Efficient View-dependent Rendering of Terrains

Yadong Wu Yushu Liu Shouyi Zhan Xiaochun Gao

Department of Computing Science
Beijing Institute of Technology

Abstract

Though considerable progress has been made with the view-dependent techniques in terrain visualization, the CPU overhead still precludes their wide application in many domains. The calculation complexity of view-dependent techniques mainly involves the calculation of node screen space error every frame, including the time-consuming screen space projection, the number of nodes whose projection error remains to be updated, and the evaluation of the valid life of the projection error. In this paper we introduce block-priority-based traversal of quadtree for reducing the traversal complexity and propose view-angle-based error metrics. Thus we successfully speed up the valid life evaluation of projection error by means of calculating the spatial relation between the viewpoint and a simplified split zone. In addition, constant frame rate has been achieved by scaling the split zone accordingly. Corresponding experimental results have shown that our methods can real-time render large scale terrain on a low-cost PC so as to satisfy the demand of most applications in this way.

Key words: Terrain, real-time rendering, view-dependent, error metric, valid life, constant frame rate.

1 Introduction

Terrain visualization plays an important role in many applications, such as GIS, vehicle simulation, battlefield visualization and entertainment. At the same time, real-time terrain rendering proves to be a difficult task for current graphics workstations. In recent years there have appeared some remarkable techniques on terrain visualization, such as visibility computing, LOD (Levels of detail), IBR (Image-based rendering), which have gained such a great progress as to be able to satisfy some applications to some extent. Of the above techniques, the most popular are the view-dependent techniques, which are often used to adjust the resolutions of different regions of the scene according to the view parameters in order to relieve the burden of graphics subsystems. The main disadvantage of these techniques lies in their overhead of CPU and memory. The problem is usually alleviated by using regular grid to lessen the storage, but the problem of inherent computing complexity is still unresolved.

This paper introduces block-priority-based traversal of quadtree and view-angle-based error metrics in terrain rendering. We simplify the split zone of nodes to an axis-aligned rectangular box, thus greatly improve the efficiency of screen space error calculation of nodes.

2 Related Work

Visibility computing [14, 12] is used to accelerate terrain rendering, but only takes effect in low-level flight or ground vehicle simulation. Image cache [13] requires long preprocessing time, and the handling of many unused pixels in the textures defined by the methods results in a performance penalty for the technique. Chen et al [1] propose Lod-sprite, a hybrid of [7] and [13]. Lod-sprite spends different time when rendering the key frames and the other frames, resulting in evident latency of key frames.

View-dependent techniques are introduced in [16, 4, 10]. The methods provide a direct measure to control the image quality and frame rates according to the screen space error of the mesh vertices. In terrain visualization, Triangulated-irregular-network (TIN) [5, 6] will use fewer triangles than regular grid, but it needs a complex data structure. Besides, TIN may produce many thin, slivery triangles. Regular grid is adopted in [7, 2, 11, 8]. Lindstrom et al [7] traverse the quadtree bottom-up. Duchaineau et al [2] use dual-queue to drive split and merge operations. Pajarola [11] manages the scene dynamically while Liu et al [8] accelerate terrain rendering based on focus criterion.

With the main idea aiming to exploit the graphics pipeline by reducing the number of polygons that are to be rendered at each frame, view-dependent techniques impose significant overhead in terms of CPU usage during visualization, as pointed out in [9, 3]. Such an approach actually limits the wide applications of the techniques. To reduce the number of the vertices to be visited, we can traverse the vertex tree based on block-based simplification [7], or in a top-down manner [2]. But the improvement is meager, for they are in nature vertex-based methods. Duchaineau et al [2] propose to defer the projection computation, but do not give a simple method explicitly. Shade et al [13] introduce one way to compute the valid lives of nodes, but it cannot be directly applied to geometry-based rendering. Hierarchical LOD is used in [3] to visualize dynamic CAD

scenes. But it seems unable to work so well in terrain rendering because this approach cannot efficiently reduce the number of polygons sent to graphics subsystem.

Our algorithm is more similar to [2] to some extent. They all adopt dual-queue of split and merge. But this paper has made remarkable improvements in traversal of hierarchical tree, error metrics and computation of projection valid life.

3 Quadtree Structure

3.1 Definition and Basic Operation

Each block is defined as a node of the quadtree, represented by the vertex centered on the block, with each of the nodes having 4 children, 1 parent except for the root node, as shown in Figure 1. Because of the vertex dependencies of regular grid, in every block the node depends on 2 corner vertices, and 4 boundary vertices depend on it, as shown in Figure 2.

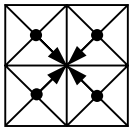


Figure 1: Node hierarchy

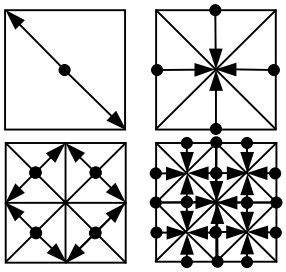


Figure 2: Vertex dependencies

In the mesh rendered at a frame, if no boundary vertices are dependent on the node, the node would be a mergeable one, as the solid dots in Figure 3. The nodes whose parents are at the end of the vertex tree are called splittable, as the hollow dots in Figure 3.

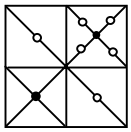


Figure 3: Splittable and mergeable nodes

All the operations in our algorithm are centered on such nodes. When splitting or merging a node, we check the related vertices, inserting or deleting them when needed. The split and merge operations are illustrated in Figure 4.

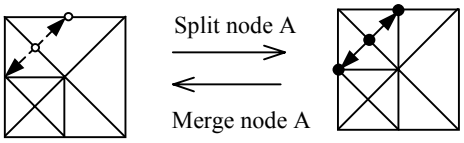


Figure 4: Node operations

To ensure that each vertex can be inserted or deleted, the vertices that depend on a leaf node in the full quadtree are defined as children of the leaf node.

3.2 Traversal of Quadtree

Block-priority-based traversal of quadtree is introduced to exploit spatial coherence. If the screen space error of a splittable node A exceeds the threshold τ specified by the user, we split it. Then we should recursively test the 4 children of the node. In the rendered mesh the brother nodes are actually to be adjacent to each other, and the errors of all nodes are in the same direction, as described in section 4. So we can select the child node B who holds the largest error of its brothers. If its screen space error is smaller than τ, we will stop the test of the rest children of node A. Since each node corresponds to a block, the priority-based traversal is called block-priority-based traversal of quadtree. The key word is the error of the block node.

Our algorithm will not recursively split one node to the bottom of the quadtree in one frame. Instead, the user is just expected to specify a depth limit. The traversal will cease when it reaches the depth limit in a frame. In doing so we can avoid not only searching the quadtree so deeply as to affect the frame rates, but also deferring the update of part of the scene so long as to make visible progressive changes of terrains.

In frustum culling it is only necessary to test the nodes rendered at previous frame, thus relieving the need to test the children of nodes that intersect the view-frustum recursively. What we have to do is just calculate the spatial relations between such nodes and view-frustum when deciding whether to split or merge them.

4 Error Metrics

4.1 Error of Vertex

As illustrated in Figure 5, deleting vertex D will introduce an error of DE. The error is:

$$\varepsilon_{\mathrm{D}} = D_z - \frac{A_z + B_z}{2}.$$

If vertex C is a leaf node in the quadtree, as defined in section 3.1, vertex D is also a child of C.

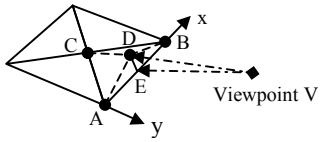


Figure 5: Deleting D will introduce an error of DE

4.2 Error of node

Each node in the mesh is associated with an error δ, $\delta \geq 0$, and

$$\delta = \max(\delta_0, \delta_1, \delta_2, \delta_3, |\varepsilon|, |\varepsilon_0|, |\varepsilon_1|, |\varepsilon_2|, |\varepsilon_3|).$$

In which δ_i is errors of its children, ε is the error of the vertex, and ε_i is errors of boundary vertices depending on it.

4.3 Screen Space Error

Since the vertex D is also a child of node C in Figure 4, then $\delta_D = |\varepsilon_D|$. From the viewpoint of V, deleting D leads to an angular error $\angle$DVE. By computing the pixel projection of $\angle$DVE, it can be found whether the simplification violates the specified pixel threshold τ. Let the field-of-view in y direction be *fovy*, and height of the window be *h*. Let θ be the angular tolerance corresponding to τ. Since θ is very small, approximately we can get

$$\theta = \frac{fovy}{h}\tau .$$

If $\angle DVE = \theta$, then viewpoint V must lie on one of the two circles in the 2D diagram in Figure 6. The radius of the circle is:

$$r = \frac{\delta}{2\sin\theta} \approx \frac{\delta}{2\theta} = \frac{\delta}{2} \cdot \frac{h}{fovy} \cdot \frac{1}{\tau} .$$

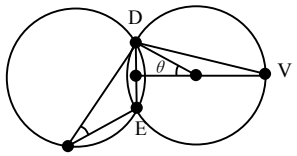


Figure 6: The angle between V and DE is θ

Given threshold τ and the error δ of the node, if viewpoint V lies outside the two circles in the 2D diagram, the screen space error of the node must be smaller than τ. Once the viewpoint enters this zone of one node, the node should split. Hence, this zone is called the split zone of the node.

It seems still troublesome to compute whether the viewpoint is inside or outside the split zone of a node, which can further be simplified to an axis-aligned rectangular box with its length and width of 4*r*, and height of 2*r* respectively. In the remainder of this paper all split zones are referred as axis-aligned boxes except specially defined.

5 Valid Life of Projection Error

To exploit frame-to-frame coherence, Duchaineau et al [2] propose to defer some of the projection error updates for some frames, provided that it will not violate the specified threshold. In this paper, the safe period is called the valid life of projection error of nodes, or valid life of nodes in short.

It proves easy to compute the valid life of projection error using our simplified split zone. Typically vehicles move much slower in vertical direction than in horizontal, so it can safely be said that the upper limit of vertical speed is just half that of the horizontal speed. And the height of the box-shaped split zone is just half the length and width, too. If so, the valid life of projection error can easily be deduced as:

```
ValidLife()
{
        dh = max(|vx – cx|, |vy – cy|);
        dv = |vz – cz|;
        if(dh>2dv)
            return (dh – 2r) / vh;
        else
            return (dv – r) / vv;
}
```

In which the viewpoint is (*vx*, *vy*, *vz*), and the center of the split zone is (*cx*, *cy*, *cz*), v_h, v_v is the upper limit of horizontal and vertical speed respectively.

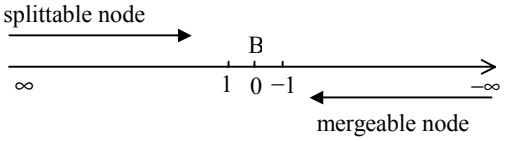


Figure 7: Life axis of node

The valid life of a splittable node can be described by means of the life axis in Figure 7. B is the base, and its value *b* is an integer. At the beginning B lies on origin, and *b*=0, indicating that a splittable node should split when the viewpoint arrives at the boundary of the split zone.

Assuming the viewpoint moves towards the splittable node along the life axis, it can be deduced that the valid life of the splittable node would move from left to right. Reaching B, the valid life of the node will be 0. This implies that if the viewpoint arrives at the bound-

ary of the split zone, the node will split. The condition can be expressed as $Q^{s}_{p=b,b=0}$, in which b=0 meaning that base B locates at the origin, p represents the current location of the viewpoint in the life axis of the node, and p=b meaning that when the viewpoint reaches B, the node would split.

Similar to splittable nodes, when a mergeable node needs to merge, the condition can be shown as $Q^{M}_{p=b+1,b=0}$. It means that when the viewpoint goes outside of the split zone, the node should merge.

6 Constant Frame Rate

Papers [10, 2, 5] describe how to achieve constant frame rate in details. The main consideration is to optimize the image quality within the triangle budget, minimizing the screen space error of the vertices. Since different frames have different projection error bounds, it is impossible to use the simplification process directly under a specified threshold in section 4.3. To sort the nodes properly according to their projection errors, it is necessary to recount the projection errors for each frame, or evaluate the valid lives of projection errors soundly. The former may help to increase the CPU overhead intensively while the later does not seem a simple task for we have to take into full account the changes of the view parameters and the projection computation.

Instead of trying to obtain optimal image at each frame, which is better, this paper believes, to balance the benefit and the cost between image quality and constant frame rate. When minimizing the screen space error, attention should also be paid to the runtime efficiency.

If more triangles are rendered than required at previous frame, the viewer is known to have reached a more complex scene. Now the number of nodes to split is $n^{S}_{p=b,b=0}$, and the number of nodes to merge is $n^{M}_{p=b+1,b=0}$, then $n^{S}_{p=b,b=0} > n^{M}_{p=b+1,b=0}$. Obviously, this will further increase the triangles to be rendered at current frame, thus greatly decreasing the frame rate.

If base B in the life axis moves to right for a unit, the number of nodes to split is 0, while the number of nodes to merge would be $n^{M}_{p=b+2,b=-1} + n^{M}_{p=b+1,b=-1}$. Since merging one node will at least decrease two triangles, the number of triangles to be rendered at current frame will be $2\times(n^{M}_{p=b+2,b=-1} + n^{M}_{p=b+1,b=-1})$ less than at previous frame.

Similarly, if fewer triangles are rendered than specified at previous frame, B would move towards left for a unit. Then the number of nodes to merge is 0, and that of nodes to split is $n^{S}_{p=b-1,b=1} + n^{S}_{p=b,b=1}$. Since splitting one node will at least introduce two triangles, the triangles to be rendered at current frame will be $2\times(n^{S}_{p=b-1,b=1} + n^{S}_{p=b,b=1})$ more than at previous frame.

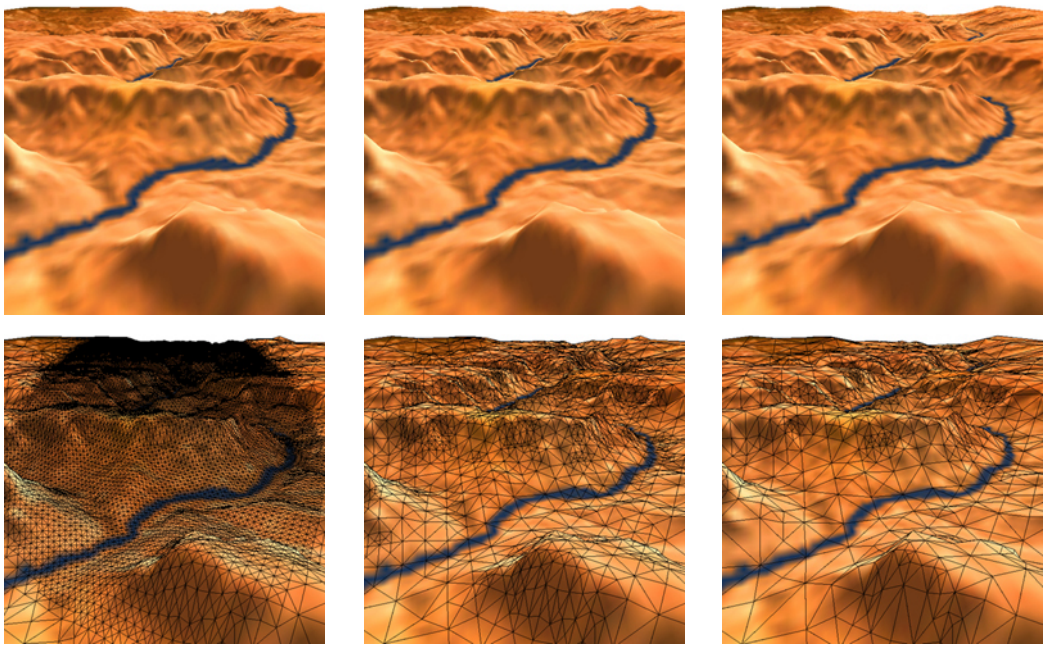
Similar to [2], dual-queue of split and merge is also adopted in this paper. The queue members are links, whose elements are splittable or mergeable nodes in nature. Each link is associated with a value, which is the valid life of the nodes in the link. At each frame the values of the links will be subtracted by 1, indicating that the valid lives of the screen space errors of the nodes will be subtracted by 1. It is just for this purpose, the data structure of split and merge queue are all defined as arrays so as to enhance the efficiency of the algorithm in our implementation.

The procedure for maintaining constant frame rate is as follows:

```
MobilizeQueues()
{
    if(n_prev>n)
        b=b−1;  // move the base B towards right for a unit
        Recompute the valid lives of nodes in the 2 links at the top
        of the merge queue, merge the nodes whose valid lives are
        greater than b.
    else if(n_prev<n)
        b=b+1;  // move the base point towards left for a unit
        Recompute the valid lives of nodes in the 2 links at the top
        of the split queue, split the nodes whose valid lives are less
        than or equal to b.
    else
        Update the valid lives of nodes in the link at the top of the
        split queue, split the nodes whose valid lives equals to b.
        Update the valid lives of nodes in the link at the top of the
        merge queue, merge the nodes whose valid lives equal to
        b+1.
}
```

Geometrically, our method is to expand or shrink the split zone towards the boundary for a unit. The result is deferring or putting ahead the operations of nodes for one frame time to make the number of triangles rendered per frame approximate to the triangle budget. If there are more triangles rendered than specified at previous frame, we prefer to reduce the split zones of all the nodes for a unit, which helps to defer the split operations to be executed at current frame to the next frame. Along with the merging of the nodes at current frame as usual, we will further merge the nodes that should merge at next frame, so as to decrease the triangles to be rendered at current frame. Otherwise, if at previous frame there are fewer triangles rendered than needed, we expand the split zones of all the nodes for a unit to increase the number of triangles to be rendered at current frame.

Needless to say, the above technique cannot guarantee optimal mesh at each frame. As compared to the solution updating the screen space errors of all the nodes and sorting at each frame, we have succeeded in



a. Full resolution after view-frustum culling *b. 6000 triangles per frame* *c. 3000 triangles per frame*

Figure 8: Experimental results

deferring or putting ahead one frame time in splitting or merging nodes according to a fixed threshold. This surely means some priority changes in the queues, resulting non-optimal image.

However, the influence is very limited. If the image at previous frame is optimal, and the base B has not been moved at current frame, it can also be expected to obtain an optimal image too. If the scene can change smoothly and slowly, there would be only a few of nodes for the operations to be split or merged at each frame. What has to be moved is only the base for a unit at each time. Besides, as there are only a few of split and merge operations to be deferred or put ahead, the image will not degraded too much compared to the optimal mesh. Thus, it can be proved that high efficiency of constant frame rate can be achieved at the cost of insignificant image degradation by using our algorithm. The corresponding experiment results also prove that the transformation is a great success.

To achieve such steady frame rate it is only necessary to specify a threshold. Assume the mesh is $d \times d$, in our experiments the threshold we have worked out is as follows:

$$\tau = k\sqrt{6\pi\sum_{i=0}^{d-1}4^i}\cdot\frac{h}{fovy}\cdot\frac{\delta_{ave}}{d\sqrt{tpf}}$$

In which δ_{ave} is the average error of all the nodes, *tpf* is the triangle budget per frame, and k is a coefficient. It can therefore be found that k = 1.5 will be a satisfactory result. More information about the formula can be found in [15].

7 Results

The algorithm was implemented on a PC of PIII450, 128M RAM, with Diamond Viper V770 based on nVidia Rava TNT2. The terrain mesh is 1025×1025. It takes only 3.12 seconds to render the first frame, in which 1.39 seconds is to be spent on reading the mesh and texture from the disk, and 1.66 seconds on preprocessing. The base B fluctuates near about the origin. For example, in a test of a circular flight above the terrain for 6000 frames at 6000 triangles per frame, the statistics is b_{max}=54, b_{min}=−41, b_{ave}≈0. On average there are 1238 splittable nodes and 525 mergeable nodes per frame, on which there are only 128.7 nodes whose valid lives are to be recomputed, 11.5 splits and 2.3 merges to be performed, and 23.3 vertices to be inserted and 23.3 vertices deleted. Table 1 lists the runtime statistics and comparison with previous researches. Since the volume of the simplified split zone is larger than the original, the average screen space error in Table 1 is overestimated.

Figure 8 shows some of the frames. Figure 8a is the scene in full resolution after view-frustum culling while Figure 8b, c the image at 6000, 3000 frames per second.

In our implementation the view-frustum is simplified to a cone with all the four corners of the scene con-

sidered outside the view-frustum. No more noticeable differences can be found between them as compared with the images before simplification except for part of the canyon bottom in the distance in the image of 3000 triangles per frame. Part reasons of the difference between Figure 8c and 8a are that we render the periphery in a lower resolution caused by our view-angle-based error metrics.

Scheme	hardware	window	frames/ sec	triangles/ frame	ave error
ours	PIII, TNT2	600×600	72	3,000	3.2
,,	PIII, TNT2	600×600	55	5,000	2.5
,,	PIII, TNT2	600×600	50	6,000	2.3
,,	PIII, TNT2	600×600	45	8,000	2.0
,,	PIII, TNT2	600×600	30	12,000	1.6
[11]	Inigo2	n/a	38	8,000	n/a
[5]	R10K-MXI	710×512	60	5,000	3.5
,,	R10K-MXI	710×512	30	12,000	1.7
[2]	R10K-MXI	1000×1000	30	3,000	n/a
[7]	Onyx-RE2	640×480	20-30	4-9,000	2.0

Table 1: Statistics and comparison with previous work

We adopt simple geometry morphing similar to [2, 5], thus avoid disturbing pops when the resolutions change in the mesh. To reduce the transport burden between CPU and graphics pipeline, we also adopt incremental triangle stripping in a locally optimization manner. In average there are 3.7 triangles in a strip.

8 Conclusions

We propose a new view-dependent method of real-time terrain rendering. The method introduces block-priority-based traversal of quadtree and view-angle-based error metrics. By simplifying the split zone of a node to an axis-aligned box, we can compute the valid life of projection error with much less cost. With these measures the algorithm achieves constant frame rates easily. The experimental results show that our method can real-time render large scale terrain on a low-cost PC. We hope our efforts will be some help to the broader applications of view-dependent techniques. Future work includes disk operation and texture map LOD processing, etc.

References

[1] Baoquan Chen, J. E. Swan II, Eddy Kuo and A. E. Kaufman. LOD-Sprite Technique for Accelerated Terrain Rendering. In *IEEE Visualization '99*, pages 291-298, 1999.

[2] M. A. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich and M. B. Mineev-Weinstein. ROAMing Terrain: Real-time Optimally Adapting Meshes. In *IEEE Visualization '97*, pages 81-88, 1997.

[3] C. Erikson and D. Manocha. HLODs for Fast Display of Large Static and Dynamic Environments. *Symposium on Interactive 3D Graphics 2001 Proceedings*, 2001.

[4] H. Hoppe. View-Dependent Refinement of Progressive Meshes, In *Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series*, pages 189-198, 1997.

[5] H. Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *IEEE Visualization '98*, pages 35-42, 1998.

[6] R. Klein, D. Cohen-Or and T. Hüttner. Incremental view-dependent multiresolution triangulation of terrain. In *The Journal of Visualization and Computer Animation*, *9(3)*, pages 129-143, 1998.

[7] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hughes, N. Faust and G. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields. In *Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series*, pages 109-118, 1996.

[8] Xuehui Liu, Enhua Wu. Hierarchical Structure with Focus Criterion for Rendering Height Field. In *Journal of Computer Science & Technology*, 13(12), pages 1-8, 1998.

[9] D. Luebke. A Survey of Polygonal Simplification Algorithms, UNC Technical Report TR97-045. Department of Computer Science, University of North Carolina at Chapel Hill, 1997.

[10] D. Luebke and C. Erikson. View-Dependent Simplification of Arbitrary Polygonal Environments. In *Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series*, pages 199-208, 1997.

[11] R. Pajarola. Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. In *IEEE Visualization '98*, pages 19-26, 1998.

[12] G. Schaufler, J. Dorsey, X. Decoret and F. X. Sillion. Conservative Volumetric Visibility with Occluder Fusion. In *Proceedings of SIGGRAPH 2000, Computer Graphics Proceedings, Annual Conference Series*, pages 229-238, 2000.

[13] J. Shade, D. Lischinski, D. Salesin, T. DeRose and J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In *Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series*, pages 75-82, 1996.

[14] A. J. Stewart. Hierarchical Visibility in Terrains. In *Eurographics Rendering Workshop 1997*, pages 217-228, 1997.

[15] Yadong Wu. Real-time rendering techniques in 3D GIS. Doctoral dissertation, Department of Computer science, Beijing Institute of Technology, 2001.

[16] J. C. Xia and A Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In *IEEE Visualization '96*, pages 327-334, 1996.