Computers & Geosciences 75 (2015) 57–65

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

A parallel algorithm for viewshed analysis in three-dimensional Digital Earth

Wang Feng[a], Wang Gang[a,*], Pan Deji[a], Liu Yuan[a], Yang Liuzhong[b], Wang Hongbo[c]

[a] *State Key Laboratory of Remote Sensing Science, The Institute of Remote Sensing and Digital Earth, Chinese Academy of Science, Beijing 100101, China*
[b] *Urban-Rural Planning Management Center, Ministry of Housing and Urban-Rural Development of the People's Republic of China, Beijing 100835, China*
[c] *Managers Training Institute, China National Petroleum Group, Beijing 100096, China*

A R T I C L E I N F O

Article history:
Received 17 April 2014
Received in revised form
16 October 2014
Accepted 29 October 2014
Available online 6 November 2014

Keywords:
Viewshed analysis
Digital Earth
GPU
Visualization
Vertex shader
Pixel shader

A B S T R A C T

Viewshed analysis, often supported by geographic information systems, is widely used in the three-dimensional (3D) Digital Earth system. Many of the analyzes involve the siting of features and real-timedecision-making. Viewshed analysis is usually performed at a large scale, which poses substantial computational challenges, as geographic datasets continue to become increasingly large. Previous research on viewshed analysis has been generally limited to a single data structure (i.e., DEM), which cannot be used to analyze viewsheds in complicated scenes. In this paper, a real-time algorithm for viewshed analysis in Digital Earth is presented using the parallel computing of graphics processing units (GPUs). An occlusion for each geometric entity in the neighbor space of the viewshed point is generated according to line-of-sight. The region within the occlusion is marked by a stencil buffer within the programmable 3D visualization pipeline. The marked region is drawn with red color concurrently. In contrast to traditional algorithms based on line-of-sight, the new algorithm, in which the viewshed calculation is integrated with the rendering module, is more efficient and stable. This proposed method of viewshed generation is closer to the reality of the virtual geographic environment. No DEM interpolation, which is seen as a computational burden, is needed. The algorithm was implemented in a 3D Digital Earth system (GeoBeans3D) with the DirectX application programming interface (API) and has been widely used in a range of applications.

1. Introduction

Digital Earth, as a multi-resolution, three-dimensional (3D) representation of the planet, allows users to locate, visualize, and interpret vast amounts of geo-referenced information. In addition to its primary functions of displaying the world's terrain and remote sensing imagery, one of the most valuable functions of Digital Earth is spatial analysis, which is the primary function of 3D Geographic Information Systems (GIS) (Shi and Liu, 2005). Viewshed analysis is one type of these spatial analyzes. This process involves predicting the total area that is visible from a single point or multiple points (Zhou and Liu, 2006). Viewshed analysis has been used in a wide range of applications, including locating telecommunication relay towers (De Floriani et al., 1994), locating wind turbines (Kidner et al., 1999), protecting endangered species (Camp et al., 1997), and searching for archeological locations (Lake et al., 1998).

A viewshed, in the virtual terrain environment, is a collection of points that are visible from a specific point. Viewshed calculations are potentially time consuming, mainly because extensive interpolation is necessary when using a gridded digital elevation model (DEM) due to the complex terrain model and the complicated geometric features. In addition to the time complexity, multi-point viewshed analyzes and the integration of viewshed calculations with Digital Earth to speed up the calculation of dynamic viewshed analyses are also major challenges to researchers. Therefore, much work has been conducted to develop an efficient viewshed algorithm. Section 2 provides a brief overview and discussion of previous work on viewshed analysis. These previous studies are valuable and can help improve the efficiency of viewshed analysis.

Graphics processing units (GPUs) have been recently used in a large number of applications because they can provide substantial computational power at an affordable cost, and their programmability has also improved (Owens et al., 2007). The parallel property of GPUs has been increasingly utilized to improve computational performance. However, traditional viewshed analysis algorithms, which interpret the viewshed as a series of

[*] Corresponding author.
E-mail addresses: luoying_gis@126.com (W. Feng), wg638@126.com (W. Gang).

http://dx.doi.org/10.1016/j.cageo.2014.10.012
0098-3004/© 2014 Elsevier Ltd. All rights reserved.

intervisibility calculations to all vertices of the DEM based on line-of-sight (LOS), are mainly executed on a computer's central processing unit (CPU) without making full use of high-end GPUs. With the development of GPUs, some GPU-based viewshed algorithms have been proposed in recent three years.

The present research focuses on solving the performance issues described above and providing users with an effective, real-time, parallel, and GPU-based viewshed analysis algorithm. This algorithm has been applied in our 3D Digital Earth application (GeoBeans) with good results.

2. An overview of previous work

Viewshed analysis, as an important branch of 3D spatial analysis, has received increasing attention among researchers. Related academic, conference, and research monographs regarding this approach are gradually increasing (Han, 2011). Current research mainly focuses on viewshed analyzes in terrain models whose data structure is a DEM or triangulated irregular network (TIN). Other previous studies have investigated building-blocks analysis and sunlight analysis (Ying, 2005).

The basic algorithm for generating a viewshed from raster elevation data, known as intervisibility, is based on the estimation of the elevation difference of intermediate pixels between the viewpoint and target pixels. A line segment between a viewpoint O and a target point A, which makes up the LOS, is created to determine the visibility of target point A. Moving along the line segment OA and testing all of the points along this line, target point A is visible only if all of the points on OA have an elevation higher than the elevation of the corresponding point on the terrain. Otherwise, target point A is invisible from the viewpoint (see Fig. 1). The LOS computation is repeated for all target points within the viewshed range of the viewpoint during viewshed analysis. This process is extremely time consuming, and its time complexity is expressed as O(n^2).

The brute-force algorithm described above is simple but computationally intense. A variety of algorithms have been developed to speed up these calculations. De Floriani et al. (1994) proposed an algorithm, named the key slope method that is a huge improvement over LOS. This method continually computes the slope along the sightline and updates the maximum slope. The slope of the current point is compared with the max slope to determine the visibility. Unnecessary computations are greatly reduced, resulting in a time complexity of O(n) using this method. A new, double increment method is presented by Yin shen to speed up the calculation. The accuracy, indeterminacy, and invariance of viewsheds are also discussed (Ying, 2005). Liu et al. (2010) proposed an improved algorithm by using the slope and elevation between the target pixel and viewshed point to reduce the required computation.

Yanlan (2001) introduced a new algorithm to determine viewshed without using sightline, named the reference plane

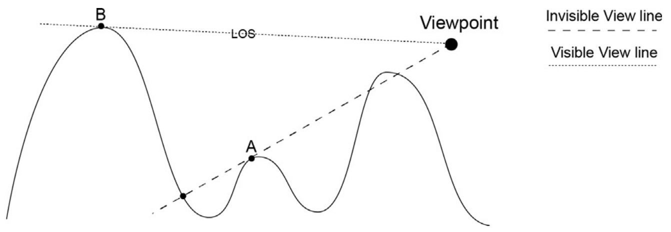


Fig. 1. Overview of the traditional LOS algorithm showing a comparison of the height of the target point with other points along the sight line to determine the visibility. This process is computationally intense. Point B is visible from the viewpoint while point A is invisible.

method, and this algorithm is considered more effective than the LOS algorithm because no DEM interpolation is needed. A reference plane, based on the spatial topological relation of the viewpoint and the target point, is generated to calculate the visibility. Unlike the LOS algorithm, this method generates viewshed without redundant computation. However, this technique is limited to DEMs and not suited to calculating a viewshed from varying resolutions.

Generally, GPUs are designed to exploit data parallelism. It has been reported that GPUs can achieve 10 times more floating-point operations per second (FLOPS) than CPUs (Govindaraju et al., 2006). With the rapid development of modern GPUs, transferring traditional algorithms that were previously executed on CPUs to GPUs is becoming increasingly popular. This technique enables GPUs to process repeated computing tasks to speed up the computation. Chen et al. (2010) implemented a rapid contour-line-extraction algorithm by using traditional methods on GPUs. Parallel processing techniques have also been applied to improve computational performance in viewshed analyzes (Mills et al., 1992, Gao et al., 2011, Zhao et al., 2013). All these methods proposed strategies for implementing traditional LOS-based interpolation viewshed algorithm with NVIDIA CUDA. Although made a progress in efficiency, those methods are limited to regular square grids (RSGs) and not suitable for triangulated irregular networks (TINs), neither for complicated scene with geometry features. Fang et al. (2011) introd Fang uced a real-time parallel algorithm for viewshed analysis known as shadow map-based algorithm. This algorithm is executed on GPUs and uses a depth buffer to store the pixel's minimum depth. Comparing the depth of the current pixel with the depth of a corresponding pixel recorded on the depth buffer, one pixel is visible only if its depth is lower than the minimum depth. This method has an advantage of avoiding most of the computation on a CPU and without consideration of the data-structure and DEM resolution. Nevertheless, one of the disadvantages of this method is its low accuracy. In Fang's method, the size and depth of the shadow map determine the quality of the final results, and low-accuracy areas are usually visible as aliasing or shadow continuity glitches.

3. GPU-based parallel algorithm for viewshed analysis

3.1. Principle

Our parallel algorithm takes a new approach to simulating viewshed analysis by creating occlusive volumes to shield the geometric features in the neighborhood of the viewpoint. In contrast to the proposed GPU-based algorithm (Yanli Zhao et al., 2013), this method avoid the interpolation operation which is time consuming. The surface of the geometry features is used to display the analysis result, called the receiver. Occlusive volumes, known as caster, are generated according to the position of the viewpoint and geometric feature outlines by casting the feature's outline along the sightline to infinity. Although this process still utilizes a sightline to generate occlusive volumes, it differs from the LOS method because DEM interpolation is not required and calculation redundancy is extremely reduced. All of the geometric features, including terrain, models, and trees, in the specific space of the required viewpoint can be used for both the caster and receiver. Therefore, our proposed algorithm performs well in complicated three-dimensional scenes, whereas traditional methods do not. Users only have to add the updated feature as a new caster when the scene is updated, and no changes to the code logic are required. With this prerequisite, our algorithm can conveniently and efficiently simulate viewshed calculations by transforming this process to identify and label the pixels that are within the

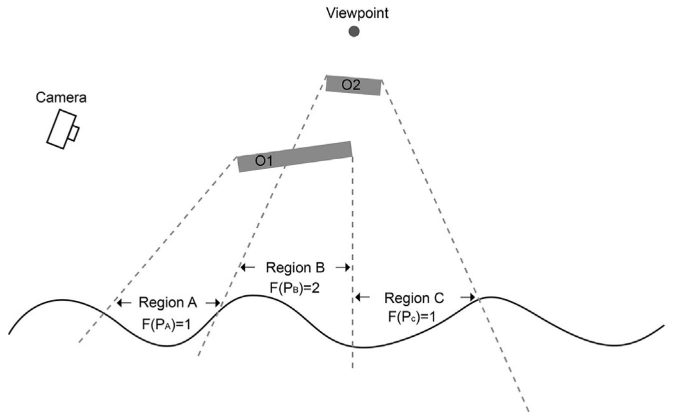


Fig. 2. Mathematic illustration of the viewshed analysis. A stencil buffer is used to label the areas that are in and out of the viewshed. Regions A, B and C are in the viewshed because their stencil buffer values are larger than 0.

occlusive volume. Changing the colors of the labeled and unlabeled pixels enables the viewshed to be displayed immediately and clearly.

Suppose that there is a collection of geometry features in the neighborhood of the viewpoint named $O=\{O_1, O_2 \ldots On\}$. In the stage of vertex shader, we can calculate the occlusive volumes of those features according to the viewpoint and name them $V=\{V_1, V_2 \ldots Vn\}$. Every point can then be rendered on the receiver, and its stencil value can be changed to a label. The function $F(P)$ is used to calculate the stencil value, and the initial stencil value of P is 0. After calculating the stencil value, we can distinguish the visible point and invisible point with function $G(P)$.

$$F(P)=\begin{cases} F(P)+1, & P\in V \\ F(P), & \text{Otherwise} \end{cases} \quad G(P)=\begin{cases} \text{visible}, & F(P)=0 \\ \text{invisible}, & F(P)>0 \end{cases}$$

Fig. 2 illustrates an example of this process. There are two geometry features, O_1 and O_2, and we can calculate two occlusive volumes, V_1 and V_2. All of the points in region A are in $V_1 (P_A \in V_1)$, so $F(P_A)$ is 1. In region B, P_B is in V_1 and V_2 ($P_B \in V_1$ and $P_B \in V_2$) and $F(P_B)$ is 2. In region C, P_C is in volume 2 and $F(P_C)$ is 1. With these results, we can conclude that the points in regions A, B and C are invisible. Other points in the terrain are visible.

In accordance with the perception of viewsheds in the real world, our algorithm can determine the viewshed based on vector data and avoid reducing accuracy by rasterizing the depth, in contrast to the shadow map-based viewshed algorithm (Fang et al., 2011).

By taking advantage of GPU programming, our algorithm performs the sub process of two-point intervisibility in the specific field in parallel, and this optimization process will reduce the time consumption significantly. The proposed algorithm integrates the calculation with a rendering model that can calculate and label the pixel in parallel. The GPU programming is implemented using a vertex and pixel shader, which is a set of software instructions that works on a graphics card to calculate the rendering effect with a high degree of flexibility. The process is known as single instruction and multiple data (SIMD), or data parallelism. Data parallelism emphasizes the distributed (parallelized) nature of the data and not that of the processing (task parallelism). In this paper, we accelerate the analysis process by calculating the stencil value of the point in parallel and simplify the process of two-point intervisibility with the determination of points are in and out of occlusive volumes. Each point on the receiver acts as a data segmentation and be sent to the stream processors to calculate F(P) which is the specified data-parallel functions called kernels. The kernels are invoked to run on GPU devices across a large

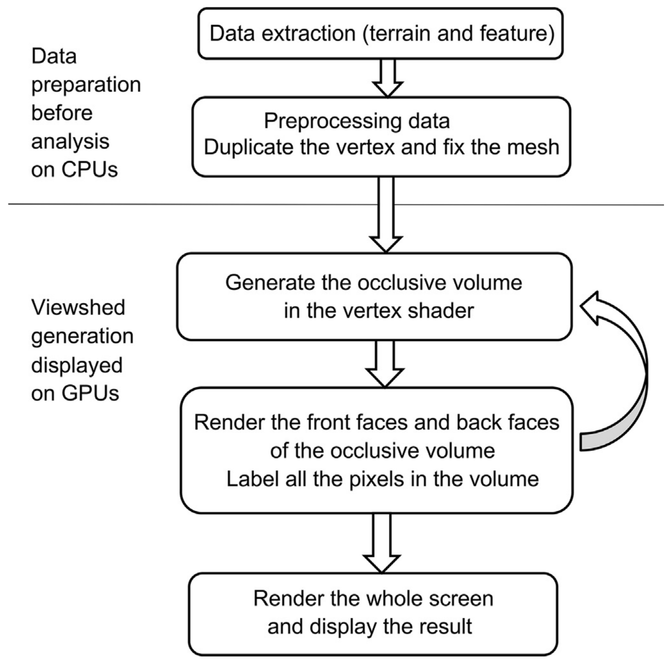

Data preparation before analysis on CPUs
Data extraction (terrain and feature)
Preprocessing data
Duplicate the vertex and fix the mesh
Viewshed generation displayed on GPUs
Generate the occlusive volume
in the vertex shader
Render the front faces and back faces
of the occlusive volume
Label all the pixels in the volume
Render the whole screen
and display the result

Fig. 3. Concrete process of our algorithm showing the steps executed on GPUs performed in parallel.

amount of parallel threads, each executing an instance of the kernels. In this paper, a simplified function utilizing occlusive volumes acts as kernels rather than the time-consuming two-point intervisibility used in Yanli Zhao's method (2013). With stencil buffer and stencil detection, a point can be easily identified if it is within or out of an occlusive volume. More details and concrete steps are demonstrated in Section 3.2.3.

Our algorithm can be efficiently and easily implemented with vertex and pixel shaders. Below is a brief introduction to our algorithm, and the entire process is shown in Fig. 3. Section 3.2 presents a detailed introduction of the concrete steps.

There are three steps involved in the algorithm.

step 1. Extract and pre-process the data; obtain all of the geometric features in the scene from Digital Earth after the viewpoint and view range are specified. This process is independent of the data structure, and only the mesh data of the geometry features are required. DEM data analysis is the traditional algorithm and more general and useful. For DEM data, a triangle mesh can be easily generated and used in the visualization and viewshed analysis of the original DEM data in the pre-processing stage. In this paper, terrain data that are extracted from the Geobeans3D platform with different LOD can act as DEM data. Essentially, the extracted data should be pre-processed to ensure that all of the features are totally enclosed.

step 2. Generate the occlusive volumes; an occlusive volume consists of three parts: a front cap, a back cap, and the side (see Fig. 4). The occlusive volume is generated at the vertex shader stage by translating the faces facing away from the light a large distance along the direction of the light. By transplanting this process onto GPUs, the occlusive volume can be generated in parallel and more efficiently.

step 3. Label and generate the viewshed; in this step, the stencil buffer is used as a mask for rendering additional geometry and to label and distinguish the pixels that are within and outside of viewshed. By rendering all of the occlusive volumes with a vertex extruding shader and setting up the

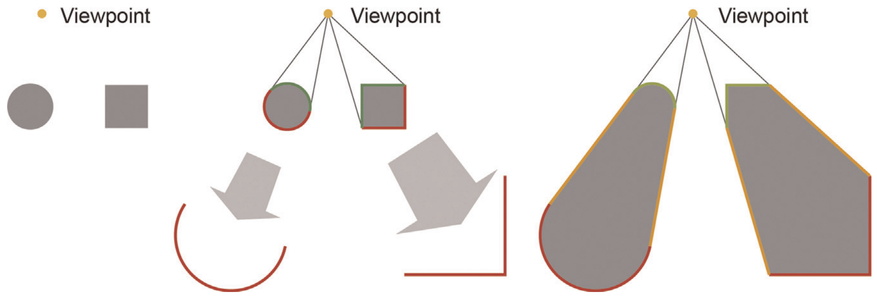


Fig. 4. Introduction of occlusive volume generation extends the back cap of mesh by a large distance along the line of sight. The red line indicates the back cap and the green line indicates the front cap. The occlusive volume must be enclosed. The orange line indicates the broadside. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

stencil buffer according to whether the pixels are within the occlusive volume, the viewshed is finally, successfully labeled. Eventually, the viewshed can be displayed in the complicated scene by flushing the screen and changing the color according to the value of the stencil buffer.

3.2. *Algorithm implementation*

3.2.1. *Data extraction and preprocessing*

Three-dimensional Digital Earth is a complex system for simulating real-world geospatial data using terrain, images, and all types of geometric features such as buildings and forest models. The methods of terrain construction can be divided into regular-grid, TIN, and mixed grid-TIN. The regular-grid, which is easier for spatial analyzes and calculations and has a small storage capacity, is more suitable for flat areas. However, precision is poor when the regular-grid is used to stimulate mountainous areas. In contrast, TIN expresses the surface morphology more precisely, especially in complicated areas. Although it easily expresses topological relationships, TIN has the disadvantages of large storage capacity and complicated computation. There are various methods for viewshed calculation according to different data structures, which differ in their output and cost.

The 3D Digital Earth platform GeoBeans3D contains global and multi-resolution terrain data. Level of detail (LOD) is used to render terrain in real time to improve efficiency. To process the viewshed analysis in complicated scenes, the first step is to collect geometric features and terrain in the neighborhood of the viewpoint. In our algorithm, the terrain data, whose data structures correspond to the rendering model, are extracted in three different resolutions. This process allows the original mesh data of complicated features, such as building models, to be used in our algorithm directly. To reduce computation, we can simplify certain tree models using the bounding box for a given tree model.

3.2.2. *Generation of occlusive volume*

The traditional approaches to generating occlusive volume are usually two-step processes that determine the silhouette edges of the geometric features according to the viewpoint and extend the silhouette for a long distance along the direction from which the light originates. These methods are typically conducted using CPUs and have proven inefficient and time consuming. Therefore, instead of determining the silhouette and generating the occlusive volume geometry on the CPU, our algorithm first generates a mesh that represents the occlusive volume regardless of light direction and uses a vertex shader to preform vertex extrusion.

The underlying concept is that triangles that face the light can be used as-is for the front cap of the occlusive volume. The vertices of triangles that face away from the light are translated a large distance, usually to the far plane of the frustum, along the light direction at each vertex; they can then be used as the back cap. However, a problem occurs at silhouette edges where one triangle faces the light and its neighbor faces away from the light. In this situation, the geometry mesh will split when performing vertex extrusion.

An effective way of solving this issue is duplicating the shared vertices for the two triangles that share an edge, so that each triangle has its own unique three vertices, and attaching a degenerated quad to each shared edge. When the common edge between the triangles becomes the silhouette edge, one triangle stays where it is and the other moves along the light direction. This process, however, creates a gap between the two triangles, whereas a closed occlusive volume cannot have any gap or hole. This problem can be fixed by adding a degenerated quad to the occlusive volume between the original two triangles, thereby creating two new triangles. Fig. 5 illustrates this process.

There are concrete steps:

- Duplicate the vertices of the original mesh and make sure every triangle has its own unique three vertices and edges for vertex extrusion.
- The normals of the new vertices are computed to be the normal of the new face. This step is necessary because the normal of the vertex is used to determine whether the vertices should be extruded. We refer to the vector of the current vertex to viewpoint as L. If the dot value of vertex normal and L is less than 0, the vertex must be extruded.
- The three edges of a triangle are added to an edge-mapping table. An edge-mapping entry contains the vertex information

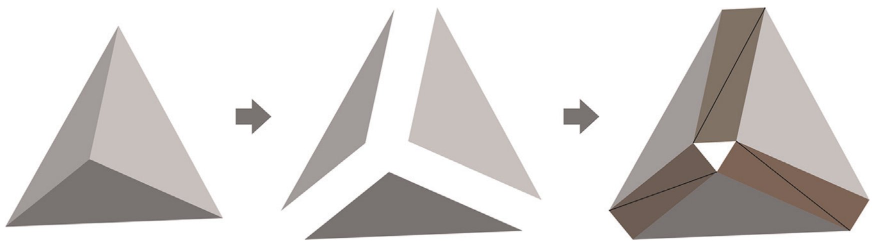


Fig. 5. Detailed illustration of mesh pre-processing, duplicating the mesh vertex, and filling in the gap by regenerating new triangles with the original four vertices.

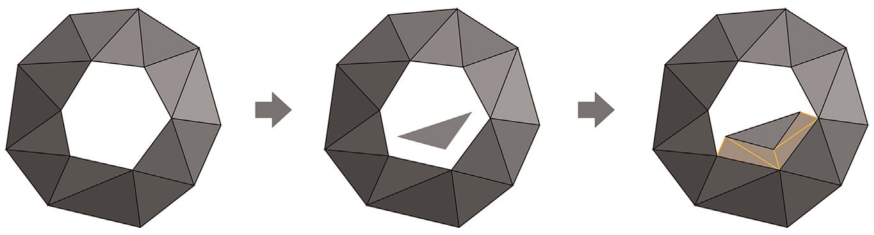


Fig. 6. Initial steps in fixing the hole of the original mesh.

of an edge. For each edge of the added face, the algorithm looks through the edge mapping table to find the entry of the source edge. If the target is hit, four vertices are elicited to generate two triangles by connecting the vertices with a specific sequence (clockwise in DirectX 9.0). The new triangles will be added to the original mesh as new elements. Finally, the edge-mapping entry should be removed from the table.

- After step 3, if there are edges in the mapping table, which means the edges are not shared in the original mesh, we must patch the volume because it should be closed. The existence of these edges implies the original mesh has holes in it. The patching algorithm examines the mapping table and identifies two edges that share a vertex in the original mesh. Then, the algorithm patches the hole by generating three new vertices and a new triangle using the two neighboring edges' vertices. To connect the patched face with the mesh, two degenerated quads should be generated. This process is illustrated in Fig. 6.

3.2.3. *Labeling and generation of the viewshed*

The stencil buffer can store an unsigned integral value for each pixel on the screen, which is frequently used as a mask to render more dynamic effects, such as shadow and swipe. In the process of rendering, a specific reference value can be compared with this stencil value. The result of the comparison can indicate whether to update the color of the corresponding pixel. This process is called stencil testing. The stencil value resets according to whether the pixels are in the occlusive volume in the first render pass. Then, the stencil test is conducted in the second render pass. If the test passes, the value of the corresponding pixel will be updated dynamically; otherwise, the value will not be updated.

After being created, all of the occlusive volumes are rendered in the rendering pipeline, and the pixels that are within any occlusive volume are labeled in the stencil buffer. Without any extra computation, this process can be performed within the programmable, 3D visualization pipeline by setting and updating the corresponding render state. The implementation of this process is performed in the pixel shader. The following is a detailed introduction to this process:

- Clear the stencil buffer and set the value as default. Disable depth-buffer and frame-buffer writing and enable depth testing and stencil testing. Prepare the stencil buffer render states for rendering the occlusive volume.
- Look through all the occlusive volumes. First, use front-face culling and set the stencil operation to increment on depth fail; then, render the back face of the occlusive volume.
- Use back-face culling and set the stencil operation to decrement on depth fail; then, render the occlusive volumes.
- Recover the render states and render the whole screen. Overlay the region whose stencil value equals 0 with red, otherwise with green.

Fig. 7 illustrates an instance of labeling with a stencil buffer. After all the occlusive volumes are rendered, the stencil buffer of

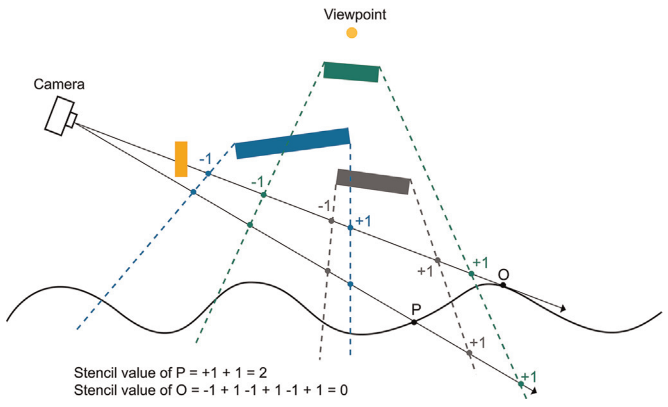

Viewpoint
Camera
-1
-1
-1
+1
+1
+1
O
P
+1
+1
Stencil value of P = +1 + 1 = 2
Stencil value of O = -1 + 1 -1 + 1 -1 + 1 = 0

Fig. 7. The process of labeling the pixel using the stencil buffer. After the process, pixel O is visible because its stencil value is equal to 0, whereas *P* is invisible because its value is not equal to 0. This process is executed on GPUs in parallel.

the neighboring space of the viewpoint is reassigned. The area in which the stencil value is not equal to 0 is the region that is shielded by the occlusive volume; this is the area that is invisible from the viewpoint.

4. Results and discussion

This algorithm, implemented with DirectX 9.0c and C#, was applied in a 3D Digital Earth application (GeoBeans3D). GeoBeans3D is constructed from a multi-stage global DEM using massive satellite data. This interactive program is the largest 3D GIS on the network in China and has been used in many important government departments. The DEM data used in GeoBeans3D have a spatial resolution of 30 m globally and a local resolution of 6 m. The resolution of the image data is 2.5 m nationwide and 0.6 m in some specific areas. The scale of the fundamental geographic data is 1:1,000,000 globally, 1:250,000 nationwide, and 1:2000 in some specific areas. As a dynamic, real-time process, the generation of viewsheds is integrated with the visualization of geometric features in GeoBeans3D. Fig. 8 provides screenshots of the application. The viewshed is efficiently generated after the viewpoint is identified. Note that the area marked with green color can be seen from the viewpoint, and the area marked with red color cannot.

The shadow map-based algorithm and our algorithm have been tested on three personal computers with different configurations (Table 1). The capability of the CPU of the testing computers does not make a substantial difference, but the graphics card varies markedly. With regard to GPU performance, the graphic card in No. 1 is best and that in No. 3 is the worst. In the experiment, we use the interactive frame-rate of the application as the evaluation parameter. The rate is measured with Fraps3.1.0. Tested at the same region with the same viewpoint and range, the two algorithms both have the best performance using No. 1 and the worst performance using No. 3 (Table 2), which means that the capability of the GPU is an important issue influencing efficiency, as

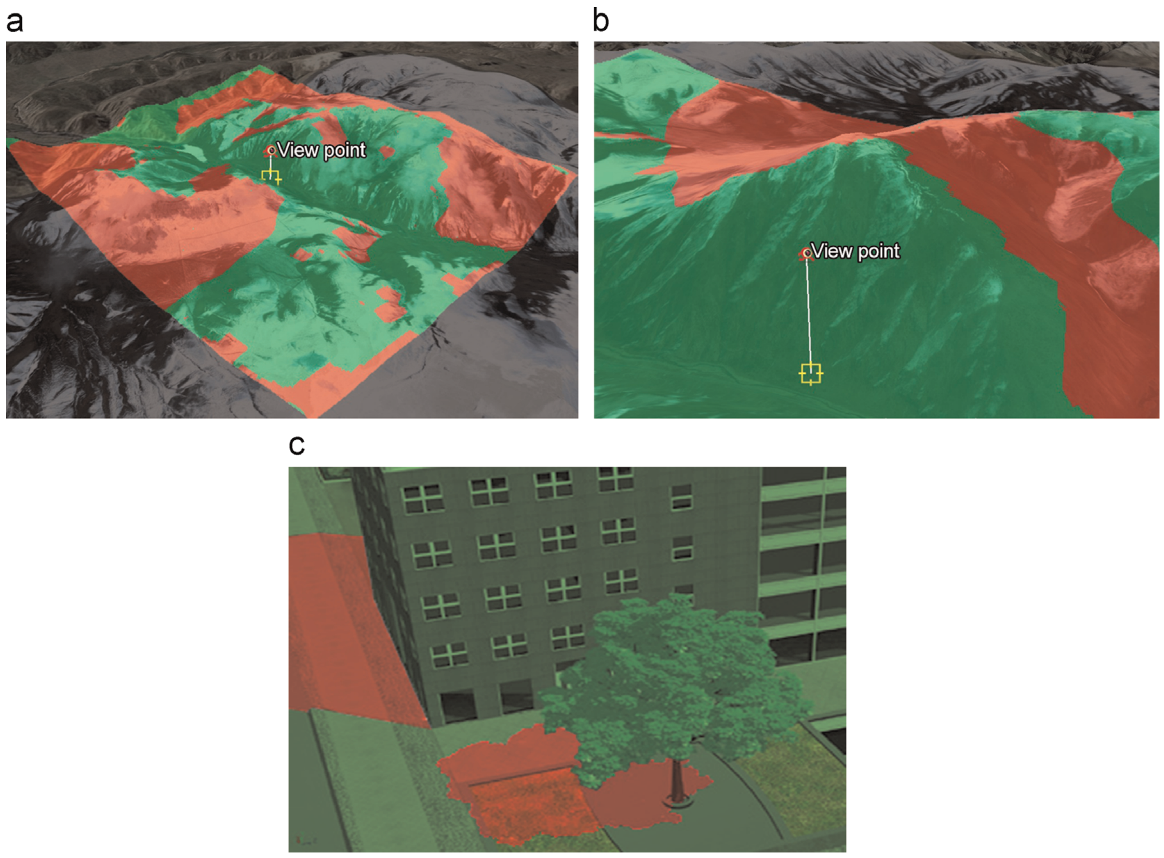


Fig. 8. Screenshots of the viewshed analysis in GeoBeans3D. The red area is invisible, and the green area is visible from the viewpoint. (a and b) The viewpoint is identified on the plateau with rich topography; (c) the viewshed with complicated features from a viewpoint over a flat plain. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Table 1
The detailed configuration of our testing platform.

	No. 1	No. 2	No. 3
CPU	Intel Core2 Q6600	Intel Core2 Quad	Intel Core2 Duo
CPU frequency (GHz)	2.4	2.5	2.4
RAM (GB)	3.25	3.25	2
GPU	NVIDIA GTX280	NVIDIA GF9800	NVIDIA GF8600
GPU memory	1 GB	1 GB	256 MB

Table 2
Testing two viewshed algorithms on our platforms.

PC	No. 1	No. 2	No. 3
Shadow map-based algorithm (ms)	8.4	16	36.3
Our algorithm (ms)	5.2	12	20.5

Table 3
Testing five viewshed algorithms in GeoBeans3D in the same platform (No. 1).

Algorithms	A1 (ms)	A2 (ms)	A3 (ms)	A4 (ms)	A5 (ms)
LOD 1	0.97	0.64	0.74	0.69	0.42
LOD 2	1.84	1.41	0.89	0.76	0.48
LOD 3	4.56	3.24	1.06	0.83	0.55

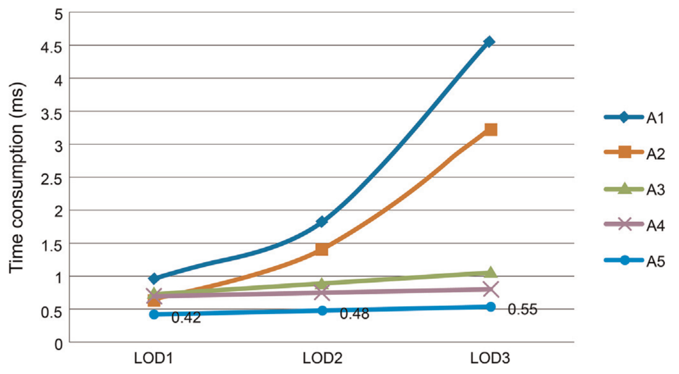
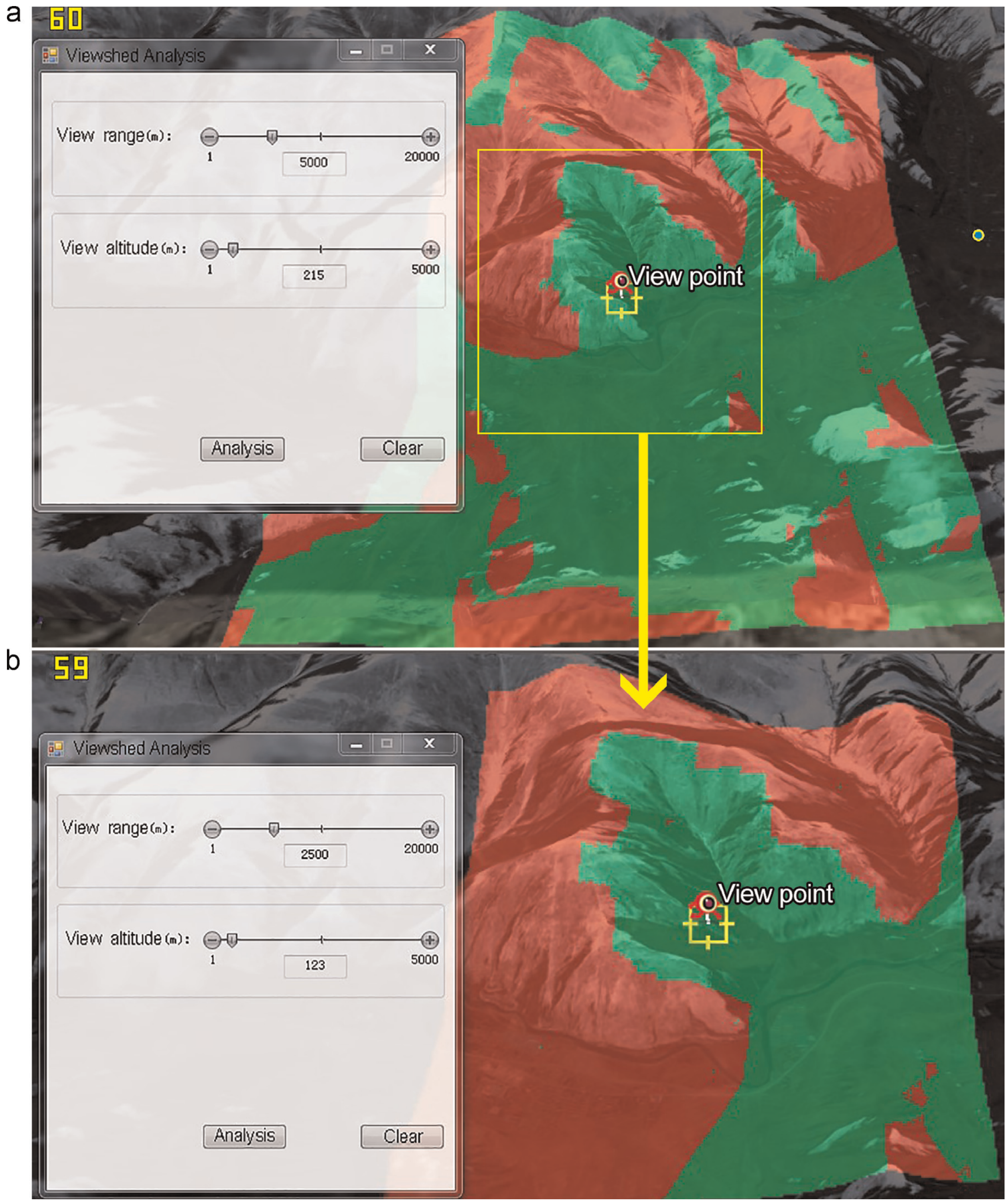


Fig. 9. Average time consumption of the viewshed analysis using five algorithms. A3, A4 and A5 are executed on GPUs and their time consumption grows more smoothly with the increase of terrain data. A5 showed the best performance in time complexity.

they are both mainly composed of processes involving vertex and pixel shaders on GPUs. Therefore, we believe that a high-performance graphic card will accelerate the viewshed analysis algorithms (including our algorithm) when performed on GPUs. Comparing the performance of the two algorithms run on the same PC, we observed that our algorithm can improve the performance by an average of 35.3%.

Additionally, we conducted other experiments to further analyze the performance of viewshed analysis algorithms. Five different viewshed analysis algorithms, including an LOS algorithm referred to as the double increment algorithm, a reference plane algorithm, a GPU-based parallel algorithm (Yanli Zhao et al., 2013), the shadow map-based algorithm, and our algorithm, were tested on computer No. 1 with the same configuration. A1 through A5 represent the five algorithms accordingly. The sample data in our experiment were extracted from GeoBeans3D at three levels of LOD. The terrain resolution is 6 m in LOD1, 2.5 m in LOD2, and 0.6 m in LOD3. For each level, twenty groups of viewpoints and



View range(m): 5000
View altitude(m): 215

View range(m): 2500
View altitude(m): 123

Fig. 10. Frame-rate with different viewpoint altitudes and ranges: (a) viewpoint altitude = 215 m, range = 5000 m, frame-rate = 60 fps; and (b) viewpoint altitude = 123 m, range = 2500 m, frame-rate = 59 fps.

view ranges are randomly chosen as the experimental data, which are tested using the five algorithms. After the computation time is calculated, we calculate the average (Table 3). As shown in Table 3 and Fig. 9, the viewshed analysis algorithms executed on GPUs have a better performance than those processed on CPUs. The time consumption in the five algorithms increases with the terrain resolution. However, algorithms executed on GPUs have much less slope in Fig. 9. Our viewshed analysis algorithm performed best among the five algorithms tested.

Generally, the time consumption of viewshed analysis increases with the view range because the larger range incorporates more terrain data and geometry features. However, this scenario has little effect on the shadow map-based algorithm and our algorithm. In these methods, the process of viewshed generation is integrated into the render pipeline and DEM interpolation is not necessary. Thus, the algorithms are able to maintain a consistent frame-rate when the viewpoint and view range are changed and additional geometric features are added to the scene. By contrast, applications using the traditional algorithms usually have a precision loss in frame-rate because most of the processor cores are used to interpolate the DEM terrain rather than render the whole scene. However, in our algorithm, rendering the scene is the main operation, and the viewshed computation has been combined with this operation; thus, changing the input parameters does not affect the efficiency of viewshed generation. The only process required in our algorithm is to change the position value of the

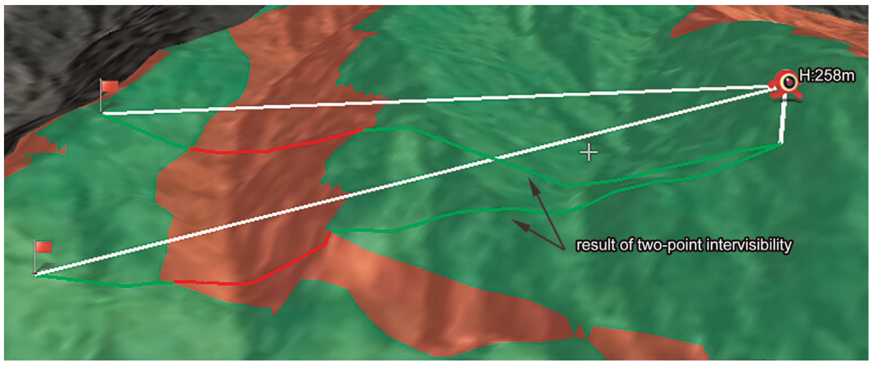


Fig. 11. Comparison of two-point intervisibility and our proposed method. The two lines under the white line are generated using the method of two-point intervisibility. The green segment of the line is visible, and the red segment is invisible. The results of two-point intervisibility and our method show a consistent match. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

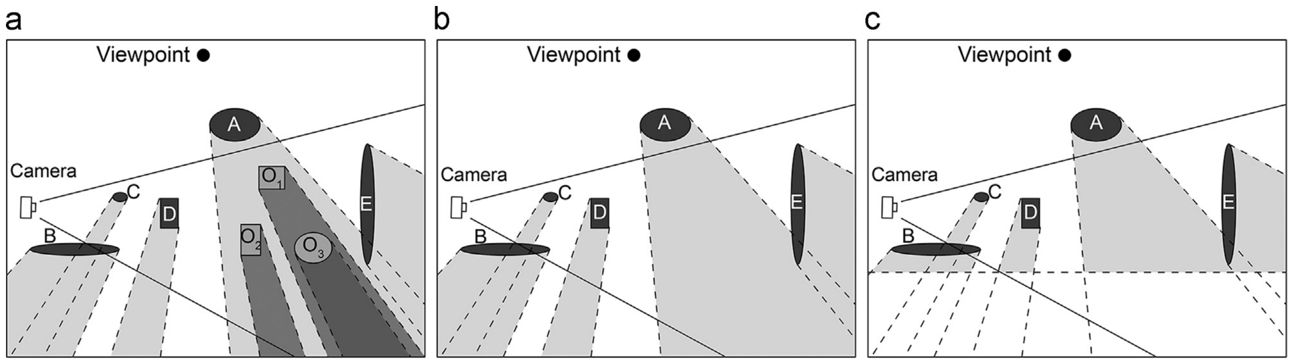


Fig. 12. Illustration of future improvements to our method: (a) indicates the original geometry features and their occlusive volumes; (b) using a culling algorithm to exclude feature O_1, O_2, O_3 whose occlusive volume is totally included by others; and (c) avoid unnecessary rendering by clamping the limits of the occlusive volume.

viewpoint transmitted into the vertex shader. In Fig. 10, we can observe that the frame-rate of the application is consistent and stable. The frame-rate remains at a value of approximately 60 when we halve the view range and change the altitude of the viewpoint.

The performance of an algorithm can be measured based on two aspects: efficiency and accuracy. The experimental results above indicate that the proposed algorithm can greatly improve the efficiency, and additional experiments should be performed to demonstrate the accuracy of our algorithm. In Section 2, we indicated that a flaw of the shadow map-based algorithm is its low accuracy. The depth buffer used to store the minimum depth has a maximum resolution of 4096×4096. The shadow map-based algorithm rasterizes the point before identifying its intervisibility. Consequently, the results are visible as aliasing, and their qualification worsens with increasing view range. Our proposed algorithm avoids this flaw by creating an occlusive volume as vector data to identify the intervisibility. The accuracy of our algorithm is only limited to the original mesh data or DEM as a traditional DEM algorithm, and it will not show a reduction in accuracy at any process stage in our algorithm. For the Geobeans3D platform, two-point intervisibility is utilized, and DEM interpolation is performed to verify the accuracy of our experimental results (Fig. 11).

Random points A and B with the same view point in our algorithm are chosen for two-point intervisibility. The height of the view point is 258 m. By overlapping the results, the visible and invisible areas of the two experiments are consistently matched in every perspective. That is to say all the points along the result line of two-point intervisibility have the same visibility with the result calculated with this proposed method. The points marked with flag in Fig. 11 are the specialized case. Their visibility using our method keeps the same with result using two-point intervisibility. Thus, the proposed algorithm is shown to have the same accuracy as the traditional DEM interpolation algorithm, which is only influenced by the terrain resolution.

Although the experiment proves that our algorithm can greatly improve the efficiency and accuracy in the generation of viewsheds, work must still be done to improve its efficiency. Microsoft DirectX 11 has increased primitive tessellation in the programmable rendering pipeline, which enables users to generate new primitives and vertices according to their needs. Therefore, we can transplant the duplication of vertices in the original mesh from the CPU to GPU and process it in parallel with the help of the geometry shader to further improve the efficiency.

Applicable for viewshed analyzes in complicated scenes, our algorithm is a batch-generating process that calculates the viewshed for all features in the neighboring space of the viewpoint. It is easy to handle cases in which the geometric features must be updated by inserting them into and removing them from the feature list. However, as geometric features in the complicated scene increase, it is extremely challenging to simulate viewshed analysis, which is reflected in two aspects. First, too many features must be rendered and rasterized. The overlapping areas will produce redundant computations when there are multiple occlusive volume objects. Second, because occlusive volumes extend away from casters toward the far plane of the view frustum, they sometimes cover much of the screen. Therefore, the rasterization of occlusive volumes is very expensive. This condition is illustrated in Fig. 12. However, in the future, we can use a volume-culling algorithm to eliminate the casters to solve the first issue. To address the second issue, we can clamp the extents of the occlusive volume to avoid unnecessary rendering in the large regions of empty spaces.

5. Conclusions

Large quantities of computing resources are required to seamlessly render global multi-resolution terrain in virtual 3D environments and geometric features on the ground. Therefore, optimizing the efficiency of rendering programs is an important task for application developers. The introduction of a GPU to the terrain analysis in Digital Earth can effectively accelerate the traditional method through parallelization.

This paper introduces a rapid and efficient algorithm based on a GPU to achieve rapid viewshed analysis of scenes with structures and vegetation. The algorithm has been used in GeoBeans3D and has potential applications in many projects. Tests have proven that our algorithm can effectively overcome the bottlenecks that traditional algorithms face and achieve viewshed rendering from the perspective of real-life geographic phenomena, which is practically important. The algorithm presented here improves the accuracy of viewshed rendering while also improving its efficiency. Moreover, with its extensive potential expandability, this algorithm applies to multi-source and multi-resolution heterogeneous data and exhibits great practical value for 3D GIS applications.

Acknowledgments

The authors gratefully acknowledge every member in the 3D Group (GeoBeans Co., Ltd.) for providing useful help. The authors are also grateful for the helpful comments provided by Dr. Chen Yun from the Australia Commonwealth Science Industries Research Organization. This work was supported in part by 863 Program Grants 2012AA12A401 and 2013AA12A403.

References

Mills, K., et al., 1992. Implementing an intervisibility analysis model on a parallel computing system. Comput. Geosci. 18 (8), 1047–1054.
De Floriani, L., et al., 1994. Line-of-sight communication on terrain models. Int. J. Geogr. Inf. Syst. 8 (4), 329–342.
Camp, R.J., et al., 1997. Viewsheds: a complementary management approach to buffer zones. Wildl. Soc. Bull. 25 (3), 612–615.
Lake, I.R., et al., 1998. Modelling environmental influences on property prices in an urban environment. Comput. Environ. Urban Syst. 22 (2), 121–136.
Kidner, D., et al., 1999. GIS and Wind Farm Planning. Geographical Information and Planning. Springer, Advances in Spatial Science, pp. 203–223. http://link.springer.com/chapter/10.1007%2F978-3-662-03954-0_11.
Chen, Z., et al., 2010. Parallel algorithm for real-time contouring from grid DEM on modern GPUs. Sci. China Technol. Sci. 53, 33–37.
Fang, C., et al., 2011. Parallel algorithm for viewshed analysis on a modern GPU. Int. J. Digit. Earth 4 (6), 471–486. http://dx.doi.org/10.1080/17538947.2011.555565.
Gao, Y., et al. 2011. Optimization for viewshed analysis on GPU. Geoinformatics, 2011, In: Proceedings of the IEEE 19th International Conference.
Govindaraju, N., et al. 2006. GPUTeraSort: high performance graphics co-processor sorting for large database management. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 325–336.
Han, J., 2011. Research on space visibility analysis in 3D simulation [D], He nan: Information Engineering University.
Liu, L., et al., 2010. An improved line-of-sight method for visibility analysis in 3D complex landscapes. Sci. China Inf. Sci. 53 (11), 2185–2194.
Owens, J.D., et al., 2007. A survey of general-purpose computation on graphics hardware. Comput. Graph. Forum 26 (1), 80–113.
Shi, J.-S., Liu, J.-Z., 2005. Development of 3 DGIS technology. Cehui Kexue/Sci. Surv. Mapp. 30 (5), 117–119.
Yanlan, W., 2001. An algorithm computing viewsheds based on reference planes. Wtusm Bull. Sci. Technol. 1, 006.
Ying, S., 2005. Key techniques and applications of spatial visibility analysis [D], Hu bei: Wu han University.
Zhao, Y.L., et al., 2013. A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. Int. J. Geogr. Inf. Sci. 27 (2), 363–384. http://dx.doi.org/10.1080/13658816.2012.692372.
Zhou, Q.M., Liu, X.J., 2006. Digital Terrain Analysis. Science Press, Beijing, BJ, pp. 181–200.