Remote interactive walkthrough of city models

Jean-Eudes Marvie, Julien Perret, Kadi Bouatouch

Remote Interactive Walkthrough of City Models

Jean-Eudes Marvie, Julien Perret, Kadi Bouatouch
IRISA
Campus de Beaulieu
35042 Rennes cedex, France
{jemarvie,juperret,kadi}@irisa.fr

Abstract

This paper presents a new navigation system built upon our client-server framework named Magellan*. With this system one can navigate through a city model represented with procedural models transmitted to clients over a low bandwidth network. The geometry of these models is generated on the fly and in real time at the client side. The navigation system relies on different kinds of preprocessing such as space subdivision, visibility computation as well as a method for computing some parameters used to efficiently select the appropriate level of detail of objects. These two last kinds of preprocessing are automatically performed by the graphics hardware.*

1. Introduction and Related work

Transmission and real-time visualization of massive 3D models such as cities are constrained by the networks bandwidth and the graphics hardware performances. These constraints have led to two research directions that are progressive 3D model transmission over Internet or a local area network and real-time rendering of massive 3D models.

With regard to progressive 3D model transmission, many papers suggest the use of geometric levels of detail (LODs) that also speed up the rendering. In [12], the LODs to be downloaded are selected according to their distance from the viewpoint. In [14] the available bandwidth, the client's computational power and its graphics capabilities are also used. In [15], the expected improvement in image quality that is achieved by transmitting an object is used to obtain better results. In [13] the amount of data to be transmitted over the network is reduced by using procedural models.

As for real time rendering of massive 3D models on a single computer, the most commonly used solution consists in subdividing the scene into cells and computing a potentially visible set (PVS) of objects for each view cell. During walkthrough, only the PVS of the cell containing the current viewpoint is used for rendering. Many systems for interactive building walkthrough [1, 16, 5, 8, 10] make use of this approach. For these systems the view cells are, most of time, the rooms of the buildings. City models can also be visualized using such a method [3, 17] where view cells are extruded road footprints. For a more general type of complex scenes, occluder fusion [11] or extended projections [4] methods allow for conservative visibility computation. Finally, in [2] the authors distinguish fully visible sets from hardly visible sets (HVS) of objects. Using the HVS classification, an appropriate LOD is selected for each hardly visible object.

2. Overview

We present a system which allows real-time walkthrough of 3D city models located on a remote machine and transmitted over a low bandwidth network, using TCP/IP protocol. The server provides access to several city models, each one being represented by one database, each database being a set of VRML97 files describing the 3D city model. Each remote client machine can connect to the server to walk through a city model using its associated database. So far, there is no interaction between clients, and each client renders its own representation of the city model.

The 3D city models we use are outputs of the generator described in [7]. Each model contains a set of roads, crossroads and buildings. With this generator, one can choose the type of models to be used for each of these objects. In addition, the generator produces some additional data such as the street network (coded as a graph) and the adjacency relationships between buildings.

In order to optimize transmissions and the rendering process, we combine visibility calculation with LODs generated on the client side by procedural models transmitted over the network. We have also developed a new method that makes use of the visibility computation results to select the appropriate LOD for each object during the rendering process.

With this aim in view, the navigation space (roads and crossroads) is subdivided into view cells and a PVS of objects (roads, crossroads, buildings) is computed for each cell. In addition, we determine the adjacency relationship between cells. During walkthrough the next visited cell, adjacent to the current one, is found using motion prediction and prefetched [5] as well as its PVS. In this way, the database is progressively transmitted to the client and the geometry used for rendering is the PVS of the visited cell.

Although the database is progressively transmitted, some PVSs may still require too much network bandwidth, which causes a latency time that cannot be compensated for by prefetching. Therefore, we use procedural models for roads, crossroads and buildings to avoid geometry transmission. The server database contains a library of procedural models as well as some sets of input parameters. Each of these sets is used by one procedural model to generate the geometry of one object. Whenever a client receives a set of input parameters related to a procedural model, it generates the associated geometry. To generate different geometric objects corresponding to the same procedural model, the client just needs to download the procedural model as well as the sets of input parameters for these geometric objects.

With regard to the rendering process, a PVS might still contain too many polygons to get an interactive frame rate (say, 25 frames per second). In order to reduce the amount of polygons to be rendered, the geometry of the PVS's objects is represented with LODs that are generated by procedural models. Usually, the suitable LOD is selected using an euclidian metric giving the distance between the viewpoint and the object center. In our implementation, we propose a different method which takes advantage of the visibility computation results. While computing visibility for one cell, the system computes an ACH (average coverage hint) for each visible object. The ACH of an object represents the average surface area of this object when projected onto the projection plane of any viewpoint within the cell. During rendering, the ACH of each object is used as a percentage of covered pixels to select it's polygon budget in order to match a target frame rate.

2.1. Visibility computation

Our visibility preprocessing consists in finding buildings, roads and crossroads potentially visible from each cell. As the objects are procedural models, their geometry is also generated during the preprocessing. In our algorithm, which is not conservative, we compute a PVS for each corner of the cell and the union of the obtained PVSs gives the PVS of the cell. The PVS of a cell corner is computed in screen space by rendering the scene for six cameras having the same COP (center of projection). The view direction of each camera is perpendicular to one face of an axis aligned box. Such a box will be called rendering box from now on. The COP shared by the six cameras is the center of the rendering box and the FOV (field of view) of each camera is equal to 90 degrees. The projection plane of a camera is a face of the rendering box. The eight rendering boxes of a cell are not exactly centered at the corners. Rather, a box center is placed close to a corner so that the box be inside the borders of the cell that are supported by the frontages of the buildings. In this way, the frontages of the buildings become occluders which reduces the size of the PVSs.

For each camera of each rendering box, all the geometric objects are rendered, which gives 48 images. In order to accelerate the rendering we use an OpenGL graphics hardware and we perform a frustum culling on the bounding box of each object. If the bounding box intersects the frustum, the highest LOD (because it avoids occlusion errors) is rendered for this camera. Each procedural model is loaded using a main root file that refers to it and is displayed with a unique color which is assigned to the memory pointer pointing to it. Consequently, the contents of all the 48 images gives the memory pointers to the objects that makes up the PVS of the cell. Note that one could use the OpenGL occlusion query extension to speed up this process.

In addition, for each camera C_i (of the 48 cameras associated with a cell), we count the number of pixels N_{ij} covered by each visible object I_j. The total number of pixels N_j^{total} covered by a visible object I_j is then:

$$N_j^{total} = \sum_{i \in [1,48]} N_{ij}$$

Let N_{obj} be the number of objects visible for the 48 cameras. The total number of covered pixels N_{pixels}^{total} for the 48 cameras is then:

$$N_{pixels}^{total} = \sum_{j \in [1, N_{obj}]} N_j^{total}$$

The ACH (Average Coverage Hint), denoted ACH_j, associated with each visible object I_j is computed as:

$$ACH_j = \frac{N_j^{total}}{N_{pixels}^{total}}$$

The properties of the ACH values are the following:

$$\begin{cases} \forall j \in [1, N_{obj}], ACH_j \in [0,1] \\ \sum_{j \in [1, N_{obj}]} ACH_j = 1 \end{cases}$$

Although our visibility computation method is not conservative, it provides good results and is fast since it exploits intensively the capabilities of the graphics hardware. Furthermore, if one needs a conservative result, he can first apply a conservative visibility algorithm to compute the PVSs

and then utilize our algorithm, using the resulting PVSs, to compute only the ACHs values.

During the process, the list of potentially visible objects as well as the ACHs values are stored in their respective cells. Each cell is saved into one distinct file and a root file that contains a set of viewpoints at which the user can start navigations is generated. In order to find quickly the cell that contains the current viewpoint, each viewpoint refers to its parent cell. If a viewpoint is used, its parent cell becomes the current cell.

3. Procedural models

The procedural model for generating buildings, that we have implemented for our tests, is coded using our extension [7] of the L-system [9] language that allows to generate VRML97 3D models and scene graph structures. This model takes as input the building footprint, its number of floors and the height of its adjacent building in order to handle party walls. Using these input parameters, it provides the geometric representation of each level of detail. In this model we generate three LODs per building. Figure 3 shows these levels for a given building using a certain set of input parameters. Each level is divided into two parts which are the frontages and the roofs. We use pattern repetition to describe each part. A ground floor is modeled using a row of windows, a door as well as another row of windows. The other floors (including roofs) are modeled using one row of windows. The model uses adjacent building heights to generate party walls. As a party wall does not contain any door or window, we make it start from the adjacent building height to reduce the number of polygons to be rendered. The frontage's wall is the same for level two and level three, so the geometry is shared by the two levels.

4. Rendering algorithm

The goal of the rendering algorithm is to render the scene (on the client side) at an interactive frame rate with a minimum number of visual artifacts. Our system is developed using our client-server framework named Magellan [7]. With this framework, the network accesses are automatically handled and different modules that access a generic scene graph handler can be implemented (on the client side). In this generic scene graph handler the nodes are generic and visual nodes are responsible for their display.

Thus, our rendering algorithm, which is coded in a rendering module, only computes the amount of polygons to be used for the next rendering to maintain a given frame rate (fps) given by the user. This amount of polygons will be called *polygon budget*. Using an average frame rate value estimated from the rendering time of some of the last computed frames and the last polygons budgets used for generating these frames, the algorithm determines the new polygon budget to be used to maintain the frame rate fps given by the user. Once this polygon budget has been computed, the algorithm passes it to the current cell rendering algorithm.

The current cell rendering algorithm makes use of two methods. The *computeVS* method utilizes the polygon budget N^{poly} and the ACHs to make the potentially visible objects of the cell share out the polygon budget. The *display* method simply invokes the display method of the visible objects. The interesting point of the process is the polygon budget sharing which is performed by the *computeVS* method of the cell. In this method, we first performs frustum culling using the bounding boxes of the potentially visible objects of the cell. For each visible object, its ACH is normalized using the number of objects that are visible. We compute the polygon budget N_i^{poly} to assign to a visible object i using its normalized ACH denoted ACH_i as follows: $N_i^{poly} = ACH_i * N^{poly}$. Then, the visible object uses this budget to select its best suitable LOD and returns N_{used}^{poly} which is the exact number of polygons of the selected LOD. The budget of polygons used for the rest of the visible objects is now $N^{poly} - N_{used}^{poly}$ polygons. This process is repeated for each visible object, starting from the object having the highest ACH and ending with the object having the lowest one. Note that this budget assignment method is interesting for any system using level of details.

5. Results

5.1. Database size

In our system, all the files of the databases are encoded using our compressed VRML97 binary format presented in [7]. Although the compressed binary format offers fast parsing possibilities and good compression ratios (around 45% of the size of the UTF8 text files), the main compression is achieved when using procedural model parameters to describe the geometry of the models. To compute the compression ratios, we have compared the size of six different city models when stored in compressed binary format with the size of these same models when their geometry is completely reconstructed using our L-system rewriting system. The results shows that a $122500m^2$ model takes 147KB in compressed format instead of 214MB when geometry is reconstructed and that a $490000m^2$ model takes 542KB instead of 1.1GB. Thus, the size of a database is around 0.07% the size of the reconstructed model. This results confirms the fact that procedural models are of high interest in remote navigation because of their small size which makes them well suited to low bandwidth networks.

5.2. Transmission quality

In order to analyze the transmission quality, we have recorded a walkthrough path (called SE) at a navigation speed of 15km/h. This path passes through the streets of a 700m square model. Then, we have performed two walkthroughs (with and without prefetching) using the path SE and a simulated bandwidth of 56Kb/s. During these walkthroughs, we count, for each new frame, the number of objects that are in the PVS of the current cell. Among these objects, we count those that have not been already downloaded and those whose geometry has not been already generated (rewritten) by a rewriting thread. This allows us to compute the percentage of needed objects that have been downloaded (called downloading quality) and those that have been rewritten (called rewrite quality). If both values are equal to 100%, the transmission quality is perfect for the new frame.

Figure 1 shows how these two percentage values evolve over time and the size of each of the objects downloaded during the walkthrough, when using or not prefetching. When looking at the downloading plot, we can observe higher values at the beginning of the walkthrough because of the texture maps used in our test scene. To overcome this problem, one could use Progressive Textures Maps [6] that fits well with our system. Nevertheless, downloading is low thanks to our procedural models and our compressed binary format. Finally, the downloading and rewriting quality plots show that prefetching allows to obtain a perfect transmission quality (100%) most of the time.

5.3. Interactivity

Recall that our system relies on the frame rate history to compute the *polygon budget* to be used for each new frame construction. To show that our system adapt to different kinds of computer to achieve interactivity, we have simulated a walkthrough using a path (called SM) on two different computers. For each computer, we have performed this walkthrough twice, noting that for the second time all the downloaded data were kept into memory. Figure 1 shows the frame rate and the *polygon budget* over time, measured for a Pentium XEON 1.7Ghz (1GB RAM, Nvidia Quadro2 Pro) and a Pentium III 800Mhz (512MB RAM, Nvidia TNT2). For these two tests, we have used a minimum target frame rate of 25fps and a frame rate history based upon the four last frames. On the frame rate plot, we can see that the target frame rate is always reached with a very small latency. On the *polygon budget* plot, we can see that for a same target frame rate, we obtain a much higher polygon budget with the more powerful computer. Note that the lower the target frame rate, the higher the visual quality (Figure 4).

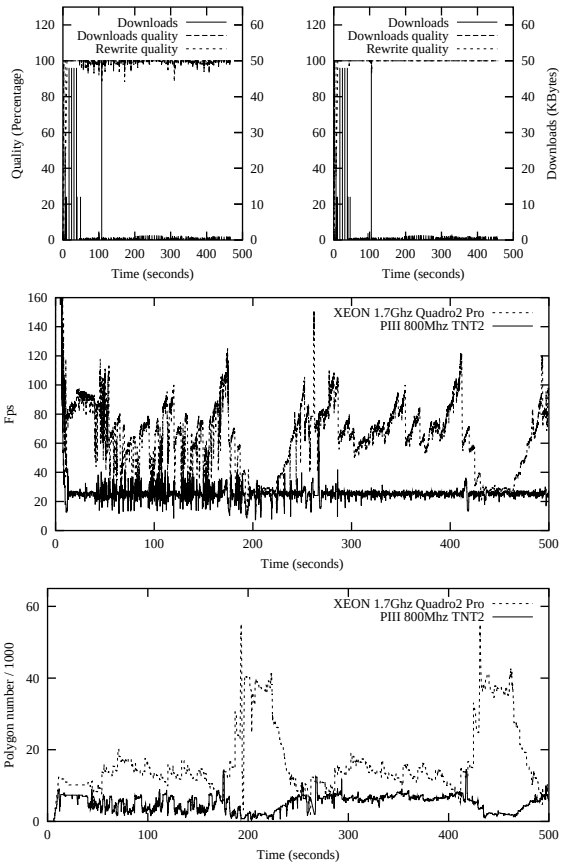

Figure 1. Top left: downloading over time, downloading quality and rewriting quality using path SE, without using prefetching. Bottom right: same, using prefetching. Middle: Frame rate over time for walkthrough SM played twice. Bottom: Polygon budget over time for the same tests.

6. Conclusion

In this paper we have shown that procedural models are of high interest when the aim is networked walkthrough of large scenes such as cities. Indeed, their size is very small, so they can be transmitted quickly over a network. These procedural models where coded with our modified version of the L-system language. The geometry of procedural models is generated on the fly at the client side. Remote real time navigation has been made possible thanks to the visibility preprocessing that have been performed using the graphics hardware and the use of LODs selected using our ACHs. Our ACH-based method is original and more efficient than the commonly used one consisting in comparing with a threshold the distance between the viewer and the objects within the scene. Moreover, this method allows on-line selection of visual quality and interactivity.

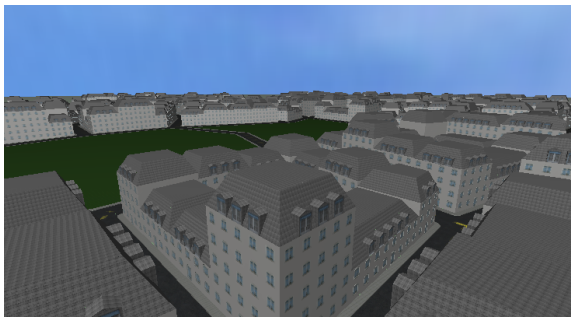

Figure 2. Bird's eye view of a city model.

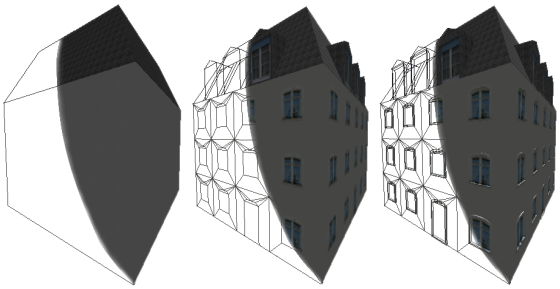

Figure 3. LODs generated for a building.

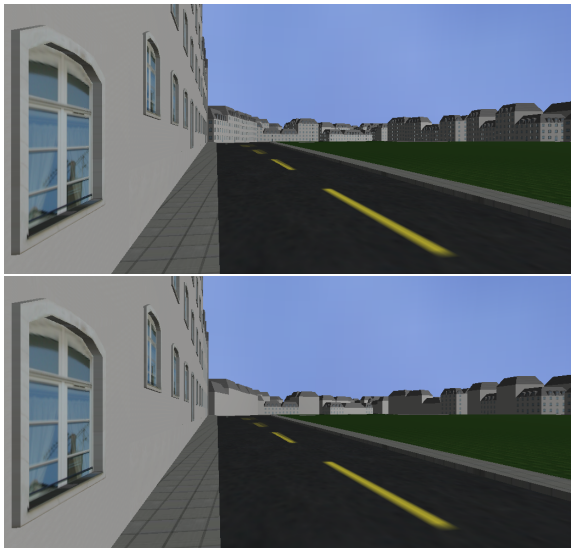

Figure 4. Walkthrough views. Top: target fps set to 25fps, obtained 26.2fps, using 56194 polygons. Bottom: target fps set to 40fps, obtained 41.7fps, using 5703 polygons.

References

[1] J. M. Airey, J. H. Rohlf, and F. P. Brook. Toward image realism with interactive update rates in complex virtual building environements. In *Symposium on interactive 3D graphics*, pages 41–50, 1990.

[2] C. Andujar, C. Saona-Vazquez, I. Navazo, and P. Brunet. Integrating occlusion culling and levels of details through hardly-visible sets. *Computer Graphics Forum*, 19(3), 2000.

[3] D. Cohen-Or, G. Fibish, D. Halperin, and E. Zadichario. Conservative visibility and strong occlusion for viewspace partitioning of densely occluded scenes. In *Computer Graphics Forum*, pages 17(3):243–253, 1998.

[4] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, July 2000. Held in New Orleans, Louisiana.

[5] T. Funkhouser. Database management for interactive display of large architectural models. In *Proceedings of Graphics Interface*, pages 1–8, May 1996.

[6] J. E. Marvie and K. Bouatouch. Remote rendering of massively textured 3D scenes through progressive texture maps. In *The 3rd IASTED conference on Visualisation, Imaging and Image Processing*, volume 2, pages ?–?, Sept 2003.

[7] J. E. Marvie, J. Perret, and K. Bouatouch. Remote interactive walkthrough of city models using procedural geometry. Technical Report PI-1546, IRISA, July 2003. http://www.irisa.fr/bibli/publi/pi/2003/1546/1546.html.

[8] D. Meneveaux, E. Maisel, and K. Bouatouch. A new partitioning method for architectural environments. *Journal of Visualization and Computer Animation*, May 1997.

[9] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[10] C. Saona-Vasquez, I. Navazo, and P. Brunet. The visibility octree. a data structure for 3d navigation. Technical report, Universitat Politechnica de Catalunya, Spain, 1999.

[11] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative volumetric visibility with occluder fusion. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 229–238. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[12] D. Schmalstieg and M. Gervautz. Demand-driven geometry transmission for distributed virtual environments. *Computer Graphics Forum*, 15(3):C421–C432, Sept. 1996.

[13] D. Schmalstieg and M. Gervautz. Modeling and rendering of outdoor scenes for distributed virtual environments. In D. Thalmann, editor, *ACM Symposium on Virtual Reality Software and Technology*, New York, NY, 1997. ACM Press.

[14] B. Schneider and I. Martin. And adaptive framework for 3d graphics over networks. *Computer and Graphics*, 23:867–874, 1999.

[15] E. Teler and D. Lischinski. Streaming of complex 3D scenes for remote walkthroughs. In *Computer Graphics Forum*, volume 20(3), pages 17–25, 2001.

[16] S. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, pages 61–69, 1991.

[17] P. Wonka, M. Wimmer, and D. Schmalstieg. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Eurographics Workshop on Rendering*, pages 71–82, June 2000.