Visualization of Vector Data on Global Scale Terrain

Baosong Deng, Dong Xu
Information Research Center, Institute of logistics science, Beijing 100071, China
E-mail: dbs310@163.com

Jinxia Zhang, Chiyang Song
North China Institute of Computing Technology, Beijing 100083, China
skylarkspring@gmail.com

***Abstract*—Vector represents a major category of data managed by GIS, which is traditionally used for representing geographic entities such as political borders, roads, rivers and cadastral information. In this paper we present a key data structure and associated render-time algorithm for the combined display of multi-resolution 3D terrain and traditional GIS vector data, which is designed to create fully 3D scenes that deliver the best possible quality and do not require dynamic texture generation and handling. The algorithm allows the system to adapt the visual mapping to the context and user needs, and enables users to interactively modify vector through the visual representation, which represents a basic mechanism for 3D GIS interface and facilitates the development of visual analysis and exploration tools.**

Keywords-vector; visulazation; terrain; matching;

I. INTRODUCTION

Real-time visualization of large scale terrains has been an active area of research for more than a decade[1]. In geographical information systems (GIS), vector data has important applications in the analysis and management of virtual landscapes. Therefore, methods that allow combined visualization of terrain and geo-spatial vector data are required. These data are typically organized as either raster or vector data. Vector data represent geometry as lists of coordinates that define points, lines and polygons. A key issue in both the computer graphics and the GIS community is eliminating unnecessary or unwanted detail from the output image [2]. GIS geometric simplification is one aspect of the more general methods, which are methods for visually representing a given geographic entity or set of entities using different visuals when displaying the entities at different map scales.

A fundamental task in GIS applications is to render vector data representing elements such as road networks, political districts and rivers. Rendering vectors such that they precisely drape over terrain is challenging. There are two general approaches to rendering vector data on a 3D mesh. One option is to convert the vector data to a texture image layer, and the second option is to render the vector data as geometric primitives [3]. For a number of reasons, we believe that vector data should be treated independently from the image data and therefore should be rendered as separate geometry by the graphics pipeline [3]. This presents a challenge because modern algorithms render a LOD terrain whose constituent triangles are changing at every frame. In order for the vector data to appear correctly on the 3D mesh, the vector geometry must therefore also change at each frame. In addition, Schneider present a method that is based on stencil shadow volume algorithm and allows high quality real-time overlay of vector on terrain[4]. The method render vector using a screen-space algorithm, so can avoid aliasing artifacts and its performance is almost independent of terrain geometry.

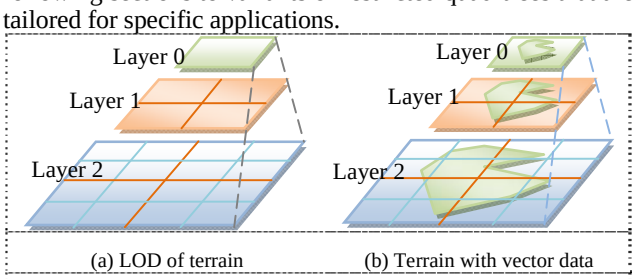
In this paper, we presents a key data structure and associated algorithms for overlaying traditional GIS data on a global, 3D terrain visualization system. Although the 3D terrain, image layers and 2D vector data are too large to fit into primary memory. This requires dynamic paging of all three data types based on the current 3D view of the database. Finally, we present the results and performance analysis of the implemented algorithm.

II. MULTI-RESOLUTION DATA GENERATION

For large-scale 3D terrain display, some technologies are come into mature. Lindstrom presented SOAR algorithm [5] and Mark presented ROAM algorithm[6], both become famous algorithm for terrain mesh simplification.

A. *Terrain LOD generation*

Dealing with huge amounts of data is often a difficult challenge. The basic data hierarchy on which our approach is based is the restricted quad-tree, which ensures that the levels of neighboring quads differ by not more than one. This is a generic and flexible data structure, and there are multiple ways to construct consistent triangle meshes from the hierarchy. It is easy to apply the techniques presented in the following sections to variants of restricted quad-trees that are tailored for specific applications.



(a) LOD of terrain (b) Terrain with vector data

Figure 1. A quad-tree structure of Terrain

Simplification algorithms take a terrain model as input and produce a simplified version of it, which provides a coarser representation of the same terrain, based on a smaller data set. Most methods are based on the iterated or simultaneous applications of local operators that simplify small portions of the mesh by reducing the number of vertices. The patch hierarchy is constructed top-down by subdividing each patch into 2×2 children patches.

To render the scene, a rendering quad-tree is built that gives at first, conservative approximation of the necessary level of detail for different parts of the scene. This quad-tree determines which quads from the data set quad-trees need to be used. An initial mesh can be constructed directly from the rendering tree. This initial mesh is then refined to produce a tailored mesh that guarantees consistency between neighboring quads. An overview and example result of the process is given in Figure 1.

B. *Vector data procession*

In order to match the DEM data levels from the pyramid structure, corresponding scales of vector data are required. In our approach, we use dynamic generation method to satisfy this requirement. One simple method is to make dense points of vector objects to be sparse depend its distance from the viewpoint. In our approach, we adopt method presented by Sun [7]. As shown in figure 1 (b), where dash line means that vector data is just indexed but not divided into quad-tree blocks.

III. TERRAIN MATCHING OF VECTOR

Geometry based methods treat vector data as graphic objects, and can be classified into points, polylines and polygons. As a pyramid structure is used for loading terrain data in real time depend on viewpoint, so at each frame just a few blocks are refreshed, this make smooth rendering is feasible when display card has a good capabilities. Generally, the most time of consumption for geometry based vector visualization approach spend on triangles searching and intersection calculation.

A. *Point vector matching*

Point vector are very common in GIS, such as city name, label, which cannot be rendered under terrain surface. Simply disabling depth test or setting render order cannot display the relative location correctly, especially when the view point is near from this position.

When high resolution tiles loaded from out-of-memory, the altitude of the point will be updated using the newest terrain mesh, and the altitude of the point must be calculated as soon as possible. For point labels on the ground, set the triangle contain this point is T, with vertex $V_0(x_0, y_0, z_0)$, $V_1(x_1, y_1, z_1)$, and $V_2(x_2, y_2, z_2)$, then the equation of the plane of this triangle can be expressed as:

$$\begin{vmatrix} x & y & z & 1 \\ x_0 & y_0 & z_0 & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \end{vmatrix} = 0$$

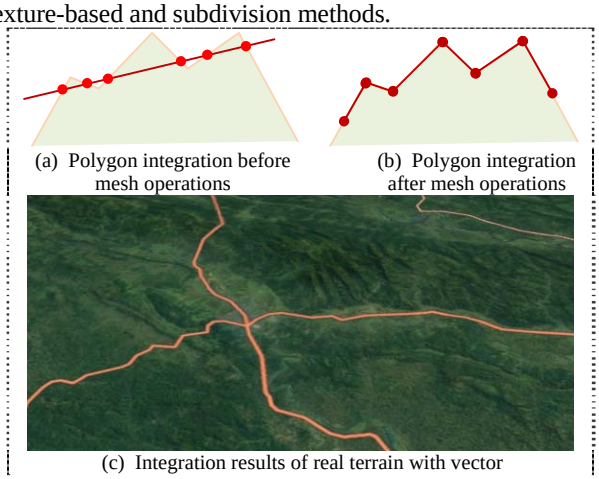
Then the altitude of the point can be computed as:

$$z = z_0 - \frac{(x-x_0)(y_{10}z_{20}-y_{20}z_{10})+(y-y_0)(z_{10}x_{20}-z_{20}x_{10})}{x_{10}y_{20}-x_{20}y_{10}}$$, With

$x_{10} = x_1 - x_0$, $x_{20} = x_2 - x_0$, $x_{10} = x_1 - x_0$, $y_{10} = y_1 - y_0$ $y_{20} = y_2 - y_0$, $z_{10} = z_1 - z_0$, $z_{20} = z_2 - z_0$.

B. *Polyline vector matching*

There is a traditional simplification algorithm, which can maintain important geometric characteristics of polylines. However, topological relationships may change, resulting in topological inconsistencies. A common approach is to render polylines to a texture, which leads to aliasing as the viewer zooms in. Rendering polylines on terrain by subdivision of points along the lines requires adjustment in response to terrain LOD tiles and the coplanar geometry will lead to z-fighting [3]. Our approach requires no pre-processing, is decoupled from terrain LOD, produces high visual quality constant pixel-width polylines with no cracking or smearing, has virtually no CPU overhead, allows for dynamic terrain and polylines, and is simpler to implement than on-demand texture-based and subdivision methods.



(a) Polygon integration before mesh operations (b) Polygon integration after mesh operations

(c) Integration results of real terrain with vector

Figure 2. matching process of Polyline vector

First, each point in the polyline is transferred to the spherical space and duplicated. One is moved away from the global centre to above the terrain surface and the other below, forming a vertical line, then a wall across the polyline with many vertical lines. The intersection of this wall with the terrain defines the desired polyline. The intersection is found in screen-space using a fragment shader that has access to the terrain's depth and silhouette textures. The tessellation control fragment shader uses it to decide the subdivision level of each patch, while the tessellation evaluation shader uses it to place each generated vertex at the proper height value. During this stage, we make the best of parallel character of GPU, then an improvement of performance will get. If a fragment is in front of the terrain and one of the surrounding fragments from the wall is behind terrain, a potential intersection is found. To avoid false positives, fragments on the terrain's silhouette are detected using the silhouette texture and discarded. Our method will not generate fragments when the wall is viewed on-edge, then a shadow volume is rendered to handle these cases.

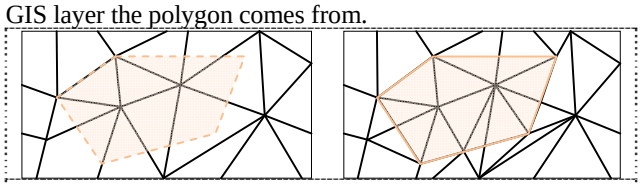
C. *Polygon vector matching*

The creation of 3D polygon geometry primitives is complex as it needs to determine it's inside surface that is usually not one plane. In order to keep coincident with LOD tiles of terrain surface, the aim of 3D polygon geometry creation in our approach is to find triangles of terrain tiles which are inside and intersect with 2D vector polygon.

The polygon borders are treated in a similar way as the polylines described above. The interior of the polygon is

triangulated and added to the new geometry recursively, to which a z-offset is added in order to avoid interference with the terrain and rendering artifacts. If the whole vector data structure is rendered in the frame buffers, each vector feature is significantly zoomed out, even just occupies several pixels. In this case, a segment may be occluded by other segments so that we cannot identify or remove all non-compliant shortcut segments at a time.

Polygonal data sets such as building blocks, green areas, forests, and roads are loaded from 2D GIS layer files and integrated into a custom Constrained Delaunay implementation for the terrain. The main properties of this method is, that the shape of the terrain is not altered, and z-buffer problems do not occur, since the resulting terrain is still a single continuous surface but with integrated areas that represent the input 2D shapes. After the geometrical integration operation we can identify the triangles lying completely inside the area and mark them as owned by the GIS layer the polygon comes from.



(a) Polygon integration before mesh operations (b) Polygon integration after mesh operations

Figure 3. matching process of polygon vector

Usually, the short segment formed by two vertices far away from each other is likely to exceed the e-Voronoi zone. In that condition, we can just link a vertex with the nearer m (m is a positive integer) vertices to create shortcut segments. If the sample interval of the map is small, the variable m should be set larger. Otherwise, m can be smaller. Compared with the geometry algorithms, our method preserves topological consistency efficiently, and avoids dealing with unnecessary shortcut segments. In the following section, we present how to keep the topological relationships.

IV. Visualization of Vector Data

An obstacle to be tackled is that the detailed geometry and texture representation of terrain surfaces demands a large amount of memory. To accomplish the goal for transmission and rendering of massive geographical maps, our approach combines view-dependent LOD simplification and out-of-core rendering algorithm. Fig. 4 illustrates the architecture of our scheme. The architecture is composed of three components: the client, the server and the database/files.

When rendering, the client sends the requests to the server for multi-resolution data falling inside the current view frustum. The server delivers the base models of the spatial objects to the client first. Then, the client interacts with the users and sends the view parameters to the server for fine models update. The simplification process of the server produces a stream of map simplification operators. These update operators are sent to the client and the selective model is reconstructed and rendered. As the viewers navigate closer to the interested region, the details will be updated and refined accordingly. After all the datasets reach the client or the added details become imperceptible to the viewers, the process stops.

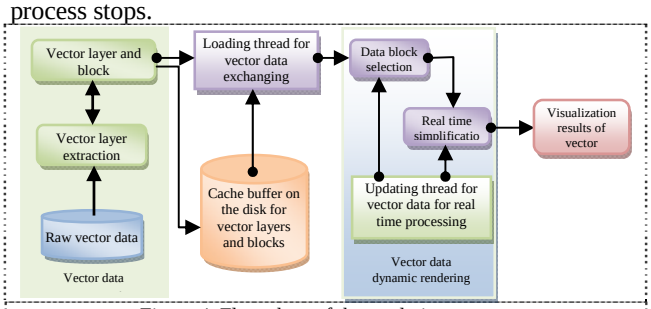


Figure 4. Flow chart of the rendering process

Eliminating the phenomenon of image popping and maintaining a higher frame rate of graphics rendering is always two major problems need to solve for visualization simulation algorithm of large-scale terrain. For the usage of very large terrain models, it will be inevitable to use more than one processor in parallel. Higher demands in memory capacity, processing speed, drawing speed and transfer efficiency appear. Multi-thread data loading was used, and for the usage of very large terrain models, it will be inevitable to use more than one processor in parallel. On one hand, independent parts of the landscape can then be updated independently. On the other hand, only multi-threading will ensure that the user-interaction and the update-process can be handled in parallel.

V. Results and Conclusions

The approach in this paper has been successfully used for rendering of 3D vector data in several areas. We implemented above idea and method in MS visual studio C++ environment, and tested the performance of our application, based on low level graphic API, OpenGL, and the window size is in all cases 1024×768 pixels. We used a 2.66GHz Pentium Dual-Core CPU, with 2GB DDR2 of RAM, NVIDIA GeForce 9300M GS programmable graphic card with 256M RAM, and SATA 500G disk.

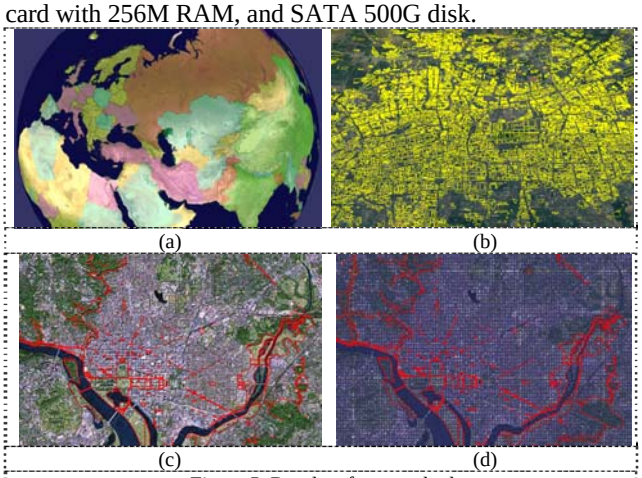


Figure 5. Results of our method

The DEM data set consists of global sample points with the ground sample distance of 20m, and the image consists of global data with the test area of resolution of 5m. The test vector layer includes 8 areas of 3628 sample points and the

polyline vector layer includes 144 roads of 18800 sample points. Before rendering the vector data, the landscape is rendered at 60 frames per second, after the vector data are overlaid on the terrain, the rendering speed keeps unchanged, only the memory cost increases 6.3MB. Figure 5 shows the rendering result of the habitation (semi-transparent yellow area) and road (red line) data on the 3d terrain landscape.

Rendering vector data can enhance many invisible information in 3D landscape map, and made 3D landscape map more close to a 3DGIS. Both pyramid structure and index of quad-tree can improve large-scale vector data of multiple scales to be load easily in real time. In order to overcome expensive computation of real time intersection of terrain mesh with vector line segments, this paper presents a framework and correlative approach for rendering large scale geographical data efficiently, including terrain grid and vector data. Using GPU based graphics hardware; we achieve vector data simplification efficiently. Based on the simplification, we establish LOD vector models. Finally, we implement dynamically reconstruction of large scale terrain with all types of vector data. We are going to further improve the algorithm as proposed in this paper. The future work includes (1) storing vector layers and blocks with different relative importance with a hierarchical data structure, (2) adapting the animation speed to the viewpoint moving speed, and (3) incorporating textures layers into the real-time rendering system.

REFERENCES

[1] Nie, J., et al., Multilevel tile load map on massive terrain visualization. Journal of Computational Information Systems, 2011. 7(2): p. 452-461.

[2] Zhang, Y., Q. Huang, and J. Han. Real-time rendering of large-scale terrain based on GPU. in 4th IEEE Conference on Industrial Electronics and Applications. 2009. Xi'an, China: IEEE Computer Society, 445 Hoes Lane - P.O.Box 1331, Piscataway, NJ 08855-1331, United States.

[3] Agrawal, A., M. Radhakrishna, and R.C. Joshi, Geometry-based Mapping and Rendering of Vector Data over LOD Phototextured 3D Terrain Models. The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Plzen-Bory, Czech Republic, 2006, 2006.

[4] Schneider, M. and R. Klein, Efficient and Accurate Rendering of Vector Data on Virtual Landscapes. 2007.

[5] Lindstrom, P. Out-of-core construction and visualization of multiresolution surfaces. in ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics. 2003. Monterey, CA, United states: Association for Computing Machinery.

[6] Duchaineau, M., et al. ROAMing terrain: real-time optimally adapting meshes. in Proceedings of the IEEE Visualization Conference. 1997. Phoenix, AZ, USA: IEEE Comp Soc, Los Alamitos, CA, United States.

[7] Sun, M., 3D Visualization Method of Large-scale Vector Data for Operation. In Proceeding of the Fourth International Conference on Cooperative Design, Visualization and Engineering, 2007, Shanghai, 2007.