International Journal of Geo-Information

MDPI

Article

A High-performance Cross-platform Map Rendering Engine for Mobile Geographic Information System (GIS)

Shaojie Li [1,2], Shaohua Wang [3,4,*], Yong Guan [5], Zhiyan Xie [2], Kejia Huang [2], Ming Wen [2] and Lixin Zhou [1]

[1] School of Software and Microelectronics, Peking University, Beijing 102600, China; lishaojie@pku.edu.cn (S.L.); lxzhou@ss.pku.edu.cn (L.Z.)
[2] SuperMap Software Co. Ltd., Beijing 100015, China; xiezhiyan@supermap.com (Z.X.); huangkejia@supermap.com (K.H.); wenming@supermap.com (M.W)
[3] Department of Geography, University of California, Santa Barbara, CA 93117, USA; shaohua@geog.ucsb.edu (S.W.)
[4] Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Science, Beijing 100101, China; wangshaohua@lreis.ac.cn
[5] Claremont Graduate University, Claremont, CA 91711 USA; yong.guan@cgu.edu
* Correspondence: wangshaohua@lreis.ac.cn; Tel.: +86-101-5989-6521 (S.W.)

Received: 22 July 2019; Accepted: 18 September 2019; Published: 20 September 2019

Abstract: With the diversification of terminal equipment and operating systems, higher requirements are placed on the rendering performance of maps. The traditional map rendering engine relies on the corresponding operating system graphics library, and there are problems such as the inability to cross the operating system, low rendering performance, and inconsistent rendering style. With the development of hardware, graphics processing unit (GPU) appears in various platforms. How to use GPU hardware to improve map rendering performance has become a critical challenge. In order to address the above problems, this study proposes a cross-platform and high-performance map rendering (Graphics Library engine, GL engine), which uses mask drawing technology and texture dictionary text rendering technology. It can be used on different hardware platforms and different operating systems based on the OpenGL graphics library. The high-performance map rendering engine maintains a consistent map rendering style on different platforms. The results of the benchmark experiments show that the performance of GL engine is 1.75 times and 1.54 times better than the general map rendering engine in the iOS system and in the Android system, respectively, and the rendering performance for vector tiles is 11.89 times and 9.52 times better than rendering in the Mapbox in the iOS system and in the Android system, respectively.

Keywords: Graphics Processing Unit (GPU); Map Rendering Engine; Mapbox; Mobile GIS

1. Introduction

The wide use of mobile GIS and mobile navigation software has ignited the public's understanding of the map, location based services, and also led to the development of the Geographic Information System (GIS) [1-3]. The GIS operation equipment has grown from the initial personal computer to the servers, mobile phones, professional handsets, and Web terminals. The hardware and operating systems are diversified, which requires a mobile map for rendering engines on different platforms.

Mobile GIS usually adopts plug-in based architecture [4]. The map data parsing engine and map rendering engine are designed as plug-in mode. Map rending engine is a critical research topic in mobile GIS [5], including spatial data models, GIS algorithms, optimization strategies, and modeling [6-8]. Although the technical development can realize the extended development of map rendering engine in the mobile GIS applications, researchers and developers need to implement a corresponding map rendering engine for different operation systems. A cross-platform map rendering engine is needed for mobile GIS.

The growth of multi-source heterogeneous high-dimensional spatial data and its Web applications is increasing [9]. Information technologies (IT) such as location based social networks [10-12], Virtual Reality (VR), and Augment Reality (AR) are also actively integrating with mobile GIS. The visualization of high-dimensional geospatial data such as oblique photogrammetry, building information model (BIM), and laser point cloud is essential to visualize in mobile GIS. High performance-based rendering methods are needed for mobile GIS with limited memory capabilities and less efficient CPUs.

The main contribution of this study is to propose a high-performance cross-platform map rendering technology as the solution to address the challenges above. Based on OpenGL graphics library, a cross-platform and high-performance map rendering engine (GL engine) is developed to achieve high performance of vector data and raster data as well as a cross-platform rendering method that assures, on different platforms, the consistency of the map rendering style. Map rendering algorithms and methods for rendering are developed to develop a high-performance, cross-platform map rendering engine. A new masking mechanism is proposed to solve the problem of triangulation. The order of magnitude improves the segmentation performance of the polygon and solves the problem of triangulation for the self-intersecting polygon and the island polygon. The texture dictionary caching technology is used to solve the problem that the font data is exchanged in the memory and the cache when the text is drawn, which greatly improves the rendering performance of the text and realizes high-performance annotation text along the line.

2. Related work

The rendering map features can be decomposed into point features, line features, polygon region features, text features, and image features [7,13,14]. These map related elements are provided in the graphical interface library of the operating system, but the graphics library algorithms and interfaces provided by different operating systems are different, which brings great inconvenience to the drawing of the map in mobile GIS.

There are various geometry-based algorithms for rendering vector data from computer graphics and geo-informatics [15-18]. However, most of these geospatial algorithms and spatial models need more space to store the pre-processing result and with high computing capabilities, for example, spatial index data for pre-processing geospatial data and hierarchical road network topologies for route planning. The strategy of map rendering between GIS and mobile GIS is different [19,20], because the management of massive geospatial data sets and rendering spatial data on reduced hardware configuration requirements of mobile terminal devices are difficult for mobile GIS.

As the world's leading provider of GIS products, Esri, its product called ArcGIS has been based on the Windows operating system for a long time. The map rendering engine in ArcGIS on the GDI graphics library is provided by the Windows system, which has caused difficulties for its subsequent cross-platform development. With the increasing demand for cross-platform products, the ArcGIS Runtime SDK product implements different map rendering engines for different operating systems, which is not conducive to the consistency of map rendering effects [21].

Li implemented a set of cross-platform graphics library (UGraphics) based on the rendering characteristics of the map for related optimization to achieve cross-platform mapping in SuperMap mobile GIS [22]. The map rendering engine uses a graphics library provided by the operating system. For example, the Windows operating system uses GDI (Graphics Device Interface). The Linux operating system uses XLib, GTK, and Qt. The mobile device Android uses Skia, and the iOS uses Quartz. With the increasing use of mobile hardware and operating systems, this problem is

increasingly hard to solve. Li developed a bitmap map rendering engine, which implements the related graphics image algorithm. This map rendering engine solves the reliance on the operating system graphics library and solves the cross-operating system problem. It has been used in SuperMap GIS platform software products for nearly ten years. However, devices such as iPad have the demand for explosive growth of graphics acceleration hardware (GPU) and Web applications. The bitmap map rendering engine cannot use the GPU for the acceleration of map rendering and the bitmap engine has low performance for large-scale data, and does not support map rendering on a web browser.

Anti-Grain Geometry (AGG) is an open source 2D graphics engine [23]. It provides a set of graphics algorithms that combine subpixel accuracy technology with anti-aliasing technology to achieve high efficiency and high quality 2D graphics processing [24]. AGG is written in C++ and the standard C Runtime Function. This gives AGG good cross-platform capabilities. Another feature of AGG is its great flexibility. AGG provides a series of loosely coupled algorithms, and all its classes are described by templates. Developers can freely combine, rewrite, and replace some or all of the algorithms to meet their specific graphics operations. Wenfeng Lu et al. compared the GDI/GDI+, Qt, AGG, and other graphics libraries, and ultimately chose AGG [25], but AGG was difficult to expand development by the complexity of using the template techniques [24,26]. The lack of support for GPU graphics acceleration, the mobile property, the Web-based applications, and 3D rendering also restricted the types of applications in Mobile GIS.

Qt is a cross-platform C++ graphical user interface software developed by Trolltech [27,28]. Qt is designed with object-oriented programming to make the functions better packaged and modularized. Qt has good cross-platform features and supports Windows, Linux, Mac, Android, iOS, and other operating systems. Qt's Qpainter strategy provides a suite of drawing API functions, such as drawArc, drawChord, drawEllipse, drawImage, drawLine, and drawText. It also supports path, color gradient, and plane coordinate transformation, which brings a sense of security to the drawing of complex graphics. Qt is rendered based on the FrameBuffer. There is a special service to copy the frame data to the graphics card. Therefore, the rendering efficiency of Qt is relatively high. In order to improve the quality of the drawing characteristics, Qt provides an anti-aliasing strategy to make the drawn graphics more comfortable. Qt is implemented based on OpenGL for 3D graphics rendering. The combination of Qt and GIS are widely used in GIS, including Quantum GIS (QGIS) [29], Merkopolo [30], and GIS related applications [31]. Qt has issues to the map rendering engine for mobile GIS, including GPU graphics acceleration, weak support for mobile applications, lack of support for 3D rendering, and lack of support for web applications.

An API is defined as a set of functions that a client program can call. OpenGL (Open Graphics Library) is a cross-language and cross-platform application programming interface (API) for drawing 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU) for hardware accelerated rendering [32]. The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although the API can be implemented entirely in software, its design is primarily implemented on hardware. Although function definitions are similar in appearance to programming language C, they are language independent. Therefore, the OpenGL has many language bindings, the most notable of which is JavaScript binding to WebGL (OpenGL ES 2.0 based API for 3D rendering in a web browser), C binding (WGL, GLX and CGL) [33,34], C binding provided by iOS and Java, and C binding provided by Android. In addition to being independent of the language, OpenGL is also cross-platform. The specification does not mention the issue of getting and managing OpenGL contexts, but rather a detail of the underlying windowing system. The OpenGL is purely focused on rendering and does not provide APIs related to input, audio, or windows. OpenGL is characterized with its text rendering, cross-platform graphics rendering, graphics hardware acceleration, Web-based graphics rendering (WebGL), mobile graphics rendering (OpenGL ES), and the issuance of free commercial licenses.

3. Methods

A high-performance cross-platform map rendering engine (Graphics Library map rendering engine, GL map rendering engine) is proposed in this study. The GL map rendering engine is based on the OpenGL graphics library, which realizes the rendering of each element of the map. When the map is rendered, the layers are rendered first, and then the rendered layer is superimposed and rendered, and the complete rendering is completed. The process of map rendering is implemented in a visual studio.

The GL map rendering engine contains a unified map element drawing method and a unified map data docking API. The map rendering engine decomposes the map elements into basic graphic elements. Under different platforms, only the OpenGL graphics library of different systems needs to be called to draw the basic graphic elements (points, lines, polygons, and text), which can form a unified style map and reach different platforms. The specific architecture for the unified map rendering is shown in Figure 1.

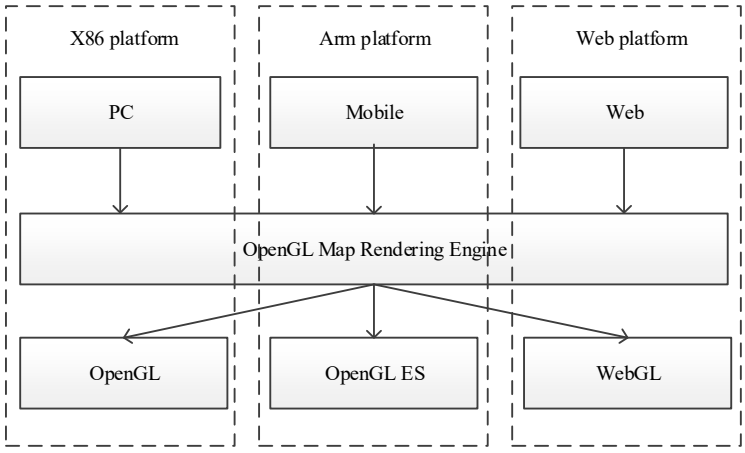

Figure 1. The graphics library map rendering engine with cross-platform principle.

3.1. The Architecture of the Graphics Library Map Rendering Engine

The modules of the GL engine include a map module, an event processing module, a GL rendering module, and a geospatial data processing module (Figure 2). The geospatial data processing module is responsible for parsing the map data into OpenGL identifiable format and is mainly responsible for triangulating the spatial geometric data, and encoding the position, the color and the similarity in the vector map data into OpenGL render objects (RO). The GL rendering module calls the OpenGL graphics rendering library to render the data as visual data, including parsing the RO array, and by controlling the rendering pipeline to render the map by calling the API to operate the OpenGL state machine. The map results are displayed to the screen through the host system form of the interactive module.

The map interaction module is responsible for rendering the map results to the monitor device or simply a display and accepting the user's interaction operation. The map interaction module provides an OpenGL display form, supplies various user interactions, including gestures, and handles user interaction operations to generate events. There two ways to respond to events. One way is to add data processing tasks, usually including events with range scale changes and need to display new data. Another way is map redrawing operation.

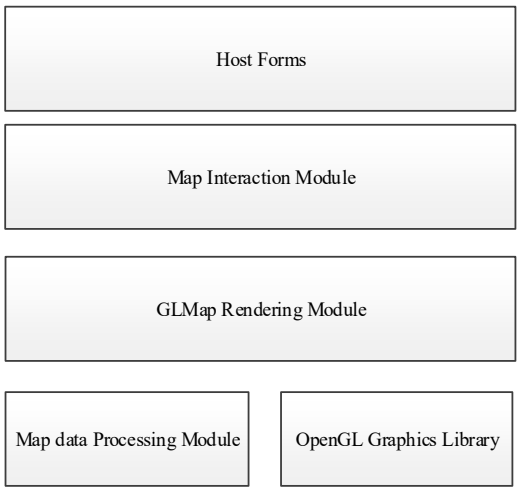

Figure 2. GL map rendering engine module.

3.2. The rendering process for the GL map rendering engine

When the host environment map window changes the mobile GIS, the rendering operation of the map is triggered. The change of the map window mainly includes the initial loading of the map or other application windows to cover the map window, and another change is the interactive operation of the map, such as zooming the map and roaming the map, editing maps and other operations.

The rendering process of the map in Mobile GIS is shown in Figure 3. When the local map change event is fired, the map enters the rendering process. The map rendering module divides the map into nine grid spaces according to the size range of the current window, and then passes the inner map of the grid space to the child thread for drawing.

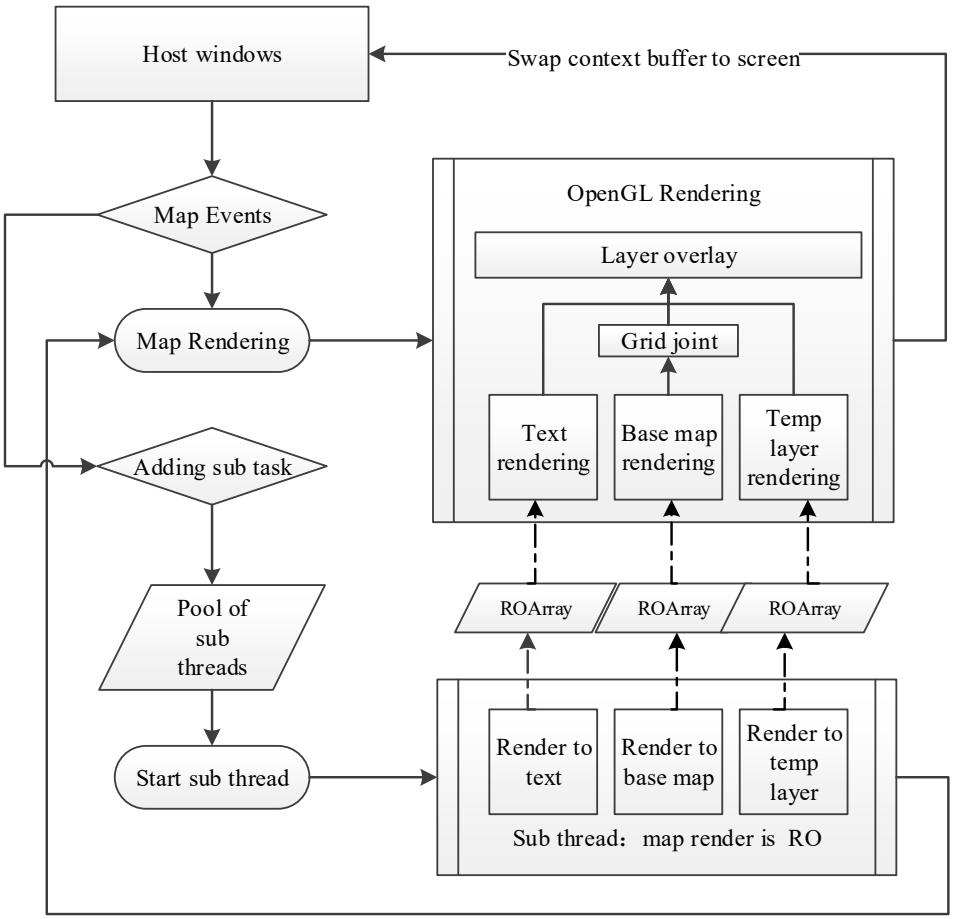

Figure 3. GL map rendering engine module.

The tasks in the child threads pool will be processed in turn. These tasks include reading geospatial data from the geospatial data sets, determining whether the data is in the visible range, cropping, and organizing the data to form a rendering RO. After the sub-thread processing is completed, the redraw operation is called back. The redraw operation is to render the processed RO through the GL function to form a pixel matrix and to render it onto the screen.

3.3. Rendering of map elements

In the high-performance rendering process of vector map data, the parts that affect the performance of the GL engine include the rendering of line geometry objects, the rendering of polygon geometry objects, the rendering of text, and the rendering of symbols. This study made special designs for these four types of elements, including wide line splitting drawing technology, surface feature mask drawing technology, dictionary text rendering technology, and texture symbol rendering technology.

3.3.1. Line rendering

Line layers can be divided into three categories: single-line layers, wide-line layers, and symbol-line layers. The GL map rendering engine draws single-line features directly using OpenGL-supported APIs, but the GL engine first splits the wide-line layer into rectangles for the wide-line layer, then splits the rectangle into triangles, and then renders it. The process workflow of triangulation of wide-line is shown in Figure 4.

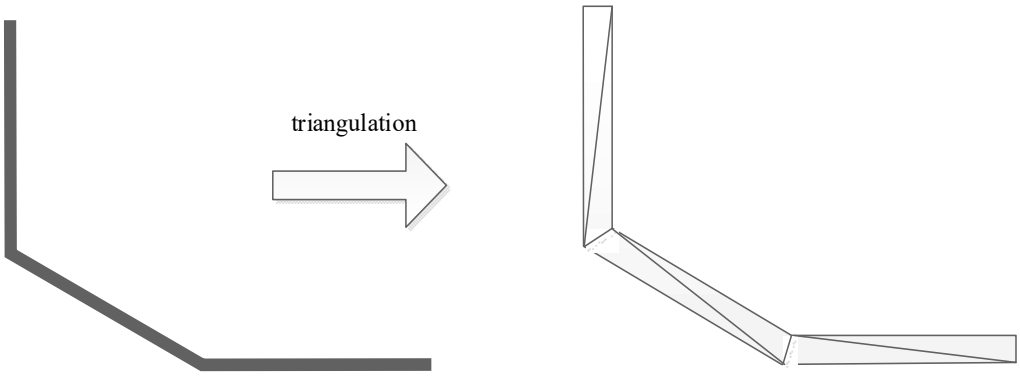

Figure 4. The triangulation of wide-line.

The Delaunay triangulation algorithm was used to split the rectangle into triangles. The algorithm for calculating triangulation relies on quickly detecting whether a point is within the circumscribed circle of a triangle and has a valid data structure that stores triangles and edges. In a two-dimensional scene, a way to detect whether a point D is located at A, B, or C is to evaluate a determinant value, sorting A, B, and C clockwise, if and only if D is at A, B. This determinant inequality holds when C is in the circumscribed circle (Figure 5).

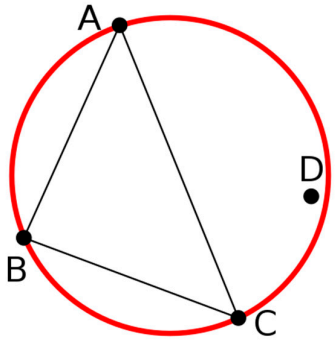

Figure 5. Delaunay triangulation judgment elements.

Formulation (1) determines if point D is in the outer circle of A, B, and C.

$$\begin{vmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{vmatrix} = \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x^2-D_x^2)+(A_y^2-D_y^2) \\ B_x-D_x & B_y-D_y & (B_x^2-D_x^2)+(B_y^2-D_y^2) \\ C_x-D_x & C_y-D_y & (C_x^2-D_x^2)+(C_y^2-D_y^2) \end{vmatrix}$$
$$= \begin{vmatrix} A_x-D_x & A_y-D_y & (A_x-D_x)^2+(A_y-D_y)^2 \\ B_x-D_x & B_y-D_y & (B_x-D_x)^2+(B_y-D_y)^2 \\ C_x-D_x & C_y-D_y & (C_x-D_x)^2+(C_y-D_y)^2 \end{vmatrix} > 0 \quad (1)$$

Yu et al. summarized three types of methods in the construction of the existing Delaunay triangulation: point-by-point insertion method, triangulation method, and the divide and conquer method for the algorithm of Delaunay triangulation [35], and proposed the direction of optimization as a hybrid construction method. However, the optimal performance can reach *O*(*n*loglogn*) in these methods.

This study uses the point-by-point insertion algorithm to perform the wide-line triangulation operation that combined with the large volume of GIS data and high-performance requirements. At the same time, in order to avoid the complexity of the splitting at the corners of the wire ends, the

performance is low. In this study, the triangulation work of the line segment intersection is replaced by adding the circle at the intersection of the line segment, which greatly improves the performance of drawing the wide line. The symbol line processing is similar to the composite point layer. It is split into basic points, lines, faces, and texts, and is processed into multiple RO.

3.3.2. Polygon rendering

If the polygon features are drawn by the triangulation method of points and line elements, there is a defect operation. If the polygon feature is drawn by triangulation, the process is the geometric polygon that expresses the polygon feature is divided into a combination of multiple triangles, and the triangle mesh completely covers the polygon and has no intersecting edges, and then renders in the video memory. The problem is that the time complexity of the triangulation algorithm is $O(n^2)$. When the polygon boundary is complicated, especially for the polygon containing the island and the hole, the time-consuming exponential growth. Moreover, the triangulation algorithm does not support the profiled surface, especially for the self-intersection polygons (Figure 6). The Delaunay triangulation is essential for complex polygons, like self-intersection polygons. In our study, multi simple polygons are generated from a self-intersection complex polygon and then Delaunay triangulation is running.

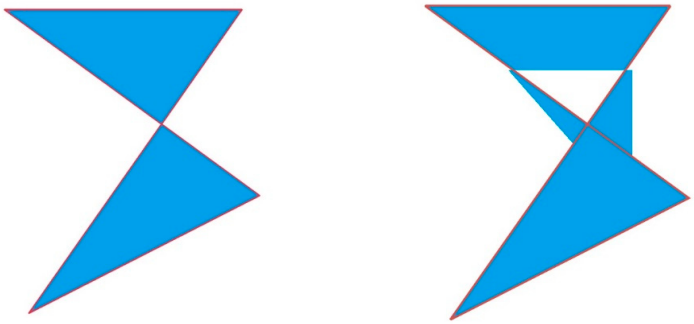

Figure 6. (a) GL engine drawing result (b) MapBox drawing result.

Most of the polygons for rendering the real-world geographic information drawing are complex self-intersection polygons. The OpenGL does not support direct filling of concave polygons [36]. There are still a large number of island polygons. For this case, a mask algorithm is proposed in this paper. The drawing mechanism, polygon drawing, replaces the traditional triangulation scheme in this way, reducing the complexity of the operation to $O(n)$. A masking technique was used in the GL map rendering engine proposes for polygon features, which divides polygon feature rendering into two steps. The first step is to engrave the shape of the image in the OpenGL stencil buffer to form a mask on the color buffer. The second step is to draw the outer bounding rectangle of the face in the color buffer to get the final rendering result. Similar to painting a piece of paper with a hollow shape on the canvas, the polygon feature is rendered in the shape of the paper (template buffer), then the entire canvas (color buffer) is colored. When the paper takes the back element, the shape is printed on the canvas. The OpenGL uses the computing power of the GPU to ensure the efficiency of the method.

The operation of template buffer engraves the shape of the picture. For a screen of length M and width N, its corresponding color buffer is described by a matrix of pixels consisting of $M*N$ points. The stencil buffer is also a matrix of $M*N$, which corresponds to each pixel to describe the pass state when it is used as a template test. The shape for the engraved picture is required to distinguish the area from the uncovered area (the point on the pixel matrix). The specific method is:

1) Create a stencil buffer with a depth of 1 bit and clear all bits of the stencil buffer matrix,

2) Use the vertex array described in 1) to draw a triangle in the stencil buffer in GL_TRIANGLE_FAN mode (the first point of the point array is the triangle vertex, and the adjacent two points are taken sequentially - the *N*-th (*N*>1) points and *N* +1 point - the line is the bottom edge of the triangle, forming a triangle network). The *glStencilFun* parameter is *GL_NEVER*, which means that the stencil buffer is only manipulated without rendering to the color buffer. The glStencilOp parameter is GL_INVERT, which means that the stencil buffer is operated in reverse color mode (the stencil buffer point value that is covered by the triangle for a single number of times is 1, and the stencil buffer point value that is evenly covered by the triangle is 0, and the stencil buffer operation mode is shown in the Figure 7. The part of the stencil buffer that is not 0 is the area covered by the face.

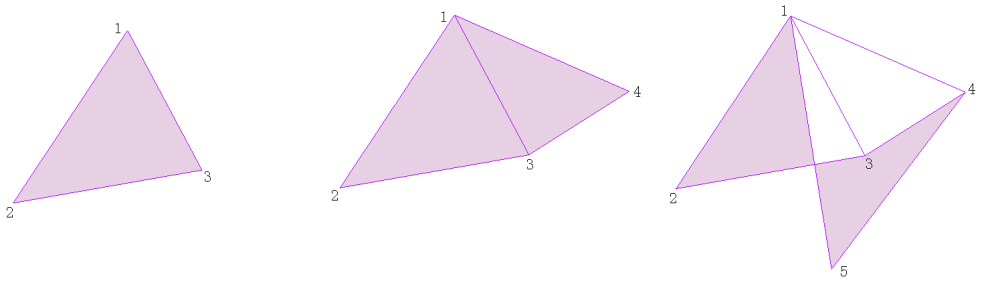

Figure 7. The polygon mask drawing process.

3.3.3. Text rendering

The way of text rendering in OpenGL is to copy the fonts Buffer produced by FreeType in memory to the memory texture before rendering. The problem with this method is that each time a text is rendered, the content exchange between memory and video memory is performed, which is inefficient and slow to render. If the glyph is managed in units of Point of Interest (POI), a texture memory cache is created for all POI data, and the same POI directly multiplexes the cache to render text, and there is no need to repeatedly use the FreeType to render text (Figuer 8). It can alleviate the problem of low rendering efficiency to a certain extent, but it cannot achieve high-performance rendering of massive POI. In addition, this method takes up a lot of memory space due to the large amount of texture.

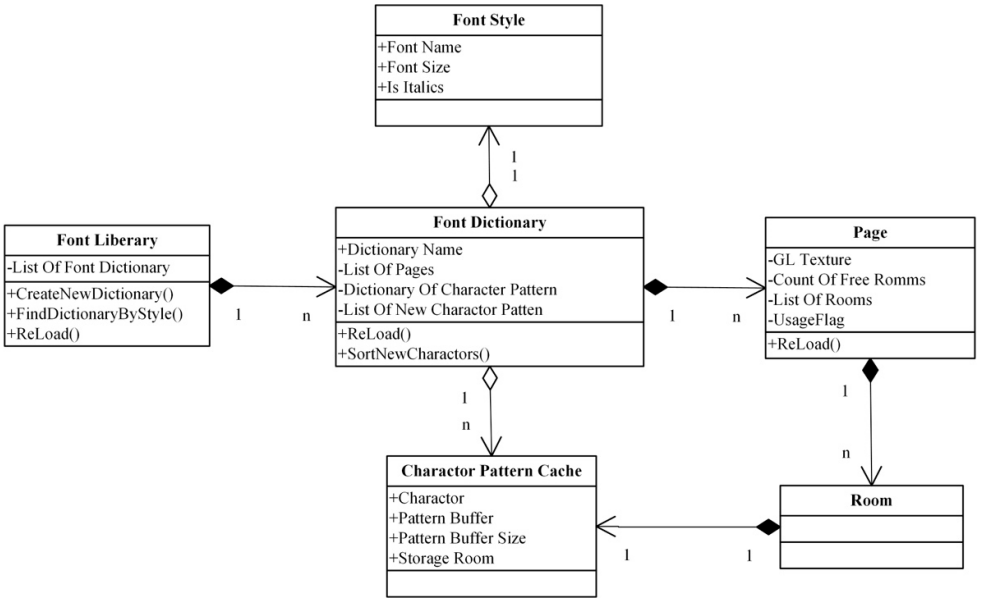

Figure 8. Polygon mask drawing process.

The GL engine proposes a dictionary-based text caching scheme based on the OpenGL. The core strategy is to copy the text font image of the same font and font size used for one rendering to each group of 16×16 and copy it to the texture page in the video memory. The multiple pages constitute the dictionary of the font under the font size. When rendering a single text, the texture is bound to the dictionary page corresponding to the text, and the text can be rendered by texture mapping. The same dictionary only needs to be updated with local textures to be reused in multiple renderings. With this scheme, the full-screen text rendering of the map can be completed in 0.1 seconds.

4. Experiments and results

To test the performance of the GL map rendering engine, this study uses the same national map data (less than 1:1 million scale using China data and more than 1:1 million scale using Beijing data), full-element map data, a total of 268 layers. The total size of spatial data 30.25 GB. In this study, the benchmark experiment is running on iOS system using Apple's 12.9-inch iPad Pro (32 GB/WIFI version) mobile device, the parameters of the iPad are 2.26 GHz clock speed, 12.9-inch screen, 2732x2048 pixel screen resolution, 4GB system memory, and 32 GB storage capacity. The benchmark experiment is also running on Android system using Xiaomi's Pad 4. The parameters of the Xiaomi's Pad 4 are 2.2 GHz clock speed, 8-inch screen, 1920x1200 pixel screen resolution, 4 GB system memory, and 64 GB storage capacity. The benchmark experiments between GL map rendering engine and the general map rendering engine (UGraphics) were compared. The results of the experiments showed that the performance of the GL map rendering engine was 1.75 times better than that of UGraphics (as shown in Table 1) in the iOS system and the performance of the GL map rendering engine was 1.54 times better than that of UGraphics (as shown in Table 2) in the Android system.

Table 1. Performance comparison between UGraphics and the GL rendering engine (iOS).

Map scale	UGraphics rendering time	GL map rendering engine	Performance ratio (UGraphics /GL)
1：20000000	12″99	17″23	0.75
1：10000000	14″37	16″65	0.86
1：5000000	11″74	10″64	1.1
1：2500000	50″31	27″31	1.84
1：1000000	27″99	8″99	3.11
1：500000	12″21	9″96	1.23
1：250000	12″02	7″58	1.59
1：100000	15″64	5″96	2.62
1：50000	14″52	5″05	2.88
1：250000	13″72	6″88	1.99
1：10000	11″23	4″12	2.73
1：5000	4″42	3″89	1.14
1：2000	3″40	3″95	0.86
Average			1.75

Table 2. Performance comparison between UGraphics and the GL rendering engine (Android).

Map scale	UGraphics rendering time	GL map rendering engine	Performance ratio (UGraphics /GL)
1：20000000	13″34	14″35	0.93
1：10000000	14″87	17″85	0.83
1：5000000	11″36	09″74	1.17
1：2500000	55″45	28″27	1.96
1：1000000	29″76	7″67	3.88

1：500000	14″41	10″02	1.44
1：250000	11″02	10″64	1.03
1：100000	15″08	10″08	1.50
1：50000	21″24	06″53	3.25
1：250000	13″10	08″93	1.47
1：10000	05″23	05″97	0.88
1：5000	04″72	05″48	0.86
1：2000	04″30	05″20	0.83
Average			1.54

In order to test the performance of the GL map rendering engine for rending vector map tiles, the benchmark of the experiments between the GL map rendering engine and the MapBox map rendering engine were compared (Figure 9). The performance of the GL map rendering engine is 11.89 times better than that of Mapbox performance rendering in the iOS system (Table 3) and 9.54 times better than that of Mapbox performance rendering in the Android system (Table 4) .

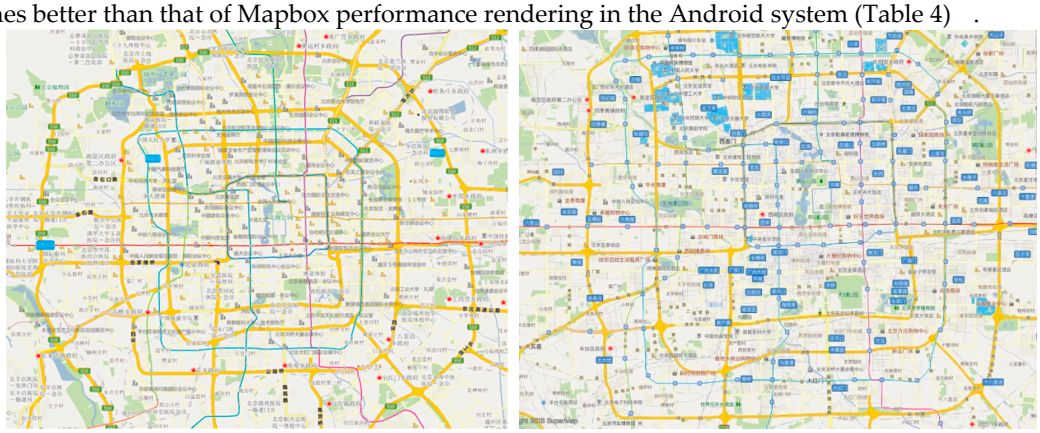

Figure 9. (a) Map in GL map rendering engine (b) Map in Mapbox.

Table 3. Performance comparison between GL map rendering engine and the Mapbox (iOS).

Map scale	Mapbox rendering time	GL map rendering engine	Performance ratio(Mapbox/GL)
1：20000000	3″89	0″42	9.26
1：10000000	3″30	0″37	8.92
1：5000000	23″30	0″43	54.19
1：2500000	5″43	0″39	13.92
1：1000000	3″43	0″44	7.80
1：500000	2″45	0″43	5.70
1：250000	2″60	0″39	6.67
1：100000	1″10	0″39	2.82
1：50000	8″00	0″41	19.51
1：250000	6″66	0″42	15.86
1：10000	2″20	0″44	5.00
1：5000	1″00	0″42	2.38
1：2000	1″00	0″40	2.50
Average			11.89

Table 4. Performance comparison between GL map rendering engine and the Mapbox (Android).

Map scale	Mapbox rendering time	GL map rendering engine	Performance ratio(Mapbox/GL)
1：20000000	4″50	0″49	9.18
1：10000000	3″82	0″43	8.88
1：5000000	16″86	0″40	42.15
1：2500000	5″43	0″46	11.80
1：1000000	3″63	0″45	8.07
1：500000	2″60	0″41	6.34
1：250000	0″94	0″51	1.84
1：100000	1″03	0″42	2.45
1：50000	2″96	0″45	6.58
1：250000	6″87	0″48	14.31
1：10000	2″36	0″42	5.62
1：5000	1″03	0″43	2.40
1：2000	1″72	0″41	4.20
Average			9.52

5. Conclusions and Further work

In this study, the cross-platform high-performance map rendering engine was proposed and the GL map rendering engine was developed, which solves the problem that the map cannot be rendered across different platform with high efficiency. It uses graphics acceleration hardware and innovative graphics rendering algorithms to achieve high-performance rendering of the map. The map rendering color correction algorithm realizes the consistency of the map rendering style under different platforms. The results of the benchmark experiments show that the performance of GL engine is 1.75 times and 1.54 times better than the general map rendering engine in the iOS system and in the Android system, respectively, and the rendering performance for vector tiles is 11.89 times and 9.52 times better than rendering in the Mapbox in the iOS system and in the Android system, respectively.

Although the GL map rendering engine has high performance for cross-platform map rendering, with the development of information technology, high-performance rendering of 3D BIM data and real-time rendering of streaming map data are still to be explored in further work.

Author Contributions: Conceptualization, Shaojie Li, and Shaohua Wang; methodology, Shaojie Li, Shaohua Wang, Yong Guan, Kejia Huang; software, Shaojie Li, Zhiyan Xie, Ming Wen, Kejia Huang, and Lixin Zhou; writing—original draft preparation, Shaojie Li, Shaohua Wang,Yong Guan, and Kejia Huang; visualization, Shaojie Li and Shaohua Wang; supervision, Shaojie Li; project administration, Shaojie Li.

Funding: Our work was supported by the National Key R&D Plan (2016YFB0502000), independent Research Project of State Key Laboratory of Resources and Environmental Information Systems, Chinese Academy of Sciences (088RAC00YA), Project of Beijing Excellent Talents (201500002685XG242), National Postdoctoral International Exchange Program (Grant No. 20150081).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Noguera, J.M.; Barranco, M.J.; Segura, R.J.; MartíNez, L. A mobile 3d-gis hybrid recommender system for tourism. *Inf. Sci.* **2012**, *215*, 37–52.
2. Frank, C.; Caduff, D.; Wuersch, M. From gis to lbs–an intelligent mobile gis. *IFGI Prints* **2004**, *22*, 261–274.
3. Tripcevich, N. Flexibility by design: How mobile gis meets the needs of archaeological survey. *Cartogr. Geogr. Inf. Sci.* **2004**, *31*, 137–151.
4. Peng, Z.-R.; Tsou, M.-H. *Internet gis: Distributed Geographic Information Services for the Internet and Wireless Networks*; John Wiley & Sons: Hoboken, NJ, USA, 2003.

5. Tao, C.V.; Li, J. *Advances in Mobile Mapping Technology*; CRC Press: Boca Raton, FL, USA, 2007; Volume 4.
6. Reichenbacher, T. Adaptive concepts for a mobile cartography. *J. Geogr. Sci.* **2001**, *11*, 43–53.
7. Rakkolainen, I.; Vainio, T. A 3d city info for mobile users. *Comput. Graph.* **2001**, *25*, 619–625.
8. Shi, W.; Kwan, K.; Shea, G.; Cao, J. A dynamic data model for mobile gis. *Comput. Geosci.* **2009**, *35*, 2210–2221.
9. Wang, S.; Zhong, Y.; Wang, E. An integrated gis platform architecture for spatiotemporal big data. *Future Gener. Comput. Syst.* **2019**, *94*, 160–172.
10. Rosser, J.; Morley, J.; Smith, G. Modelling of building interiors with mobile phone sensor data. *ISPRS Int. J. Geo. Inf.* **2015**, *4*, 989–1012.
11. Yang, X.; Fang, Z.; Xu, Y.; Shaw, S.-L.; Zhao, Z.; Yin, L.; Zhang, T.; Lin, Y. Understanding spatiotemporal patterns of human convergence and divergence using mobile phone location data. *ISPRS Int. J. Geo. Inf.* **2016**, *5*, 177.
12. Li, M.; Sun, Y.; Fan, H. Contextualized relevance evaluation of geographic information for mobile users in location-based social networks. *ISPRS Int. J. Geo. Inf.* **2015**, *4*, 799–814.
13. Oulasvirta, A.; Estlander, S.; Nurminen, A. Embodied interaction with a 3d versus 2d mobile map. *Pers. Ubiquitous Comput.* **2009**, *13*, 303–320.
14. Meng, L.; Reichenbacher, T. Map-based mobile services. In *Map-Based Mobile Services*; Springer: Berlin/Heidelberg, Germany, 2005; pp 1–10.
15. Vaaraniemi, M.; Freidank, M.; Westermann, R. Enhancing the visibility of labels in 3d navigation maps. In *Progress and New Trends in 3D Geoinformation Sciences*; Springer: Berlin/Heidelberg, Germany, 2013; pp 23–40.
16. Agrawal, A.; Radhakrishna, M.; Joshi, R. *Geometry-Based Mapping and Rendering of Vector Data Over Lod Phototextured 3D Terrain Models*; Vaclav Skala Union Agency: Plzen, Czech Republic, 2006.
17. Wartell, Z.J.; Kang, E.; Wasilewski, A.A.; Ribarsky, W.; Faust, N.L. *Rendering Vector Data Over Global, Multi-Resolution 3D Terrain*; Georgia Institute of Technology: Atlanta, GA, USA, 2003.
18. Schneider, M.; Klein, R. *Efficient and Accurate Rendering of Vector Data on Virtual Landscapes*; Vaclav Skala Union Agency: Plzen, Czech Republic, 2007.
19. Li, Q. Opportunities in mobile gis. In *Dynamic and Mobile Gis*; CRC Press: Boca Raton, FL, USA, 2006; pp 47–62.
20. Eleiche, M.A. *Network Analysis Methods for Mobile Gis*; NYME: New York, NY, USA, 2011.
21. Shichao, G.X.S.Z.Y. On rendering inconsistency of c/s structural multi-client in arcgis. *Comput. Appl. Softw.* **2011**, *2*, 32.
22. Li, S. A cross-platform graphics library for mobile gis. In *SuperMap Conference*; Geomatics and Spatial Information Technology, 2011; pp 43–45.
23. Geometry, A.-G. *Adaptive Subdivision of Bezier Curves: An Attempt to Achieve Perfect Result in Bezier Curve Approximation*. July: Singapore, 2005.
24. Lu, L.; Liu, X. Agg (anti-grain geometry) based platform-independent graphics interface design. *Microcomput. Inf.* **2009**, 25(*6*), *266–267*.
25. Lu, W.; Wang, K.; Wu, Y.; Dou, C. Research on efficiency and cross-platform of graphics rendering engine. *Comput. Eng. Des.* **2016**, *37*, 1400–1404.
26. Yue, S.; Yang, J.; Chen, M.; Lu, G.; Zhu, A.-x.; Wen, Y. A function-based linear map symbol building and rendering method using shader language. *Int. J. Geogr. Inf. Sci.* **2016**, *30*, 143–167.
27. Blanchette, J.; Summerfield, M. *C++ Gui Programming With Qt 4*; Prentice Hall Professional: Upper Saddle River, NJ, USA, 2006.
28. Thelin, J. *Foundations of Qt Development*; Apress: New York, NY, USA, 2007.
29. Hugentobler, M. Quantum gis. *Encycl. GIS* **2008**, 935–939, doi:10.1016/j.ecoinf.2009.07.004
30. Merkopolo. Merkopolo Project. Available online: https://gitorious.org/merkopolo/merkopolo (accessed on 12, 07, 2019).
31. Teodoro, A.C.; Duarte, L. Forest fire risk maps: A gis open source application–a case study in norwest of portugal. *Int. J. Geogr. Inf. Sci.* **2013**, *27*, 699–720.
32. Nurminen, A.; Helin, V. Technical Challenges in Mobile Real-Time 3D City Maps With Dynamic Content. In *IAESTED Software Engineering*; Citeseer: Pennsylvania State University University Park, PA, USA, 2005.
33. Angel, E.; Shreiner, D. *Interactive Computer Graphics with Webgl*; Addison-Wesley Professional: Boston, MA, USA, 2014.

34. Congote, J.; Segura, A.; Kabongo, L.; Moreno, A.; Posada, J.; Ruiz, O. Interactive visualization of volumetric data with webgl in real-time. In Proceedings of the 16th International Conference on 3D Web Technology, New York, NY, USA, 20–22 June 2011; pp. 137–146.
35. Jie, Y.; Pin, L.; Changwen, Z. A comparative research on methods of delaunay triangulation. *J. Image Graph.* **2010**, *15*, 1158–1167.
36. Chen, X.; McMains, S. Polygon Offsetting by Computing Winding Numbers, ASME 2005. In Proceedings of the international design engineering technical conferences and computers and information in engineering conference, Long Beach, CA, USA, 24–28 September 2005.