ISBN: 978-972-8924-39-3 © 2007 IADIS

B-QUADTREE. A DATA STRUCTURE FOR URBAN FLIGHTS AND WALKTHROUGHS

J.L. Pina, F.J. Serón, E. Cerezo
Grupo de Informática Gráfica Avanzada (GIGA)
Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
jlpina@ivo.cps.unizar.es, seron@unizar.es, ecerezo@unizar.es

ABSTRACT

A data structure based on urban blocks which vastly improves rendering speed in urban walkthroughs and flights is presented in this paper. The city is split by means of a quadtree partition and the block is adopted as the basic urban unit. One advantage of blocks is that they can be easily identified in any urban environment, regardless of the origins and structure of the entry data. Results are promising and lead to a factor three increase in rendering speed.

KEYWORDS

Visualization Urban Quadtree Block Walkthrough Fly

1. INTRODUCTION

The aim of this paper is to find a data structure capable of interactively carrying out urban walkthroughs and flights without the use of predefined paths. In order to fulfill both possibilities, all the elements of the city are grouped into a basic unit: the block. Setting out from a typical Quadtree decomposition (Pajarola, R., 1998), a data structure that we named B-Quatree or Block-Quadtree was built. The use of this data structure leads to a great increase in FPS, both in flights and walkthroughs.

One of the requirements imposed on the new data structure was that it was capable of being applied to urban data with low structure or no structure at all. Therefore, the proposed data structure can be applied to data acquired from several sources: 2D GIS, terrain measurements, etc. Another advantage of our method is that identification of buildings is not required. Our basic unit, the urban block, can be easily identified under any circumstance, which is not the case with individual buildings. Moreover, the use of blocks has an impact on the number of nodes in the structure: there are not as many there would be if we were using buildings, but there are enough for fast tree culling.

2. RELATED RESEARCH

A great variety of solutions can be found in the bibliography related to urban walkthroughs and flights. The challenge still remains: to visualize a large amount of geometric elements all together and in real time. Methods can be classified into four main groups of techniques: LOD or level of detail, billboards, occlusion culling and preprocessing.

LOD is one of the most frequently used techniques: continuous LODs have been tested (Dollner, J. and Buchholz, H., 2005), but these reduce rendering speed; so have hierarchical models (Funkhouser, A. and Séquin, C., 1993) that choose the proper resolution according to the node of the tree being displayed. However, it is very difficult to avoid bothersome drops between frames of different resolutions.

Billboards (Sillion, F. G. and Drettakis, B. B, 1997) and impostors (Jeschke, S. et al., 2005), are based on the use of images of objects instead of the 3D objects themselves. They are very effective in reducing rendering time. The correct use of this technique allows for realistic visualizations with less graphics cost. In fact, the task of visualizing models with many polygons is not really solved, but avoided. A minor variation of billboards is the image reuse technique (Shade, J. et al, 1996): the images of the object are stored in real time for the duration of a frame, and then the images are reused providing changes remain below a certain threshold.

The occlusion culling technique resorting only to software techniques (Teller, S. and Séquin, H., 1991) or taking advantage of the GPU (Bittner, J., et al, 2004), involves culling away objects that are not visible due to their being hidden. The identification of hidden objects from the camera's point of view is the major challenge involved. A new identification is needed for each frame every time the camera or an object moves. Nonetheless, occlusion culling is one of the most common techniques, because in an environment full of graphic elements which are very close to one another, as far as height is concerned, certain objects —those nearest to the camera— will hide more distant ones. Although the algorithm required to fix hidden objects is time consuming, the time saved is worthwhile (Cohen-Or, 2002). This technique is widely used in urban environments, but it is not suited to urban flights in real time.

Preprocessing is the name given to all those techniques that rely on data structuring to facilitate the visualization phase. They have proven to be very helpful for visualization without being time consuming. One form of preprocessing is the Out-of-Core technique, which has been intensively used in other areas. The Out-of-Core technique is used whenever it is desirable to store scene data in discs. These data are dynamically loaded in real-time when needed. Adequate distribution of the data in the disc allows for efficient pagination (Davis, D. et al, 1999). All the related information is stored in the same disc page; afterwards, when needed, it is recovered (El-Sana, J. and Chiang, Y., 2000). Preprocessing can be used to improve the performance of the other techniques already mentioned, reducing the number of visible polygons in the model in the case of occlusion culling, by building trees of occluders, such as a binary tree (Bittner, J. and Havran, V., 2001). Preprocessing is the acceleration option chosen in this paper: we will be using an efficient tree type data structure called B-quadtree. The structure will be generated on the basis of a quadtree decomposition (Finkel, R. and Bentley, J.L, 1974). In terrain visualization, the quadtree decomposition (Ayala, D. et al, 1985) is the most commonly used decomposition, whereas in urban scenes the structure most often used is the R-tree (Guttman, A. and Yormark, B., 1984).

3. THE B-QUADTREE STRUCTURE

The data structure developed has been called B-Quadtree because it is a decomposition of a city's blocks in quadtrees. In this paper, the term block is used to name the group of urban elements completely surrounded by streets, i. e., the usual meaning of urban block.

In order to build the structure, certain tasks have to be carried out first. In particular, we must:

- identify all the blocks in the town
- assign every graphic element of the town to an urban block
- calculate the bounding-box of each block.

In this paper the block is considered the minimum and indivisible unit of the city as well as the basis of the proposed structure. In Figure 1a a part of the city under consideration (Zaragoza) with the blocks already identified is displayed.

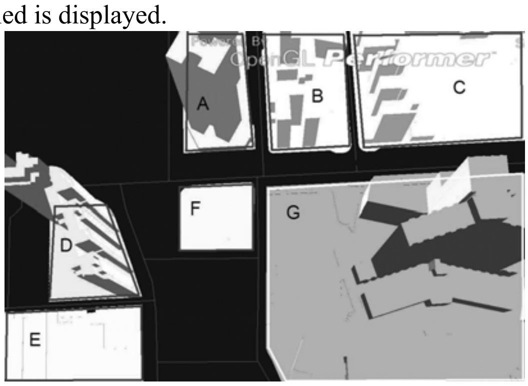

Figure 1a. Identifying the blocks in a city

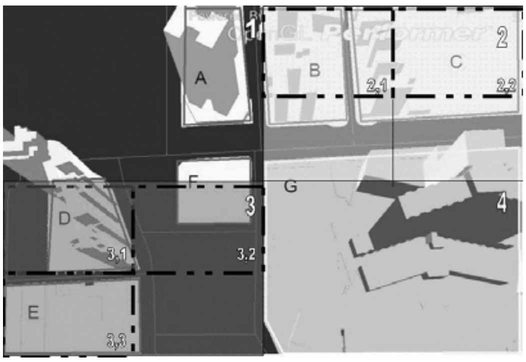

Figure 1b. Quadtree decomposition: every quadrant is identified

To build the B-quadtree, the first step is to take the minor rectangle bounding the city. This rectangle is divided into four equal rectangles called quadrants; all quadrants are recursively subdivided in this manner. The B-quadtree tree is formed by recursive division of the city into quadrants. A quadrant is considered to be indivisible if it contains less than two blocks. Should a division cross a block, the entire block is assigned to a quadrant. The division ends when each block has been assigned to a quadrant and to one only. At the end of the process, every final quadrant must contain a single block or remain empty. Figure 1b presents a B-quadtree decomposition performed on the example shown in Figure 1a.

Figure 2a shows a graphic representation of the B-quadtree structure. Quadrants containing a block are called leaves, and are represented by squares; quadrants that have been subdivided are called nodes and are represented by ellipses. Empty quadrants are not stored in the tree, thus categorizing the quadtree as an adaptive quadtree.

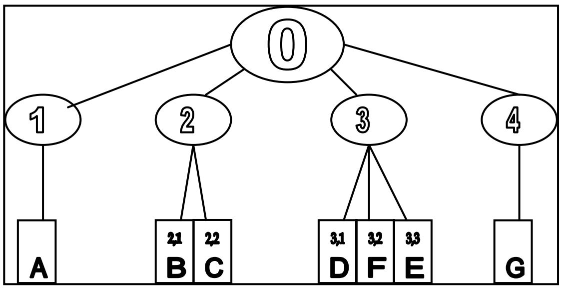

Figure 2a. Tree representation of the example seen in Figure 1

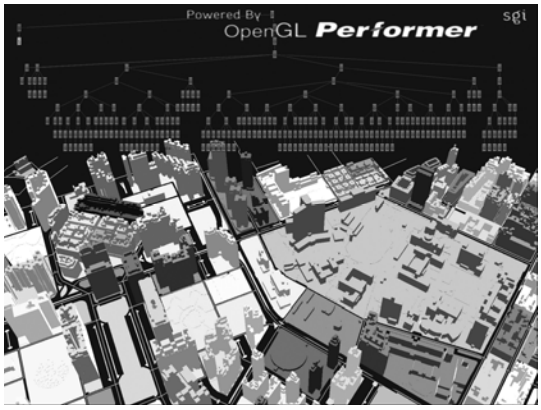

Figure 2b. Tree implementation in Performer

Each leaf of the tree stores the entire geometry of the block as well as the bounding-box calculated for that block. Each node stores a pointer to its descendants and to the bounding-box of the node, which in turn is the result of the union of its descendants' bounding boxes. Therefore, the resulting structure is a tree of the quadrants' bounding-boxes which ends with the bounding-boxes and contains the geometry of the blocks as well. A linear codification based on a numeric key is used to identify all the elements of the tree, nodes and leaves. Storage of the key of each element of the tree is required due to the non-representation of empty blocks.

These are the properties of the B-quadtree data structure:

- Quadrants are disjointed and they are the same size at a specific level
- The complexity of the search for an element is proportional to the depth of the tree, being O(n) in the worst case, where n is the maximum depth of the tree, which tends to be small. Union, intersection and complement share the same complexity, all of which leads to very efficient tree culling.
- Although very similar to an R-tree, the most important difference is that in an R-tree, nodes belong to non-disjointed areas, which makes access to elements belonging to several nodes more difficult.

4. VISUALIZATION WITH PERFORMER

OpenGL Performer, specifically, the perfly application, has been used to test the proposed B-quadtree data structure. In order to load the structure, a file with all the geometry and the B-quadtree structure elements arranged in nodes and leaves is supplied to perfly. A dll reads it, quadrants are loaded as pfGroup and blocks are loaded as pfGeoset, pfBuilder creates the tree in Performer format. Therefore, any other scene graph can be used by simply generating a new dll to load the new format.

Figure 2b shows the B-quadtree tree implemented by Performer while flying over the city. In order to facilitate the identification of urban blocks these are coloured instead of texturised.

5. RESULTS

The data used belong to the city of Zaragoza, (Spain), and have been kindly provided by the city council. The initial file was in Miscrostation format and its data were converted to a 2,013,900 point text format. The B-quadtree structure built was comprised of 145 nodes and 96 leaves. Auxiliary files for the 96 blocks as well as one file with the 300Mb 3D model and 1,688,218 triangles were created. Two other files were generated on the basis of the latter file, one of which contained 610,325 triangles, and the other 2,358,953. All tests were performed with a computer provided with a DualP4 Xeon Pentium 4 processor at 2.8 GHz and a memory of 2Gb. The graphic card is a GeForce Fx 6800 Ultra with a memory of 256 Mb. The operative system is Windows XP. Table 1 shows the evolution of FPS speed for each of the models. The first row shows results without using B-quadtree structure; the second row shows results using the new structure. All the results are in frames per second (FPS), and belong both to flights over the city as well as walkthroughs at ground level. Each time the B-Quadtree structure is used, a substantial increase in frames per second becomes evident.

Table1. Rendering speed in FPS according to the data structure used and the number of triangles of the model

	Triangles		
	610.325	1.688.218	2.358.953
No B-quadtree	9 fps	4 fps	2 fps
B-quadtree	9-20 fps	4-15 fps	2-12 fps

Table 1 shows the results obtained without any additional acceleration technique. The use of B-quadtree structure allows us to apply Performer's usual culling algorithm, from top to bottom and from left to right, based on the bounding-box of the nodes. The culling begins testing the root node, after its descendents are tested. If necessary, the more leftward node, along with its descendent is tested, and all the following nodes are recursively tested, ending with the most rightward node. This culling explains the speed interval obtained in the second row: the closer the camera is to the city, the more nodes are culled away, and the larger the improvement in speed. If no data structure is used, a constant number of FPS is obtained because triangles are not culled away when they disappear from the screen. Better results, in FPS terms, are obtained when more polygons are visualized: FPS may be multiplied by up to 6, a figure which corresponds to the bigger model. It should be noted that the rate of improvement obtained is similar or superior to those obtained in other relevant research. Thus, in their paper, El-Sana et al. (El-Sana, J. and Chiang, Y., 2000) based on a view-dependent tree and out-of-core storage, reported improvements between 1 and 5.6 times in FPS depending on the size of the model and system memory. Downs et al., in research based on a binary tree of occluders (Downs, L. et al, 2001), obtained FPS improvement factors of 3.5, 4.5 or 5.5, depending on the desired image quality (less improvement leading to image quality due to transition drops between frames).

6. CONCLUSIONS

We have presented a new data structure, the B-quadtree. This structure is especially suited to urban environments. These are its main features:

- It is defined by considering the city block as the basic and logical unit. The advantage of the block as opposed to the traditional unit, the building, is that it is easily identified regardless of the data source format.
- The usefulness of the structure has been tested with low structured city data, which makes its application appropriate to almost all city data.
- Tests have been carried out with a standard application, and it could equally be used with any scene graph application.

The results obtained by the tests show that when using the B-quadtree structure to perform city walkthroughs and flights, rendering times are divided by up to a factor of 6.

ACKNOWLEDGEMENTS

This work has been partly financed by the Spanish Dirección General de Investigación, contract number Nº TIN2004-07926 and TIN2004-07672-C03-03 and by the Government of Aragón by way of the WALQA agreement (ref. 2004/04/86).

REFERENCES

Ayala, D., Brunet, P., Juan, R., and Navazo, I. 1985. "Object representation by means of no minimal division quadtrees and octrees". *ACM Trans. Graph*. 4, 1 (Jan. 1985), 41-59

Bittner, J. and Havran, V., "Exploiting Temporal and Spatial Coherence in Hierarchical Visibility Algorithms", *Proceedings of Spring Conference on Computer Graphics SCCG 01*, IEEE Computer Society, 2001, 213--220

Bittner, J. and Wimmer, M. and Piringer, H. and Purgathofer, W., "Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful", *Computer Graphics Forum Proceedings of EUROGRAPHICS 2004*, 3, 23, 615—624

Cohen-Or, D. and Chrysanthou, Y. and Silva, CT. and Durand, F., A Survey of Visibility for Walkthrough Applications, *IEEE Transaction on Visualization and Computer Graphics*, 2002

Dollner, J. and Buchholz, H., "Continuous level-of-detail modeling of buildings in 3D city models *": Proceedings of the 13th annual ACM international workshop on Geographic information systems*, Bremen, Germany, 2005, 173—181

Davis, D. and Ribarsky, W. and Jiang, T. Y. and Faust, N. and Ho, S., "Real-Time Visualization of Scalably Large Collections of Heterogeneous Objects", *IEEE Visualization 99*, San Francisco, 1999, 437—440

Downs, L. and Moller, T. and Sequin, C.H., "Occlusion horizons for driving through urban scenery", *SI3D 01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 2001, 121—124

El-Sana, J. and Chiang, Y., "External Memory View-Dependent Simplification", *Computer Graphics* Forum, 19, 3, 2000

Finkel, R. and Bentley, J.L, "Quad Trees: A Data Structure for Retrieval on Composite Keys", *Acta Informatica* 4, 1, 1974, 1-9

Funkhouser, T.A. and Séquin, C.H., "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments", *Computer Graphics Annual Conference Series*, 27, 1993, 247—254

Guttman, A. and Yormark, B., "R-Trees: A Dynamic Index Structure for Spatial Searching", *SIGMOD 84 Proceedings of Annual Meeting*, Boston, Massachusetts, June 18-21, 1984, 47-57

Jeschke, S. and Wimmer, M. and Schumann, H. and Purgathofer, W., Automatic Impostor Placement for Guaranteed Frame Rates and Low Memory Requirements, *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games,*2005, 103—110

Pajarola, R., "Large scale terrain visualization using the restricted quadtree triangulation", *VIS 98: Proceedings of the conference on Visualization 98*, Research Triangle Park, North Carolina, United States, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, 19—26

Sillion, F. G. and Drettakis, B. B., "Efficient Impostor Manipulation for Real-Time", *Visualization of Urban Scenery. Computer Graphics Forum Proceedings of EUROGRAPHICS 97*, 16, 3, 1997, 207—218

Shade, J. and Lischinski, D. and Salesin, D. H. and DeRose, T. and Zinder, J., "Hierarchical image caching for accelerated walkthroughs of complex environments", *SIGGRAPH 96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, 75—82

Teller, S.J. and Séquin, C.H, "Visibility preprocessing for interactive walkthroughs", *Computer Graphics 4*, 25, 1991, 61—68