A VIEW-DEPENDENT LEVEL-OF-DETAIL MODEL FOR RENDERING IN THREE-DIMENSIONAL GIS

Wenzhong SHI[1] Bisheng YANG[1,2] Qingquan LI[2]

[1]Department of Land Surveying and Geo-Informatics The Hong Kong Polytechnic University, Kowloon, Hong Kong
lswzshi@polyu.edu.hk
[2]R&D Center for Spatial Information and Network Communication Wuhan University, Wuhan, Hubei, China 430079
ybs_sw@sohu.com,qqli@whu.edu.cn

Commission II, WG II/6

KEY WORDS: View-dependent LoD, hierarchical surface model, multi-resolution data model, GIS

ABSTRACT:

In a three-dimensional GIS, data volume is a critical issue particularly when we have to render the large amount data for visualizing a city in real time. For efficient analyzing and visualizing these data in real time, this study intend to develop a solution for fast visualizing large dataset three dimension (3D) model by a proposed view-dependent level-of-detail (LoD) model.
In this paper, we first propose a structure and corresponding algorithm to construct 3D data model of LoD for view-dependent rendering. Second, the algorithm is implemented within a prototype software Space/Info, which is developed by authors, and these are described in detail. Third, the performance and the newly proposed algorithm are tested via an experimental study in terms of roaming speed. Finally, the analysis and some conclusions are drawn..

1. INTRODUCTION

Recent advances in 3D data acquisition, reconstruction, visualization and virtual reality make the data volume and category increase dramatically. The corresponding geographical datasets often contain many layers or themes and with a large data volume. The data volume of 3D model perhaps exceeds the capability of any current computer for interactive rendering. Obviously, the size of data volume determines the speed of interactive rendering. In fact, the rendering speed is of vital importance because we need a high frequency interaction with the datasets, whether in terms of moving around, flying or using dynamic behavior as a method to encode certain attribute data (Spitaleri, 1993; MacEachren, 1994). To shorten the gap between the increased 3D data volume and rendering capability of computer, the complexity of 3D model has to be simplified. Currently, research on the simplification complex surface model has been motivated by attempting to shorten the gap between real-time rendering and volume of 3D data. A series of algorithms have been implemented for this development. These may include, for example, triangle mesh decimation (Schroeder, 1992, 1997), vertex clustering (Rossignac, 1993), adaptive subdivision algorithm (Eck, et al, 1995), quadric error metrics (Garland, 1998), voxel-based object simplification (He, et al, 1995) and algorithm based on quad-tree (Lindstrom, 1996). With the simplification algorithms, mentioned above, we can obtain a hierarchy of approximations to original 3D data model. Moreover, each level has a uniform resolution. These approximations often have a number of distinct LoD while rendering. Many of these algorithms have been used in terrain surface simplification and achieve reasonable results. However, many of them do not consider the viewer's conditions at rendering.

Several algorithms about running time LoD model have been published. They are dynamic view-dependent simplification (EI-Sana, 1999; Xia, 1997) and progress mesh (Hoppe, 1998). But they were not used for digital city model simplification directly where are with its own particular problems. For instance, in digital city, there are several thousands of objects in it, including complex architectural buildings, roads and terrain. It is very likely that only a limited number of objects or models among the whole can be seen from a particular viewpoint and at a visualization session. Therefore the viewer conditions have to be considered at running time. Xia (1997) constructed an adaptive level-of-detail surface model using merge tree, which is proved can generated continuous level-of-detail model at running time. Moreover, the resolution is related to viewer direction and the different region of the model can be connected seamlessly, which is improved by EI-Sana (1999). The merge tree can be constructed offline, at rendering, whether the node is collapsed or split is decided by the projection error using a Euclidean distance between the nodes, but without considering the feature lines of surface model such as ridges, water and others. It may cause problem if the feature lines are ignored, and an example is given in Figure 1. In such a process, the feature line (edge) can be lost if the feature line is not given a special consideration. At the same time, it introduces dependencies list for preventing surface fold at running time, which is proved that they are very expensive (in terms of data volume and retrieve speed) to store.

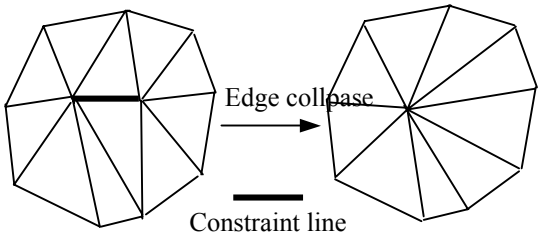

Figure 1. Edge collapsing without considering constraint line

In order to implement the merge tree algorithm for the digital city study, this algorithm has been further extended in this study. As a result, the extended mode is made suitable for constructing level-of-detail model for a digital city. With our further development, the algorithm can make the terrain feature invariable retained and the efficiency of storage space is improved. In the same time, the method for encoding adjacent vertexes array and for determining the visible faces of building model are also developed as a further extension to the original algorithm.

The following parts are arranged as the following, in section 2 the data structure and method of the algorithm are presented; in section 3 the implementation of the model are presented; in section 4, the performance and results of the algorithm are tested and analyzed. Finally some conclusions and outlook are presented.

2. DATA STRUCTURE OF THE ALGORITHM

The data structure for an algorithm is very importance, because it affects the efficiency of algorithm greatly. As a result, various data structures have been developed for modeling terrain surface and buildings. For a building model, the Layer-Combined model (Yang, et al, 2000) was used in city modeling. For terrain model, we use four arrays containing vertex, triangles, edge, vertex index about edge and triangle to store the terrain surface model and their topological relationships.

- Vertex array contains the index of vertex and its three dimensional coordinate (x, y, z);
- Triangle array contains the index of three vertexes of a triangle t and its normal;
- Edge array contains the index of the source and destination nodes of an edge and the flag of constrain line (0 or 1), moreover, the index of source node is less than that of destination node; and
- Vertex index contains the index of a vertex (ID), triangles with the labeling id and the edges with the labeling ID.

In our algorithm, the vertex hierarchy of terrain and the vertex hierarchy of building are introduced for storing displayed vertexes and triangles. In the algorithm, the vertex hierarchy has two meanings: one is for terrain surface and the other is for building model, because the building models are discrete models.

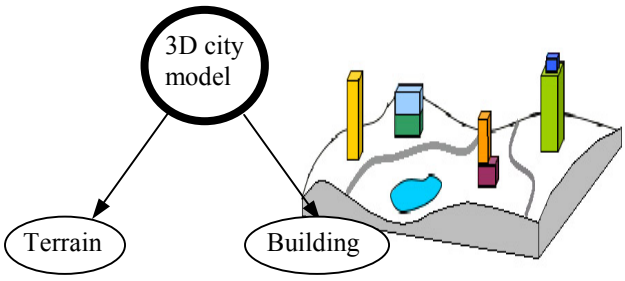

Figure 2. Vertex hierarchy of a digital city model

2.1 Vertex Hierarchy of Terrain

Vertex hierarchy of terrain is constructed by the splitting or collapsing of vertex each time, the new vertex generated by collapsing or splitting consists of the node of vertex hierarchy. The vertex hierarchy of terrain is different from the model of Xia (1997). Our algorithm does not store the dependencies lists of collapsing edge, but only keep the relation of root vertex with neighbor vertexes at every depth. At the same time, the encoding and compressing algorithms about neighbor vertexes are developed to encode or compress the vertex index when collapsing or splitting vertex. Before introduce the structure of terrain vertex hierarchy, several terms are defined as follow.

- parentnode, which means that when one vertex pair (p1, p2) is collapsed, a new vertex p will be generated, the vertex p is the parent of p1 and p2, and p1 and p2 is the child of vertex p. One is the leftChild, the other is the rightChild.
- rootnode, when a vertex p has no ParentNode, the vertex p is the rootnode.
- adjacent vertex, all the vertexes, which is linked with the vertex p, is named as the neighbor vertex of p, and each of vertex of adjacent vertex and vertex p consists of one edge.

The structure of terrain vertex hierarchy is as the following.

```
Struct vertex{
        Int id;                 //the index in vertex array
        Int virtualid;          //the index after collapsing
        int ParentNode;         //the index of parent of vertex
        int childNode[2];       // the indexes of children of
                                vertex
        Int nNums;      //the number of neighbor vertex
        Int* id;        //the id array of neighbor vertex
        BYTE byteStatus;        //the status of vertex
        Int  nDepth;    // the depth of vertex hierarchy
        Int  nPos;      //the position of right child in adjacent
vertexes array
    };
```

In vertex hierarchy structure, the element of byteStatus is indicated whether the vertex is a rootnode or not. If it is not a root vertex (parentnode is valid), the adjacent vertex array can be encoded form its parent vertex, which is illustrated as Figure. 3.

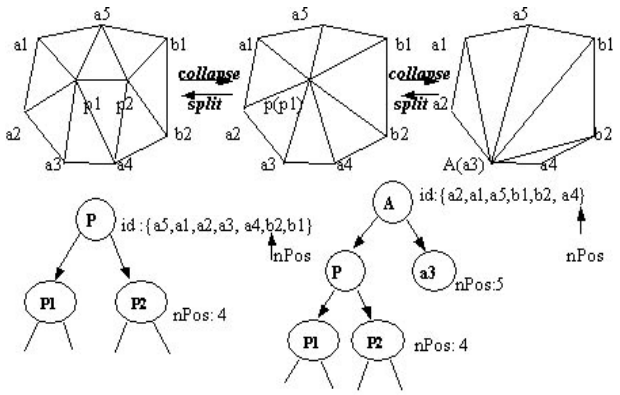

Figure 3. Adjacent relationships among vertex after collapsing or splitting

In figure 3, the adjacent vertex and the position (nPos) are recorded, adjacent vertex of node indicates the link relationship of the node, nPos is used to encode the adjacent vertex of leftChild and rightChild of the node. The vertexes prior the position of nPos is the adjacent vertex of leftChild, the other is the adjacent vertex of rightChild. For example in figure 3, nPos equals 4 and the adjacent vertex of vertex p is{a5, a1, a2, a3, a4, b2, b1}, based on the regulations above, the adjacent vertex of

p1 (the leftChild of p) is {a5,a1,a2,a3,a4} and the adjacent vertex of p2 (the rightChild of p) is {b2,b1}.

In vertex hierarchy, only the root adjacent vertexes of each depth are stored, the adjacent of other nodes in vertex hierarchy are not stored. However, their adjacent vertexes can be encoded dynamically during running period. For compressing, the adjacent vertexes of child are stored in their parent node when edge is collapsed. At the same time, right child position in the adjacent vertex of parent node is recorded, then the left child adjacent vertexes are released if its depth equals zero. Here, we define the depth of vertex in full resolution model equals zero. When edge (p1, p2) collapses into p, we do not calculate new vertex but select p1 or p2 as new vertex for p. The validity of edge collapse (for details see Xia 1997). The criterion of select is (1) if p1 and p2 are not boundary point, the minor error of vertex will be deleted. The method of calculating error is in Garland (1999), (2) If p1 or p2 is a boundary point, then delete interior point, so as to keep the integrity of whole model. Whether the vertex is split or collapsed based on it's project error on the screen.

On the other hand, we can see that at each depth of vertex hierarchy only the parent node contains adjacent vertex array. Although the adjacent vertex array of the left child and right child are not kept in vertex hierarchy, the adjacent vertex array of theirs can be easily encoded from their parent node adjacent vertexes array. It is easy to see that the adjacent vertexes array occupied a large portion of whole vertex hierarchy storage space, so at every depth the parent-child relationship should be decided among as many vertexes as possible. The encode algorithm of adjacent vertexes array will be explained detail in the Section 3.

2.2 Vertex Hierarchy of Building

For the reason that the building model is stored as LC model (Yang, et al 2000), moreover, Lod information is stored between different layers and each layer consists of a single object such as roof, window or others. All the building models comprise the total model in digital city. In fact, we can think the vertex hierarchy of building as object hierarchy (a complex building model can be divided several simple objects according to LC model), so every building model consists of a tree structure, the overall building model is composed of a forest data structure. The overall building model can be illustrated as Figure 4.

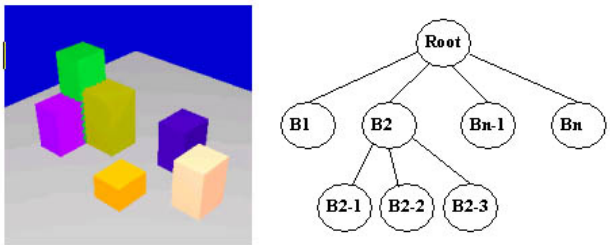

Figure 4. Hierarchy relationships among vertex in the building mod

In Figure 4, each tree of the forest is a building model, leafs of each tree consist of the layers of single complex building model. Moreover, every tree contains certain level of error and attribute information such as material and texture of the building surface. The information about error of each layer of a building is stored in leaf of the tree. For instance figure 4, supposing the limited error of B2-1, B2-2 and Bn are all within the set error at a certain visualization session, they can be rendered at the same time, others will not be rendered. On the other hand, it is easy to see that only certain number of faces of each visible layer can be seen during rendering, the other faces are invisible – the faces that are behind of visible faces of the object. In fact, the normal of face plays an important role in visualization, the determination if a face is visible is explained in detail in Section 3.

3. IMPLEMENTATION OF THE ALGORITHM

According to the above discussed data structure, the algorithm was implemented for rendering LoD digital city model. According to Section 2, the vertex hierarchy is constructed by edge collapsing or splitting. Initially, the terrain model and building model are stored in full resolution in the model. The edge collapsing method is then applied to construct vertex hierarchy offline. At the same time, considering the constrain line when edge collapsing. The neighbor region of collapsed edge effected should not contain constrain lines. Although it effects the simplification of terrain surface, it can keep the important feature of terrain. In splitting a vertex, the encoding algorithm has to be implemented to acquire the adjacent vertexes array. At the same time, according to the analysis above, the vertex hierarchy tree is constructed based on vertex split or collapsed. In the algorithm, the criterion of vertex split or collapsed is determined by the projection error of vertex split or collapsed. Projection error indicates the changes on the project screen after the vertex is split, on the average, the pixel is used as the unit of projection error. It is evident that when the part of model is far way the viewer, it occupies little area on the project screen, so the project error is related with the viewing conditions (viewer height and direction). In the following parts, the relationship between viewing conditions and projection error is explained in detail.

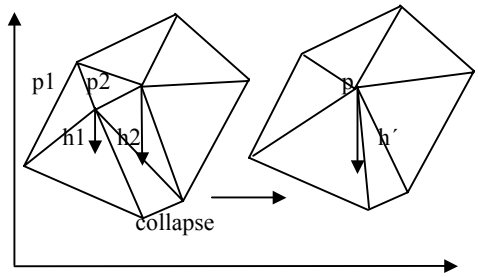

Figure 5. Height error after vertex collapse

3.1 The Criterion of Projection Error

The projection vertex error of the vertex plays a vital role when building the vertex hierarchy tree. So as to keep the maximal similarity of model between adjacent two Lod model, the projection error on the projection plane, which is brought by the vertex collapse/split, should not exceed a limited value between adjacent two Lod model, otherwise, the "pop" will emerge. For terrain, no matter what the vertex collapse/split, the plane position(x,y) does not vary, but the value in z direction is variable, so the projection error is defined as the size on the projection plane of height error, which is brought by the change in z direction of different resolution model. Figure 5 illustrate

the height error, which is the result because of vertex collapse/split, the height error will inevitable bring a certain value (projection error) on the projection screen. Next, we will demonstrate the relationship between projection error and viewer conditions.

In Figure 5, the right part is the result of vertex collapse. After vertex collapse, there is a variable value in Z direction. In order to calculate the variable value, we independently calculate the distance of collapsing vertex (p1 and p2) to the plane, which is comprised by the adjacent vertex array, Supposing the distance is h1 and h2. After the p1 and p2 is collapsed, the distance of vertex p to the plane also is calculated, supposing the result is h´. So the height error of vertex collapse is

$$\Delta h = h^{'} - \min\{h1, h2\} \qquad (1)$$

Figure 6 illustrates the relationship between height error and viewer conditions, β is the pitch angle of observer direction center, α is the angle between observer direction of height error and observer direction center, h is the height error, f is the distance between observer and projection plane, and d is the perpendicular distance between observer and the center of height error. Supposing the projection value on the projection plane of the height error h is δ. According to figure 6, the next relationships can be reasoned. Supposing the scale between projection plane and display screen is 1:1, the **δ** is the pixel size of height error on the screen.

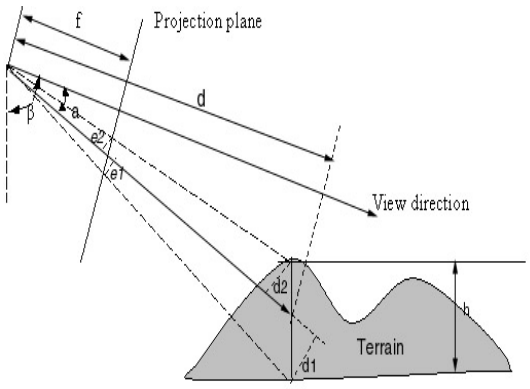

Figure 6. The relationship between height error and projection error

$$\delta = 4 * h * f * d * \sin(\beta - \alpha)/((4 * d^2 - h^2 * \cos(\beta) * \cos(\beta)) * \cos(\alpha)) \qquad (2)$$

Based on the Eq.2, it is evident that the projection error not only is related with the viewer distance, but also it is related with the viewer direction and height error. So once the threshold of projection error is determined, when vertex hierarchy tree is built, the projection error of vertex is calculated before compared with the threshold, if the projection error does not exceed the threshold, the vertex will not be split, otherwise it will be collapsed.

3.2 Encoding Adjacent Vertex Relationships

In rendering, the vertex hierarchy is traversed every frame. If project error of the distance between parent node and child nodes exceeds the specified threshold, this parent node has to be split. On one hand, the triangles contain the parent node will be deleted from the triangle list. The triangles that the child of its constructed will be added in the triangle list for rendering, the encoding algorithm is detailed as the following.

Supposing the parent node contains adjacent vertexes array a ={a1, …, aN}, and the position of its right child at this array is m(0<m <N), and the index of its rightChild is p2 , the index of leftChild is p1. According to the regularity of Section 2.1, the adjacent vertexes array of rightChild is R={am,..aN,a1,p1}, moreover any adjacent two point (Rn-1,Rn) with p2 consist of a triangle, if the adjacent vertexes array of its leftChild is zero, the adjacent vertexes array is L = {a1, a2, ..., am, p2} and any adjacent two point (Ln-1, Ln) with p1 consist of a triangle, otherwise, adjacent vertexes array of leftChild equal its adjacent vertexes array.

3.3 Determining Visible Faces in Building Model

According to vertex hierarchy of building model, the layer of each building can be decided whether it is to be rendered. The project error on the screen can be calculated by the minimum box enveloped it. If the project error is less than set the specified threshold, it will be omitted. The visible faces of the visible layer in a building will be further determined by using surface's normal. For the reason that the building models occupy a large portion of the overall region, it is necessary to judge the visibility of faces. For instance, what is illustrated in figure 7. Supposing the building is in the view scope, the visible faces can be judged by its normal.

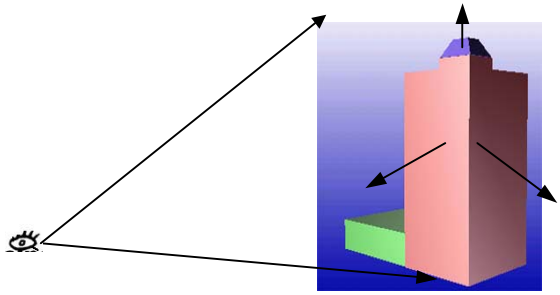

Figure 7. Determination of visible faces

Based on the viewer position (x, y, z) and face center point (x_f, y_f, z_f), the normal N_V = (x-x_f, y-y_f, z-z_f) can be acquired. Thus the angle Φ between N_V and N_f (normal of face) equals.

$$\Phi = \text{acos}(N_V \bullet N_f) \qquad (3)$$

In Eq.3, if $\Phi > \pi/2$, the face is invisible. When rendering, all the faces of visible layer in each building are calculated by Eq.3. The visible surfaces are added in triangle list for rendering, others will not be tackled.

Terrain Triangle	Building Triangle	Screen Resolution	Threshold pixel (*T1*)	Threshold pixel (*T2*)	Time1 (Second)	Time2 (Second)
32385	0	800x600	2.0	4.0	2.23	5.78
534607	4000	800x600	2.0	4.0	29.30	63.43

Table 1. The running time of under different threshold

4. EXPERIMENTAL STUDY AND ANALYSIS

Based on the algorithm we discussed in this paper, the algorithm was implemented in c^{++} language and OpenGL. It is also tested with several datasets. In the following, several experimental results are listed. These results are obtained from a model of real world.

Table.1 illustrates the running time about the simplification of terrain surface at different set threshold (*T*). Figure 9 illustrate the snapshot of 3D city model in Hongkong, which is created in Space/Info software.

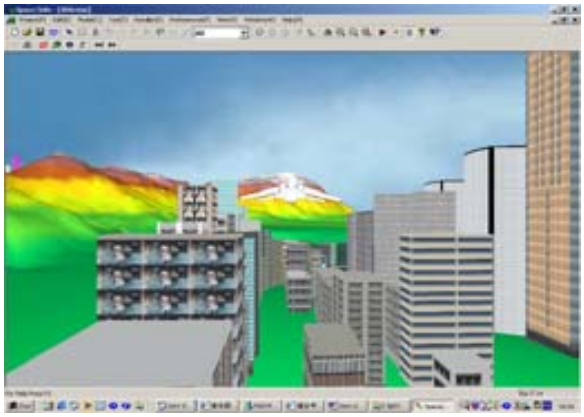

Figure 9. The 3D city model

5. CONCLUSIONS

In this paper, a data structure and corresponding algorithm for construct 3D data model in Lod rendering process was proposed. The algorithm was then implemented within a prototypes software Space/Info and these are described in detail. The performance and the newly proposed algorithm were tested via an experimental study in terms of its running time at various thresholds and the comparison was made about the triangles between terrain and buildings at different running time.

The developed algorithm can be used to build LoD model of digital city in real time. It can also be used to build continuous multi-resolution model for digital city. Moreover, the resolution is related with viewer conditions. Compared with the previous algorithms, one of the advantages of this algorithm is that it can retain the important feature of the terrain surface to be built. The other advantage is the storage structure of adjacent relationships during splitting and collapsing is improved.

Furthermore, the method for determining the visible faces of the buildings was implemented in this study, which proved is able to reduce the rendering triangle numbers greatly and improve the rendering speed. This method is especially efficient for digital city modeling with high density of buildings.

Although exciting results have been achieved from this algorithm, there are still problems unsolved in this algorithm, for example the criterion about selecting the collapsing edge is worth to be further studied, since different criterion can generate different results and running time. Moreover, temporal coherence needs to be further exploited, for the reason that a high degree of frame to coherence during the roaming session is very important. It can eliminate the “pop” aircraft and improved the rendering speed.

ACKNOWLEDGEMENTS

The work described in this paper was substantially supported by funds from The Hong Kong Polytechnic University (Project No.: 1.34.9709), RGC of Hong Kong (Project No.: PolyU5050/98E) and China National Natural Science Foundation (No.69833010).

REFERENCES

Andreas Voigtmann, L. Becker, 1997. A hierarchical model for multiresolution surface reconstruction, *Graphical models and Image Processing* ,Vol.59, No.5, pp.333-348.

EI-Sana Jihad, A.Varshney, 1999.Generalized view-dependent simplification. *Eurographics’99*, Vol.18 ,No.3, pp.1-12.

Eck M, et al, 1995. Multi-resolution analysis of arbitrary mesh, In: *Computer Graphics*, Vol.29, ACM Press, New York pp.173-182.

Floriani Leila, Paola Magillo, 2000.VARIANT: A System for Terrain Modeling at Variable Resolution, *GeoInformatic*, Volume 4 ,Issue 3, pp.287-315.

Floriani Leila, Paola Magillo, 2000. Compressing trianglulated irregular networks, *GeoInformatic*, Volume 4 ,Issue 1, pp.67-88.

Garland M, P. Heckbert, 1998. Simplifying surfaces with color and texture using quadric error metrics, *Visualization’98*, Oct 1998.

GAHEGAN, M, 1999.Four barriers to the development of effective exploratory visualization tools for geosciences. *INT. J. Geographical Information Science*, vol.13, No.4, pp.289-309.

Hoppe, H., 1996. Progressive meshes, In: *SIGGRAPH’96 Proceeding*, pp99-108.

Hoppe, H., 1998. Smooth view-dependent level_of_detail control and its application to terrain rendering. In: *IEEE Visualization* ,pp.35-42.

He T, et al. 1995.Voxed-based objects simplification. In:*Proc. Visualization95 IEEE CS Press*, Los Alamitos Calif pp296-303.

Koninger, A.,1998. 3D-GIS for urban purpose. *GeoInformatic* 2,pp.1 79-103.

Luebke David, C,Erikson,1997.View-dependent simplification of arbitrary polygonal environments. In: *Computer Graphics*. ACM press, New York pp.199-208.

Lindstrom Peter, et al, 1996. Real-time continuous level of detail rendering of height fields. *Computer Graphics* Vol.20 ACM Press New York,pp. 109-118.

Lindstrom Peter, G.,Turk, 2000.Image-driven simplification. *ACM Trans.Graphics* Vol.19 No.3 ,pp.203-241.

Maceachren AM 1994 Times as a cartographic variable. In:Visualization. In: *Geographical Information Systems*. England:Wiley, pp.115-130.

Mahdi Abdelguerfi, WYNNE Chris 1998 Representation of 3-D elevation in terrain databases using hierarchical triangulated irregular networks: a comparative analysis. *INT. J. Geographical Information Science* Vol. 12 No.8 ,pp.853-873.

Popvie Jovan, Hoppe Hugues 1997 Progressive simplicial complexes. In: *SIGGRAPH 97 Proceeding* ,pp.217-224.

Rossignac J, Borrel Paul 1993 Multi-resolution 3d approximations for rendering complex scene. *Modeling In: Computer Graphics*. Springer-Verlag, Berline ,pp.455-465.

Schroeder W, Jonathan A Zarge 1992 Decimation of triangle meshes. Computer Graphics (*SIGGRAPH'92 Proceeding*). 26(2),pp.65-70.

Spitaleri R 1993 Reference models for computational visual simulations. In: P.Palamidese(eds) *Scientific Visualization Advanced Software Techniques*, pp.3-14.

Xia J, El-Sana J, et al 1997 Adaptive real-time level-of-detail-based rendering for polygonal model. *IEEE Transactions on Visualization and computer graphics*.Vol.3 No.2 ,pp.171-183.

Yang, B.S, Q.Q,Li, 1999. Buildings modeling for 3d city model. *Geo-Spatial Information Science*, Vol.2, No.1,pp.109-114.

Yang, B.S, Q.Q.Li ,2000. Building model creating and storing in 3d urban GIS. In:*International Archives of Photogrammetry and Remoting*.Vol.XXXIII, Part B4 ,pp.341-349.