RENDERING 3D VECTOR DATA USING THE THEORY OF STENCIL SHADOW VOLUMES

Chenguang Dai, Yongsheng Zhang, Jingyu Yang

Zhengzhou Institute of Surveying and Mapping,
Zhengzhou, PR. China- (dai_zism, yszhang2001, jyyang)@163.com

Commission II, WG II/5

KEY WORDS: Stencil Buffer, Shadow Volume, Vector Data, Real-time Visualization, Terrain Rendering

ABSTRACT:

In geographical information systems vector data has important applications in the analysis and management of virtual landscapes. Therefore, methods that allow combined visualization of terrain and geospatial vector data are required. Such methods have to adapt the vector data to the terrain surface and to ensure a precise and efficient mapping. In this paper, a rendering method based on the stencil shadow volumes theory is presented, which allows high-quality real-time overlay of vector data on virtual landscapes. The method includes three steps which are the generation of the vector data polyhedra, creation of the mask in stencil buffer and the application of the mask to the scene to render the vector data. The research and experiment show that our approach is a screen-space algorithm and is of pixel-level precision. Since it does not suffer from aliasing artifacts like texture-based techniques and independent of the complexity of the terrain data set, it can achieve good performance for the rendering of 3D vector data.

1. INTRODUCTION

In geoscience analytical vector data represents one of the main categories managed by geoinformation systems. The vector data is typically represented by lists of coordinates defining points, lines, polygons, etc. These primitives are traditionally used for describing geographic entities, for example buildings, rivers, and vegetation or soil types. In GIS, vector data has important applications in the analysis and management of virtual landscapes. Therefore, methods that allow combined visualization of terrain and geospatial vector data are required. Such methods have to adapt the vector data to the terrain surface and to ensure a precise and efficient mapping.

Real-time terrain rendering techniques have been extensively studied for a long time but methods for projecting additional vector data on a virtual landscape have gained less attention. The few methods existing so far can basically be divided into texture-based and geometry-based approaches. The texture-based approach is to rasterize the vector data into a texture and use standard texture mapping techniques to project it onto the mesh during rendering. The geometry-based approach is to map the vector data to 3d geometric primitives and to render them as separate geometry with an additional offset. Both techniques comprise a number of challenges.

Rasterization of the 2d vector data into a texture in a preprocessing step and then rendering the terrain using standard texture mapping implies several drawbacks. The frequently needed combination of several layers of geo-spatial data demands a separate texture for each data layer resulting in high memory requirements. Multitexturing techniques have to be applied to project the textures for different geospatial information onto the textured terrain. What is worse, the accuracy of the vector data is bound by the texture resolution leading to unpleasant results when zooming in and single texels become visible.

A texture-based approach to visualize vector data was proposed by Kersting et al. (Kersting, 2002). Textures containing the vector data are generated on-the-fly using p-buffers. An on-demand texture pyramid that associates equally sized textures with each quadtree node is used to improve visual quality when zooming in. However, many expensive p-buffer switches have to be performed, which leads to decreased rendering performance. Even with more recent and efficient extensions (e.g. framebuffer objects) each switch still requires a complete pipeline flush. A texture-based approach is presented by Schneider (Schneider, 2005) that also creates textures on-the-fly in an offscreen buffer. A perspective reparameterization adopted from perspective shadow mapping is applied taking into account the current point-of-view.

Most 3d representations are based on a level-of-detail terrain model which is needed to handle large terrain data sets and whose geometry is refined according to the viewpoint. If vector data is mapped to such a multiresolution structure, it has to be adapted to the current level-of-detail in order to avoid rendering artifacts. Unfortunately, this procedure leads to an increase in the number of geometric primitives compared to the original 2d vector representation. Furthermore, a suitable z-offset has to be added to the created primitives during rendering in order to avoid z-buffer stitching artifacts.

Wartell et al. (Wartell, 2003) presented an algorithm and an associated data structure that allows rendering of polylines on multiresolution terrain geometry. Since their system is based upon a continuous level-of-detail terrain rendering engine, an adaption of the polyline to the current state of geometry is required at runtime resulting in additional computational costs. In addition to the previously mentioned texture-based approach, Schneider et al. also presented a geometry-based approach for rendering engines based on static level-of-details (Schneider, 2005). The vector data geometry is mapped to each LOD in a preprocessing step and integrated in the used quadtree ensuring rendering of corresponding terrain and vector data LODs. Since the number of geometric primitives that have to be created grows with the terrain complexity, this method is not suited for very high resolution data sets, especially for vector data covering large areas.

In this paper, we treat the rendering of vector data on virtual terrain as the vertical projection of the vector data on the surface of the terrain. A rendering method based on the stencil shadow volume theory is studied, which allows high-quality real-time overlay of vector data on virtual landscapes. The method includes three steps which are the generation of the vector data polyhedra, creation of the mask in stencil buffer and the application of the mask to the scene to render the vector data.

The paper is structured as follows. First, we briefly introduce the research background and review the related work. Then, in section 2 we explain the theory of stencil shadow volumes. We describe our approach in detail in section 3, show experiment result in section 4 and draw conclusions in section 5.

2. THEORY OF STENCIL SHADOW VOLUMES

Shadow volumes are a technique used in 3D computer graphics to add shadows to a rendered scene. They were first proposed by Frank Crow in 1977 (Crow, 1977). A shadow volume is an enclosed area of space that looks somewhat like a cone or a pyramid. The tip of the shadow volume is the light source, the faces of the shadow volume are determined by the outline of the object that is casting the shadow. Any part of the scene that falls inside the shadow volume is shadowed, any part of the scene that falls outside the shadow volume is lit.

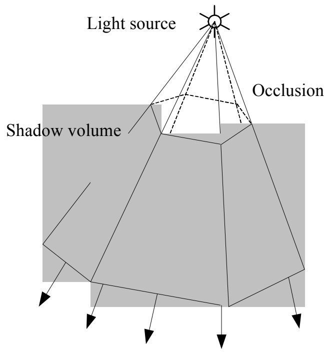

Figure 1. The illustration of shadow map

The method to test whether a pixel in the screen is inside the shadow volume or not is called shadow test. The shadow test is often realized by using the technology of the depth buffer and the stencil buffer, which can be divided into z-pass algorithm and z-fail algorithm according to the light ray used in the test (Everitt, 2002).

2.1 Z-pass algorithm

Figure 2 shows the theory of z-pass algorithm. For a pixel in the screen, we consider the ray starting from the camera (viewpoint). When the ray enters into the shadow volume, the stencil value increases one, otherwise, when the ray runs out the shadow volume, the stencil value decreases one. Finally if the stencil value is greater than zero, it means that the corresponding pixel is inside the shadow, otherwise if the stencil value is equal to zero, the pixel is outside the shadow. See Figure 2, the ray leaving the camera for point P enters into the shadow volume of object 2, into the shadow volume of object 1, out the shadow volume of object 2, into the shadow volume of object 3, out the shadow volume of object 3, out the shadow volume of object 1, the final stencil value of P is +1+1−1+1−1−1=0, which means that the point P is outside the shadow. Again for the point Q, the ray runs into the shadow volume of object 2, into the shadow volume of object 1, out the shadow volume of object 2, into the shadow volume of object 3, and the stencil value is +1+1−1+1 =2>0, which means the point Q is inside the shadow.

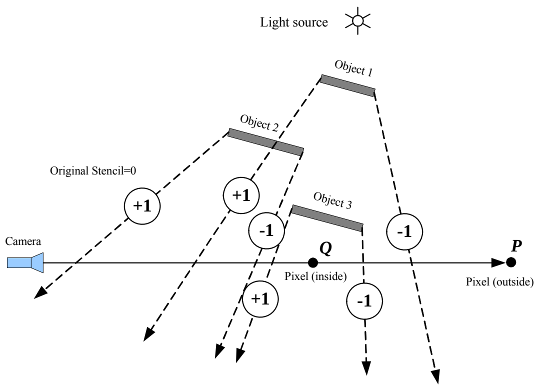

Figure 2. The z-pass algorithm

Note that in the z-pass algorithm, if the shadow volume intersects with the near clipping plane of the view frustum, the additional surface must be added to the shadow volume to ensure the correct result, the reason of which is that after clipped by the view frustum, the shadow volume is likely to be open. See Figure 3, if the additional cap is not added, the stencil value will not increase when rendering the shadow volume, and the shadow test of those points in the affected area will fail.

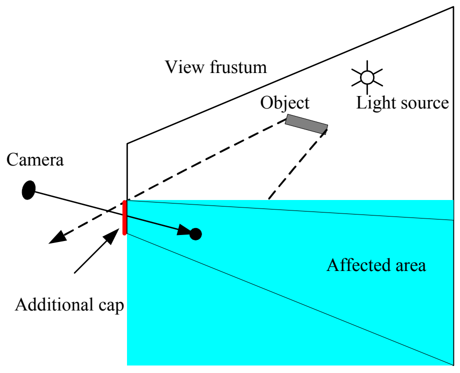

Figure 3. The z-pass algorithm affected by the view frustum

2.2 Z-fail algorithm

If the camera (view point) is inside the shadow volume, the performance of z-pass algorithm would fail. See Figure 4, if the z-pass algorithm is applied, the stencil value of Q is −1+1=0, which means Q is outside the shadow, but in fact it is inside the shadow. Here we should use the z-fail algorithm (Heidmann, 1991). In the z-fail algorithm, we extend the ray from the camera to the infinite. Starting from the pixel, those shadow volumes farther than the pixel are considered. When the ray runs out the shadow volume, the stencil value increases one, otherwise, when the ray runs into the shadow volume, the stencil value decreases one. If the final stencil value is greater than zero, the corresponding pixel is inside the shadow; otherwise if the final stencil value is equal to zero, the pixel is outside the shadow. If the pixel is outside of the all shadow volumes, the father shadow volume is not exist and the stencil value keeps unchanged, that is the value is zero and the pixel is outside the shadow.

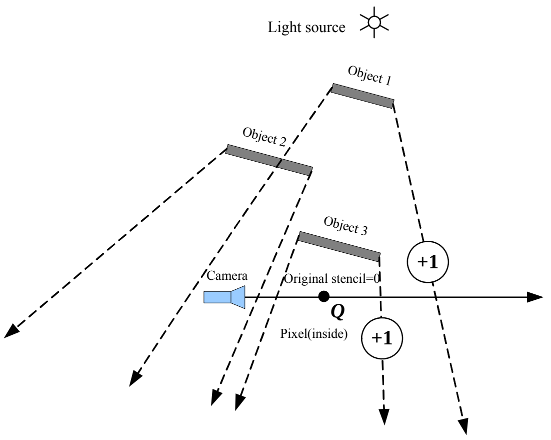

Figure 4. The z-fail algorithm

Note that in the z-fail algorithm, the shadow volume must be closed at both ends because the stencil value is counted when the pixel fail the depth test. See Figure 5, if the top cap and the bottom cap are not added, the stencil value of the P and Q would be zero, but the correct value is one because they are both in the shadow.

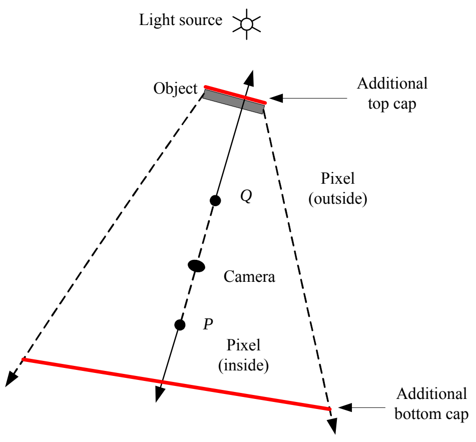

Figure 5. The front and back caps added for z-fail algorithm

2.3 Comparison of z-pass and z-fail algorithm

In the shadow volume algorithm the z-pass method has a higher performance than the z-fail method. But the z-pass algorithm fails when the shadow volume intersects the near clipping plane. This near clipping problem was the reason for the development of the z-fail technique which processes shadow volume fragments that fail (instead of pass) the depth test. This approach moves the problems from the near to the far clipping plane which can be handled robustly by moving the far plane to infinity. However, this robustness comes at the expense of performance since in the z-fail case the shadow volumes must be closed at both ends. In this paper, we make use of the advantages of the both algorithm to realize the rendering of 3D vector data on the virtual landscape.

3. RENDERING 3D VECTOR DATA BASED ON THE THEORY OF STENCIL SHADOW VOLUMES

The approach of rendering 3d vector data based on the theory of stencil shadow volumes consists of three parts (Schneider, 2007):

(1) Constructing the vector polyhedra: extrude the vector data into the polyhedra according to the bounding box of the terrain data where the vector data lie.
(2) Creating the mask: render the polyhedra to the stencil buffer to create a mask, which is consistent with the vertical projection of the vector data onto the terrain surface.
(3) Rendering the vector: apply the mask to the scene and raster the screen area covered by the vector data by using the appropriate stencil test method.

The detail work flow of the approach is as the Figure 6:

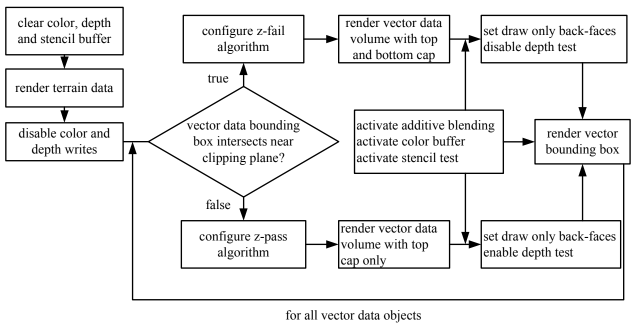
clear color, depth and stencil buffer
render terrain data
disable color and depth writes
vector data bounding box intersects near clipping plane?
true
false
configure z-fail algorithm
render vector data volume with top and bottom cap
set draw only back-faces disable depth test
activate additive blending activate color buffer activate stencil test
render vector bounding box
configure z-pass algorithm
render vector data volume with top cap only
set draw only back-faces enable depth test
for all vector data objects

Figure 6. The work flow of the approach of rendering 3d vector data based on the theory of stencil shadow volumes

3.1 Construction of vector polyhedra

In the first step we need to extrude the vector data geometry into vector polyhedra that are afterwards rendered into the stencil buffer to generate an appropriate mask. Construction is started by duplicating each vertex of the vector data. One vertex of each of the created pairs is translated towards the geocenter, the remaining vertices are moved into the opposite direction. The group of upper and lower vertices constitutes the polyhedron's top and bottom cap. The amount of translation has to be chosen such that the top and bottom cap are located completely above and below the terrain surface respectively. Applying the described construction the resulting polyhedron encloses the part of the terrain surface that is supposed to contain the vector data.

In order to minimize the high rasterization workload potentially caused by large polyhedra, we reduce the size of the polyhedra. To accomplish this, we move the top and bottom caps towards the terrain surface from both sides as far as possible but without intersecting it. In our implementation we utilize the bounding boxes of the quadtree cells inherent in the terrain rendering engine (Dai, 2004). In particular, the bounding boxes encode an upper and lower bound of the enclosed terrain and therefore provide conservative but reasonable upper and lower bounds for the polyhedra caps as well. In the case of linestrips, before constructing the polyhedra the linestrips are needed to be extended to the strips according to the attribution of the data (for example, the width of the road), just as the Figure 7 shows.

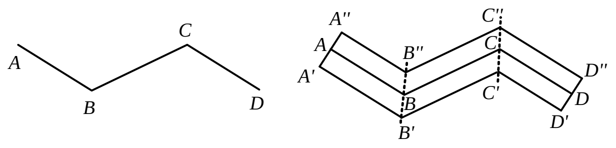

Figure 7. The extension of linestrip vector data

When constructing the polyhedra of linestrips vector data, we consider each line segment separately. The height values of the corresponding vertices of the top and bottom cap are the minimum and maximum height values of the bounding boxes containing the projection of the line segment. See Figure 8, point A, B, C, D are the vector data point, point A_1, B_1, C_1, D_1 are the upward-moved points and A_2, B_2, C_2, D_2 are the downward-moved points of each vector data point, the above moved points construct the polyhedra of the vector data enclosing the terrain data.

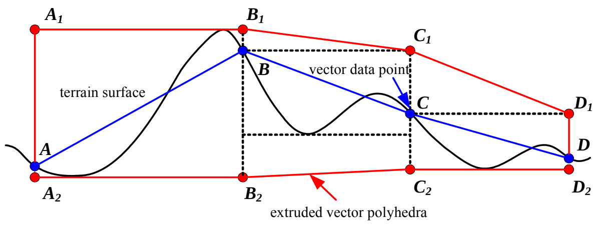

Figure 8. The 2d sketch map of constructing polyhedra of linestrip vector data

In the case of polygons we use the minimum and the maximum height value of the bounding boxes enclosing the projection of the whole polygon. The constructed polyhedra are tesselated ensuring a consistent winding order with all face normals pointing outwards.

3.2 Creating the mask in stencil buffer

Now that we have created the polyhedra from the vector data they can be rendered into the stencil buffer. When the viewpoint is inside the shadow volume or the shadow volume intersects the near clipping plane of the view frustum, the z-fail algorithm is used, otherwise the z-pass algorithm is used. The steps of creating the mask are as follows:

(1) Color, depth and stencil buffer are cleared and the terrain is rendered initializing the depth buffer with the required depth values.
(2) Depth buffer writing is disabled, but the depth test still remains active. Rendering is then restricted to the stencil buffer only.
(3) Rendering vector polyhedra to create the mask. The polyhedron's faces are rendered using different stencil operations depending on whether they face towards or away from the camera. To this end, face culling is enabled and the polyhedron is rendered twice, one time with back-face culling enabled, the other time with front-face culling enabled. If the z-pass method is used, because the polyhedron does not intersect the near clipping plane, the values in the stencil buffer are modified when the depth test passes. The stencil value is incremented for fragments belonging to front-facing polygons and decremented for fragments belonging to back-facing polygons. If the z-fail technique is applied, values in the stencil buffer are modified when the depth test fails. The stencil value is incremented for fragments belonging to back-facing polygons and decremented for fragments belonging to front-facing polygons.

3.3 Rendering the vector

After creating the mask in the stencil buffer we apply it to the scene. Therefore, we reactivate writing to the color buffer and activate additive blending. The stencil test is configured to pass only when the value in the stencil buffer does not equal zero. Instead of drawing a screen-sized quad to apply the mask to the scene, we rasterize the bounding box of the respective polyhedron in order to save rasterization bandwidth. This is performed with depth test enabled and drawing only front-faces in the z-pass case and with depth test disabled and drawing only back-faces in the z-fail case.

The pseudo code of creating the mask and rendering the vector using z-fail algorithm is as follows:

```
//create the mask
glClear(GL_STENCIL_BUFFER_BIT);
glColorMask( GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE );
glEnable( GL_CULL_FACE );
glEnable(GL_DEPTH_TEST);
glDepthMask(GL_FALSE);
glDepthFunc(GL_GEQUAL);
glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS, 0, 0);
//set the stencil buffer operation
glStencilOp(GL_KEEP, GL_KEEP,GL_INCR);
//render the back-faces of the polyhedra
glCullFace( GL_FRONT );
DrawVectorPolyhedra();
//set the stencil buffer operation
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
//render the front-faces of the polyhedra
glCullFace( GL_BACK );
DrawVectorPolyhedra();
//draw the vector data
//render the front-faces of the bounding box of the vector polyhedra
glDepthMask( GL_TRUE );
glColorMask( GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE );
glCullFace( GL_FRONT );
glDepthFunc(GL_GEQUAL);
//set the stencil buffer operation
glStencilFunc(GL_NOTEQUAL,0, 1);
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );
DrawBoundingBoxofVectorPolyhedra();
//resume the default setting
glEnable( GL_CULL_FACE );
glCullFace( GL_BACK );
glDepthFunc(GL_LESS);
glDisable(GL_STENCIL_TEST);
```

4. 3D VECTOR DATA RENDERING EXPERIMENT AND ANALYSIS

The approach in this paper has been successfully used for the rendering of 3d vector data in several areas. We test the algorithm on a Pentium 4 (3.0 GHz, 1GB RAM) computer with windows XP SP2 operation system and NVIDIA Geforce 8500GT graphic card. The size of the rendering window is 1200×1000. The DEM data set consists of 14997×17556 sample points with the ground sample distance of 20m, and the DOM data set consists of 119964×140420 pixels with the resolution of 2.5m. The test habitation vector layer includes 8 areas of 1208 sample points and the road 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 9 shows the rendering result of the habitation (semi-transparent yellow area) and road (red line) data on the 3d terrain landscape.

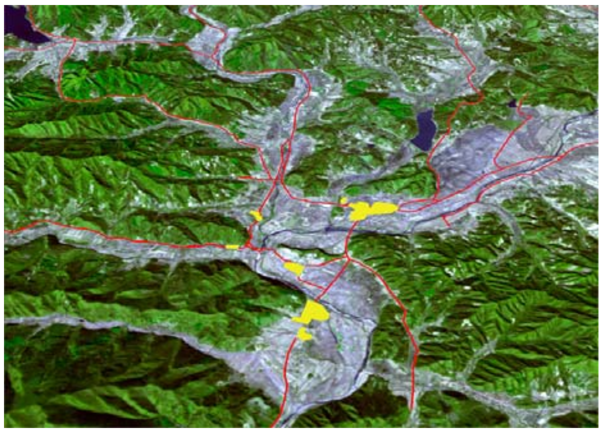

Figure 9. The experiment result of rendering vector data based on the theory of stencil shadow volumes

5. CONCLUSIONS

Experiments show that the presented algorithm allows real-time and high-quality vector data visualization as provided by other geometry-based methods. However, it does not suffer from their shortcomings, namely the expensive adaptation process and the increased primitive count coupled with the terrain complexity. In comparison to texture-based techniques that immediately render the vector data into a texture the method demands slightly more primitives to be rendered but provides superior quality. Interactive editing and manipulation of the vector data is also possible with our method. It only requires updating the shadow volume of the modified vector data object allowing interactive response.

REFERENCES

Crow F., 1977. Shadow algorithms for computer graphics. In: Proceedings of SIGGRAPH 1977, pp. 242–248.

Dai C., 2004. Algorithm for real-time visualization of massive terrain dataset. Journal of computer-aided design & computer graphics, 16(11), pp. 1603-1607.

Everitt C., 2002. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. http://developer.nvidia.com. (accessed 15 Jan. 2008)

Heidmann T., 1991. Real shadows real time. IRIS Universe, 18, pp. 28-31.

Kersting O., 2002. Interactive visualization of vector data in GIS. In: Proceedings of the 10th ACM International Symposium on Advances in GIS.

Schneider M., 2005. Real-time rendering of complex vector data on 3d terrain models. In: Proceedings of the 11th International conference on virtual systems and multimedia, pp. 573–582.

Schneider M., 2007. Efficient and accurate rendering of vector data on virtual landscapes. Journal of WSCG, pp. 59-65.

Wartell Z., 2003. Rendering vector data over global multi-resolution 3d terrain. In: Proceedings on the Symposium on Data Visualization, pp. 213–222.