Using Expressive Rendering for Remote Visualization of Large City Models

Jean-Charles Quillet, Gwenola Thomas, Xavier Granier, Pascal Guitton, Jean-Eudes Marvie

Using Expressive Rendering for Remote Visualization of Large City Models

Jean-Charles Quillet* Gwenola Thomas* Xavier Granier* Pascal Guitton* Jean-Eudes Marvie†
IPARLA Project (LaBRI - INRIA futurs) THOMSON Corporate Research

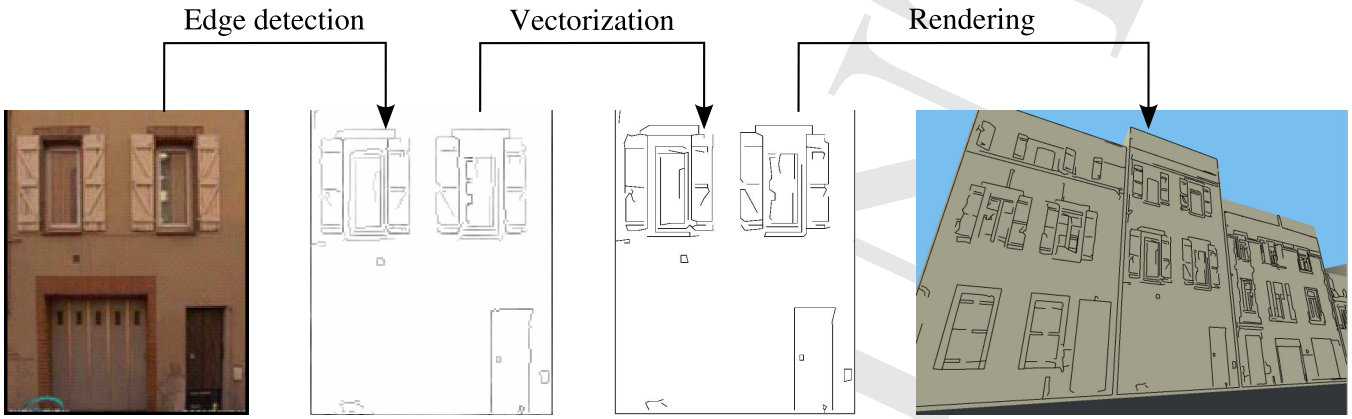


Figure 1: Production pipeline of stroke-based buildings. The edges of facade images are extracted and vectorized. This vector data is used instead of textures to represent the facade in the final 3D model.

Abstract

In this paper, we present a new approach for remote visualization of large 3D cities. Our approach is based on expressive rendering (also known as Non-Photorealistic Rendering), and more precisely, on feature lines. By focusing on characteristic features, this solution brings a more legible visualization and reduces the amount of data transmitted on the network. We also introduce a client-server system for remote rendering, as well as the involved pre-processing stage that is required for optimization. Based on the presented system, we perform a study on the usability of such an approach in the context of mobile devices.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; C.2.4 [Computer-Communication Networks]: Distributed Systems—Client/server; I.4.6 [Image Processing and Computer Vision]: Segmentation—Edge and feature detection.

Keywords: Non-photorealistic rendering, line-based rendering, mobile and connected devices, client-server architecture, streaming

1 Introduction

The aim of the majority of previous work in remote visualization is concerned with improving realistic rendering using complex 3D models, complex lighting systems, progressive texture transmission, and so on. The resulting choices are visible in the definition of X3D[1] and VRML2[2] format. However, full realism will always be very resource-demanding, due to the natural complexity of real world.

On the other hand, for a number of applications, such as trip or assisted navigation, the solutions still mostly rely on pure 2D information like maps[3]. These can even be available on cell phones or PDA's, and, more generally, on mobile devices. Combined with GPS, they can assist the user during a trip.

We believe that there is a middle ground for remote visualization of 3D content such as complex urban environments, navigation, illustration and education such as for Cultural Heritage. In this paper, we propose a new approach for such a context with a special focus on cities.

In our remote visualization context, the server disposes of the entire geometry and sends a subset of the model to the client on demand. Usually, the buildings of a virtual city are simple textured blocks, which are constructed from extrusions of their footprints. The textures are photographs of real building facades. With high quality photographs, the visual quality of the resulting cities is satisfactory. However, the transmission of high-resolution textures can be prohibitive due to the network bandwidth. Furthermore, overly detailed textures can hide the important features that are be transmitted. The content has to be adapted to what the user can perceive, and, in the context of mobile devices, it has to be adapted to the reduced size of the display of mobile devices.

Thus, we base our approach on expressive or non-photorealistic rendering (NPR), and more precisely, on Feature Lines (e.g., for 3D models [DeCarlo et al. 2003]). By focusing on characteristic features of the content, feature lines can create more legible images than realistic rendering: focusing on the content to be displayed, frees the screen from visual data that is not required. The 3D con-

*e-mail: {quillet,gthomas,granier,guitton}@labri.fr
†e-mail: jean-eudes.marvie@thomson.net

[1] http://www.web3d.org/x3d/
[2] http://www.web3d.org/x3d/specifications/vrml/
[3] http://www.mappy.com, http://mappoint.msn.com

tent can be augmented with information, such as text (for example, for a city walk-through, streets with their name and their direction). Also, since only the features are displayed, this approach can reduce the amount of data that has to be transferred in a client-server architecture. By storing the feature lines in a vectorial form, the size of a building representation can decrease significantly.

In this paper, we present a pipeline that is able to extract the feature lines from photographs of building facades, to transmit and to render them as full 3D models on remote clients. This pipeline has been implemented in a prototype system that allows us to perform an experimental study on the usability of such an approach, in terms of rendering performance, data organization and transmission, and required network bandwidth.

2 Previous work

Our application is specific to navigation in large virtual cities in a client-server configuration. We state that the use of feature lines for the rendering will improve performances. Most of our work relies on feature line extraction and vectorization, and on the level of detail for the extracted lines. In this section, we present some of the existing previous work.

2.1 NPR for remote rendering

Our use of NPR for remote rendering is motivated by the fact that the transmitted data can be smaller than its photorealistic equivalent. In the case of small displays, the abstraction offered by NPR is also invoked for more efficient visualization [Hekmatzada et al. 2002; Diepstraten et al. 2004].

We are closely related to these solution since we are transmitting feature lines from remote rendering. However, in Hekmatzada et al. [2002] and Diepstraten et al. [2004], the lines are extracted on the server side and are essentially silhouettes; those lines are view-dependent. The consequence is a strong dependence between the client and the server because new data must be sent for each new viewpoint. In the context of virtual cities, the features lines, which are related to facades, are view-independent. Indeed, the potential visible lines that are transmitted are valid for a large range of viewpoints. There is, therefore, less communication between the server and the client. Moreover, because the data we transmit is vectorial, we can perform progressive transmission based on levels of details. In [Hekmatzada et al. 2002], they also render a town using sharp edges of the geometric model. In that instance, the details of the facade (windows, doors, materials) were not represented.

2.2 Feature extraction and vectorization

Most of the techniques for edge detection are based on a convolution kernel in order to locally estimate the gradient magnitude. The gradient represents the intensity variation within the image, which conveys the characteristic features. Following this convolution, the contours are recovered by applying a threshold. These filters work well for general purpose, but nevertheless, they are very sensitive to noise. Moreover, resulting contours are usually regions of contours, that is, their width is larger than one pixel. This is generally a problem for an approximation with poly-lines. The most popular edge detectors are Sobel [Pingle 1969], Roberts [1965] and Prewitt [1970]. Each detector uses a slightly different convolution-kernel. The Canny edge detector [1986] detects one pixel width contours; our solution is based on it.

Once the edges are detected, they have to be approximated by curves or poly-lines for the final vectorization. As stated in [Tombre et al. 2000], many algorithms have been developed and provide case-specific solutions. This vectorization is generally done in two steps. First, the pixels are chained. This process is not well detailed in existing publications (see the extensive survey in [Tombre et al. 2000]), except for the work of Chakrabarti et al. [2000]. But this algorithm requires too many iterations. The second step is the polygonal approximation. All the different solutions can be classified into three categories: (i) minimization of the distance between the approximation and the curve (e.g., [D. H. Douglas 1973; de Figueiredo 1995]); (ii) minimization of the area between the approximation and the curve (e.g., [Wall and Danielsson 1984]); and (iii) best approximation of the angular (e.g., [Dunham 1986]) or the curvature (e.g., [Asada and Brady 1986]) variation. Each approach provides a different solution but gives similar results for the quality.

2.3 Level of detail for line-based rendering

Once the feature lines have been extracted, they have to be rendered efficiently. With the development of NPR rendering, some solutions have been proposed for the simplification of the line-based style. In Deussen et al. [2000], the presented model is very specific to the underlying structure for trees and vegetation, and cannot simply be used in our context. Praun et al. [2001] present an image-based method to handle LOD in hatching. Even if their approach allows real-time display of hatching styles, their tonal art maps are only mip-mapped textures, and will not work for vectorial data. In the "WYSIWYG NPR" system [Kalnins et al. 2002], and for generic hatching in general, the user specifies the appearance of the object for several viewpoints, and relevant LODs are then blended for a given view. There is no automatic process for lines.

For a set of co-planar feature lines, image-space simplification of lines can be adapted in order to obtain the different LODs. A first approach consists in a perceptual grouping of lines. Most of the work in this area deals with the extraction of closed paths in drawings [Saund 2003; Elder and Zucker 1996], focusing on grouping criteria such as good continuation and closure. However, other criteria are more relevant for simplification purposes, e.g. proximity and parallelism. Unfortunately, even if each criterion has been studied in isolation [Rosin 1998], their relative influence has yet to be determined. In Barla et al. [2005], the authors extend the previous work in order to create a real line-set simplification based on a perceptual metric. Unfortunately, all these solutions create new lines for each LOD and thus increase the needed bandwidth for the data transmission.

Another approach is to remove, for each LOD, the "non-significant" lines (e.g., [Preim and Strothotte 1995; Wilson and Ma 2004]).The work of Preim et al. [1995] is most closely related to our research. Our approach is inspired by the metric they suggest, using line length for the selection (see Section 5.2).

3 Architecture

3.1 Overview

Figure 2 presents the general overview of the system. The first part consists in the modeling and optimization of the 3D model. We first extrude the buildings from the city footprints. This results in a set of buildings with textures corresponding to their facade. Then, feature lines are extracted from the textures and used to represent the facades more legibly (see Section 4).

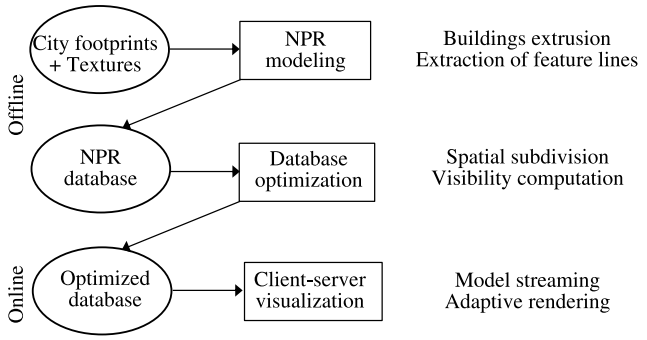


Figure 2: General overview. The offline processes consist of modeling and preparing the database to be used in a client-server application (online). First, the feature lines are extracted from the textures. Then, the geometry is split up and the visibility relationships are computed.

The next part consists of optimizing the database for the streaming. The database needs first to be split up into connected cells, so that the server is able to send to the client only the required cells. For further optimization, visibility relationships between the cells are computed. As a result, only the visible geometry is sent to the client and rendered. The related processes are described in Section 3.2 and Section 3.3.

Finally, the remote-rendering solutions for client-server applications are explained in Section 5, and the results of our approach are discussed in Section 6.

3.2 Scene subdivision

The objective of this preprocessing step is to produce a 3D database that can be streamed using the visibility information. A large number of solutions have been developed (see the extensive survey in [Cohen-Or et al. 2003]). We based our solution on VRML97 and on the extension for visibility streaming defined by Marvie et al. in [2004]. We describe briefly some aspects of this approach.

The preprocessing can be decomposed into two steps: First, the navigation space is produced by subdividing the city model according to its topology. Second, the PVOS (Potentially Visible Object Set) is computed for each cell of the navigation space.

In order to generate the navigation space for a city (a 2.5D dataset), we exploit the BSP subdivision similar to the one presented in [Meneveaux et al. 1998]. The city is subdivided into a set of cells (each cell containing its associated geometry) for which it establishes adjacency relationships. The resulting cells are used to compute cell-to-objects visibility relationships [Wonka et al. 2000; Cohen-Or et al. 2003]. Each cell will then refer to a set of potentially visible objects (PVOS) that are the buildings of the city. Note that the navigation space can be restricted to the streets by using the approach described by Décoret and Sillion [2002].

For the generation of such a database and the optimization of the preprocessing step, we rely on three different representations of the city model:

1. The polygonal model of the entire city, in order to compute the navigation space.
2. The simplified model of the buildings (only their polygonal faces), in order to perform the visibility computation.
3. The full model of the buildings (their polygonal faces together with their vectorial appearance), for the final rendering.

3.3 Visibility streaming

The process is done online on the client side. There are two stages which are processed in a sequential way: one loading step, necessary for the navigation, and then, a pre-fetching step to optimize data transfers. During this first step, the current cell is downloaded in a synchronous way together with its PVOS. The user can now begin the navigation.

During the navigation, a prediction is made about the next cell to be visited, based on the users's motion. This future visited cell and its adjacent cells are downloaded in an asynchronous way with their PVOS. The future visited cell has a higher priority.

3.4 Adaptive memory management

While the user is walking through the scene, the new cells are streamed progressively. These cells are stored in memory, so downloading them a second time will not be required when the user comes back. The client builds up a graph using the adjacency relationships of the cells available on its side. When the client needs a new cell but runs out of memory, the farthest cells from the current cell (i.e., the deepest in the graph) are removed until enough memory is freed.

4 Feature extraction of the facades

The whole image-processing pipeline is shown in Figure 1. Feature lines are extracted from the facade pictures using an edge detector. Then, the resulting contours are vectorized into poly-lines. This procedure allows the size of the data to be reduced as well as to make the representation independent from the resolution. These two steps are presented in this section.

4.1 Edge detection

Among the tested edge detector we tested [Pingle 1969; Roberts 1965; Prewitt 1970; Canny 1986], we identified the Canny edge detector [Canny 1986] to be best suited for our purposes. It works in a multi-stage process that we recall here shortly. First of all, the image is smoothed by Gaussian convolution. Then, the gradient is computed on the image, using the following convolution kernels.

$$G_x = \frac{1}{2}\begin{bmatrix} -1 & -1 \\ +1 & +1 \end{bmatrix} G_y = \frac{1}{2}\begin{bmatrix} +1 & -1 \\ +1 & -1 \end{bmatrix}.$$

This highlights high frequencies, which often represent contours. The resulting gradient magnitude and direction are

$$|G_{i,j}| = \sqrt{G_x(i,j)^2 + G_y(i,j)^2} \quad (1)$$

$$\theta_{i,j} = \arctan\left(G_y(i,j)/G_x(i,j)\right). \quad (2)$$

Then, a pixel value is kept only if it represents a local maximum in the gradient direction. This ensures that one pixel wide contours are obtained, which is a desirable quality for feature lines extraction.

The resulting magnitude image is filtered by removing values that are outside a minimum and a maximum threshold. This ensures that the same edge will not be broken by noise.

The parameters involved with this calculation are the size of the Gaussian kernel, its standard deviation, and the two thresholds. These values can be set once and applied to a complete set of data. This provides coherent results. An example can be seen in Figure 1.

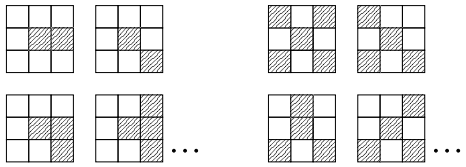


Figure 3: Sample masks used to find initial nodes of the graph: (a) for end-point detection, (b) for detection of junctions

4.2 Pixel chaining

For the vectorization process, the pixels obtained from the edge detection are chained to obtain the poly-lines. To this end, we create and fill a graph structure.

First, we identify end-points and junctions of the pixel chains using 3×3 masks, like those depicted in Figure 3 on the contour image. The detected pixels are the initial nodes of the graph, the edges are the pixel chain that links two nodes.

Starting from an end-point, we process the connected components following the contour. For remaining components without end-points (e.g., the loops, like the windows of Figure 4), a second traversal of the image is necessary. This time, the starting pixel is the first unprocessed one. Figure 4 illustrates the different stages of this procedure.

The identified junctions are not always positioned on the actual intersections between the segments, due to the fact that we only search for one-pixel neighborhood. Figure 5 illustrates this problem. Figure 5(a) shows the detected junctions using the mask method. The real position of the intersection (Figure 5(b)) can only be obtained by knowing a vector representation of the involved segments. We obtain it easily after the polygonal approximation in the post-processing stage by merging locally close junctions.

Thus, each pixel chain of the graph is processed to obtain a polygonal approximation [Wall and Danielsson 1984] (some segments defined by two vertices). The vertices are then added as nodes in the graph structure (see Figure 4-(e)).

4.3 Post-processing and cleaning

In order to optimize vectorization and to preserve the general structure of a facade, a post treatment is necessary. Its purpose is to simplify the obtained approximation and to correct the errors due to the initial noise and the possible lens deformation of the image, and the errors due to the vectorization stage.

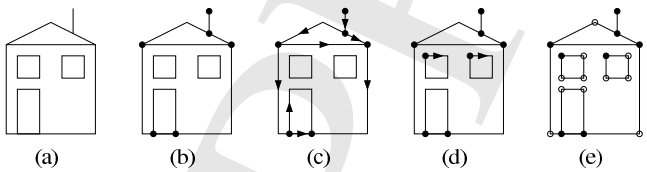


Figure 4: Stages for automatic stroke extraction: (a) Extracted edges. (b) End-points and junctions detection. (c) Breadth-first traversal. (d) Loop processing. (e) Polygonal approximation (introduced characteristic points are white).

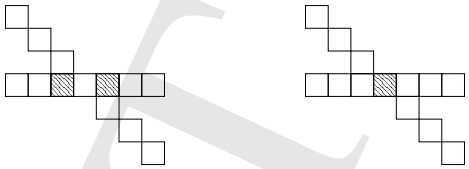


Figure 5: Problem when positioning junctions: (a) Detected junctions. (b) More likely position.

The post-processing steps that we selected are the followings:

- *Suppression of small size segments*: these are introduced by noise and do not bring anything to the legibility of the image.
- *Junctions merging*: as explained in Section 4.2, the masks method does not allow us to find the right position of the intersections between the segments. The vectorial representation enables us to merge locally close junctions.
- *Straight line detection*: since the input images suffer from lens distortions and noise, the detected lines are not always perfectly straight. Indeed, the polygonal approximation of a chain perceived as a single line can result in several small connected segments that have similar orientations. To limit the number of segments and to improve the legibility, we merge connected segments that have similar orientation.
- *Vertical and horizontal line fitting*: As we work on buildings, we have a great number of horizontal and vertical lines. When a line is close to vertical or horizontal, we simply straighten it.

5 Rendering of Lines

We now have a set of lines for each facade. An easy way to get a 3D model from this data is to build a street in which each building is a simple bloc. For each facade, we store the extracted lines.

5.1 Rendering the full set of lines

We can use the set of lines to render a facade in different ways:

1. The lines can be drawn in an image to be used instead of a texture. This image is indeed less complex than the original and thus offers a good compression rate using any image compression algorithm. We can consider this an easy way to try different stylization techniques, but the quality is restricted by the texture resolution.
2. A line can be represented as a geometric primitive on the surface of the building (a GL_LINE in our implementation). The projected lines have a constant width of 1 pixel. With this vectorial representation, the visualization can be adapted to any resolution.

Because it is adequate for multi-resolution, we have chosen the second method. Figure 6 shows an example of the rendering of a street using lines on Pocket PC. Rendering all lines without taking into account facade distance and orientation results in a high density of lines in image space that reduces legibility. This leads us to use level of detail techniques in order to solve this issue.

(a) Without level of detail (b) Using level of detail

Figure 6: A street rendered on a Pocket PC. (a) The display is quickly saturated with lines. (b) Using level of detail permits us to reduce line density.

5.2 Level of detail

The lines previously extracted are rendered as constant width lines on the 3D model no matter the distance they are viewed from. In some instances, the screen may become saturated with lines when viewed at distance. This will decrease the legibility of the image. LOD provides a solution to this problem. We present here our approach based on the VRM97 standard.

Depending on the distance a building is viewed at, we only show a subset of the associated lines. The farther we are from a building, the smaller the subset will be. In this way, we preserve a reasonable line density on the screen. Moreover, less geometric primitives are drawn, thereby improving the rendering time.

In order to create the LODs, we have to define a criterion for line selection. In our approach, we use the length of the lines. The fully detailed level contains the whole set of lines; this level is used for close and orthogonal view of the facade. Thus, for each level of detail, we define the range of valid line lengths. This results in removing the smallest lines between two successive LODs. This technique provides visual continuity during the transition between two levels of detail.

This approach is global to a facade and does not consider the closeness between the lines, which can lead to unwanted local agglomerations of lines. We are exploring a solution that selects lines for a given level of details depending on line density in the image [Barla et al. 2005]. Nevertheless, our first approach is well adapted to the current rendering system, providing us with meaningful LOD, and giving better rendering performances (see next section) than the full line rendering.

6 Results

Since the approach that we have presented can be considered mostly as an alternative to textured rendering applied to urban scenery, we need to compare both methods. The validation of scene legibility depends on psychological matters and is not the purpose of the current paper. We rely on the fact that we can still recognize a facade with only the extracted lines (see Figure 6 and Figure 1). We compared line and texture-based rendering on the following points: data size and rendering performance.

For these tests, we used a DELL Axim X50V in order to show the performance of the presented approach on mobile devices. It has a 624 MHz ARM processor, 64 MB of RAM memory and an Intel 2700G GPU. It supports full hardware acceleration of OpenGL-ES and handles a resolution up to 640x480. The application is tested with two OpenGL-ES implementations: the Intel SDK for complete hardware acceleration and a software-based library developed by Hybrid Graphics[4].

For the relevance of the tests, all the data in each set has to be coherent, that is the images of facades must come from the same source (same model, same street, same acquisition device...). This allows us to use the same parameters for the vectorization of a complete set of input images, with comparable results. Obviously, it will not look homogeneous when mixing high and low resolution images or images from different cities.

All of the pictures of building facade come from two different sources. The first one is the website of the French "yellow pages[5]" which offers images of facades for several towns. It is possible to easily obtain images from the same street. However, they are of very small resolution (250x320) and poor quality: the imperfections of JPEG compression are often visible. In order to confront the results to data of better quality, we have taken photographs of facades using a digital camera. The resulting pictures have a resolution of 3008x2000. From these input data we generated three streets using:

1. 10 images of the Borda street in Bordeaux coming from the online yellow pages.
2. 22 images of the Viadieu street in Toulouse coming from the online yellow pages.
3. 13 photographs of the "cours de la Libération" at Talence taken with our digital camera.

We used the city model described by Hachet and Guitton [2001] with our sets of texture. These textures are also processed in order to extract the characteristic lines.

6.1 Fixed viewpoint

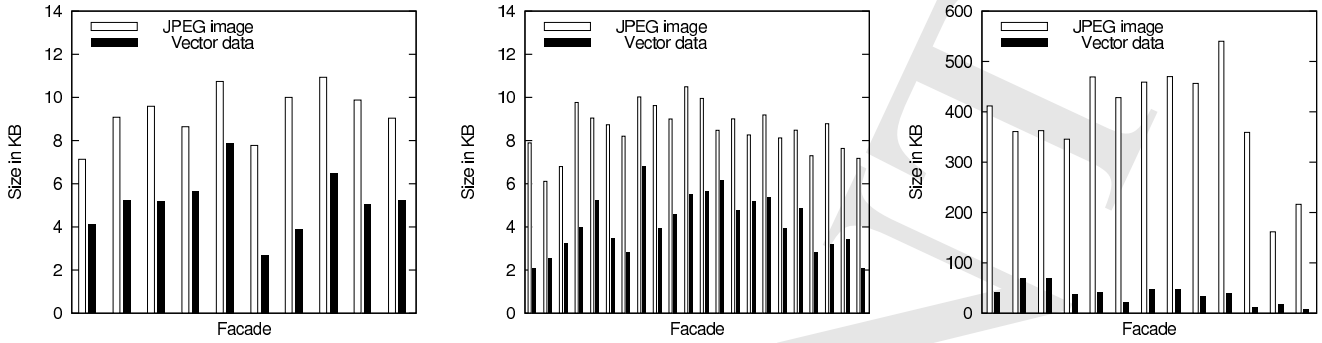
In this section, we study the general behavior of our approach when no streaming is requested from the server. This represents the optimal case.

6.1.1 Data size

Comparison with the JPEG compression As an alternative to textured rendering, we have to compare the size of the generated lines with the size of the original JPEG-compressed images. The results are summarized in Figure 7. The vector data is stored in a simple binary format. It contains a header of two integers which are the width and height of the original image. Then, each line is described by four float values: the coordinate of the two end-points of the line. The resulting file is compressed using the zlib[6].

The vector files resulting from the low resolution images (Figure 7(a) and Figure 7(b)) are on average two times smaller than the JPEG images: 4.16 KB in average for the vector data compared to 8.55 KB for the JPEG compression. We obtain smaller data sizes with our method. Of course, the files contain far less information than compressed images ; the representation is independent of the resolution, and it contains only the legible characteristic

[4] http://www.hybrid.fi
[5] http://www.pagesjaunes.fr
[6] http://www.zlib.net/



(a) Borda street (10 images) (b) Viadieu street (22 images) (c) Cours de la Libération (13 images)

Figure 7: Comparison between the size of vector data and JPEG images

features. Conversely, an image depends strongly on its resolution. Thus, for high-resolution images, the ratio JPEG/vector is bigger than 10 (387.76 KB in average for JPEG compression compared to 37.55 KB for vector data).

Comparison with drawn textures The lines can be drawn into an image to be used as a texture. The resulting images are less complex than original ones, so they offer a good compression ratio using any image compression algorithm. Such a representation is interesting because we can foresee using this approach to easily try several stylization methods. The Figure 8 presents a comparison between the sizes of files compressed using JPEG and the sizes of the images in which we drawn the vector data. The PNG format has been adopted instead of JPEG for this purpose because, even if the JPEG method handles the compression of images that contain big color patches, as ours do, strong artefacts can be introduced along the sharp edges. Indeed, the images we produced have a white background on which the lines are drawn in black. For the PNG compression, we used the adaptive filter as recommended in the PNG specification published by the W3C[7].

The resulting data size for PNG without anti-aliasing are fairly low, 3.13 kb on average for the low-resolution images (Figure 8(a) and Figure 8(b)) and 60.56 kb for the high-resolution ones (Figure 8(c)). Therefore, it is conceivable to use them as textures. When an anti-aliasing is applied to the drawn lines, the color map grows as well as the size of the images. In high resolutions, the resulting images are still smaller than the JPEG ones, 260.85 kb for PNG compared to 387.76 kb for JPEG on average. This is not the case for the low-resolution images because of their poor quality.

6.1.2 Rendering speed

The rendering tests presented in Table 1 have been done on the three data sets using textures, lines without LOD and lines with LOD on the Pocket PC.

Textured rendering is faster than line-based rendering on the software-based OpenGL-ES implementation. This is not surprising due to the large number of geometric primitives used to render the scene. The frame rate increases when using LOD and is higher than textured rendering. For the high-resolution images, their use as textures is not practicable on Pocket PC, whereas when using lines with LOD the result is still interactive.

[7]www.w3.org

Using the GPU, the performance is better when using the textures or the lines with LOD. In this case, the texture based rendering leads to better results than the line-based rendering. Surprisingly, rendering the full set of lines on the software-based implementation is faster than using the dedicated hardware.

6.2 Moving viewpoint: the complete city model

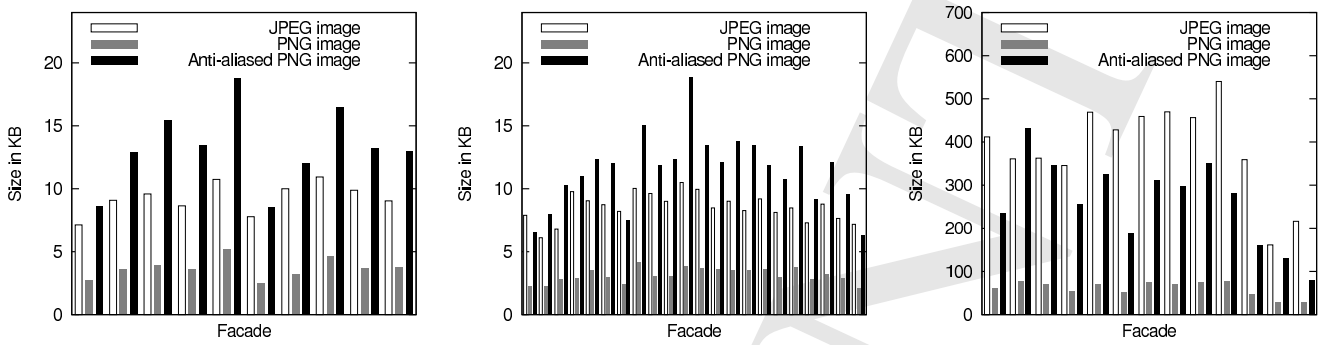
In this section, we study the general behavior of our approach when streaming is requested from the server. This represents the real client-server case.

(a) Using original images (b) Line-based rendering

Figure 9: Same view of a street. (a) Using the original images as textures (b) Using line-based rendering.

6.2.1 Data size

In order to test the different methods, we recorded a path in the scene and used it for the different models: buildings with their original textures (photographs) and buildings along with their line descriptions. We used two versions of the later model: one which contains the whole set of lines for each building and one which takes advantage of the LOD technique described previously. As a reference for the highest possible performance, we used a model containing the buildings only. During the navigation, we recorded different statistics. Considering the size of downloaded files, we have measured 7.55 KB on average for the lines, and 35.61 KB for



(a) Borda street (10 images) (b) Viadieu street (22 images) (c) Cours de la Libération (13 images)

Figure 8: Comparison between JPEG images and vector data drawn into PNG images, optionally using anti-aliased lines

	Borda street	Viadieu street	Cours de la Libération
Number of lines	11188	6190	58949
Number of lines per facade	508.55	619	4534.54
Performance using the software-based implementation			
Textured rendering	5.7 fps	5.6 fps	-
Rendering using lines	4.6 fps	3.8 fps	1.3 fps
Rendering using lines and LOD	7.2 fps	7.2 fps	3.1 fps
Performance with hardware acceleration			
Textured rendering	51.0 fps	28.2 fps	-
Rendering using lines	4.7 fps	2.8 fps	0.7 fps
Rendering using lines and LOD	15.8 fps	18.8 fps	2.0 fps

Table 1: Frame-rate comparison on the Pocket PC

(a) Performance using the software-based implementation

	Buildings	+ Lines	+ Lines with LOD	+ Textures
Memory used (MB)	23.50	25.19	26.25	26.95
Frame rate (fps)	2.88	1.17	2.28	2.18
Graph scene traversal (ms)	6.77	47.43	65.32	13.96
Rendering time (ms)	286.39	930.04	346.54	387.74
Average bitrate (bps)	23980	27292	25220	31696
Min bitrate (bps)	2448	1577	1573	1573
Max bitrate (bps)	87000	548535	319640	319640

(b) Performance with hardware acceleration

	Buildings	+ Lines	+ Lines with LOD	+ Textures
Memory used (MB)	20.92	23.65	24.41	26.28
Frame rate (fps)	22.31	1.25	5.45	11.97
Graph scene traversal (ms)	5.98	47.09	57.92	9.83
Rendering time (ms)	27.84	1631.25	257.88	90.15
Average bitrate (bps)	19723	19125	17306	26902
Min bitrate (bps)	9757	3629	2715	8583
Max bitrate (bps)	92200	448454	297178	1168214

Table 2: Rendering performance on the Pocket PC

the textures (see Table 2). As for previous tests on the street model, the lines are much smaller than the textures. This can also be seen on the required memory. Using textures results fills the memory about 2 MB more than the lines.

6.2.2 Rendering speed

Considering the rendering speed using each technique, once again, the geometry shows the highest frame rate we are able to reach. Using the lines without LOD results in a frame rate of 1.17 fps in the software-based implementation, and 1.25 fps in the other case (see Table 1). The rendering using the textures leads to good results (2.18 fps and 11.97 fps) and is still higher than using the lines with LOD (2.28 fps and 5.45 fps). This can be explained by considering the very poor quality of original textures. Additionally, the line node could be more optimized. Indeed, we are currently using non-connected lines: each line is described by two end points. Because of the original graph structure, some end-points are common to different lines. Therefore, the rendering cost could be reduced by using line strips (GL_LINE_STRIP).

The rendering is divided in two stages, the scene graph traversal and the rendering time. As seen in Table 2, the scene graph traversal time for the texture (13.96 ms and 9.83 ms) is lower than for the lines with LOD (65.32 ms and 57.92 ms). Indeed, the scene graph for the textures is simpler.

However, the rendering times are very close using the software implementation: 387.74 ms for the textures, and 346.54 ms for the lines with LOD. By implementing an optimized dedicated node, we would decrease the time spent in the graph traversal and consequently we would increase the frame rate significantly. Using hardware acceleration, the textures obtain better results (90.15 ms) than the lines with LOD (257.88 ms).

6.2.3 Network

For each method, we measured the bitrate in the following way. Each time a file is requested by the client, we compute the time interval between the beginning of the download and the node initialization. By dividing the file size by this value, we obtain a result that represents the rate at which the data are arriving from the server to the moment they are available in the scene graph of the client, ready to be rendered. It includes: data transmission over the network, data decompression, VRML97 parsing and node initialization. The OpenGL-ES library uses 16 bits integers as index for vertex arrays. The VRML97 standard uses 32 bits indexes. As a choice of implementation, we decided to let the client adapt himself to the data. So, there is a conversion from 32 to 16 bits on the fly on the client side. As seen in Table 2, the texture node requires less time be spent in parsing VRML and in converting integers and so it makes its bitrate higher than for the lines. Even if there is less data to be transmitted, the time spent in all these processes makes the bitrate smaller for the lines. Once again, making a dedicated node would improve the results.

On our tests, the maximum bitrate we reach is about 1.1 Mb/s. Since the actual network bitrate used is lower than this value. We should be able to navigate through the city even with a low network bandwidth.

7 Conclusion

In this paper, we have presented a new approach for remote rendering of large 3D content (like cities), based on expressive rendering. In this approach, the original textures of the facade are processed and the feature lines are extracted. The resulting data set (buildings with their characteristic lines) is optimized for remote visualization and stored on the server side. The city is then streamed on-demand on a remote client.

We have experimented with this approach on mobile clients like PDA. In our experiments, we can still recognize buildings, showing that the resulting rendering still conveys the required information. This is useful for 3D content display on small devices. It shows also that the amount of data to transmit is greatly reduced, making this solution well suited for limited-bandwidth networks like the ones used by mobile devices. Moreover, it shows that we can still obtain interactive rendering, even with pure software 3D rendering.

Future Work

First, we have to improve the robustness and the efficiency of our extraction of the feature lines. This would increase the legibility of the resulting visualization. Based on this improved solution, a cognitive study has to be performed.

Then, our results have shown that using the lines as 3D primitives requires of the scene graph which is too costly. A specific node for the vectorial description of an object's appearance would greatly decrease this drawback. This new node would take advantage of the up-coming vectorial processor based on the OpenVG[8] standard by the way of vectorial textures, and would allow the integration of a larger range of NPR styles. We are also working on specific LOD solutions for such a node.

Acknowledgment

We want to acknowledge Matt Kaplan, Patrick Reuter, and all the Web3D reviewers for the time spent on this paper. This work is funded by France-Telecom R&D and is part of the INRIA Cooperative Research Initiatives MIRO. We also want to thank Intel for providing us with up-to-date PDAs. This work is partially supported by Conseil Regional d'Aquitaine.

References

ASADA, H., AND BRADY, M. 1986. The curvature primal sketch. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 1, 2–14.

BARLA, P., THOLLOT, J., AND SILLION, F. 2005. Geometric Clustering for Line Drawing Simplification. In *Proceedings of the Eurographics Symposium on Rendering*.

CANNY, J. 1986. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6, 679–698.

CHAKRABARTI, A., JAIN, A., AND RAY, A. 2000. A Novel Algorithm to Generate the Cover Map of a Remotely sensed image for gis applications. In *Indian Conference on Computer Vision, Graphics and Image Processing 2000*.

[8] htpp://www.khronos.org

COHEN-OR, D., CHRYSANTHOU, Y. L., AND DURAND, C. T. S. D. 2003. A Survey of Visibility for Walkthrough Applications. *IEEE Trans. Visualization and Computer Graphics 9*, 3, 412–431.

D. H. DOUGLAS, T. K. P. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer 10*, 2, 112–122.

DE FIGUEIREDO, L. 1995. *Adaptive sampling of parametric curves*. Morgan Kaufmann, 173–178.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3, 848–855.

DÉCORET, X., AND SILLION, F. 2002. Street generation for city modelling. In *Architectural and Urban Ambient Environment*.

DEUSSEN, O., AND STROTHOTTE, T. 2000. Computer-generated pen-and-ink illustration of trees. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 13–18.

DIEPSTRATEN, J., GRKE, M., , AND ERTL, T. 2004. Remote line rendering for mobile devices. In *IEEE Computer Graphics International (CGI)'04*.

DUNHAM, J. 1986. Optimum uniform piecewise linear approximation of planar curves. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 1, 67–75.

ELDER, J. H., AND ZUCKER, S. W. 1996. Computing contour closure. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, Springer-Verlag, London, UK, 399–412.

HACHET, M., AND GUITTON, P. 2001. From cadastres to urban environments for 3d geomarketing. In *Proceedings of IEEE/ISPRS joint workshop on remote sensing and data fusion over urban areas*, 146–150.

HEKMATZADA, D., MESETH, J., AND KLEIN, R. 2002. Non-photorealistic rendering of complex 3d models on mobile devices. In *Conference of the international association for mathematical geology*, vol. 2, 93–98.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. Wysiwyg npr: drawing strokes directly on 3d models. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 755–762.

MARVIE, J.-E., AND BOUATOUCH, K. 2004. A VRML97-X3D extension for massive scenery management in virtual worlds. In *Proceedings of the ninth international conference on 3D Web technology*, ACM SIGGRAPH, 145–153.

MENEVEAUX, D., MAISEL, E., AND BOUATOUCH, K. 1998. A new partitioning method for architectural environments. *Journal of Vizualisation and Computer Animation 9*, 4 (Novembre), 195–213.

PINGLE, K. K. 1969. Visual perception by a computer. In *AII*, 277–284.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 581.

PREIM, B., AND STROTHOTTE, T. 1995. Tuning Rendered Line-drawings. In *Proceedings of Winter School in Computer Graphics –The third Central European Conference on Computer Graphics '95*, 227–237.

PREWITT, J. 1970. Object enhancement and extraction. In *Picture Processing and Psychopictorics*, B. Lipkin and A. Rosenfeld, Eds. Academic Press, New York, 75–149.

ROBERTS, L. 1965. Machine perception of three–dimensional solids. In *Optical and Electro-optical Information Processing*, J. Tippet, D. Berkowitz, L. Clapp, C. Koester, and A. Vanderburgh, Eds. MIT Press, Cambridge, Massachusetts, USA, 159–197.

ROSIN, P. L. 1998. Grouping curved lines. *Machine Graphics and Vision 7*, 3, 625–644.

SAUND, E. 2003. Finding perceptually closed paths in sketches and drawings. *IEEE Trans. Pattern Anal. Mach. Intell. 25*, 4, 475–491.

TOMBRE, K., AH-SOON, C., DOSCH, P., MASINI, G., AND TABBONE, S. 2000. Stable and robust vectorization: How to make the right choices. In *GREC '99: Selected Papers from the Third International Workshop on Graphics Recognition, Recent Advances*, Springer-Verlag, London, UK, 3–18.

WALL, K., AND DANIELSSON, P.-E. 1984. A fast sequential method for polygonal approximation of digitized curves. *Comput. Vision Graph. Image Process. 28*, 3, 220–227.

WILSON, B., AND MA, K.-L. 2004. Representing complexity in computer-generated pen-and-ink illustrations. In *in Proceedings of the International Symposium on NonPhotorealistic Animation and Rendering (NPAR) 2004,*.

WONKA, P., WIMMER, M., AND SCHMALSTIEG, D. 2000. Visibility preprocessing with occluder fusion for urban walkthroughs. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, Springer-Verlag, London, UK, 71–82.