mainly controlled by the *V* parameters (the corresponding colour table thus contains only one column). Thus, the handling of line joins for this type of line can be customized by assigning *V* values for the vertices in the triangles of a line join, and the *U* value can be assigned based on the line length or even assigned as a constant value for each vertex (i.e. *U* parameterization is not required for solid lines). The handling of a line join for this type of line is presented in Figure 8. For different solid line types that shown in the left of the Figure 8, the handling method of line joins is same. The shape of this line join is customized by the different tessellations, and the continual filling of this line join is customized by the *V* parameter.

#### 4.3.2. Dashed lines

In contrast to the general dashed lines that are defined using a range of line segments with specific dash widths, the dashed linear map elements are described by repeatedly drawing a linear map symbol along the line direction. Regarding this line type, the handling of line joins can be customized using different fill patterns. As shown in Figure 9, the *U* value is the dominant parameter for the repetitive filling of linear map symbols. Considering the feature of a linear map symbol, the *U* parameter should be dynamically computed

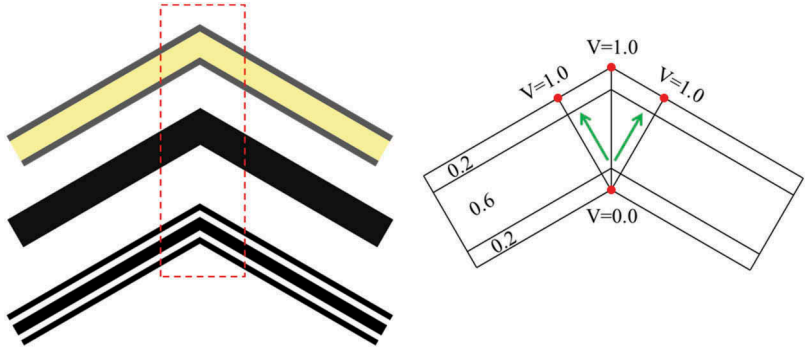



**Figure 8.** Addressing line joins for solid linear map elements.

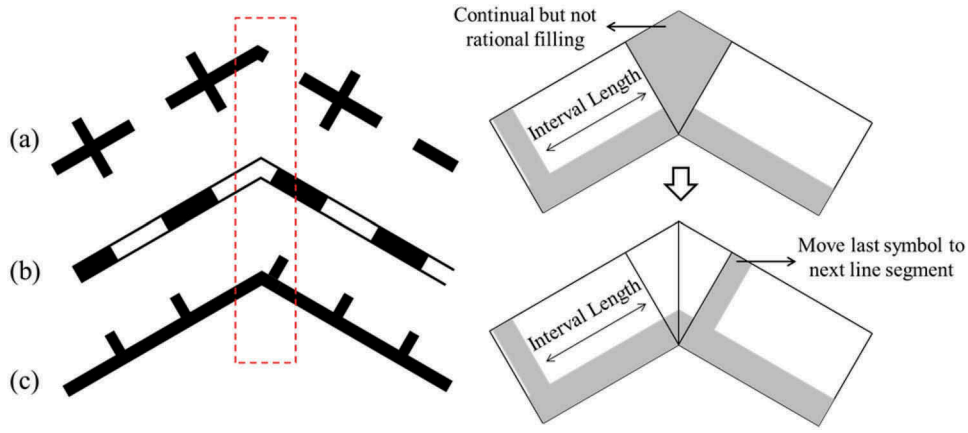



**Figure 9.** Handling line joins for dashed linear map elements.

according to the symbol structure. For the 'abandoned railway symbol' (black-and-white dashed line), line joins should be filled with a black colour (shown in Figure 9(a)). For the 'railway symbol' (black-and-white dashed line with black outlines), line joins can be freely filled with black or white (shown in Figure 9(b)). For the 'boundary symbol' (single dashed line with a bottom baseline), line joins should be filled with the bottom line rather than the bulgy segment (shown in Figure 9(c)). Based on these rules, the *U* value of the vertices in a line join can be computed based on the length of the line string and the length of a symbol.

### *4.3.3. Gradient colour lines*

Gradient colour lines are represented by filling wide lines with colour ribbons or gradient-changing colours, and this 'change' should usually follow a direction (e.g. along the line direction or perpendicular to the line direction). By assigning a set of successive values of the *U* parameters, the gradual change in line joins can be filled with the gradient colour. In addition, if map applications require line joins to be drawn as obvious turning information, the drawn line joins can be filled with a constant colour to distinguish them from other line segments. As shown in Figure 10, both gradual and sudden changes can be implemented with the proposed customizable method. At the left-top of Figure 10, the line join is filled with gradually changing colours; in the left-middle of Figure 10, the line join is filled with a constant colour value, which represents a sudden change effect. For the perpendicular gradient colour lines, line joins are mainly handled by the *V* value. The gradual changing effect of a line join can be customized by assigning different *V* values, which is

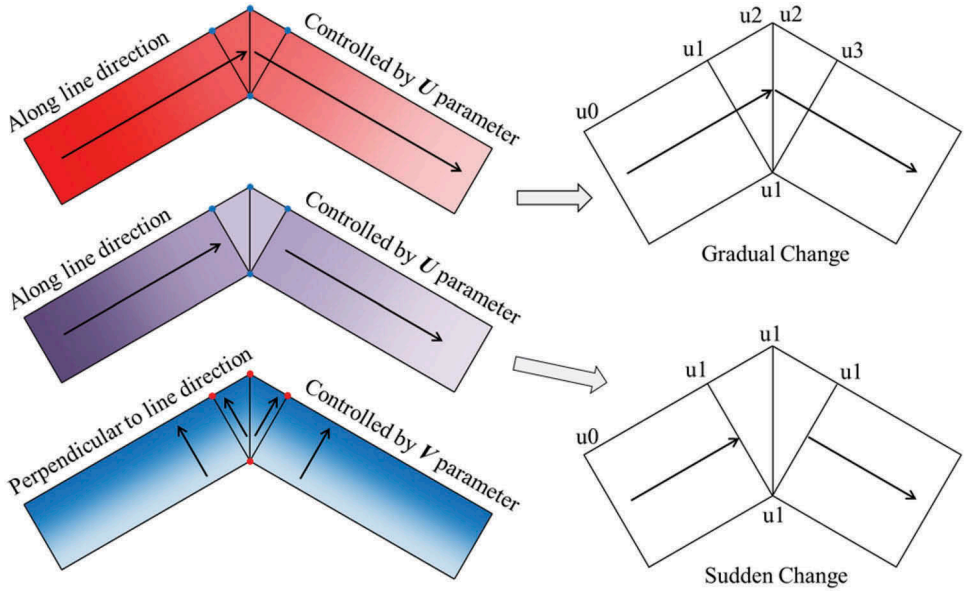



**Figure 10.** Handling line joins for gradient colour linear map elements.

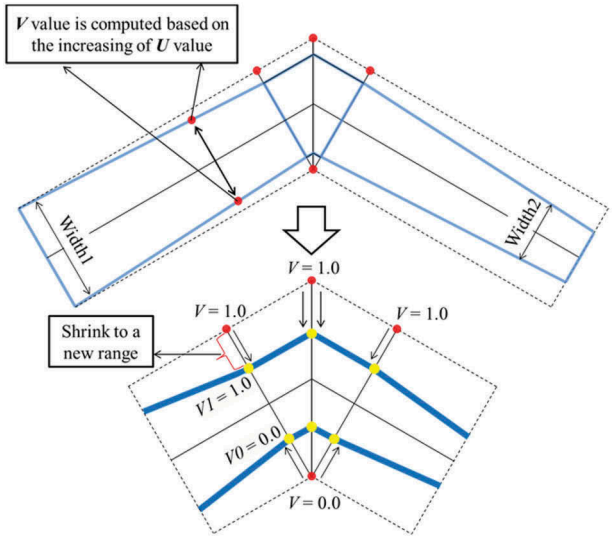



**Figure 11.** Handling line joins for transition linear map elements.

similar to the situation along the line direction. A gradually changing perpendicular to the line direction is presented at the left-bottom of Figure 10.

#### 4.3.4. Transition lines

The most distinct feature of transition lines compared with other line types is a changing line width along the line direction. Using the shader language, the changing of line widths can be implemented by filling the transparent colour if the *V* parameter of the current pixel is beyond the current pixel's corresponding line width (the current pixel's line width can be computed based on the *U* parameter). As shown in Figure 11, the *V* value of a pixel is determined by the changing *U* value, and the range of *V* (0.0–1.0) is reduced from the original edge to the new edge. The vertices of a line join can be assigned a *V* value equal to 1.0 (up) or 0.0 (bottom). Solid lines, dashed lines and gradient colour lines can all be applied to render transition lines. Based on this *V*-value-shrinking method, the handling of line joins for transition lines can be conducted using the same methods as for solid lines, dashed lines and gradient colour lines. The mitre line join type is illustrated in Figure 11. For other line join types, the *V*-value-shrinking method is processed the same way as the mitre type. With this method, the line width of a line join would be processed as a constant value to make a gradual transition.

## 5. Experiments

To validate the capabilities and practicability of the proposed method, an experimental map rendering system was designed. The system was realized using the C++ programming language and implemented the shader program for this customizable method in OpenGL ES 2.0. The basic implementation processes of the program are presented in Figure 12. The basic implementation of the symbol structure is introduced in Figure 12(a); the column-row information, colour table and other symbol-related information can be generated in the CPU and then passed to the GPU to facilitate the rendering of linear map symbols. *MAX_COL* and *MAX_ROW* are defined to build the column array and row array, and the *CellColor* type is developed to transmit the colour information of each cell in the symbol structure. In addition, in the symbol structure, the *uiColType* is employed to indicate if a column can be used to fill line joins. This parameter is mainly used to symbolize dash arrays by handling the background colour (the background colour of a symbol should be determined by the background colour of the bottom map layer).

The basic process of building a wide line with a specific line width is explained in Figure 12(b). The *b–c–a* point set (which is introduced in Section 4.1 as the *Pb–P–Pa* point set) is extracted from the mathematical line string to build wide line segments and line joins (using a range of triangles). The vertex and index information of these triangles is stored in the Vertex Buffer (an *SVertex3D* array) and Index Buffer (a *ushort* array). For the continual and rational handling of a line join, the *U* value of the vertex must be adjusted based on the symbol structure information. A sample handling process of a line join is introduced in Figure 12(c). After computing each vertex's position and the *U* and *V* values of the line join, the *U* value of the vertex must be transformed based on the symbol structure. Then, this transformed *U* value can be used to judge which column the original *U* value belongs to; if it belongs to a column that cannot be used to fill line joins, the *U* value must be adjusted to allow the line join to be drawn rationally. This adjustment process can be extended according to the demands of different drawing effects.

### 5.1. Experiment on the capability of drawing different linear map symbols

To represent different geospatial objects, a variety of linear map symbols are employed in map applications, and different line join drawing methods are implemented for different linear map elements. This research developed a linear map symbol library to help map designers build different maps. Typical linear map symbols are presented in Figure 13. Compared with the original shader function-based method, this paper's method can handle line joins more rationally (as shown in the bottom of Figure 13).

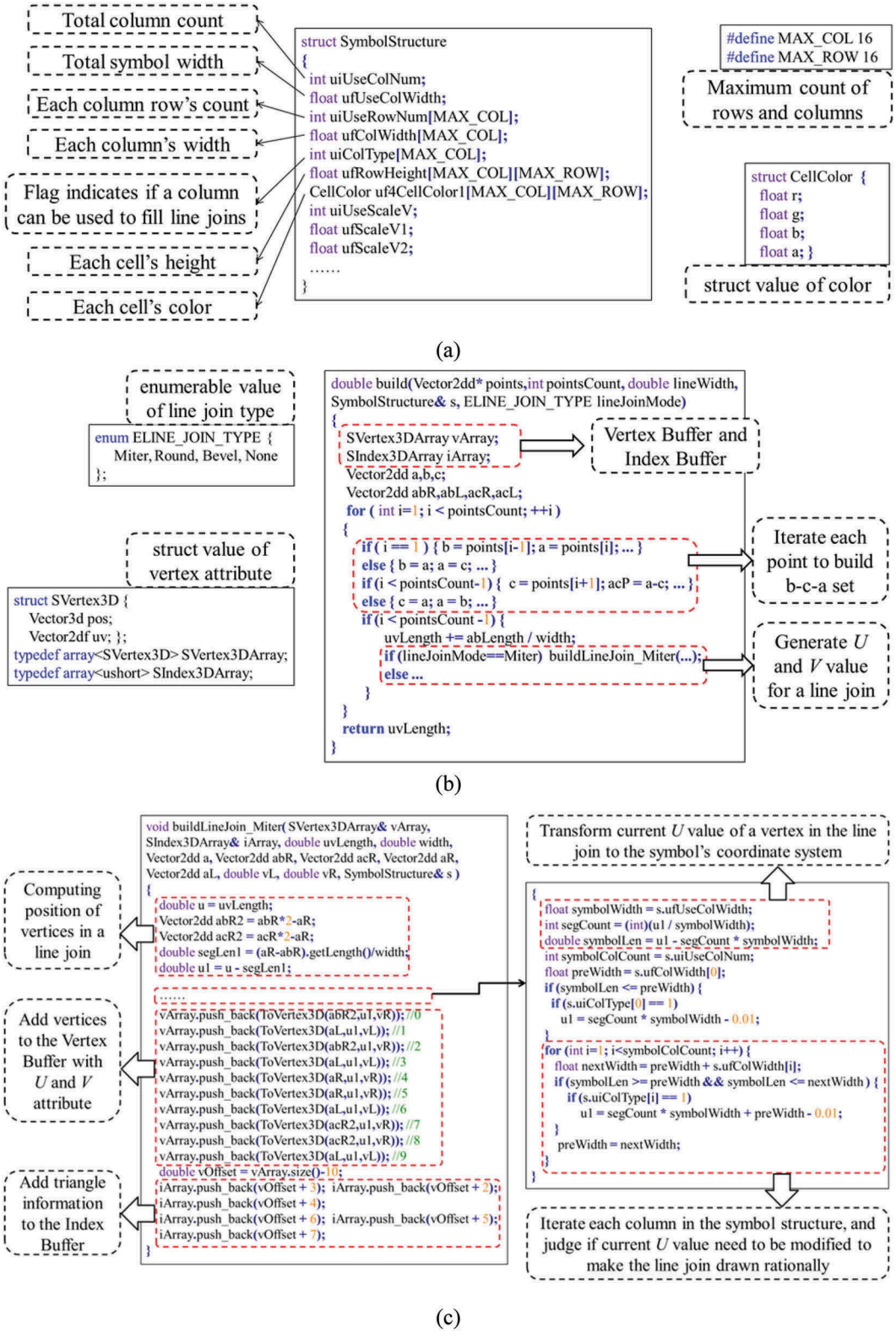



**Figure 12.** Basic implementation processes for building linear map elements.

When applying a linear map symbol to draw linear map elements, the drawing effects of line joins are mainly controlled by customizable parameters: different line join types (mitre, round, bevel or none) and different continual filling modes (filling line joins with different colours according to the rational drawing rules of a specific linear map symbol). Various line join types for a multi-track railroad symbol are presented in

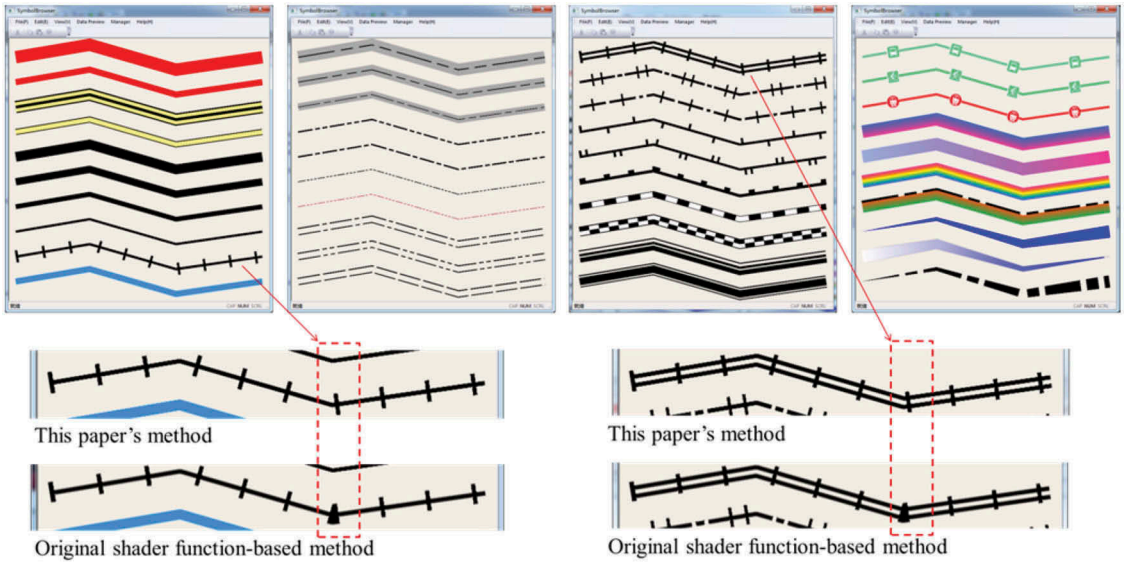



**Figure 13.** Rendering effect of typical linear map symbols.

Figure 14(a), and the drawing effects of other rendering methods (GDI+, AGG and Cairo) are presented in Figure 14(b) to compare continual filling and rational drawing. Because the foundation of GDI+ and the AGG and Cairo methods is the scan line algorithm, the handling of line joins with these methods is limited to rational rendering. With the method proposed in this paper, a range of customizable parameters can be employed to facilitate the continual and rational rendering of line joins.

Based on the proposed method, a prototype map application program was designed and developed. Some snapshots of the symbolization of map elements are presented in Figure 15. In this prototype application, various linear map elements can be rendered with different linear map symbols. The boundary lines, railway lines, road lines and river lines are presented in Figure 15, and the line joins within them are handled based on their own characteristics. The line joins of the boundary lines are filled with a black colour (left-top in Figure 15), while the line joins in the dashed line that represents a ship's course can be freely filled with a blue colour or a transparent colour (left-bottom in Figure 15).

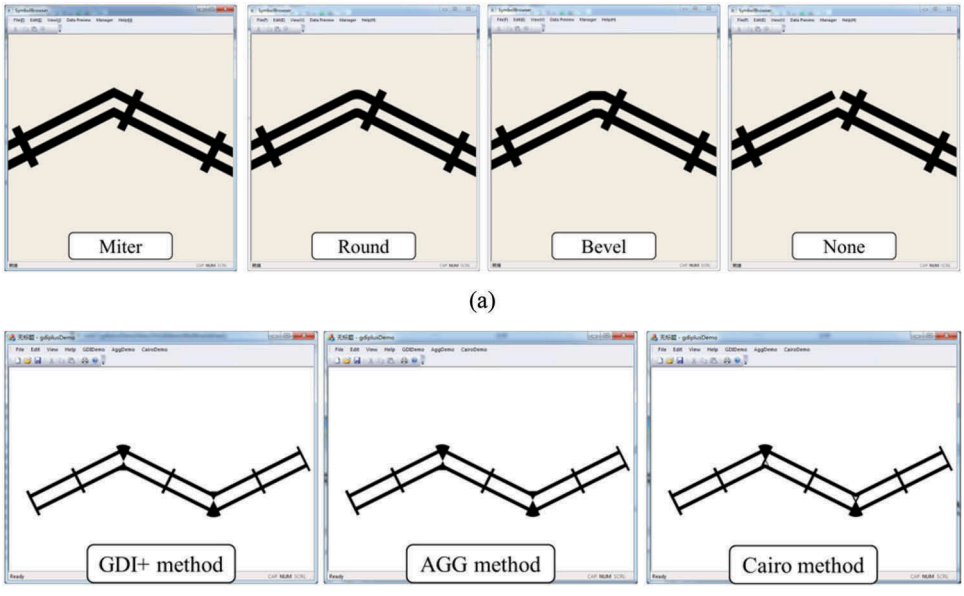



**Figure 14.** Applying the customizable line join handling method for drawing map elements.

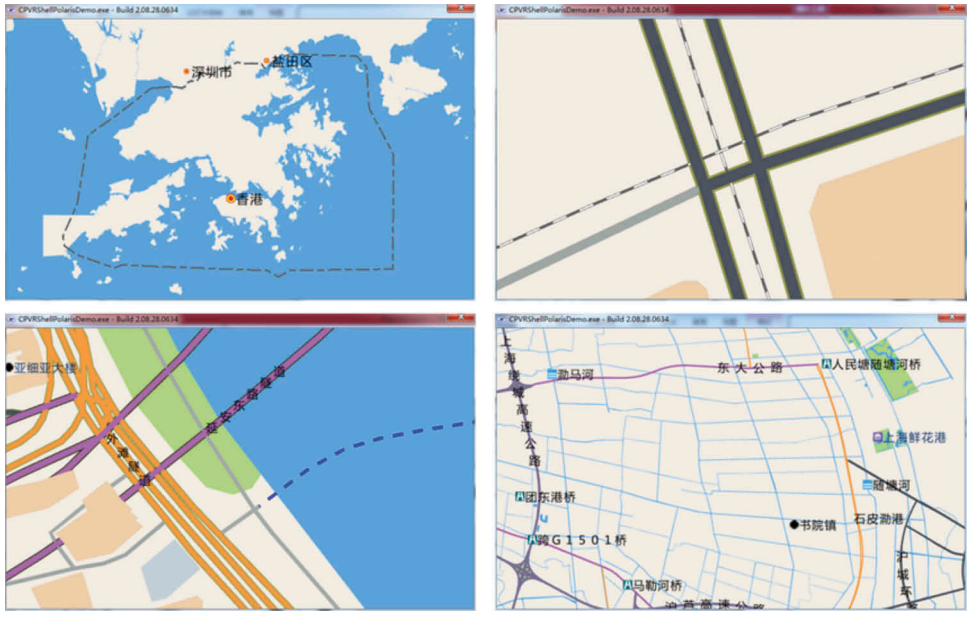



**Figure 15.** Prototype map rendering application.

### *5.2. Experiment on the rendering efficiency*

Drawing efficiency is important to the interaction between map readers and map applications and further supports more dynamic representations of geospatial objects. The proposed method for handling line joins in the representation of map elements focuses on both the drawing effect and the drawing efficiency. Using the customizable line join handling method, which is mainly implemented in the shader program and executed on the GPU, the efficiency of drawing linear map elements can be improved compared with the software methods. The GDI+ method, Cairo method and AGG method are employed to explain such an improvement.

The rendering results for different polyline data (using different linear map symbols) based on these methods are presented in Figure 16. Using the proposed method, line joins can be represented according to the various demands of map designers. For the primary road symbol, line joins are handled as solid lines; for the railway symbol, line joins are handled as dashed lines; for boundary symbols, line joins are handled as gradient colour lines and for the river symbol, line joins are handled as transition lines.

Table 1 shows the time consumption comparison results for these drawing methods. To simulate real map operations (such as moving, zoom in, and zoom out), every line was rendered 1000 times with each rendering method. The experiment was conducted 10 times to obtain the average time costs. According to the statistical results in Table 1, the proposed method can significantly improve the drawing efficiency. The improvement was mainly caused by the differences in rendering mode: GDI+, Cairo and AGG should draw a geometric line multiple times to achieve the effect of repeatedly filling symbols, whereas the proposed method handles the *pixel filling* process using a symbol-structure-related function with the shader language. Regarding the primary road symbol, the previous three methods should draw a line with a wider line width and subsequently draw the same line with a thinner line width. In addition, the handling of line joins in these methods should also be performed multiple times. However, when using the proposed method, the rendering of a primary road can be conducted only one time.

For the GDI+, Cairo and AGG methods, the time costs are influenced by both the count of vertices and the symbol construction (as shown in Table 1, the AGG and Cairo methods are much more efficient than the GDI+ method which is mainly because the AGG and Cairo methods conduct the filling of geometries directly from the scanline algorithm). An example using the AGG method follows: Sample data 2 has 2468 vertices, and Sample data 3 has 349 vertices. Because Sample data 3 uses the gradient colour line type (the boundary symbol requires drawing 4 lines of different line colours in this example), the time cost involved exceeds that of Sample data 2 (the railway symbol requires drawing 2 lines of different line widths and colours in this example). The time cost of the proposed method is mainly dominated by the count of the triangles. Using the proposed method, the time cost of drawing Sample data 3 is much less than that of drawing Sample data 2.

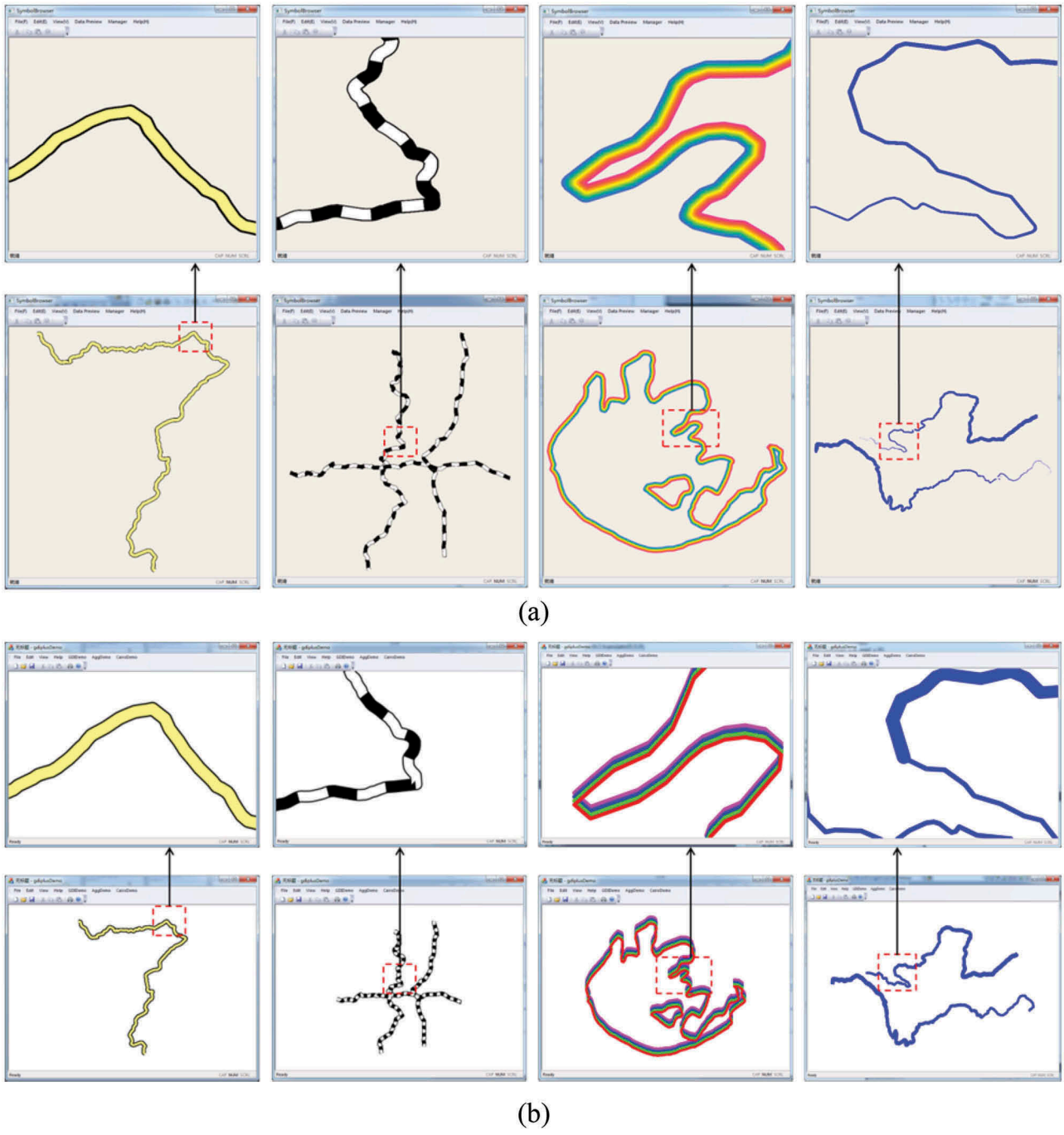



**Figure 16.** Rendering different polyline data.

The time consumption for rendering the same data using four different line types is presented in Table 2. The time cost for rendering a linear map element increases with the complexity of the linear map symbols (the construction of a railway symbol is more complex than a primary road symbol). However, distinguishing whether boundary or river symbols are faster is more difficult. The time costs of the four line types are essentially stable (computations that are performed on a GPU are very fast).

In Table 3, an experiment is conducted to determine the drawing efficiency for different line join types applied to data for two line types (railway data and boundary data). A mitre line join requires two triangles to represent itself, a round line join requires six triangles, a bevel line join requires three triangles, and a none line join requires four triangles. The construction of a round line join requires much trigonometric computation; therefore, the round type requires more time. In addition, due to the efficiency of GPU computations,

**Table 1.** Time consumption comparison between different drawing methods.

| | Vertex count & triangle count | Time cost (in milliseconds) | | | |
|---|---|---|---|---|---|
| | | GDI+ | Cairo | AGG | Proposed method |
| Sample data 1 (primary road) | V: 836 T: 6674 | 16,447 | 10,512 | 3258 | 1341 |
| Sample data 2 (railway) | V: 2468 T: 19660 | 15,013 | 25,532 | 5286 | 5054 |
| Sample data 3 (boundary) | V: 349 T: 1964 | 23,166 | 14,378 | 5985 | 1341 |
| Sample data 4 (river) | V: 249 T: 2764 | 7240 | 6564 | 3922 | 1373 |

**Table 2.** Time consumption comparison between different line types using the proposed method (in milliseconds).

| | Sample data 1 | Sample data 2 | Sample data 3 | Sample data 4 |
|---|---|---|---|---|
| Solid line | 1341 | 5039 | 1347 | 1358 |
| Dashed line | 1420 | 5054 | 1373 | 1404 |
| Gradient colour line | 1432 | 5194 | 1341 | 1389 |
| Transition line | 1404 | 5054 | 1357 | 1373 |

**Table 3.** Time consumption comparison among different line join types (in milliseconds).

| | Sample data 1 | Sample data 2 | Sample data 3 | Sample data 4 |
|---|---|---|---|---|
| Mitre | 1108 | 3791 | 1277 | 1311 |
| Round | 1420 | 5054 | 1347 | 1358 |
| Bevel | 1185 | 3807 | 1310 | 1310 |
| None | 1248 | 3931 | 1326 | 1310 |

the time costs of these four line join types are essentially equal.

## 6. Conclusion and future work

This paper focussed on the handling method of line joins, which can significantly influence the drawing effect and efficiency of map applications. A customizable method is proposed to address the difficulty of continual rendering line joins and to consider the rendering rationality and efficiency. The method uses a customizable tessellation method for the shapes of line joins and customizable parameters of line vertices for the continual and rational rendering of line joins. Our experiments rendered various linear map symbols and further compared the drawing effects and drawing efficiency with other methods. The customizable handling method for line joins that was proposed in this paper achieved more continual and rational drawing effects and improved the drawing efficiency of map applications.

However, map rendering and visualization research are synthetic work, so future research is needed, especially regarding the following aspects:

(1) Interactive and friendly user interfaces for defining how line joins are handled for certain linear map elements or certain map layers. The proposed method can provide flexible program APIs for building different linear map symbols and assigning different parameters for drawing line joins; however, a graphical user interface (GUI) application remains needed to help map designers more conveniently design different maps. Related Symbolizer elements in SLD specification should be included within the GUI application.
(2) *Integrating with map generalization methods*. In cartography, 'generalization' retains the notion of abstraction and extracting the general overriding principles, similar to usages in other disciplines. Map generalization is typically associated with simplification and scale reduction, which is important for displaying spatial information more clearly and improving the drawing efficiency. When applying the proposed method in map applications, map generalization methods should be well considered to avoid overlapping rendering that is caused by different line joins that are too close to each other. Using map generalization methods can also be helpful to improve the representation effect of a map.
(3) *Anti-aliasing rendering*. The proper handling of line joins can enable the rational drawing of line strings to improve the rendering effect of map applications. Another important factor in rendering effects is smoothly drawing map elements, which can be achieved via anti-aliasing rendering. Traditional anti-aliasing methods are more focussed on the edges in a wide-width line string. However, linear map symbols are constructed with more complex inner structures, so the rendering of these symbolized map elements should also consider inner anti-aliasing.

## Acknowledgements

We appreciate the detailed suggestions and comments from the editor and the anonymous reviewers.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## Funding

This work was supported by the National Basic Research Program of China: [973 Program, 2015CB954102], the National Natural Science Foundation of China: [Grant Numbers 41571433 and 41271446] and the Priority Academic Program Development of Jiangsu Higher Education Institutions [Grant Number 164320H116].

## References

AGG. "Anti-Grain geometry." Accessed 27 October 2015. http://www.antigrain.com/.

Aydinoglu, A. C., and M. S. Bilgin. 2015. "Developing an Open Geographic Data Model and Analysis Tools for Disaster Management: Landslide Case." *Natural Hazards & Earth System Sciences Discussions* 15 (2): 335–347. doi:10.5194/nhess-15-335-2015.

Batra, V., M. J. Kilgard, H. Kumar, and T. Lorach. 2005. "Accelerating Vector Graphics Rendering Using the Graphics Hardware Pipeline." *ACM Transactions on Graphics* 34 (4): 146. doi:10.1145/2766968.

Bos, B., H. W. Lie, C. Lilley, and I. Jacobs. 2015. "Cascading style sheets, level 2 CSS2 specification." Accessed 27 October 2015. http://www.w3.org/TR/CSS2/.

Cairo. 2015. Accessed 27 October 2015. http://cairographics.org/

Chen, M., Y. Wen, and S. Yue. 2015. "A Progressive Transmission Strategy for GIS Vector Data under the Precondition of Pixel Losslessness." *Arabian Journal of Geosciences* 8 (6): 3461–3475. doi:10.1007/s12517-014-1467-y.

Deng, B. S., T. Q. Deng, and D. Xu. 2013. "GPU-Based Techniques for Vector Data Visualization on Global Scale Terrain." *Applied Mechanics & Materials* 380–384: 1689–1692. doi:10.4028/www.scientific.net/AMM.380-384.

Deoliveira, J. 2008. "Geoserver: Uniting the Geoweb and Spatial Data Infrastructures." In *Proceedings of the 10th International Conference for Spatial Data Infrastructure*, St. Augustine, February 25–29.

Dietze, L., and A. Zipf. 2007. "Extending OGC Styled Layer Descriptor (SLD) for thematic cartography." In *Proc. 4th Int. Symp. on LBS and Telecartography*, Hong Kong, November 8–10. Heidelberg: Springer.

Dong, W., J. Ran, and J. Wang. 2012. "Effectiveness and Efficiency of Map Symbols for Dynamic Geographic Information Visualization." *Cartography and Geographic Information Science* 39 (2): 98–106. doi:10.1559/1523040639298.

Dymon, U. J. 2003. "An Analysis of Emergency Map Symbology." *International Journal of Emergency Management* 1 (3): 227–237. doi:10.1504/IJEM.2003.003301.

Garlandini, S., and S. I. Fabrikant. 2009. "Evaluating the Effectiveness and Efficiency of Visual Variables for Geographic Information Visualization." In *Spatial Information Theory*, 195–211. Berlin: Springer. doi:10.1007/978-3-642-03832-7_12.

GDI. 2015. Graphic device interface. https://msdn.microsoft.com/en-us/library/aa925824.aspx Accessed 27 October 2015.

gl-agg. 2015. Accessed 27 October 2015. https://code.google.com/p/gl-agg/.

Goodchild, M., R. Haining, and S. Wise. 1992. "Integrating GIS and Spatial Data Analysis: Problems and Possibilities." *International Journal of Geographical Information Systems* 6 (5): 407–423. doi:10.1080/02693799208901923.

Goodchild, M. F., M. Yuan, and T. J. Cova. 2007. "Towards a General Theory of Geographic Representation in GIS." *International Journal of Geographical Information Science* 21 (3): 239–260. doi:10.1080/13658810600965271.

Guo, M., L. Wu, and Z. Xie. 2015. "An Efficient Parallel Map Visualization Framework for Large Vector Data." *Geomatica* 69 (1): 113–117. doi:10.5623/cig2015-108.

Harrie, L., H. Stigmar, and M. Djordjevic. 2015. "Analytical Estimation of Map Readability." *ISPRS International Journal of Geo-Information* 4 (2): 418–446. doi:10.3390/ijgi4020418.

Haunert, J.-H., and L. Sering. 2011. "Drawing Road Networks with Focus Regions." *IEEE Transactions on Visualization & Computer Graphics* 17 (12): 2555–2562. doi:10.1109/TVCG.2011.191.

Hearnshaw, H. M., and D. J. Unwin. 1994. *Visualization in Geographical Information Systems*. Chichester: John Wiley & Sons.

Herring, J. 2011. "Opengis Implementation Standard for Geographic Information-Simple Feature Access-Part 1: Common Architecture." *OGC Document* 4 (21): 122–127.

Iosifescu Enescu, I., N. H. Panchaud, M. Heitzler, C. M. Iosifescu Enescu, and L. Hurni. 2015. "Towards Better WMS Maps Through the Use of the Styled Layer Descriptor and Cartographic Conflict Resolution for Linear Features." *The Cartographic Journal* 52 (2): 125–136. doi:10.1080/00087041.2015.1119468.

Iosifescu, I., M. Hugentobler, and L. Hurni. 2010. "GIScience 2010 Tutorial Development of Web GIS and Web Mapping Applications with QGIS mapserver." In *Sixth International Conference on Geographic Information Science*, Zurich, September 14–17.

Lienert, C., B. Jenny, O. Schnabel, and L. Hurni. 2012. "Current Trends in Vector-Based Internet Mapping: A Technical Review." In *Online Maps with APIs and WebServices*, edited by M. P. Peterson, 23–36. Berlin: Springer Verlag.

Lin, -S.-S., C.-H. Lin, Y.-J. Hu, and T.-Y. Lee. 2014. "Drawing Road Networks with Mental Maps." *IEEE Transactions on Visualization and Computer Graphics* 20 (9): 1241–1252. doi:10.1109/TVCG.2014.2312010.

Luna, F. D. 2008. *Introduction to 3D Game Programming with DirectX 10*. Burlington, MA: Jones & Bartlett Publishers.

MacEachren, A. M. 2004. *How Maps Work: Representation, Visualization, and Design*. New York: Guilford Press.

MacEachren, A. M., and D. R. F. Taylor, Eds. 1994. *Visualization in Modern Cartography*. Oxford: Pergamon.

Mei, Y., and L. Li. 2007. "Cartographic Symbol Library considering Symbol Relations Based on Anti-Aliasing Graphic Library." *Geoinformatics*. doi:10.1117/12.759674.

Neteler, M., M. H. Bowman, M. Landa, and M. Metz. 2012. "GRASS GIS: A Multi-Purpose Open Source GIS." *Environmental Modelling & Software* 31 (7): 124–130. doi:10.1016/j.envsoft.2011.11.014.

Nilsson, P., and D. Reveman. 2004. "Glitz: Hardware Accelerated Image Compositing Using OpenGL." In *Proceedings of the FREENIX Track: USENIX Annual Technical Conference*, Boston Marriott Copley Place, Boston, MA, June 27–July 2, 29–40.

OGC. 2015a. OpenGIS Styled Layer Descriptor (SLD) Implementation Specification 1.0 [D-IS]. OpenGIS Consortium. Accessed 27 October 2015. http://www.opengeospatial.org/standards/sld/.

OGC. 2015b. OpenGIS Symbology Encoding Implementation Specification 1.1.0 [IS]. OpenGIS Consortium. Accessed 27 October 2015. http://www.opengeospatial.org/standards/symbol/.

Peng, L. 2014. "Key Technology in a Lightweight WPF-Based Electronic Map Engine." *Applied Mechanics and Materials* 678: 70–74. doi:10.4028/www.scientific.net/AMM.678.70.

Qt Painter. 2015. Accessed 27 October 2015. http://doc.qt.io/qt-4.8/qpainter.html.

Robinson, A. C., R. E. Roth, J. Blanford, S. Pezanowski, and A. M. MacEachren. 2012. "Developing Map Symbol Standards through an Iterative Collaboration Process." *Environment and Planning B: Planning and Design* 39 (6): 1034–1048. doi:10.1068/b38026.

Rost, R. J., B. M. Licea-Kane, D. Ginsburg, J. M. Kessenich, B. Lichtenbelt, H. Malan, and M. Weiblen. 2009. *Opengl Shading Language*. Upper Saddle River, NJ: Pearson Education.

Rougier, N. 2013. "Shader-Based Antialiased Dashed Stroked Polylines." *Journal of Computer Graphics Techniques* 2 (2): 91–107.

Schnabel, O. 2005. "Map Symbol Brewer–A New Approach for a Cartographic Map Symbol Generator." In *Proceedings of ICA Conference*, Miami, June 20–25.

Schneider, M., M. Guthe, and R. Klein. 2005. "Real-Time Rendering of Complex Vector Data on 3d Terrain Models." In *Proceedings of the 11th International Conference on Virtual Systems and Multimedia*, Ghent, October 3–7, 573–582.

Steenbeek, J., M. Coll, L. Gurney, F. Mélin, N. Hoepffner, J. Buszowski, and V. Christensen. 2013. "Bridging the Gap between Ecosystem Modeling Tools and Geographic Information Systems: Driving a Food Web Model with External Spatial–Temporal Data." *Ecological Modelling* 263: 139–151. doi:10.1016/j.ecolmodel.2013.04.027.

Steiniger, S., and A. J. S. Hunter. 2012. "Free and Open Source GIS Software for Building a Spatial Data Infrastructure." In *Geospatial Free and Open Source Software in the 21st Century*, edited by E. Bocher and M. Neteler, 247–261. Berlin: Springer.

Stoll, C., S. Gumhold, and H. P. Seidel. 2005. "Visualization with Stylized Line Primitives." *Visualization* 2005 (695–702). doi:10.1109/VISUAL.2005.1532859.

SVG. 2015. Scalable Vector graphics specification – W3C, Working Draft 06 July, 1999. Accessed 27 October 2015. http://www.w3.org/TR/SVG/.

Virtanen, J.-P., H. Hyyppä, A. Kämäräinen, T. Hollström, M. Vastaranta, and J. Hyyppä. 2015. "Intelligent Open Data 3D Maps in a Collaborative Virtual World." *ISPRS International Journal of Geo-Information* 4 (2): 837–857. doi:10.3390/ijgi4020837.

Voženílek, V. 2005. *Cartography for GIS – Geovisualization and Map Communication*. 1st ed. Olomouc: Univerzita Palackého.

Wen, Y., M. Chen, G. Lu, H. Lin, and S. Yue. 2013. "A Characteristic Bitmap Coding Method for Vector Elements Based on Self-Adaptive Gridding." *International Journal of Geographical Information Science* 27 (10): 1939–1959. doi:10.1080/13658816.2013.774006.

Woo, M., J. Neider, T. Davis, and D. Shreiner. 1999. *Opengl Programming Guide: The Official Guide to Learning Opengl, Version 1.2*. Boston, MA: Addison-Wesley Longman Publishing.

Yoo, J. J., S. Krishnadasan, J. Brothers, S. Jung, S. Ryu, and J. Kim. 2014. "Path Rendering for High Resolution Mobile Device." *In SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications* 13. doi:10.1145/2669062.2669085.

Yuan, M. 2001. "Representing Complex Geographic Phenomena in GIS." *Cartography and Geographic Information Science* 28 (2): 83–96. doi:10.1559/152304001782173718.

Yue, S., J. Yang, M. Chen, G. Lu, A. Zhu, and Y. Wen. 2016. "A Function-Based Linear Map Symbol Building and Rendering Method Using Shader Language." *International Journal of Geographical Information Science* 30 (2): 143–16. doi:10.1080/13658816.2015.1077964.

Zipf, A. 2005. "Using Styled Layer Descriptor (SLD) for the Dynamic Generation of User-And Context-Adaptive Mobile Maps–A Technical Framework." In *Web and Wireless Geographical Information Systems*, Lausanne, December 15–16, 183–193. Berlin: Springer. doi:10.1007/11599289_16.