PBTree – A new progressive and hierarchical representation for network-based navigation in urban environments

J. Royan*, C. Bouville†, P. Gioia

France Telecom R&D, Rennes, France

Abstract

This paper describes a novel progressive and hierarchical representation for densely built urban areas. It aims at navigating through huge urban environments with network-based client-sever systems. Thanks to this method, navigation is no longer limited to walkthrough, but allows to fly over the city. This can be achieved through a set of dedicated algorithms allowing the decomposition of city into a multi-resolution representation. The method efficiently exploits the fact that automated modeling technologies of urban scene are generally based on the processing of 2D data (video, photographs, cadastral map, ...), to model $2D\frac{1}{2}$ data (building footprint, height, altitude, ...) which turns out to be more compact. From this pre-computed representation of buildings, a server can progressively send the perceptible details to the client for all the regions visible from a given viewpoint.

1 Introduction

Many techniques are now available to automatically generate huge urban environments from the acquisition of $2D$ data (video, photographs, cadastral maps, etc). For example, photogrametry-based approaches allow to determine the representation of the city from aerial photographs or videos[5]. In the case of urban area, these technologies can produce very large datasets consisting of building footprints with associated altitudes and heights. Other acquisition methods using laser scanner produce $3D$ city models with much more details. Unfortunately, they are much too complex and ill-organized for interactive visualizations. Even with the $2D\frac{1}{2}$ representation mentioned above, it is impossible to quickly send to the client the whole database and then visualize the entire model in real-time given the size of datasets (more than 100 000 buildings for a city like Paris). Therefore, the ”download and play” paradigm used in current VRML-based applications is unrealistic for complex urban models.

Although this problem has been widely studied for the particular case of manifolds[6] [3], the case of large urban environments modeled by sets of building models are of a completely different nature. The somewhat irregular nature of such geometry are closer to so called *polygon soups* than regular topological meshes. Indeed, these scenes are composed of millions of buildings but each of them has very few polygons and little geometric redundancy, which is not suitable to mesh decimation. Different methods have been developed aiming at interactive urban walkthroughs with basic home computer configurations. However, these methods have a number of restrictions in terms of scene complexity and visualisation. Moreover, such methods are not suited to networked-based navigation. It is commonly admitted that an unrestricted low bandwidth network-based visualisation system must have the following features :

- **Compression:** the $3D$ model of Paris is very bulky (more than 8Go), which precludes transmission across the network in a reasonable time. Dedicated geometry and texture compression techniques are of absolute need.
- **Progressiveness:** it's essential that the client has almost immediate access to navigation just after its connection to the server, whatever the coarseness of the model at this first stage. As data are received from the server, the city model is progressively refined by adding the appropriate details. To preserve coding efficiency, such progressive representation must not be redundant in terms of geometry and texture information. (incremental process).
- **Adaptability:** the refinement of the scene must be carried out in depth order i.e. from

* e-mail: jerome.royan@rd.francetelecom.fr
† e-mail: christian.bouville@rd.francetelecom.fr

near to far regions, and must be invertible so as to avoid drift when the viewpoint is changed. Furthermore, this refinement data stream should be adapted to the available network bandwidth.

Moreover, flying over virtual cities is also essential for many applications such as flight simulators or geopositioning. This issue has been scarcely addressed in the literature and we shall see below that it is harder to solve than walkthrough.

The goal of this paper is to propose a new approach that fulfill the aforementioned requirements. It includes the following contributions:

- a new fully scalable representation for network-based navigation through or over virtual cities
- a set of dedicated algorithms allowing to compute this progressive representation from the classical $2D\frac{1}{2}$ representation provided by most automated city modeling methods,

The remainder of the paper is organized as follows: after reviewing some background works on the visualization of huge $3D$ environments, and looking at the potential adaptation to a network-based system that allows flying-over navigation, Section 3 gives the specifications of a progressive and hierarchical representation of densely built areas. In Section 4, we present a method that allows unsupervised generation of such representations from $2D\frac{1}{2}$ city models. Then, we comment our results in Section 5, and finally, we conclude with some future work directions.

2 Related work

In this section we present the background works on large scenes visualization. We focus on the case of urban scenes, analysing how they fulfill the requirements mentioned above.

Impostors: In order to limit the number of polygons to be processed by the graphic hardware, a well-known solution consists in using a hybrid approach that combines geometric and image-based rendering often called *impostor*[8]. Solutions consisting in adding depth information to impostors, such as multi-layered impostors[1] and point-based impostors[11], were proposed to limit the artifacts cropping up due to parallax problems. However, the impostor representation has the drawback of requiring considerable fast access storage resource, which is not workable on a network-based system given the cost of image-based representations in terms of network resource.

Point-based representation: In order to process only points instead of triangles, point-based rendering uses sets of points without explicit connectivity. Those sets of points, known as *surfels*, have attributes such as depth, texture color, normal, and others[9]. However, point-based rendering is well-suited to complex objects that are viewed from a great distance compared to the object size. This is not the case of most buildings representation where details are represented by textures mapped on a raw geometric model. Therefore, point-based representations are quite inapropriate in terms of network resource when walkthrough navigation through urban areas is to be considered.

Conservative visibility: Visibility culling is another technique used to reduce the geometry to be rendered by the graphics hardware. In the case of urban walktroughs, most of the far geometry is occluded by the near buildings. Some methods allow to compute the potential visible set of objects (PVS) in real-time for a given viewpoint, possibly with rendering hardware support[12]. However, in a centralized networked application, only the server can compute PVSs since it contains the whole database, which causes processing bottlenecks in case of many client connections. In this case, pre-computation of PVS associated to regions (called view cells) is a better solution[10] [13]. In a networked application, the client sends its current viewpoint location to the sever, which determines the corresponding view cell and forwards the PVS to the client. This method is efficient for walkthrough navigation but becomes inapropriate for fly over navigation since occlusion complexity is much lower in this case.

Levels of detail: The basic idea of LOD is to adapt 3D object complexity to the size of its projected image on the screen, thus eliminating barely perceptible details. As the viewpoint is changed, different representations of the same object (*Levels Of Detail*) are used. The case of mesh object LODs has been extensively studied in the literarure[3], and a progressive and view dependent representation can be obtain using a tree data structure containing the successive atomic operations of simplification[6] [7]. However, most city buildings cannot be regarded as meshes but rather more as prisms made up of a few polygons. Therefore, it wiser to apply simplifications to each building separately, rather than to groups of buildings so as to allow efficient detail culling when buildings are seen at a very far distance.

3 Data Structure

Before describing the progressive and hierarchical building representation, we begin with a description of the input data provided by the automated city modeling process.

3.1 $2D\frac{1}{2}$ Input Data

Given the size of the cities to be modeled, automated methods are often used for cost reasons although the resulting model accuracy is affected, especially as far as facade texture mapping is concerned. A great majority of these methods make use $2D$ data (aerial videos and snapshots, cadastral map, ...), in order to produce a $2D\frac{1}{2}$ representation (using image segmentation to determine the building footprints, and photogrammetry methods to retrieve building altitudes and heights). The compactness of this $2D\frac{1}{2}$ representation makes it highly suitable to network-based viusalization (only the $2D$ vertices of the building footprints as well as some procedural parameters). A ratio of 1:10 compared to the $3D$ representation can be reached as regards the representation data size. Still, this coding efficiency is obtained at the expense of additional client side processing to rebuild the full 3D models.

3.2 Progressive and hierarchical $2D\frac{1}{2}$ Representation

After a context presentation, we describe in this section the progressive and hierarchical building representation for urban navigation over network that respects the different characteristics presented in introduction : progressiveness, compression, adaptability, and flying-over. This representation can be used by the server to send the refinement data to the client.

3.2.1 Context

Let S be the set of vertices in $\mathbf{R}^2$ of the scene represented in $2D\frac{1}{2}$, and let $P \in S^n$ be the set of polygons that can be created from set S. A building is expressed as a triple $b = (\ F, h, a)$ where F specifies its building footprint in P, h its height, and a its altitude. Let $\widehat{B}$ in b^k denote the set containing exactly the k initial buildings of the scene. The initial set of buildings $\widehat{B} = B^n$ can be simplified into a coarser set of buildings by applying a sequence of n successive simplification transforms:

$$(\widehat{B} = B^n) \xrightarrow{\sigma_{n-1}} \dots \xrightarrow{\sigma_1} B^1 \xrightarrow{\sigma_0} B^0 \quad (1)$$

Since each simplification transform σ is invertible, the building set of the scene can be represented as a simple building B^0 together with a sequence of n σ^{-1} records:

$$B^0 \xrightarrow{\sigma_0^{-1}} B^1 \xrightarrow{\sigma_1^{-1}} \dots \xrightarrow{\sigma_{n-1}^{-1}} (B^n = \widehat{B}) \quad (2)$$

Each refinement transform σ^{-1} introduces addtional vertices as well as some building footprint modifications (cf. Fig 1). The tuple $(B^0, \{\sigma_0^{-1}, \dots, \sigma_{n-1}^{-1}\})$ forms a progressive and hierarchical representation of $\widehat{B}$. The tuple $(\{b_i, \dots, b_{i+m}\}, b_j)$ defines a simplification transform σ_{n-1} where $\{b_i, \dots, b_{i+m}\}$ represents the buildings in B^n, and b_j the corresponding simplified building in B^{n-1}. Conversely, the tuple $(b_j, \{b_i, \dots, b_{i+m}\})$ represents a refinement transform σ_n^{-1}.
The resulting sequence of building set $B^0, \dots, B^n$ is particularly adapted to view-dependent selection of levels of detail. In addition, geomorphs can be used in order to visually obtain smooth transition during the swapping between different levels of detail.

3.2.2 Progressive and hierarchical representation

The progressive and hierarchical representation is based on a tree data structure that holds the merging and simplifications of buildings: the *PBTree* or *Progressive Building Tree* (cf. Fig. 1). Each node of this tree, called *Building Node*, stores the $2D\frac{1}{2}$ representation of a building at a certain level of detail. Moreover, a vertex array is added to this tree defining the building footprints associated to Building Nodes. Each building node has to contain a $2D\frac{1}{2}$ representation allowing the $3D$ reconstruction of the associated building, and is composed of the following elements:

- a tree node structure,

- a building footprint (a polygon indexed to the vertex array),
- a height,
- an altitude,
- a roof model,
- any parameters allowing a more complex procedural $3D$ modelisation of the building.

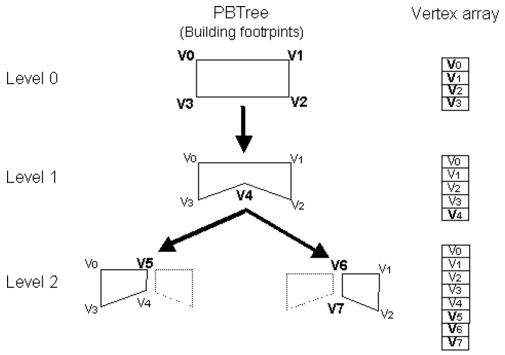

Figure 1: $2D\frac{1}{2}$ progressive and hierarchical representation using a PBTree. In bold, the new vertices required for expanding a Building Node.

This representation allows a progressive and view dependent transmission of buildings to the client. Indeed, according to different criteria, such as the distance from the viewpoint to a building represented by a building node bn, the server can decide to send the necessary information to expand bn, and so render a more refined building on the client.

One more interest of PBTree lies in the storage of only few redundant data. In fact, to expand a building node, it is not necessary to transmit all vertices that define the building footprints of its children, but only the vertices that are not still present in the vertex array (cf. Fig. 1).

Moreover, this representation respects the constraint of locality, as the client may not have the entire progressive representation in memory, but only a part of the PBTree. Of course, the server knows the whole PBTree, and its leaves correspond to the initial buildings in the scene.

This tree representation introduces a new constraint on the PBTree computation process that we call the *circuitless condition*: a node has a single parent, each executed simplification must generate one and only one building node (cf particular case (*g*) of Fig 3). A hierarchy satisfies the *circuitless condition* if and only if

$$\forall k, (a,b) \in (B^{k-1})^2 \Rightarrow a \in B^k \vee b \in B^k \quad (3)$$

This condition has to be checked during the whole simplification computation process.

4 PBTree Computation

Now that the input and the desired output representation have been presented, we are attached first to describe the possible simplifications that can be applied to the $2D\frac{1}{2}$ representation for levels of detail generation. Then, a description of the PBTree precomputation using these possible simplifications is detailed.

4.1 Simplification Transforms

The goal of simplifications is to reduce the city geometric complexity, by simplifying the building footprints and reducing their number. For this, we use the three following transformations:

Merging two consecutive facades of a building: this is done by removing a vertex of a building footprint. This removal should fulfill the following conditions so as to comply with some elementary topological constraints:

- only one vertex can be removed at each facade collapse. Moreover, this vertex can't be share by another building footprint (to fulfill the *circuitless condition*).
- the facade merging cannot generate an intersection between the new simplified building and its neighborhood.
- the simplified building footprint area has to be the nearest one to the non simplified building area.

Merging two buildings connected by at least one facade: this is done by merging the building footprints of the two concerned buildings. The height of the resulting building is the average of the heights of the two buildings weighted by the area of their footprints. The resulting altitude is the minimal altitude of the two buildings, ensuring that the simplified building still lies on the ground:

$$\forall \sigma(b_k, \{b_i, b_j\}), b_k \in B^{n-1}, b_i \in B^n, b_j \in B^n$$

$$\begin{cases} a_k = MIN(a_i, a_j) \\ h_k = \frac{h_i * \mathcal{A}(f_i) + h_j * \mathcal{A}(f_j)}{\mathcal{A}(f_i) + \mathcal{A}(f_j)} \\ ti_k = \begin{cases} ti_i \\ \quad \text{if } \mathcal{P}(f_i) * h_i > \mathcal{P}(f_j) * h_j \\ ti_j \\ \quad \text{otherwise} \end{cases} \end{cases} \quad (4)$$

Where a_k is the altitude of b_k, h_k its height, f_k its building footprint, ti_k its facade texture index, $\mathcal{A}(f_k)$ and $\mathcal{P}(f_k)$ the area and perimeter of its building footprint.

Merging two buildings not connected by a facade: this simplification is less implicit than the two previous ones. To determine the potential merging of non connected buildings, a $2D$ triangulation of the free space between the building footprints is applied. As the vertices of the triangulation have to belong to set V (the vertices defining the initial building footprints), this constrained triangulation does not add any *Steiner* points (i.e. this is not a Delaunay triangulation, cf. Fig. 2). Therefore, a mesh defined with the building footprints and the triangles output from this triangulation of the free space is obtained, which provides a complete topology of the scene.

Consequently, this triangulation is used to merge two non connected buildings. Actually, the two building footprints will be merged with the triangles (cf. Fig. 3 (c),(e) and (f)), when the following conditions are met:

- there must be at least one edge shared with one of the footprints,
- there must be at least one vertex shared with the other footprint

The new altitude and height of the generated building are derived according to Equation 4. Other methods can be used to merge two non connected buildings, but they have to fulfill the constraint that they must not generate new vertices. Figure 3 shows

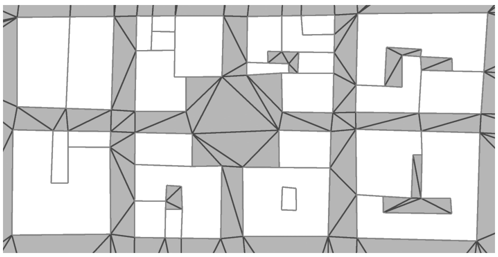

Figure 2: Triangulation constrained by the building footprints without Steiner points addition giving us a complete topology of the scene.

various possible simplification cases. The use of the free space triangulation allows to solve the particular cases *(e)* and *(f)*, but has the drawback of not taking into account all the possible simplifications, such as case *(h)*. Note that this outstanding case does not stop the simplification process, but does not allow to merge in priority the nearest buildings. Similarly as in case *g*, that does not follow the *circuitless condition*.

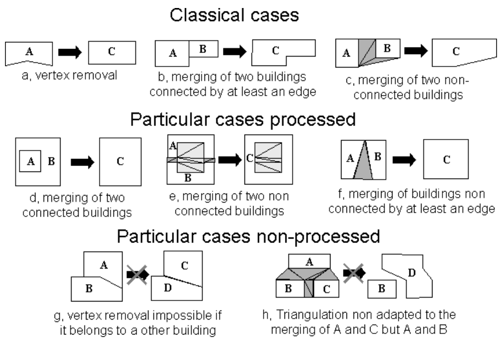
Classical cases
A
C
a, vertex removal
A
B
C
b, merging of two buildings connected by at least an edge
A
B
C
c, merging of two non-connected buildings
Particular cases processed
A
B
C
d, merging of two connected buildings
A
B
C
e, merging of two non connected buildings
A
B
C
f, merging of buildings non connected by at least an edge
Particular cases non-processed
A
B
C
D
g, vertex removal impossible if it belongs to a other building
A
B
C
B
D
h, Triangulation non adapted to the merging of A and C but A and B

Figure 3: Simplification examples.

4.2 Simplification algorithm

This simplification algorithm takes as input the set of initial buildings represented in $2D\frac{1}{2}$ to generate the corresponding PBTree. The algorithm is initialized by assigning the input buildings to each leaf of the PBTree, and by searching for all potential simplifications of the scene. But, to determine which potential simplification has to be applied in priority, a cost function is assigned to each potential simplification. This function must take into account the following criteria :

- The height difference between the potentially merged buildings ($diff\_height$),
- The difference of altitude between the potentially merged buildings ($diff\_alt$),
- The additional area generated by the potential merging ($area$), since the larger is this area, the more visible will be this merging during the visualization, resulting in popping effects.
- The minimal distance between the potentially merged buildings (min_dist), allowing to preserve the streets as long as possible during the simplification process.

During the process, the simplification data are stored in a list ordered by the cost function: the *PSL* or *Potential Simplification List*. Thus, one applies in priority the potential simplification that has the lowest cost, knowing that the cost function has a minimum when $diff\_height$, $diff\_alt$, $area$, and min_dist are minimal.

After testing different cost functions, the following one has proven to be particularly well-suited to the building case:

$$cost(diff\_height, diff\_altitude, dist\_min, area) = area.\exp(\alpha.diff\_height + \beta.diff\_alt + \gamma.dist\_min) \quad (5)$$

where α, β, γ are weighting factors.

Figure 4 describes the simplification algorithm that allows to generate the PBTree used to transmit progressively the virtual city geometry on the network. Figure 5 shows the data during the simplification process. At each iteration, the first simplification of the PSL is applied (adding a node in the PBTree), followed with the update of the PSL. Actually, applying a simplification may affect the free space triangulation, as well as many other potential simplifications contained in the PSL. The algorithm ends when the PSL is empty, and at that time, the PBTree contains the whole progressive and hierarchical representation used for the network-based city navigation.

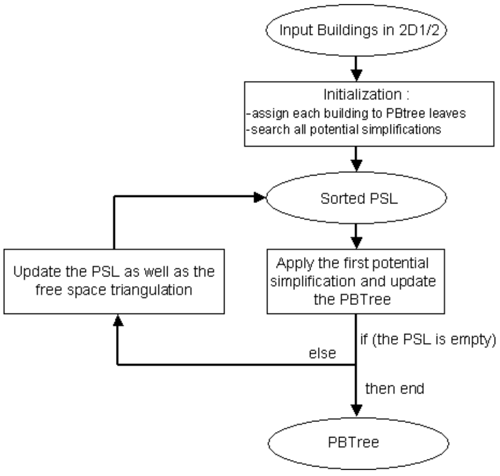
Input Buildings in 2D1/2
Initialization :
-assign each building to PBtree leaves
-search all potential simplifications
Sorted PSL
Update the PSL as well as the free space triangulation
Apply the first potential simplification and update the PBTree
if (the PSL is empty)
else
then end
PBTree

Figure 4: Simplification algorithm to generate the PBTree.

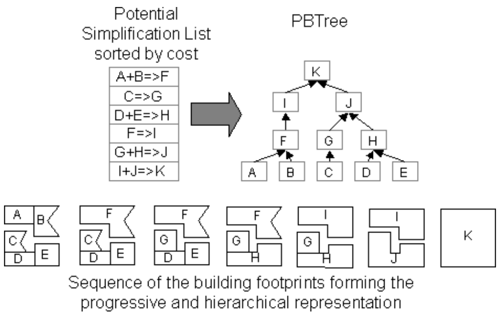
Potential Simplification List sorted by cost
A+B=>F
C=>G
D+E=>H
F=>I
G+H=>J
I+J=>K
PBTree
K
I
J
F
G
H
A
B
C
D
E
Sequence of the building footprints forming the progressive and hierarchical representation

Figure 5: Principle of the simplification algorithm.

5 Experimental Results

5.1 Computation time and storage

The PBTree has been computed for three cities. The input $2D\frac{1}{2}$ representation format is the *Shape* format from ESRI. This format allows associating various parameters (height, altitude, facade texture index, ...) to different types of primitives such as polygons (the building footprints). Table 1 presents the number of buildings and facades, the PBTree precomputation time and the visualization frame rate for three different cities.

Computation time can be long for huge cities,

	# buildings	# facades	computation time	fps
Marseille	1360	8362	28s	32
Nice	15603	97742	7mn 52s	27
Rennes	35203	280267	2h18mn27s	23

Table 1: Number of buildings, facades, PBTree precomputation time, and frame rate for three different cities.

due to huge data structures updates. Nevertheless this is acceptable for a pre-processing stage, and the building tree can be easily stored in an appropriate compressed format (using local references to store the vertex coordinates). In Table 2 , the size of the generated files representing the PBTree are compared to the *VRML* format, whether they are compressed or not. *VRML* is not the best format as regards compression, but it is the only available lossless representation in our case, as building representations cannot take advantage of existing compression methods applied on meshes. These results show the great efficiency of this representation in compression, that makes it prone to be used for network-based navigation in huge virtual urban environments.

	No compression				
	$3D : VRML$		$2D\frac{1}{2}$		
	Normal	Hierar.	Shape	Hierar.	Rates
Marseille	1,86	7,28	0,635	0,283	0,039
Nice	21,7	110	5,64	4,08	0,037
Rennes	57,8	351	12,8	12,8	0,036

	Gzip compression				
	$3D : VRML$		$2D\frac{1}{2}$		
	Normal	Hierar.	Shape	Hierar.	Rates
Marseille	0,205	0,626	0,143	0,089	0,142
Nice	2,34	8,78	0,89	1,28	0,146
Rennes	6,28	26,34	2,17	3,02	0,115

Table 2: Size of files in Mb for the selected cities with various representations: VRML, VRML with static LOD, Shape, and our progressive and hierarchical representation. Last column presents the rate between the two hierarchical representations.

5.2 Visualization

We have developed a visualization system demonstrating the results obtained with the PBTree. This application allows to navigate freely and fly over the city, using a simple $3D$ building reconstruction (footprint extrusions with texture mapping). However, a more complex building model can be compute, using procedural methods, as facade patterns or roof reconstruction using the straight skeleton of footprints [2]. Figure 7 shows various screenshots of the application taken during the navigation. Note on the vectorized views that the city reconstruction is view-dependent. Figure 6 shows the evolution of the city model refinement, highlighting the good behaviour as regards the street structure. As shown in Table 1, this new progressive and hierarchical building representation allows a smooth navigation over huge urban environments, even with an important depth field. Due to the local nature of the reconstruction, it can be assumed that there is no size limits for a city, except the storage size of the PBTree on the server.

6 Future Extensions

At present, a standalone visualization system has been implemented. In this system, all textures are stored in memory. However, on a future client-server system, we plan to use *JPEG2000* as a solution to solve the scalability problem posed by large terrain textures. As the facade textures can be considered as a wall texture combined with windows and doors elements, procedural methods can be used to efficiently encode these textures.
Another research direction to be explored is the procedural $3D$ reconstruction on the client-side. For the moment, we are working on the roof procedural modeling. Procedural reconstruction of complex facades will also be studied, so as to be able to model buildings with more details.
Finally, as this representation is efficient for networked applications, it will be integrated on the existing platform for landscape network-based navigation using a $3D$ wavelet compression: *Wavier*[4]. Furthermore, we are working on the conservative visibility pre-computation, to be able to minimize the geometry to be transmitted to the client during a walkthrough navigation. Our objective is to implement a complete client-server system integrating the previously mentioned techniques along with dedicated protocols and cache managers that will allow to navigate over realistic urban scenes using different terminal from the personal computer to the mobile phone.

7 Conclusions

We have proposed a new progressive and hierarchical building representation, allowing smooth city walkthrough or flying-over navigation on a client-server system. It consists first in sending a $2D\frac{1}{2}$ building representation to the client, that allows fast transmission. The second idea is to generate a progressive and hierarchical representation organized by a tree structure called PBTree that describes the model simplifications. Experimental results obtained with this representation demonstrate its great efficiency, and show the advantages featured by this approach:

- flying over cities is allowed, with an important depth of field,
- the city geometry transmission is progressive and hierarchical, allowing full scalability and

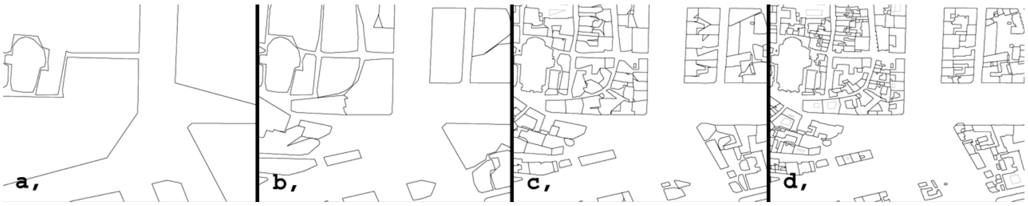

Figure 6: Refinement model evolution showing the progressive transmission of the city.

view-dependent reconstruction. Therefore, the navigation within and over densely urban environments can begin just after the connection to the server,

- the transmitted data can be easily compressed, using a local reference system for the vertex coordinates. Experimental results show strong compression ratio, that makes this new representation well adapted to network-based applications,
- the complexity of the $3D$ reconstruction processed on the client is not reduced to a block, but can be enhanced using procedural parameters to generate roofs or complex facades.

Acknowledgements

We would like to thank Archivideo who provided us with the three $2D\frac{1}{2}$ city models that allowed us to experiment the building tree computation on real cities.

References

[1] X. Decoret, G. Schaufler, F. Sillon and J. Dorsey Multi-layered impostors for accelerated rendering In *Eurographics'99*, volume 18, 1999

[2] P. Felkel and S. Obdrzalek Straight skeleton implementation In *14th Spring Conf. Computer Graphics*, pages 210-218, 1998

[3] M. Garland and P. Heckbert Simplifying Surfaces with Color and Texture using Quadric Error Metrics In *IEEE Visualization 98 Conference Proceedings*, 1998

[4] P. Gioia, O. Aubault, C. Bouville Real-time Reconstruction of Wavelet Encoded Meshes for View-dependent Transmission and Visualization In *ICIP 2002 Conference Proceedings*, Rochester, NY, 2002

[5] A. Gruen, E. Baltsavias, and O. Henricsson Automatic Extraction of Man-Made Objects from Aerial and Space Images(II) In *Proceedings of the Monte Verita Workshop*, 1997

[6] H. Hoppe Efficient Implementation of Progressive Meshes In *Computer and Graphics*, volume 22, pages 27-36, 1998

[7] D. Luebke, and C. Erikson View-Dependent Simplification of Arbitrary Polygonal Environnements In *Computer Graphics SIGGRAPH 97*, volume 31, pages 199-208, New-York, 1997, ACM Press

[8] Paulo W. C. Maciel and Peter Shirley Visual Navigation of Large Environments Using Textured Clusters In *1995 Symposium on Interactive 3D Graphics*, pages 95-102, August 1995, ACM Press

[9] H. Pfister, M. Zwicker, J. Van Baar and M. Gross Surfels: Surface Elements as Rendering Prinitives In *Computer Graphics, SIGGRAPH 2000 Proceeding*, pages 343-352, Los Angeles, 2000, ACM Press

[10] Seth J. Teller and Carlo H. Sequin Visibility Preprocessing For Interactive Walkthroughs In *Computer Graphcis (proceedings of SIGGRAPH 91)*, 25(4):61-69, 1991, ACM Press

[11] M. Wimmer, P. Wonka and François Sillion Point-Based Impostors for Real-Time Visualization In *EuroGraphics Workshop on Rendering*, 2001

[12] P. Wonka, M. Wimmer, Michael and F.X. Sillion Instant Visibility In *Eurographics 2001*, volume 20, 2001

[13] P. Wonka, M. Wimmer, D. Schmalstieg Visibility Preprocessing with Occluder fusion for Urban Walkthroughs In *Eurographics Workshop on Rendering*, 2000

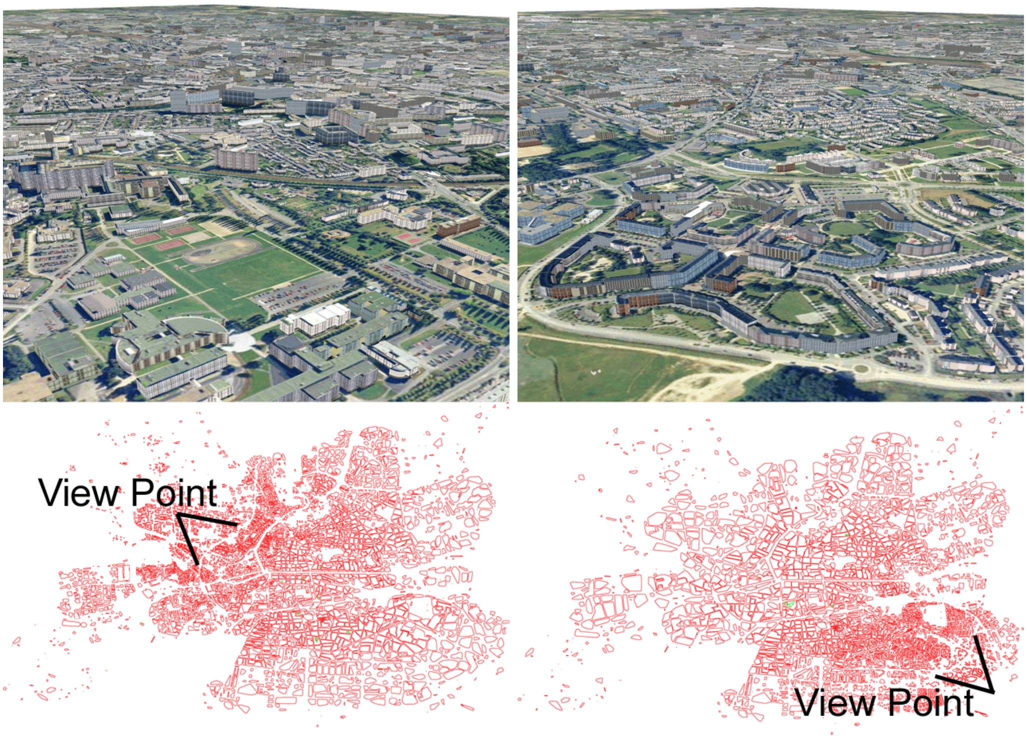

Figure 7: Screenshots of the visualization system (city of Rennes, 35000 buildings), with the $3D$ and the $2D$ respective view of the building footprints showing the view-dependent refinement.

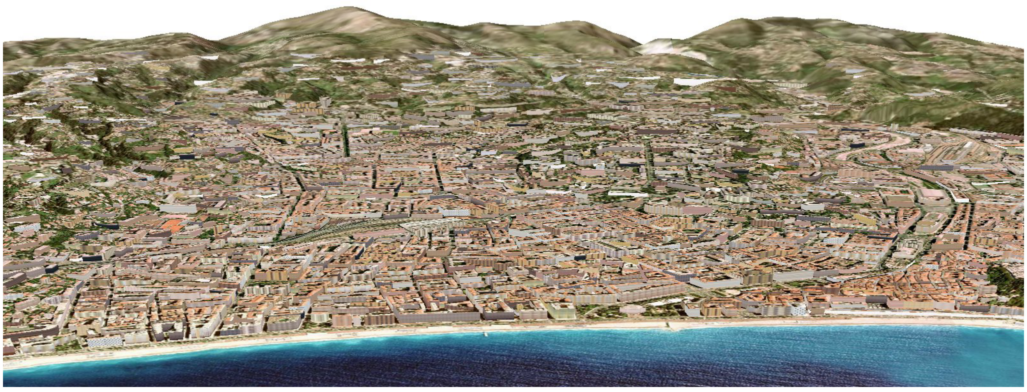

Figure 8: Screenshot of the visualization system (city of Nice, 15000 buildings).