# A Client-Server Approach to Image-Based Rendering on Mobile Terminals

Kadi Bouatouch, Gérald Point, Gwenola Thomas

# *A Client-Server Approach to Image-Based Rendering on Mobile Terminals*

Gwenola Thomas — Gérald Point — Kadi Bouatouch

**N° 5447**

January 2005

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--5447--FR+ENG

# A Client-Server Approach to Image-Based Rendering on Mobile Terminals

Gwenola Thomas [*] , Gérald Point [*] , Kadi Bouatouch[†] [*]

Thème COG — Systèmes cognitifs
Projets IPARLA

Rapport de recherche n° 5447 — January 2005 — 29 pages

**Abstract:** This report shows how to use IBR methods to make possible the rendering of complex scenes on a mobile terminal, such as a PDA, while using a client/server architecture. The PDA represents the client of a server which computes a very small set of key images (to avoid latency time that would affect interactivity) of a complex 3D scene and transmits them on demand to the client across a low bandwidth network. The client utilizes these images to use a warping technique to compute new images as seen by intermediate cameras (using an IBR technique) whose positions and directions are chosen interactively by the user by moving the stylus of a PDA. The most difficult problem is how to place the cameras (capturing the key images) which allow an efficient warping avoiding artifacts, such as holes, due to occlusions and exposures. Providing a general solution to the problem of camera placement is a hard task. In this report we address only the case of urban scenes.

**Key-words:** client/server, 3D scenes, Rendering, Warping, Image Based Rendering, Low-Bandwidth Network , PDA, Camera Placement

[*] LaBRI, INRIA Futurs, Bordeaux
[†] IRISA, Rennes

# Une Approche Client/Serveur pour le Rendu Basé Image sur Terminaux Mobiles

**Résumé :** Ce rapport montre qu'il est possible d'utiliser des méthodes IBR (rendu basé image) pour effectuer le rendu de scènes complexes sur un terminal mobile, tel qu'un PDA, dans le cas d'une architecture client/serveur. Le PDA représente le client d'un serveur qui calcule un ensemble restreint d'images clés (pour éviter un temps de latence important qui pourrait dégrader l'interactivité) d'une scène 3D complexe et les transmet à travers un réseau bas débit sur demande. Le client utilise ces images clés pour en déduire par interpolation (en utilisant donc des méthodes IBR) de nouvelles images intermédiaires vues par des caméras dont l'orientation et la position sont choisies par l'utilisateur de manière interactive en déplaçant un stylet. Un problème délicat est celui du placement des caméras pour éviter l'apparition de trous dus aux problèmes d'occlusion et d'apparition de nouveaux objets lorsqu'on effectue l'interpolation. Concevoir une solution générale à ce problème est une tâche très difficile. Dans ce rapport nous proposons une solution de placement de caméra dans le cas de scènes urbaines.

**Mots-clés :** client/serveur, Scènes 3D, Rendu, Interpolation, Rendu Basé Image, Réseau Bas Débit, PDA, Placement de Caméra

## Contents

## 1 Introduction

PDAs (Personal Digital Assistants) are handheld computers that are increasingly widespread since the last decade. Many applications already run on PDAs but complete high quality rendering of 3D models still remain beyond their capabilities. To make rendering on PDAs possible, one solution is to rely on a client/server architecture in which the server computes images of a 3D scene then sends them to a client, say a PDA, which visualizes them on its small screen. As this solution is highly demanding in terms of network bandwith, a preferable solution is to distribute rendering among the server and the client. Indeed, the server computes a set of key images that are sent to a client which computes in-between images using IBR techniques (Image Based Rendering).

While traditional rendering methods need data representing the geometry and the photometry of the objects making up a 3D scene, IBR methods take as input a set of images (synthetic or real) sometimes augmented with depth maps. When rendering complex scenes, the computation cost of traditional rendering is proportional to the number of objects within a scene, while it is only proportional to the image resolution for IBR methods. IBR techniques proved that they are fast and easy to implement. They only consist in calculating, for intermediate camera positions, in-between frames from key frames and depth-maps.

This paper shows how to use IBR methods to make possible the rendering of complex scenes on a PDA in the framework a client/server architecture. The PDA represents the client of a server which computes a very small set of key images (to avoid latency time that would affect interactivity) of a complex 3D scene and transmits them on demand to the client across a low bandwidth network. The client utilizes these images to compute new images as seen by intermediate cameras (using an IBR technique) whose positions and directions are chosen interactively by the user by moving the stylus of a PDA.

Image-Based-Rendering seems a good compromise between classical 3D rendering on a PDA and streaming images that are all computed on the server side. In other words, IBR is a good compromise between processing time on a PDA and time of data transmission through a low bandwidth network such as GPRS or wireless networks (Wifi or others).

Another advantage of IBR techniques is the possibility to interact with a 3D scene while it is hardly possible when streaming images from the server to the client. Warping key images to compute in-between ones gives the user the feeling of navigating through a 3D scene.

The most difficult problem is how to place the cameras (capturing the key images) which allow an efficient warping avoiding artifacts, such as holes, due to occlusions and exposures. Providing a general solution to the problem of camera placement is a hard task. In this report we address only the case of urban scenes.

This report is organized as follows. Section 2 presents some related works regarding rendering and image-based rendering on PDAs as well as solutions to the camera placement problem. Our Client/Server architecture is described in section 3. Section 4 presents in detail our solution to the camera placement problem in the case of urban scenes only. Some implementation details and results are given in section 5. Finally we conclude in section 6.

## 2 Related works

In this section we report first on 3D rendering methods on PDAs, then on IBR techniques while focusing on the solutions relevant to our method. Next, we address the problem of key image selection. This problem can be stated as a camera placement problem. The solution we propose to this problem is only valid for urban scenes.

### 2.1 3D rendering on PDA

Rendering complex 3D scenes on PDAs cannot be straightforwardly performed by reusing existing software packages running on Personal Computers because the PDAs are not yet supplied with floating point units and dedicated graphics accelerators have just been available. Moreover, even if external memory cards of 256 Mega bytes can be used on these devices, access times are still too high.

However, 3D rendering of small scenes is possible on PDAs. A new generation of PDAs based on the novel Intel XScal PXA 250 processor with a 400 Mhz clock speed providing dedicated support for multimedia and 3D graphics applications has appeared. Some 3D APIs dedicated to programming on PDA already exist. One of these tools is a 3D graphics library similar to OpenGL [15, 22]. To render larger scenes, methods relying on simplified geometry based on levels of details [27] or Non-Photorealistic-Modeling [9] can be used. However, these methods provide images that do not seem realistic and the scene complexity is still very limited.

To overcome the limitations of PDAs and to make possible rendering of complex scenes on these devices, one can make use of a client/server architecture. Martin [19] has classified into three major categories the methods for rendering 3D models in client-server environments. The first category is called *client-side methods* [12]. The methods of this category do not involve any rendering on the part of the server. The geometry as well as the textures are downloaded to each client that requests them and the client is responsible for rendering it. Such methods are not well suited for PDAs. When using the methods of the second category, called *server-side methods*, the 3D model is fully rendered on the server side and the resulting images are sent to the clients [6, 2]. In [6], the sever generates the frames, encodes and transmits them to the client. The encoded frames are transmitted as a video stream to the client which decodes the stream and displays it. In [2], each client uses previous views of the scene to predict next view using image-based rendering techniques. The server performs the same prediction and sends only the difference between the predicted and the actual views. Compressed difference images require less bandwidth than the compressed images of each frame. Such methods get interesting when the clients have limited resources and limited graphics performances, which is the case of PDAs. As for the third category, called *hybrid-side methods*[17], parts of the 3D model are rendered on the server and the other parts are downloaded and rendered on the client side. Such methods have the advantage of reducing the geometric complexity of the data to be transmitted by replacing parts of the geometry with images. However, deciding which part of a model should be rendered on the server or on the client is not a trivial task. One possibility is that the server could render high and low resolution versions of a 3D model and send the residual error image and the low-resolution geometry to the client [17]. In this case, the role of the client is to render the coarse model and to add the residual image to restore a full quality rendering. Such methods are not suited for PDAs, because for complex scenes, a lot of geometry (even coarse models), textures as well as residual images have to be transmitted to the PDAs.

As seen above, a *server-side method* has to be used for rendering complex scenes on a PDA. Recall that this kind of method is based on streaming images to the client. Streaming images can offer realism and is well studied to rendering on PDAs [3, 6, 16, 4]. Even with a high bandwidth network, streaming is not the perfect solution. A lot of time is spent by the client to download and render images [16], which dramatically reduce the frame rate on the client side. Moreover, when walkthrough is the targeted task of streaming applications, the problem of interactivity is not efficiently dealt with. Note that in the case of NPR (Non Photorealistic Rendering) some works have been done on streaming silhouettes, creases and feature lines (rather than images) to a PDA [9, 12]. In our opinion, when realistic rendering is targeted, streaming has to be combined with IBR techniques as already done in [16, 4].

The drawbacks of 3D rendering (small scenes, no realism) and streaming (bottleneck for images transmission and lack of interactivity) led us to propose the use of IBR methods for rendering complex 3D environments on mobile terminals such as PDAs.

## 2.2 Image-Based-Rendering

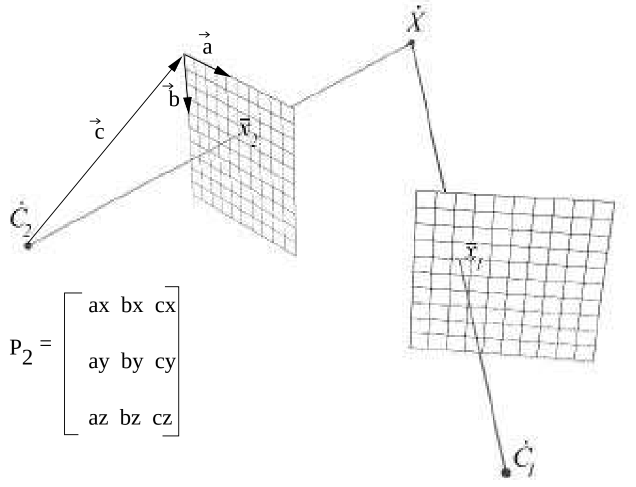
Image-Based-Rendering (called IBR from now on) is based on the assumption that there is a slight difference between two successive images when walking through a 3D environment. Indeed, rendering at a current position and an orientation can be performed using IBR techniques which consist in warping nearby pre-rendered images (called key images or reference images). IBR techniques are used in two main applications: 3D walkthrough and 3D object reconstruction from images [7, 21].
A famous IBR walkthrough system is QuickTime VR [5] which allows a rendering yielding 360-degrees cylindrical panoramic images. When using this method the degrees of freedom of navigation are rotations, around the up axes, and zooms (in and out). This limitation in degrees of freedom for navigation is due to the way the intermediate images are computed, say affine transforms of panoramic images [13]. Warping techniques overcome this drawback using 3D information.

The pixels of the warped image are computed by re-projection of the pixels of key images (see figure 1). Warping is only possible if depth information is available for each pixel of the key images [20], [18]. Equation 1 is the general equation of 3D warping, where image 1 is the key image and image 2 the wrapped one.

$\dot{C}_1$ and $\dot{C}_2$ are the centers of projection of the 2 images, $P_1$ and $P_2$ are the inverse projection matrices. These matrices define the intrisic parameters of the camera (see figure 1).

$$\frac{\bar{u}_2}{\delta(\bar{u}_2)} = P_2^{-1} P_1 \frac{\bar{u}_1}{\delta(\bar{u}_1)} + P_2^{-1}(\dot{C}_1 - \dot{C}_2) \tag{1}$$

$$with\ \bar{u}_1 = \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} \quad and\ \ \bar{u}_2 = \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} \tag{2}$$



$\dot{X}$
$\vec{a}$
$\vec{b}$
$\vec{c}$
$\bar{x}_2$
$\dot{C}_2$
$\bar{x}_1$
$\dot{C}_1$
$P_2 = \begin{bmatrix} ax & bx & cx \\ ay & by & cy \\ az & bz & cz \end{bmatrix}$

Figure 1: 3D warping of a point $\dot{X}$ and intrisic parameters of the camera $P_2$

In the equation 1, $\delta(\bar{u})$ is the disparity of pixel $\bar{u}$. $\delta(\bar{u}) = \frac{S}{z}$, where $S$ is the focal length and $z$ the depth of the pixel. During re-projection on image 2, several 3D points can reproject onto the same pixel. To solve this visibility problem without using a Z-Buffer, McMillan [20] proposes to warp each image in occlusion compatible order.
Warping generates exposure errors. Exposure errors occur when the motion of the viewpoint reveals regions of the model that were not seen in the reference images. To solve this problem, the use of several and well chosen reference images is necessary (camera placement). Layered Depth Images (LDI) combine together a number of reference images [25] [23]. Multiple pairs (color, depth) are associated whith each pixel of an LDI. Advantages of LDIs are that they naturally avoid redundancy between reference images, they can be warped in occlusion compatible order and are capable of reducing the number of exposure errors. The main drawback of LDIs methods is that they are demanding in memory size. Interesting strategies aiming at choosing pertinent reference images have also been proposed in [10]. There are described in section 2.3.

The closest related work is by Hudson and Mark [14] who also propose to use IBR techniques in the case of a client / server approach. They propose an algorithm for reference camera placement. This fundamental problem as well as related works are discussed in section 2.3.

## 2.3 Camera placement: Determining reference images

Camera placement consists in covering every visible surface with a minimum number of cameras to avoid exposures and occlusions when using IBR. This problem can also be regarded as an extension of the Art Gallery Problem. The formal solution to this problem is the aspect graphs [24] that store all the visibility relations between all the objects in the scene. An aspect graph contains all the visual events (exposures, occlusions) that occur in a scene and that can therefore be used to pre-render images from cameras whose locations depend on the visual events.

The solutions to the problem of camera placement depend on the applications. When the application is 3D-recontruction a huge set of images are captured by for example turning a camera around the 3D object [21] or using few significant images [7]. When the application is navigation, the cameras that produce the reference images can be placed at certain predefined positions [18] [14] or the reference images can be once for all pre-computed according to a certain strategy [10], [1].

In [14] three sets of depth images are used. Each set contains four images that form the faces of a cube. These faces are the image planes of four cameras positioned at the cube's center. The current navigation camera lies within a triangle whose vertices are the centers of three reference cubes. This allows to efficiently warp the current navigation camera. The targeted application of [14] is walkthrough of indoors environments. Unlike this method, our objective is walkthrough of city models. With this aim in view, we propose a new camera placement strategy well suited to urban scenes.

Our method is inspired by Fleishman's work [10]. The scene is divided into viewing areas. In each viewing area, a small (not necessary minimal) set of cameras samples is chosen. Our camera placement algorithm takes two considerations into account: (i) every polygon should be covered (say, seen by at least one reference camera), (ii) every covered polygon should be covered at a sufficient coverage rate. A coverage rate of a polygon is the ratio between its area and that of its projection onto the image plane of the reference camera that sees this polygon. To avoid redundancy, only one camera is associated with a given polygon in the scene. While in [10] cameras are placed on the boundary of arbitrary walking zones, we place reference cameras on the urban street network. The placed reference cameras will capture images similar to those a pedestrian could see while walking along a street.

## 3 Overall client-server architecture

The overall client-server architecture and the global algorithm of our system is described in figure 2. The server owns the 3D environment and runs a camera placement algorithm capable of determining pertinent reference cameras that capture reference images of the environment, used for image-based rendering. The camera placement algorithm as well as the associated data structures are described in the next section. The reference images, once computed by the server, are sent to the client that warps them to compute new intermediate images.

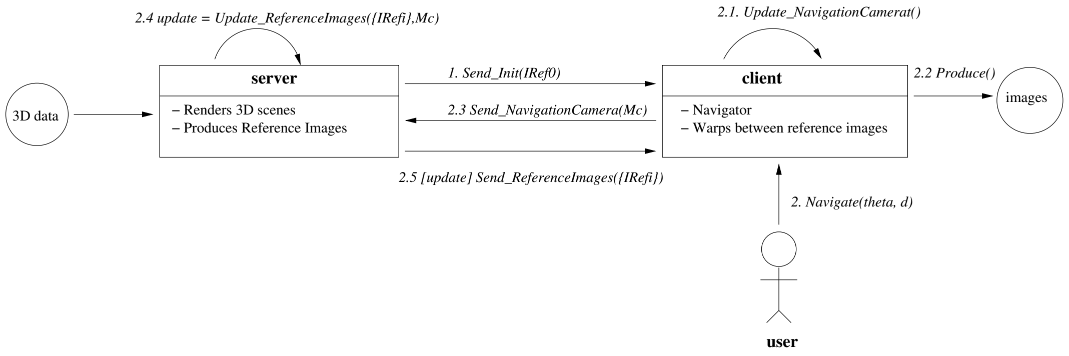

2.4 update = Update_ReferenceImages({IRefi},Mc)
2.1. Update_NavigationCamerat()
3D data
server
– Renders 3D scenes
– Produces Reference Images
1. Send_Init(IRef0)
2.3 Send_NavigationCamera(Mc)
2.5 [update] Send_ReferenceImages({IRefi})
client
– Navigator
– Warps between reference images
2.2 Produce()
images
2. Navigate(theta, d)
user

Figure 2: Overview of the rendering architecture

Here is a more detailed description of the figure 2:

1. The IBR process is initialized when the server sends the client an initial reference image together with its corresponding camera parameters (`1.Send_Init(IRef0)`). From now on, `IRef` represents a reference image and its corresponding camera parameters.
2. On the client side, the user can navigate through the 3D environment by changing the orientation and the position of the camera (`2.Navigate(theta,d)`). In the present application, navigation is performed in a urban environment. The position of the camera is constrained to lie on a horizontal plane. Sudden changes of camera orientation are not allowed. These few degrees of freedom of the camera limit changes between two successive images to make the IBR approach possible. These limitations are also coherent with the way people walk in a city.
3. Whenever the user moves the navigation camera, the client computes a new image by warping some of the available reference images (`2.1.Update_NavigationCamera()`, `2.2.Produce()`). The available reference images are not always appropriate for warping, that is to say the reference cameras that produced the available reference images can be too far from the current navigation camera, which may cause the appearance

of holes on the warped image. These holes are due to sub-sampling or exposures. For the sake of simplicity of implementation and rapidity of execution, we use adaptive blurring filters to fill the appeared holes.

4. To maintain an appropriate set of reference images on the client side, the client transmits the parameters of the new current navigation camera to the server whenever the user moves the camera (`2.3.Send_NavigationCamera(Mc)`). The set of reference images, available on the client side, is appropriate if each reference image of this set significantly contributes to the construction of the warped image. The contribution of a reference image is measured as the percentage of pixels of the reference image that re-project onto the warped image.

5. The server owns the 3D urban scene and a set of edges that define the geometry of the street network. Depending on the current navigation camera `Mc` on the client side and on the reference images previously sent `IRefi`, the server is able to determine whether the reference images, available on the client side, have to be updated or not (`2.4.update=Update_ReferenceImages({IRefi})`). A reference image has to be replaced (hence there is a need for updating the available set of reference images) on the client side when it is not appropriate, say when it does not significantly contribute to the construction of the warped image. If some updates are necessary, the server sends the client new reference images (`2.5.[update]Send_ReferenceImages({IRefi})`). The way the cameras are positioned in the environment and the way the server selects them to compute reference images are provided by the camera placement algorithm. This latter is described in detail in section 4.

**Communication protocol** In a client/server architecture, the data transmission time as well as the synchronization between a client and the server are crucial when the objective is to maintain a satisfactory interactivity on the client side.

While navigating through the virtual world, the client (PDA) continuously sends the server the position and the orientation of the new current navigation camera. According to the camera placement algorithm, the server decides if the client needs new reference images, then sends them to it if necessary. In order to prevent blocking communications, each process is divided into two threads as depicted on figure 3:

- The client sends the position and the orientation of the new current camera to the server.
- The server receives the current position and orientation of the new current navigation camera from the PDA, then wakes up its camera placement thread.
- If the camera placement algorithm decides that the client needs new reference images, then it sends them to it.
- The PDA receives the set of newly rendered reference images.
  Since it may receive images at any time, the thread downloading reference images from

the server and the thread, in charge of rendering and warping, share two sets of images; when the reception of the reference images is completed the two sets are swapped. In this way, the rendering and warping process is not blocked when new reference images are being downloaded..

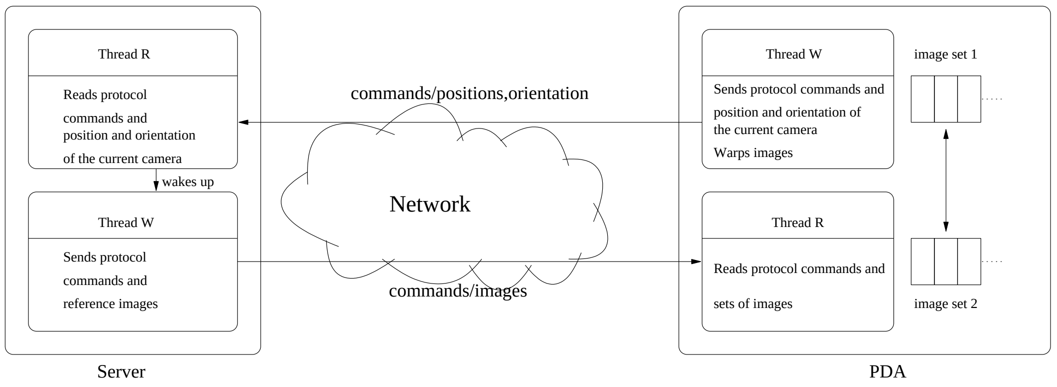

Thread R
Reads protocol commands and position and orientation of the current camera
wakes up
Thread W
Sends protocol commands and reference images
Server
commands/positions,orientation
Network
commands/images
Thread W
Sends protocol commands and position and orientation of the current camera
Warps images
image set 1
Thread R
Reads protocol commands and sets of images
image set 2
PDA

Figure 3: Client and server communications are handled by two threads.

The reference images are the main bottleneck of communications between the PDA and the server. In order to reduce the time of image transfer, the color and depth data of the reference images are compressed using the `zlib` library[26]. Each reference image requires 300kb for a standard 320×240 resolution; the use of `zlib` reduces the image size from 60 to 80 %.

## 4 Selecting the reference images on the server side

A common solution to camera placement (to compute reference cameras), in the context of IBR, is to place a set of reference cameras around the current navigation camera. The reference cameras can produce a cube of images centered at the position of the current navigation camera or can be positioned on a predefined view trajectory. These solutions do not take into account the topology of the environment. In our particular case of an urban environment, the topology is represented by the street network. To visually cover all polygons with a sufficient coverage rate, we position cameras on the street network which corresponds to viewing areas. A street network can be recovered from building footprints using Voronoï graph [8]. Once the street network is found, cameras can be positioned and oriented on it in order to visually sample the scene (see section 4.2). During navigation, the server selects reference cameras on the streets network to compute reference images required by the client for warping (see section 4.3).
The method, we propose to select reference cameras on the street network, offers two main advantages: (i) limitation of redundancies between reference images, (ii) limitation of the number of reference images. When the user walks in a street (for example between two buildings), only two reference cameras are necessary. These advantages allow to limit the amount of data transmitted from the server to the client and offer a greater autonomy to the client.

### 4.1 Streets network extraction

The algorithm that extracts the street network from building footprints is similar to Décoret's one [8]. Since the navigation environment is a city characterized by its building footprints, our method operates on a 2D horizontal plane that contains the buildings footprints. Each building footprint is a convex polygon composed of a set of edges (see in figure 4). In figure 4, the building footprint $B_1$ [1] is composed of four edges $E_1$, $E_2$, $E_3$ and $E_4$ . The geometry of the street network consists of a set of street fragments (edges) delimited by building edges. In figure 4, a street fragment *sf1* has been found between edges $E_1$ and $E_5$. As described in [8], these street fragments are obtained from the Voronoï diagram of the Delaunay triangulation of the sampled building edges.

### 4.2 Camera placement

Once the street network has been extracted, it is used to place the reference cameras. Because not all the building edges lie along a street we consider two kinds of reference camera:

1. **Free cameras**. Free cameras are not linked to any street. When a building's edge is not bordered by a street we consider that there is enough space in front of it to position a camera that looks toward it (see second picture in figure 5). In this configuration

[1] for language convenience we will write `building` instead of `building footprint`

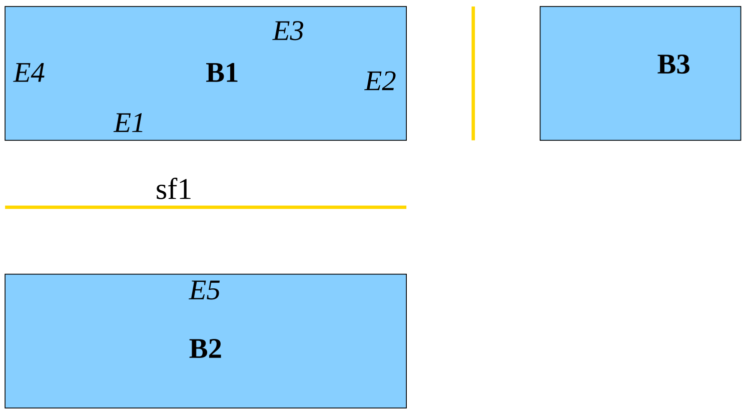


Figure 4: Street network extraction from buildings footprints

the visual coverage rate (quality measure inspired by the notion of coverage quality described in [10]) is high. When the building is totally isolated, cameras can be positioned all around it.

2. **Street cameras**. When a building is close to a street, cameras are placed on the street axes and oriented parallel to the street direction. In this configuration the visual coverage of the building edges that border the street is not optimal but the view is coherent with the view of a walker that navigates through streets (see in figure 5).

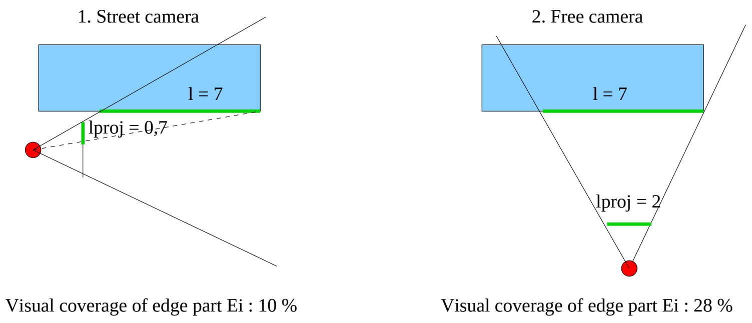

1. Street camera
l = 7
lproj = 0,7
Visual coverage of edge part Ei : 10 %
2. Free camera
l = 7
lproj = 2
Visual coverage of edge part Ei : 28 %

Figure 5: Visual coverage rate for a street camera and a free camera

The visual coverage rate $vc$ of an edge is the ratio between the length $l$ of the edge as seen by a camera and the length of its perspective projection $l_{proj}$ ($vc = \frac{l_{proj}*100}{l}$). Figure 5 illustrates this concept when visualizing the same edge from two different positions and orientations of a camera: the visual coverage of a street camera is lower than the visual coverage of a free camera. In figure 5, the visual coverage rate of the street camera is $\frac{0,7*100}{7} = 10\%$, while the visual coverage of the free camera is $\frac{2*100}{7} = 28,56\%$.

Reference cameras can be positioned either on a street fragment or in a free region. The selection process will be described in the next section. We describe here the placement process.

**Preliminary definitions.** Because our approach operates on a horizontal plane, the cameras are linear and their view frustum is defined with edges instead of planes for a 3D camera (see figure 6). As illustrated in figure 6, a camera $Cam_i$ is defined by its center of projection $C_i$, its look-at direction $\vec{at}$, its aperture $\theta$, its focal distance $f_c$, its near edge $E_{near}$ and its far edge $E_{far}$. The projection plane (edge) coincides with the near edge. The length of the near edge is $l_n$ while that of the far edge is $l_f$.

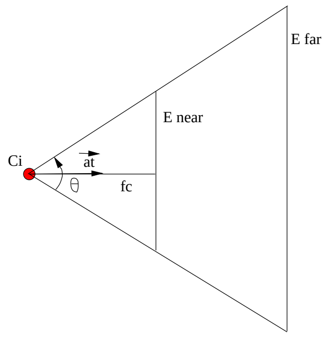


Figure 6: Linear camera definition

A street fragment $sf$ is defined by a direction $\vec{d_s}$. A building edge is defined by a normal $\vec{d}_{\perp e}$ and a length $l_e$.

**Placing street cameras on a street fragment.** Street cameras are placed along street fragments (at a certain height corresponding to that of a human eye) so that they can see the maximum of building facades that border the streets.

The center of projection (COP) $C_i$ of a reference camera is placed onto the street fragment $sf$ (see figure 7).

In Figure 7, the navigation (or warping) camera $C_w$ [2] sees the edge $E_1$ from point $P_1$ to point $P_2$. By construction, we designate $P_1$ as the closest to $C_w$. The reference camera $Cam_i$ is positioned so that it sees the edge $[P_1, P_2]$. The reference camera look-at direction

[2]For convenience, we often designate a camera by its center of projection

$\vec{at}$ is $\vec{d_s}$ if the dot product between $\vec{P_1P_2}$ and $\vec{d_s}$ is positive, to $-\vec{d_s}$ otherwise. Let $P_{ts}$ be the projection of $P_1$ on $sf$. In fact, $P_{ts}$ helps positioning a reference camera. In the simplest case, the COP of the camera $Cam_i$ can be positioned at : $C_i = P_{ts} - \vec{at} * f_c$. This equation is valid if the aperture angle $\theta$ of the camera is wide enough, if not (see figure 7), the camera has to be moved backward to see the point $P_1$: $C_i = P_{ts} - \vec{at} * f'$. The new distance between the point $P_{ts}$ to be seen and the camera COP is $f' = \frac{\|[P_1, P_{ts}]\|}{\tan(\theta/2)}$ .

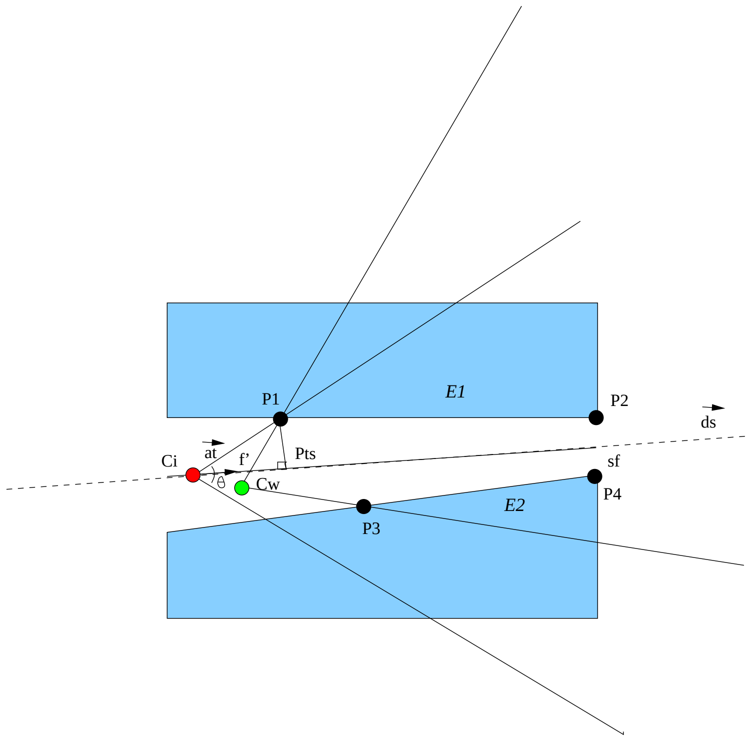


Figure 7: Street camera placement on a street line

In figure 7, the edge $E_2$ is also seen by $C_w$ from point $P_3$ to point $P_4$. Another reference camera $C_{i+1}$ will also be placed. The management of this set of cameras is described in the next section.

**Free camera placement** Some building edges are not associated with any street. In this case, the strategy of camera placement is similar to that used for street fragment. The difference is that the viewing direction of the reference camera is perpendicular to the edge building as seen by the navigation camera and passes through its middle point. The figure 8 illustrates three distinct configurations for the placement of free cameras :

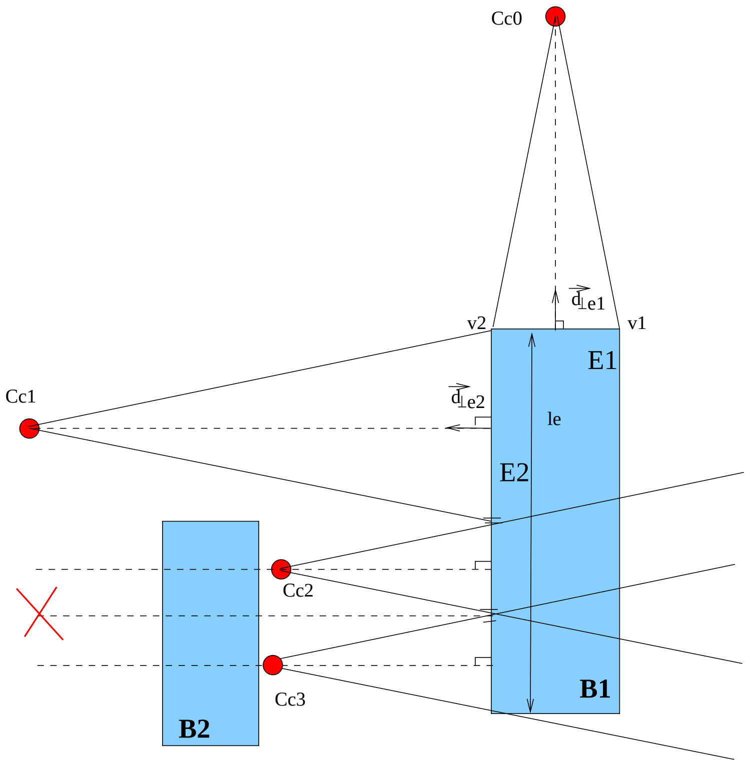


Figure 8: Free camera placement around a building $B_1$

1. The length $l_e$ of the building edge to be seen is smaller than $l_f$ (length of the far edge of the camera view frustum). Only one camera is necessary to see this edge. This camera looks perpendicularly to this edge and its view direction passes through the middle of this edge. On figure 8, the camera $C_{c0}$ sees the whole edge $E_1$ of building $B_1$.

2. The length $l_e$ of the building edge to be seen is greater than $l_f$. The building edge is divided into $nb$ regular parts, with $nb = \frac{l_e}{l_f} + 1$. On figure 8, the edge $E_2$ is first divided into two parts and the camera $C_{c1}$ sees the first part of edge $E_2$.

3. There is an occluder between the desired position of the camera's COP and the edge to be seen. In this case, the edge to be seen is subdivided again. This process is recursively repeated until there is no occlusion problem between the placed cameras and the considered edge. On figure 8, the cross symbolizes a desired position of a camera that was occluded by the building $B_2$. The cameras $C_{c2}$ and $C_{c3}$ have been determined after having sudivided the lower part of edge $E_2$.

**Replacing free cameras by street cameras.** Even if free cameras offer some best visual coverage rates than street cameras, the drawback is that more cameras are necessary to see the same area. For example in figure 8, three cameras are necessary to visually cover the edge $E_2$ whereas only one street camera positioned along this edge would be sufficient. We then propose to construct virtual streets around isolated edges. This solution allows to manipulate only one kind of camera and facilitates the process of camera selection (see section 4.3).

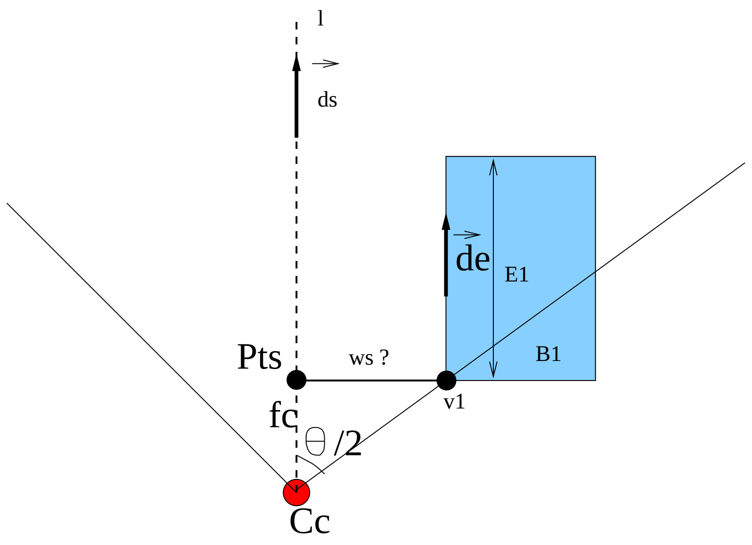


Figure 9: Positioning street cameras around isolated buildings

Indeed, in figure 9, we want the edge $E_1$ of the building $B_1$ be seen with street cameras rather than with free cameras. To this end, a street line $l$, parallel to the edge $E_1$, is created. The distance $w_s$ between the line $l$ and the edge $E_1$ depends on the focal $f_c$ and the aperture $\theta$ of the camera. As for street cameras placement, a point to be seen $P_{ts}$ gives the relative position of the camera's COP on the line $l$ ($C_c = P_{ts} - \vec{d_s} * f_c$). The distance $w_s$ between the line and the edge is calculated so that the camera sees $v_1$ (first vertex of $E_1$) on its projection plane. This gives the following expression for $w_s$: $w_s = tan(\theta/2) * f_c$.

## 4.3 Selection of reference cameras during navigation

The navigation (or warping) camera $Cam_w$ is controlled on the client side by the user. Whenever this camera is moved, a new image is produced on the client side by warping available reference images. Reference cameras are placed on the server side which uses them to compute reference images. The server uses the set of street fragments to place reference cameras on it. For each navigation camera $C_w$ the reference cameras $C_i$ are placed so as to see at least what $C_w$ sees.

Here is the algorithm for the determination of the set of reference cameras $C_i$ that correspond to a given camera navigation $C_w$:

1. **Determination of $\mathcal{V}$, the set of visible edges**. To determine the portions of edges $\{E_i, P_i, P_{i+1}\}$ ($E_i$ is one edge and $P_iP_{i+1}$ one portion) that are seen by $C_w$, rays are drawn from its COP to find intersections with building edges. For example in figure 10, the edge $E_2$ is totally seen by $C_w$; the triplet $\{E_2, P_3, P_4\}$ is inserted into the visible set. The edge $E_1$ is partially seen; $\{E_1, P_1, P_2\}$ is inserted into the visible set of $Cam_w$. The edge $E_3$ is partially seen ; $\{E_3, P_3, P_5\}$ is inserted into the visible set.

2. **Determination of $\mathcal{S}$ the set of street cameras**. For each visible edge within the set $\mathcal{V}$, a street camera is constructed as described in the previous section. In figure 10, $C_0$ is placed so as to see the edge $[P_1, P_2]$ on $E_1$. $C_0$ also sees all the portions of edges as seen by $C_w$ and is therefore a good candidate, it will be chosen as a reference camera. In the example of figure 10, three street cameras will be placed so as to see the portions of the edges $E_1$, $E_2$ and $E_3$. There are not all drawn on the figure for the sake of clarity.

3. **Reduction of $\mathcal{S}$**. If all the portions of edges $\{E_i, P_i, P_{i+1}\} \in \mathcal{V}$ seen from one camera $C_i \in \mathcal{S}$ have been already seen by another camera $C_j \in \mathcal{S}$, the camera $C_i$ is removed from $\mathcal{S}$. In the example illustrated in figure 10, the camera $C_1$ will be removed from the set because it only sees $\{E_2, P_3, P_4\}$ which is already seen by $C_0$. The camera that sees the portion of edge $E_3$ will also be removed. The final set of reference cameras $\mathcal{S}$ will only contain the camera $C_0$.

For each new warping camera it receives (see message event `2.3.Send_camera(Mc)` on figure 2), the server checks if the set of reference images he previously sent to the client is appropriate (see `2.4.update=Update_ReferenceImages({IRefi})` on figure 2). Recall that a reference image is appropriate if it significantly contributes to the production of the warped image on the client side. If the portions of edges seen by the current warping camera are not seen by previous reference cameras, they are updated, say the server sends new images to the client.

## 5 Implementation and results

In this section we give more details about the implementation of the server. Some aspects of the implementation on the client side are discussed in section 5.2.

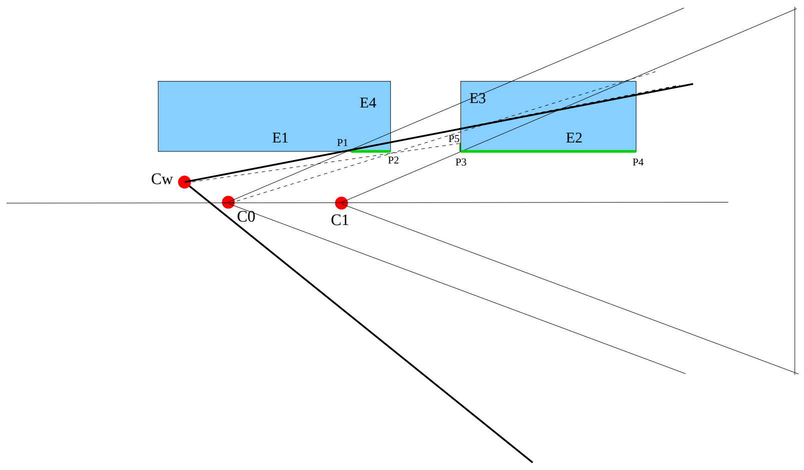


Figure 10: Selection of reference cameras.

## 5.1 Server data structures

When the server receives the position and the orientation of the current navigation camera from the client, it performs several tasks :

1. It computes the set of building edges which are visible from the received current navigation camera (section 5.1.1).
2. It determines if the previously sent reference images are sufficient to view all these visible edges within this set; if yes then the server does nothing else (section 5.1.2).
3. If no, then, as explained in the previous sections, the server assigns each building edge a camera with an ad-hoc position and orientation (section 4).
4. Finally at most four reference cameras are selected and the associated reference images are computed and sent to the client. In order to limit the network bandwidth as well as the computations required on the client side, we restrict the number of reference images. We use an empiric (but relatively natural) criterion to select the reference cameras (section 5.1.3).

### 5.1.1 Visible building edges

Given a camera we need to compute the visible building edges of the urban environment. Each building of the environment defines a footprint which is encoded by a list of 2D edges. To determine which edges are visible from a given camera we use a quadtree (as depicted on figure 11).

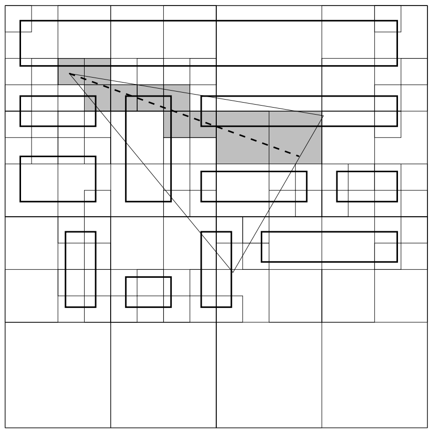


Figure 11: A quadtree used to compute visible edges of building edges

Each leaf of the quadtree contains one edge vertex and the list of building edges sharing it. In order to determine the set of visible edges, the view frustum is sampled by shooting rays. Recall that we are using linear cameras. The projection edge (or image edge) is subdivided into linear pixels. Rays originate at the COP of the camera and pass through the pixels. For each ray we compute the leaves of the tree that it crosses. Then, starting from the leaf containing the camera COP, we look for the closest edge intersecting the ray.

Each edge is parameterized by $P = P_0 + tP_1$, where $t \in [0, 1]$ and $P_0$ and $P_1$ are the vertices (endpoints) of the edge. Each intersection point $P$, between a ray and an edge, is represented by its associated $t$ value. An edge may be intersected by several rays, then yielding intersection points whose associated $t$ values range from `tmin` to `tmax`. For each intersected edge we store its associated `tmin` and `tmax` values which represent the visible portion of the edge. Since the projection edge of the camera is discretized into pixels, some edge endpoints could be missed (see figure 12). So, if we detect a *gap* between two consecutive rays (for example the two rays intersect different edges) then the parameter $t$ is set to its extremal value 0 or 1 (w.r.t the concerned endpoint of the edge).

### 5.1.2 Camera update checking

At any time the server knows the camera parameters corresponding to the most recent reference images already sent to the client, say the PDA. When the server gets the position and orientation of the new current navigation camera $C$, it determines if it is necessary to send new reference images to the client. To this end, it computes the edges visible to $C$. Then it checks if all these edges are seen by the reference cameras (for which reference images have been computed) already stored on the PDA.

This test is achieved by computing, for each stored reference camera $C_i$, its set of visible edges. It consists in checking, for each edge whose portion $[tmin, tmax]$ is visible to $C$, if

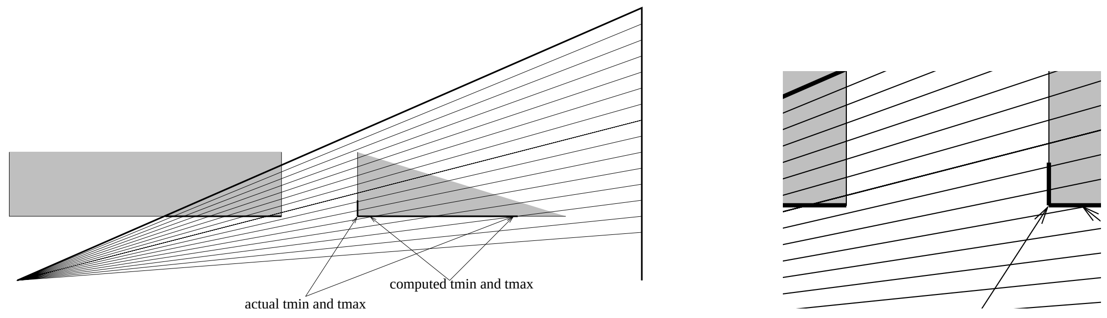


Figure 12: Intersection of rays with building edges. On the left : the sampling of the projection edge and the intersection of rays with buildings. On the right : edge endpoints are missed by the rays.

there exists at least one camera $C_i$ that sees the portion $[tmin', tmax']$ of the same edge such that $[tmin, tmax] \subset [tmin', tmax']$

If there exists an edge whose portion $[tmin, tmax]$ is not entirely seen by a camera $C_i$ then a new set of cameras is computed for $C$ (see section 4).

#### 5.1.3 Contribution of a camera

Our algorithm computes one reference camera for each visible building edge. As explained in section 4, we check redundancies existing between cameras; this allows to remove useless redundant cameras. Unfortunately, this process does not prevent from determining too many cameras.

In order to limit the network bandwidth required as well as computations on the client side we restrict the number of reference cameras to 4. The criterion used to select reference cameras is their contribution to the construction of the warped image on the client side. The contribution of a reference camera is estimated as the sum of the lengths of building edges that it sees. Once the contributions are computed we keep the four cameras of highest values.

## 5.2 Results

### 5.2.1 Our client/server application

The figure 13 shows our client and server application. The upper-left window belongs to the server and shows reference images sent to the client. The lower-right window is also part of the server; it shows the footprints of the urban environment, street fragments attached to footprints and the position of cameras. One can see several cameras: the white camera is the current navigation camera, the green cameras are those sent to the client and the red cameras are those rejected by our algorithm (see 4 and 5.1.3). The upper-right window shows the warped image computed by the client. The PDA and the server communicate through a WIFI wireless network.

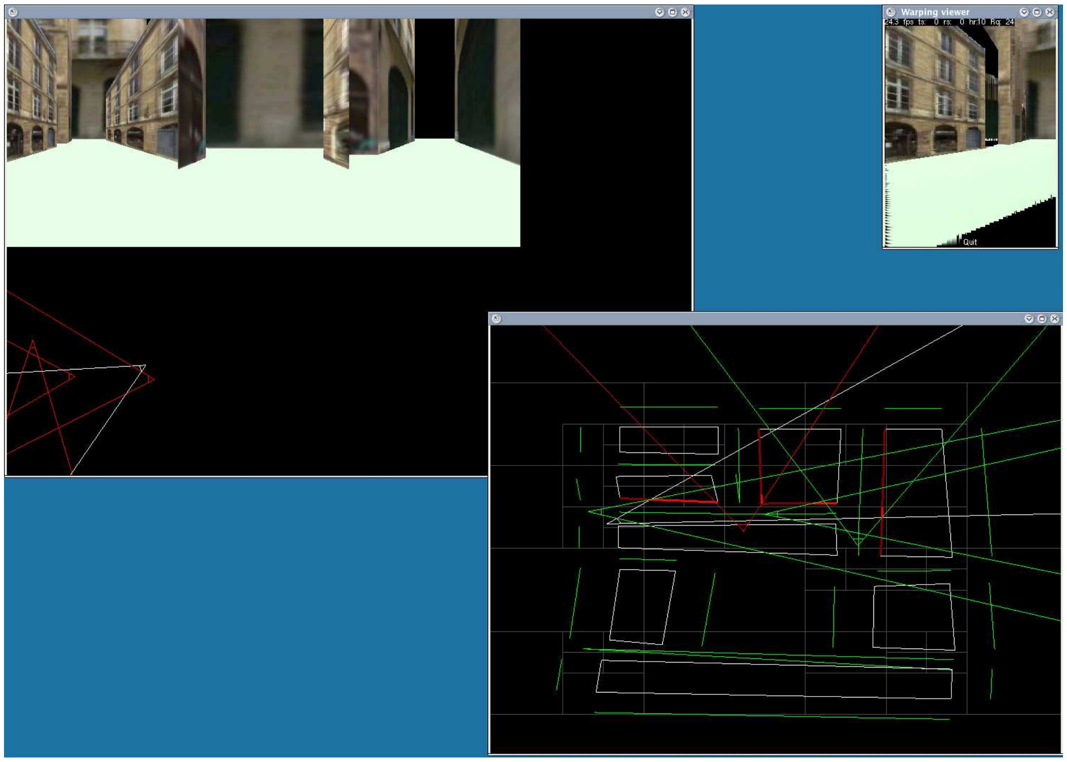


Figure 13: The client/server application : the upper part of the upper-left window shows the 3 reference images computed by the server and sent to the client; the lower part shows the positions of reference cameras (in red) and the position of the current navigation camera (in white) to be warped. The reference cameras are also shown in the urban environment on the lower right window; the frustums are shown using green color. Finally the upper-right window shows the result on the client side.

### 5.2.2 Test scenes

Our algorithm has been evaluated using three types of scene :

1. The first model is generated using a simple program. No texture is used. The scene represents a grid 100×100 of square units; several cubes have been added regularly on the floor. Some pictures of these scene are given in the figure 14. This model has been used to evaluate the warping algorithms.

2. The second type of scene is a VRML model generated with a procedural method written in Java. This procedural method generates simple buildings from 2D footprints and assigns a texture to each facade. We have generated several models of varying number of buildings. One can find pictures of such a model on the figure 13. The textures used in these models have a small resolution (249×281) and correspond to photographs taken from streets of Bordeaux (a city of France). This model has been used to evaluate the camera placement algorithm.

3. The third scene is a 3D model automatically generated from the cadaster of Bordeaux [11]. The generated model represents a surface of 4 km$^2$ containing 8712 buildings. With additional hand-made objects the model is composed of 108040 polygons. Textures are some photos taken from streets of Bordeaux. This model has been used to evaluate the warping algorithms.

### 5.2.3 Some remarks about floating-point and fixed-point arithmetics

We have implemented two warping algorithms. The first one uses floating-point encoding while the second uses fixed-point encoding. The latter has been implemented because a PDA is not supplied with a floating point unit (FPU).

The two main drawbacks of using fixed-point encoding are the following :

- In order to optimize the algorithm, some coefficients of the matrix used to backward and forward project pixels are partially precomputed with some camera parameters known in advance. These parameters are chosen in such a way the operations fit within the fixed-point precision allowed.

  The matrix used by our algorithm is the result of $M = (P_2 M_2 M_1^{-1} P_1^{-1})$ where $P_i$ are the projection matrix and $M_i$ the view matrix of the navigation camera and of the reference camera. To warp a pixel from an image to another we must add scaling $S_1$ and $S_2$ at both ends of the $M$ transformation. So, $M$ and $S_i$ depend on :

  - The camera position and orientation;
  - The camera frustum : near, far, left, right, bottom and top planes;
  - The image size : width and height.

  So, we have twenty parameters (10 for the two cameras). To optimize the algorithm we assume that :

  - The user moves only on a plane; so only 2 dimensions are taken into account for the position of cameras.
  - The reference and resulting images have the same size and are generated with the same camera frustum.

  By fixing the frustum parameters (in an empirical way w.r.t to the scale of the model) and the image size (which is the size of screen of the PDA), the warping matrix depends only on the position/orientation of the navigation camera and on the position/orientation of reference camera. Using `Maple` one can easily simulate the warping operations then estimate (coarsely) the bounds of each coefficient of the matrix and their ability to fit into our 16 bits fixed-point encoding.

- The 3D model must be chosen in order to fit into the fixed-point precision. In the previous point we saw that the warping matrix depends on the scale of the frustum, the position and the orientation of the cameras. These parameters partially depend on the scale and precision of the 3D model (e.g. the near and far plane).

### 5.2.4 Quality of the rendering

The quality of the warped images depends mainly on three parameters :

1. The difference between the position and orientation of the current navigation camera and those of the reference cameras. These differences strongly depend on the algorithm used for camera placement.
2. The number of reference cameras and the space they cover.
3. The algorithm used to fill, in the resulting warped image, the holes due to subsampling or exposure. We use a blurring filter.

Of course the two last parameters have a significant impact on the frame rate on the client side.

The figures 14 and 15 exhibit some problems concerning the quality of the rendering. The figure 14 shows that the quality of the result becomes unacceptable when a blurring filter is not applied to the warped image. The degradation of the resulting warped image is noticeable while the reference cameras lies close to the current navigation camera.

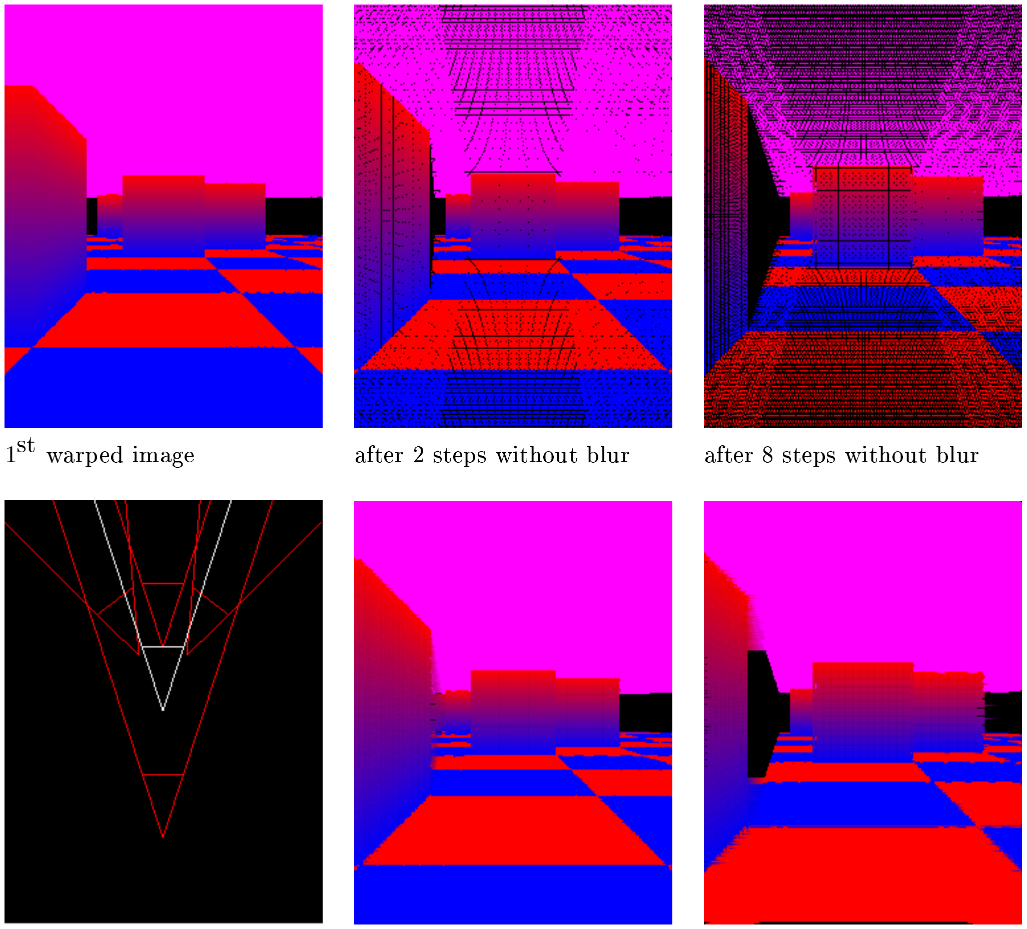
The figure 15 shows the influence of camera placement on the behavior of the fixed-point based warping algorithm. On this figure we used four reference cameras, two of them are perpendicular to the view direction of the current navigation camera. We can notice that artifacts appear on the resulting image. While these artifacts remain acceptable without blurring filter, they have a detrimental side-effect when applying a blurring filter.

### 5.2.5 Frame rate

The PDA used for our experiments is a Toshiba e800 (Intel PXA263 400 MHz, 128 Mo RAM) with an integrated WiFi antenna. We used the 320×240 resolution of its 4 inches TFT screen.

When using fixed-point arithmetic and 4 reference images, the best frame rate we obtained on the PDA is less than 4 fps (frames per second). Note that this result is totally independent of the 3D model considered.

When the server application and the client application are executed on two different PCs running under Linux and communicate through a local area network, the best frame rate



1st warped image

after 2 steps without blur

after 8 steps without blur

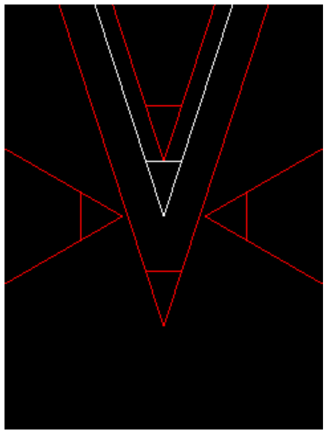
Position of reference cameras (red) compared to the first position of the user (white)

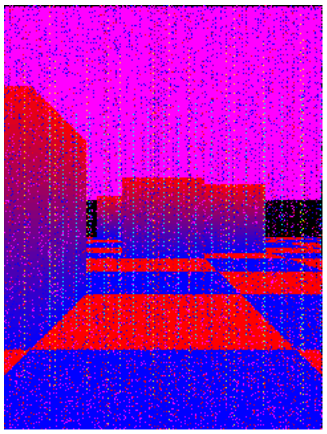
after 2 steps with blur

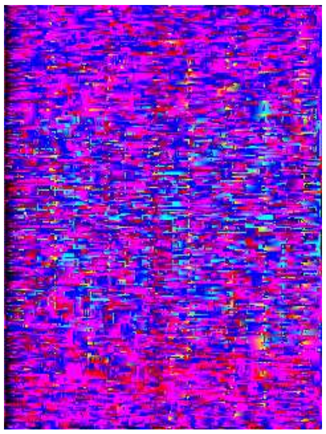
after 8 steps with blur

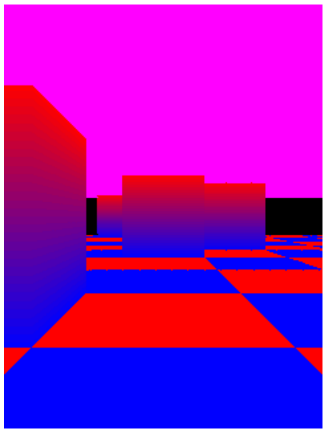
Figure 14: Image warping with or without blur filters

we obtained is around 20 fps. This result is obtained when the client application warps only one reference image. This frame rate falls to 8 fps when four reference images are used. These results correspond to average frame rates computed after multiple experiments. It is easy to conclude that the higher number of reference images used for warping, the lower values of the obtained frame rates. In other words, the warping of multiple reference images slows down our image-based rendering application. For this reason, the objective of our camera placement algorithm is to reduce the number of reference images, needed for


Position of reference cameras (red) compared to the first position of the current navigation camera (white)


1st warped image using fixed-point arithmetic and no blurring filter


1st warped image using fixed-point arithmetic and blurring filter


1st warped image using floating-point arithmetic and no blurring filter

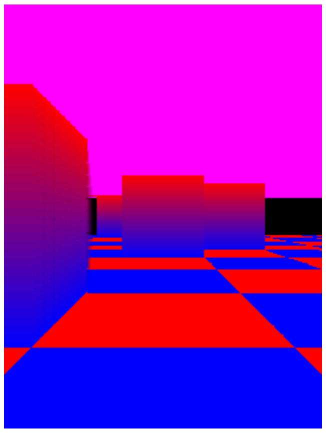

1st warped image using floating-point arithmetic and blurring filter

Figure 15: Image warping using fixed or floating point operations.

warping, to meet the constraint of interactivity while providing high quality warped images. After multiple experiments, an average of 3 reference cameras (providing 3 reference images) are needed for warping.

## 6 Conclusion

In this paper, we have shown that remote rendering of complex scenes is possible on a PDA. As the current PDAs are not equipped with FPUs we have resorted to fixed point arithmetic to speed up the warping algorithm running on the PDA. But as explained in section 5.2.3, the use of fixed point arithmetic limits the type of scene one can consider. In other words, it is difficult to encode a scene with a large disparity in the coordinates of the 3D objects making the scene. Our experience has shown that the presence of FPU in a PDA is crucial when one wants to remotely render outdoor and indoor scenes. We think that this problem will be solved in the near future with the advent of more powerful PDA supplied with FPUs (Floating Point Unit).

Another contribution of our work is our camera placement algorithm. It has proved its efficiency since it limits the number of reference images needed and allows warping without artifacts due to occlusions or exposures thanks to the judicious placement of reference cameras. Unfortunately, the obtained frame rates are still low because of the warping algorithm which is time consuming especially when the number of reference cameras needed gets high. This affects, unfortunately, interactivity which is our main goal. We are working on a faster version of the warping algorithm. We are sure that the constraint of interactivity will be met with PDA equipped with FPUs and with a faster version of the warping algorithm.

## References

[1] D. G. Aliaga and A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 307–316. ACM Press/Addison-Wesley Publishing Co., 1999.

[2] H. Biermann, A. Hertzmann, J. Meyer, and K. Perlin. Stateless remote environment and view compression. Technical report, Media Research Lab., Dept.of computer science, New York University, 1999.

[3] M. Brachtl, J. Slajs, and P. Slavik. Pda based navigation system for a 3d environment. *Computers and Graphics*, 25(4):627–634, 2001.

[4] C.-F. Chang and S.-H. Ger. Enhancing 3d graphics on mobile devices by image-based rendering.

[5] S. E. Chen. QuickTime VR — an image-based approach to virtual environment navigation. *Computer Graphics*, 29(Annual Conference Series):29–38, 1995.

[6] D. Cohen-Or, Noirmak, and T. Zvi. A server-based interactive remote walkthrough.

[7] P. Debevec, C. Taylor, , and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image- based approach. In A. SIGGRAPH, editor, *SIGGRAPH'96, Computer Graphics Proceedings*, Annual Conference Series, pages 11–20, New Orleans, Louisiana, August 1996.

[8] X. Décoret and F. Sillion. Street generation for city modelling. In *Architectural and Urban Ambient Environment*, 2002.

[9] J. Diepstraten, M. Görke, , and T. Ertl. Remote line rendering for mobile devices. In *IEEE Computer Graphics International (CGI)'04*, 2004.

[10] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, page 12. IEEE Computer Society, 1999.

[11] M. Hachet and P. Guitton. From cadastres to urban environments for 3d geomarketing. In *Proceedings of IEEE/ISPRS joint workshop on remote sensing and data fusion over urban areas (Urban 2001)*, pages 146–150, November 2001.

[12] D. Hekmatzada, J. Meseth, and R. Klein. Non-photorealistic rendering of complex 3d models on mobile devices.

[13] G. R. Hofmann. The calculus of the non-exact perspective projection. In *Proceedings of the European Computer Graphics Conference and Exhibition (Eurographics'88)*, Nice, France, 1988.

[14] T. Hudson and B. Mark. Multiple image warping for remote display of rendered images. Technical Report TR99-024, UNC-CS, 1999.

[15] Klimt - the open source 3d graphics library for mobile devices. http://studierstube.org/klimt/screenshots.php.

[16] F. Lamberti, C. Zunino, A. Sanna, A. Fiume, and M. Maniezzo. An accelerated remote graphics architecture for pdas. In *ACM/SIGGRAPH Web3D 2003 Symposium*, pages 55–61, 2003.

[17] Y. Mann and Cohen-Or. Selective pixel transmission for navigating in remote virtual environments. *Computer Graphics Forum*, 1997.

[18] W. R. Mark, L. McMillan, and G. Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–ff. ACM Press, 1997.

[19] I. M. Martin. Adaptive rendering of 3d models over networks using multiple modalities. Technical report, IBM T.J. Watson Research Center, 2000.

[20] L. McMillan. An image-based approach to three-dimensional computer graphics. Technical Report TR97-013, 19, 1997.

[21] R. Pito. A solution to the next best view problem for automated surface acquisition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(10):1016–1030, 1999.

[22] Pocket gl 3d library for pocket pc. http://pierrel5.free.fr/PocketGLb/ReadMeFirst.htm.

[23] V. Popescu, A. Lastra, D. Aliaga, and M. de Oliveira Neto. Efficient warping for architectural walkthroughs using layered depth images. In *Proceedings of the conference on Visualization '98*, pages 211–215. IEEE Computer Society Press, 1998.

[24] R. D. Schiffenbauer. A survey of aspect graphs. Technical Report TR-CIS-2001-01, Polytechnic University, Department of Computer and Information Science, Westchester, NY, 2001.

[25] J. Shade, S. Gortler, L. wei He, and R. Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM Press, 1998.

[26] zlib - A Massively Spiffy Yet Delicately Unobtrusive Compression Library. http://www.gzip.org/zlib/.

[27] C. Zunino, F. Lamberti, and A. Sanna. A 3d multiresolution rendering engine for pda devices. In *SCI 2003*, volume 5, pages 538–542.

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399