

**A RATE-DISTORTION FRAMEWORK FOR
TRANSMISSION AND REMOTE VISUALIZATION
OF 3D MODELS**

Pietro Zanuttigh

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI ELETTRONICA ED INFORMATICA
TESI DI DOTTORATO IN INGEGNERIA INFORMATICA ED ELETTRONICA
INDUSTRIALI

**A RATE-DISTORTION
FRAMEWORK FOR
TRANSMISSION AND REMOTE
VISUALIZATION OF 3D MODELS**

Pietro Zanuttigh

Contents

1	Introduction	5
2	Existing remote visualization techniques	11
2.1	Transmission of a triangular mesh with the corresponding texture . . .	11
2.2	Image-Based Rendering techniques	12
2.2.1	The plenoptic function	12
2.2.2	Rendering without geometry	14
2.2.3	Rendering with implicit geometry	17
2.2.4	Rendering with explicit geometry	17
2.3	Remote visualization by image transmission	18
2.3.1	Basic structure	19
2.3.2	Prediction of the views	21
2.3.3	Residual selection strategies	22
2.3.4	Compression of the transmitted data	25
2.4	Compression and transmission of 2D images with JPEG2000 and JPIP	27
2.4.1	Compression standard brief overview	27
2.4.2	Scalability features in JPEG2000	29
2.4.3	The JPIP transmission protocol	30
2.4.4	Client-Server Interaction in JPIP	31
3	A new approach to remote 3D visualization	33
3.1	Proposed scheme	36
3.2	Scene representation	38
3.3	Server overview	45
3.4	Client overview	47
4	Distortion-sensitive view synthesis	49
4.1	Warping of a single view	50
4.2	Warping from multiple views	54
4.3	Distortion-based selection of the stitching sources	60
4.4	Accounting for image compression distortion	60
4.4.1	Propagation of the distortion through DWT synthesis, warping and DWT analysis	61
4.4.2	Effects of extra-band energy	66
4.4.3	Distortion estimation with depth maps	69

4.5	Holes related issues	72
4.6	Accounting for Depth Uncertainty	78
4.7	Lighting related issues	83
4.8	Regularization of the selection choices	85
4.9	Performance issues	86
5	Distortion-sensitive geometry synthesis	89
5.1	Proposed approach	89
5.2	Holes related issues in depth estimation	94
6	Server policies	97
6.1	The server optimization problem	98
6.2	Optimization of Reinforcing Enhancements	103
6.3	Optimization of Disruptive Enhancements	106
6.4	Complete Server Solution	112
6.5	Geometry transmission policy	113
7	Experimental results and performance analysis	115
7.1	View fusion experiments	118
7.2	Geometry synthesis experiments	125
7.3	Server policy experiments	129
7.4	Performance analysis	134
7.4.1	Client performance analysis	135
7.4.2	Server performance analysis	139
8	Conclusions	141
	Bibliography	142

Abstract

This thesis presents a new approach to the remote visualization of 3D scenes based on a rate-distortion framework. The proposed interactive browsing environment is based on a client server couple where the server holds the scene description in the form of a set of views and depth maps. Both geometry and texture information is scalably compressed using JPEG2000 and can be progressively transmitted to the client as the interactive browsing goes on. The proposed system extends the JPIP standard, originally proposed for the interactive browsing of JPEG2000 compressed images, to achieve an efficient transmission of 3D scene information.

The user at client side can freely explore the scene from any viewpoint and direction. As soon as he requires a view of interest, the client application sends the viewing parameters to the server that decides how to optimally allocate transmission resources between the various elements of the scalably compressed image and depth maps bit-streams on the basis of the required view and of the data already transmitted to the client. The JPIP communication paradigm fits inside this framework rather well because it allows the server to form its own decisions regarding the best way to improve the synthesized views at the client, either by augmenting the quality of the images (or depth maps) already available at client side, or by sending information corresponding to new images or depth maps. Finally the client application combines the received information into the required rendering using 3D warping and multi-resolution image fusion techniques based on the wavelet transform. It can perform the rendering at any time exploiting the already transmitted data without waiting for the server communications.

A novel rate-distortion framework allows to estimate the distortion in the final rendered view. This model takes into account different sources of distortion, including the compression distortion in the received views, translational errors due to the geometry uncertainty and colours changes due to lighting and reflections. This framework will be exploited both to select the best source view for the samples in the rendered images at client side and to decide which elements of the compressed scene description need to be transmitted at server side. A two step real time optimization algorithm exploits this distortion framework to allocate the available bandwidth between the different images: the first step takes into account the effects of improving the images currently used to synthesize the required rendering, while the second considers the switch to new, better aligned, views. The two solutions are finally combined together to find the optimal solution to the selection of the information that needs to be transmitted. A prototype application has been built and used to experimentally validate the proposed approach.

Sommario

In questa tesi viene descritto un approccio innovativo per la visualizzazione remota di ambienti tridimensionali basato su un modello di tipo rate-distortion. L'approccio proposto è basato su un sistema di tipo client-server dove la descrizione della scena è memorizzata sul server sotto forma di un insieme di viste e mappe di profondità. Sia l'informazione sulla geometria che le viste sono compresse in modo scalabile con JPEG2000 e possono essere trasmesse al client in modo progressivo durante la navigazione della scena. Il sistema proposto estende le funzionalità dello standard JPIP, originariamente pensato per la trasmissione di immagini compresse in JPEG2000, in modo da poterlo utilizzare per trasmettere in modo efficiente ambienti tridimensionali.

L'utente può esplorare liberamente la scena da ogni direzione e punto di vista. Non appena richiede una vista di interesse, l'applicazione client invia i parametri della vista al server che decide come distribuire in modo ottimale la banda disponibile tra i diversi elementi dei bit-stream compressi relativi alle diverse immagini e mappe di profondità sulla base della vista richiesta e dell'informazione precedentemente trasmessa al client. Il modello di comunicazione usato da JPIP si integra molto bene in questo sistema perché permette al server di decidere in modo autonomo come migliorare la qualità del rendering al client, migliorando la rappresentazione di immagini (o mappe di profondità) già disponibili o invece trasmettendo dati relativi a nuove immagini o mappe di profondità. Infine l'applicazione client sfrutta l'informazione ricevuta dal server per generare la vista richiesta riproiettando le viste disponibili e fondendole insieme utilizzando tecniche multirisoluzione basate sulla trasformata Wavelet. Il client può effettuare il rendering in qualsiasi momento sfruttando l'informazione ricevuta in precedenza senza attendere l'arrivo di nuovi dati dal server.

È stato sviluppato un modello di tipo rate-distortion per stimare la distorsione nelle viste generate dal client. Questo modello considera diverse fonti di errore, come la distorsione introdotta dalla compressione con perdite nelle viste ricevute, gli errori sulla posizione dovuti ad imprecisioni nella geometria e l'effetto dell'illuminazione e delle riflessioni. Questa stima viene utilizzata sia dal client per selezionare la miglior vista da cui prendere ogni punto del rendering, sia dal server per decidere quali elementi della rappresentazione compressa della scena devono essere trasmessi. Un algoritmo di ottimizzazione in tempo reale diviso in due fasi sfrutta il modello della distorsione per allocare la banda disponibile tra le varie immagini. La prima fase considera il guadagno che si può ottenere migliorando le immagini attualmente utilizzate per ricostruire il rendering, mentre la seconda considera il passaggio a nuove immagini più vicine alla vista richiesta. Infine le due soluzioni sono combinate insieme per de-

cidere quali elementi della descrizione della scena devono essere trasmessi. La validità dell'approccio proposto è confermata dai risultati sperimentali ottenuti con un primo prototipo dell'applicazione.

Chapter 1

Introduction

Three dimensional representations have gained an increasing interest in the last years, but while a lot of research activity has been carried on how to build and render them, the remote visualization of this kind of data is still an open issue. These representation are usually associated to huge amounts of data and how to extract from these datasets the information needed to follow the interactive inspection of the scene is a problem posing many challenging conceptual and practical questions. The work presented in this thesis is concerned with the problem of efficient interactive retrieval and rendering of 3D scenes.

A first basic requirement to build an efficient browsing system is a good understanding of how transmission resources should be distributed between the different elements of the scene representation. This includes the optimal subdivision between texture and geometry, studied for example in [1], but also the definition of which elements of the scene description are more useful in the rendering of a particular view. Even if some answers have previously been provided for the case in which the entire 3D model, including all texture and geometry components, is to be transmitted over a bandwidth constrained channel (for example [2], [3]), in an interactive framework these issues must be solved taking into account also other aspects like the information transmitted to the client during the previous browsing.

The basic scheme of a remote 3D visualization system is made of a client-server couple, connected via a bandlimited channel: at client side, the user can freely inspect the 3D scene and interactively determines the particular view of interest. The server should provide the data needed to follow the user's browsing of the scene. Finally the client should exploit this information to show the required views. A key difference between this framework and other interactive applications such as video streaming is that the user is expected to navigate between a variety of different views, but we do not know in advance of time which views will be selected and how much time (transmission resources) the user will choose to devote to any particular view. Another critical issue is the presence of two completely different kind of data in the scene description, the geometry information and the texture. Some preliminary studies have been carried out in this field (see [4]) but how transmission resources should be distributed between texture and geometry information remains a difficult problem.

There exist several different remote visualization schemes, but none of them is

completely satisfactory. The most common approach is based on the standard representation of 3D scenes as a triangular mesh together with the texture information. This representation is very compact and has no redundant information, but at the same time it does not allow an easy random access to the scene description. The simplest solution consists in just transmitting from the server to the client a complete 3D model together with its texture and then performing the rendering at client side. Though very common, this approach suffers from poor response time, since visualization cannot commence until everything has been sent, and can require a very powerful client to render in an efficient way large datasets. The effectiveness of this scheme can be improved by compressing the 3D mesh descriptions and the textures, possibly exploiting scalable compression techniques. There are many methods to compress the geometry information, for example the approaches described in [5, 6, 7], but all these solutions aim more at obtaining an efficient compression of the complete scene description than at an interactive visualization system.

Another group of approaches is instead based on *Image Based Rendering* techniques. The basic idea behind these systems is to represent the 3D scene with a suitable collection of images. The representation of the scene as a set of images (sometimes associated with depth information) offers a straightforward way to access to subsets of the data. Unfortunately it also introduces a redundancy in the scene representation. To exploit efficiently this scheme it is also necessary to find an efficient way to combine the information from the finite set of available images into the rendering of a new arbitrary view. There exist many different IBR techniques, Section 2.2 contains a brief overview of them, but usually they introduce some constraints on the freedom of exploration or require the availability of a huge dataset of images.

Another possible solution (described in detail in Section 2.3) is to move the rendering engine at server side and transmit just the rendered images. However this approach introduces other drawbacks such as a huge workload at server side, high bandwidth requirements and a strict coupling between the server and the client that makes it highly sensitive to network congestion and delays. Predictive compression techniques can help in dealing with these issues, but the performances of systems based on this approach are still not completely satisfactory.

In the last years new highly scalable compression and transmission tools, such as JPEG2000 [8] together with its interactive protocol JPIP [9], have offered an efficient solution to the remote browsing problem for the simpler problem of bidimensional images. They allow to effectively browse huge images by downloading only the data really necessary for the required section or resolution of the image, and to avoid the transmission of redundant information by exploiting previously transmitted data to satisfy new requests.

In this thesis a new framework for interactive browsing will be proposed. It will extend some of the features and ideas behind these approaches from 2D images to a three dimensional framework. In the proposed scheme the server delivers incremental contributions from two types of pre-existing data: scalably compressed views of the scene and a scalably compressed representation of the scene surface geometry. Initially a representation of the geometry based on a triangular mesh will be considered but then it will be replaced by a set of incremental contributions from a group of scalably

compressed depth maps.

All the compressed information available at server side is divided in many contributions corresponding to different spatial regions at different quality and resolution levels exploiting the JPEG2000 scalability features. All these elements are transmitted in a progressive way while the interactive browsing goes on. It is important to underline that the server does not generate new views or compress differential imagery. Instead, it just determines and sends the elements from the compressed data that allow to obtain the best possible rendering of the required view at client side. The client application decompresses and stores the data transmitted by the server and then exploits the geometry information to combine the received information about the different views into the renderings required by the user. In the proposed scheme the synthesis of the required rendering can be performed by the client at any time exploiting the previously received information. This helps to decouple the client and server components of the system, thus making the system robust to network congestion and latencies.

An efficient implementation of the proposed approach should give an answer to two fundamental questions:

1. How should the client combine information from available original view images into a new view of interest, using the available description of the surface geometry?
2. How should the server distribute available transmission resources between the compressed elements of the various available views and the geometry information in order to obtain the best image quality at client side? Included in this question is that of whether the server should transmit elements from a new original view which is more closely aligned with the requested view, rather than refining nearby original views for which the client already has more data.

The geometry information allows to warp the available images to the viewpoint of interest, but a good answer to the first question requires also to find a way to select which images (or part of them) are going to be used to reconstruct the required rendering. The solution of the second question is different from the case of a straightforward compression of the whole scene description, and must be solved taking into account the previously transmitted data. For example if the client happens to request one of the original view images, it can be incrementally served directly from its scalably compressed representation. Interestingly, though, this might not always be the best policy. If the client has already received a good quality representation of one or more nearby original views it may be more efficient to send only the geometric information required for the client to synthesize the requested view from the available one and then send additional compressed information to further augment the quality of these nearby original view images. It follows that even if the server has a huge number of original view images, an efficient service policy would effectively subsample them based on the interactive user's navigation patterns. Exploiting the scalable compression features the server may even choose to send some elements from the requested view while expecting the client to derive other aspects of the view from the previously delivered view images.

In particular, we have developed a paradigm to estimate the distortion in the rendered views [10], [11]. The proposed scheme permits to map the various sources of distortion in the different available views and in the geometry description to the final rendered view. The distortion estimates offer a straightforward solution to the client issue, where the requested rendering can be reconstructed by selecting the contribution from the available data that minimize the final distortion. The same model can be also exploited in the answer to the server question, again by transmitting the elements from the available compressed data that minimize the distortion in the rendered views. It is worth mentioning that some answers to the second question posed above have previously been provided for the case in which the entire 3D model, including all texture and geometry components, are to be transmitted over a bandwidth constrained channel. In particular, Tian and AlRegib [2] extend an approach proposed by Balmelli [3], in which the bandwidth is divided between texture and geometry components of a global model on the basis of the optimization of a visualization objective. The key difference between this and similar approaches and the proposed framework is that, while in these formulations, the global visualization objective either explicitly or implicitly considers a wide range of viewing directions simultaneously, in the proposed approach we are concerned with the optimization of an interactive client's view of interest. Indeed the global perspective becomes increasingly unhelpful as the scene which must be navigated grows. Eventually it becomes unreasonable to expect that any individual client will ever visit more than a fraction of the scene. Moreover, interactive navigation means that the client may move very close to the surface at some instants and much further away at others; these clearly alter the optimal balance between texture and geometry information.

The works from Ramanathan and Girod [12],[12] on optimized server distribution policies for predictively compressed light fields are perhaps the most closely related to this work, even if Ramathan's approach does not rely on the availability of geometry information. Another key difference between that work and the one presented in this thesis is the distortion-sensitive rendering approach used at client side in our work.

Chapter 2 contains a brief review of the most common remote visualization techniques for three dimensional scenes and standard images. After the description of the most common image-based rendering techniques, a particular focus is given to the remote visualization by image transmission schemes, that represent the starting point for the development of the proposed approach. Finally the chapter contains a brief overview of the basic concepts behind JPEG2000 and its interactive protocol JPIP because they will be extensively used to achieve a compact and scalable representation of texture and geometry information in the proposed system. Chapter 3 presents a general overview of the architecture of the proposed remote visualization system and the basic building blocks of the client and server applications. Chapter 4 introduces the distortion estimation framework and explains how it can be used together with 3D warping and multiresolution stitching techniques to combine the different available views into the required renderings at client side. In Chapter 5 the framework introduced in the previous chapter is applied also to geometry information, thus allowing to reconstruct a single representation of the 3D scene from a set of depth maps. Chapter 6 focuses on the server issue and describes a policy to decide how to allocate the available transmis-

sion resources between the different elements of the available views. Finally Chapter 7 presents the experimental results obtained with the prototype of the system and a brief analysis of the computation requirements of the proposed scheme.

Chapter 2

Existing remote visualization techniques

In this chapter contains a brief discussion of the most common remote visualization schemes for three dimensional scenes. After a brief overview of the classic approach and of the most common image-based rendering techniques, a particular focus is given to remote visualization by image transmission and to the remote visualization of standard images using JPEG2000 and the JPIP protocol.

The image transmission approach is described in detail because it were the subject of previous research activity and the remote visualization scheme proposed in this thesis represents both an evolution and an answer to the limits of this approach. The JPEG2000 image compression standard and its interactive protocol JPIP allow an efficient remote visualization of 2D images. They have been included in this chapter because the approach to remote visualization presented in this work relies on the features of these standards to achieve an efficient progressive transmission of the scene information.

2.1 Transmission of a triangular mesh with the corresponding texture

The simplest and most common approach to remote visualization is based on the standard representation of the scene as a triangular mesh with the corresponding texture. This representation is compact and has no redundant information, but also does not allow an easy random access to subsets of the data. The most common remote visualization tools (for example VRML viewers like *CosmoPlayer* or the *Cortona VRML client*) just download the complete scene description in a suitable format (it can be a standard file type like VRML or X3D, or any proprietary format) and then perform the rendering at client side.

This approach has some drawbacks: the bandwidth required by complex scenes' description can be huge and it is difficult to guarantee a pre-assigned quality of service (e.g., starting latency less than a given time, etc.) because the amount of data to be transmitted depends on the size and characteristics of each scene; it sends the whole

model irrespectively of fact the user may only want to see a small portion of it; it uses inefficiently the communication channel, because the connection is never used after a burst of data sent at the beginning; complex scenes can require a powerful client to be smoothly browsed with the limited client resources; finally it is difficult to protect the copyright on the 3D data (when the user downloads the complete model the author loses the control over its diffusion).

The effectiveness of this scheme can be improved by compressing the 3D mesh descriptions and the textures. There are many compression tools for triangular meshes. Some of them allow also a progressive refinement of the scene description [13, 5, 7], but they aim more at an optimal compression of the complete scene description than at an interactive browsing framework. Texture information is usually represented by one or more standard images and can be compressed using any suitable image compression standard.

2.2 Image-Based Rendering techniques

The construction of a complete three dimensional representation of a real world scene is usually a long and complex task which requires ad-hoc instrumentation and a lot of manual labour. To avoid this task many researchers have tried to develop techniques to represent the 3D world as it appears to our eyes without reconstructing the complete scene geometry. *Image-based rendering* (usually abbreviated to *IBR*) techniques are based on the idea of rendering novel views from a set of available images of the scene without relying on a 3D geometry description. This allows to avoid the construction of a 3D model of the scene and to obtain a more photorealistic rendering. However the set of available views is limited for practical reasons and the critical problem in these approaches is how to reconstruct arbitrary views from the available images. Before describing the most common IBR techniques it is useful to introduce the concept of *plenoptic function*, that represents the general framework for all IBR approaches.

2.2.1 The plenoptic function

The starting point to deal with the problem of reconstructing arbitrary views of an environment is to understand the relationship between the light that fills the space around us and what can be seen by our eyes. Adelson and Bergen developed a complete representation of what our eyes are able to see from the world around them through the concept of *plenoptic function* [14]. They started from the idea that the eye could be approximated with a *pinhole camera* which selects all the rays of light passing through a certain point in the space. In other words the eye "reveals the structure of the pencil of light at the pupil's location". If we switch back to the pinhole camera and we consider a single black and white picture shot from it, it records the intensity of the light as it appears from a single viewpoint at a certain time, averaged over all the wavelengths of the visible spectrum. The recorded intensity can be represented as a two dimensional function of the viewing direction in polar coordinates $P(\theta, \phi)$. A colour picture adds the information on how the intensity depends on the wavelength,

and a movie adds the time dimension, thus leading to a 4D function representing all the information available to the eye in a certain point in space $P(\theta, \phi, \lambda, t)$. Finally if we consider this function in every possible point of the three dimensional space we obtain the complete form of the plenoptic function:

$$P = P(\theta, \phi, \lambda, t, V_x, V_y, V_z) \quad (2.1)$$

The information contained in this function is a complete representation of the visual world and permits to reconstruct every possible view at every position and instant of time. It is important to underline that the plenoptic function does not depend on the orientation of the eye, because it represents the complete 360° sphere of rays of light around the point (also the rays from behind) and by changing the gaze direction only their relative position is changed. The plenoptic function can be considered as the only link between physical objects and the human eye. In other words the objects fill the space with ray of lights which are represented by the plenoptic function and the eye takes samples of it.

Of course the information associated to this 7D function is huge and practical ways of representing it should be able to reduce the dimensionality and subsample in an efficient way the plenoptic function. An obvious simplification is to drop the time dimension by restricting to the case of static scenes. By switching from a multispectral representation to the simpler case of colour RGB images we can exclude another dimension, but we are still dealing with a 5D function.

Many different ways of representing and sampling the plenoptic function have been developed [15]. In fact the set of available images in image-based rendering techniques can be considered a sampling of the plenoptic function and the target of all IBR solutions is to reconstruct the plenoptic function from the available samples. This is a very difficult task and some techniques rely also on the aid of some form of geometry information, while others are able to reconstruct the plenoptic function only partially (this usually turns into some constraints in the browsing freedom). Every technique is characterized by a different trade-off between the amount of data associated to the representation and the accuracy of the possible reconstruction of the plenoptic function (see Table 2.1). Most techniques permits to reduce the number of dimensions associated to the representation, usually at the cost of introducing constraints on the freedom of exploration of the scene.

A common way to group the different techniques is on the basis of the amount of geometric information used for them. A possible solution [15] is to divide them into three main categories: the techniques that use no geometry at all; the ones that exploit some implicit form of geometry (the most common is a set of correspondences between samples in different views); and finally the ones which explicitly use some geometry, for example depth information associated to the samples in the images. The standard 3D representation based on texture and geometry can be considered part of this last group and in fact it is difficult to set a dividing line between IBR and the “classic” 3D rendering. However this grouping is not a strict subdivision and some techniques are difficult to characterize in this way. In the following sections there is a brief overview of the most important techniques.

Representation type	Year	# of dims	Data size	Geometry representation	Constraints
Plenoptic function	1991	7	Huge	None	None
Light field	1996	4	Huge	None	Outside bounding box
Lumigraph	1996	4	Huge	None	Outside bounding box
Concentric mosaics	1999	3	Medium	None	Bounding plane
Panoramic images	-	2	Small	None	Fixed viewpoint
View interpolation	1993	2	Small	Correspond.	Close to avail. views
View morphing	1996	2	Small	Correspond.	Close to avail. views
3D warping	-	2	Small	Depth values	**
Layered Depth Images	1998	2	Small	Depth values	**
View Dep. Texture Mapping	1996	2*	Medium	3D model	None
Standard 3D rendering	-	2*	Medium	3D model	None

*2D texture + geometry information **samples visible in available views

Table 2.1: Characteristics of the most common IBR techniques

2.2.2 Rendering without geometry

The techniques in this group allow to represent a three dimensional scene with only a set of images without any information on the geometric structure. This approaches permit to completely avoid the complex task of reconstructing the geometry, but usually the lack of information on the scene structure is compensated by using a huge set of images or by strict constraints on the freedom of exploration.

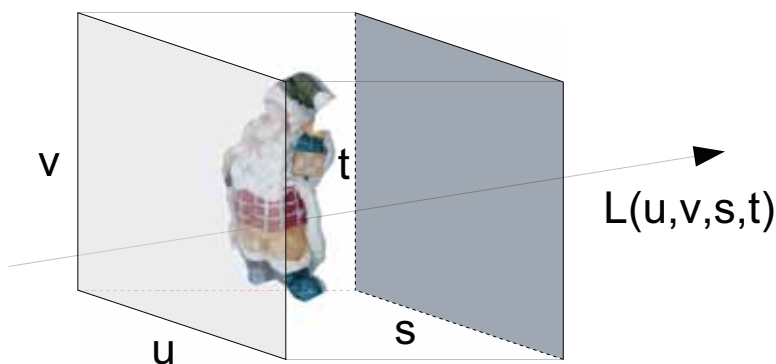


Figure 2.1: Parametrization of light fields

Light fields

An important observation about the rays of light in the plenoptic function is that the radiance does not change along a ray unless blocked. This permits to reduce the 5D plenoptic function (without time and spectral dependency) to a 4D function in regions free of occluders. If we consider that most geometric models are bounded the reduction

can be performed with the only constraint of placing the camera outside the convex hull (or simply the bounding box) of the object. Levoy and Hanrahan [16] represent this function by parametrizing the light rays with their intersection with two arbitrary planes (see Figure 2.1). By calling (u, v) the coordinate system in the first plane and (s, t) in the second, we obtain the 4D light field function:

$$L = L(s, t, u, v) \quad (2.2)$$

The function can be considered as a set of pictures of the (s, t) plane taken from every point on the (u, v) plane. For a complete representation of the plenoptic function inside the bounding box six of these couples of planes are needed. The light field can be built by rendering synthetic images from a 3D representation or from a collection of picture taken from a camera array. To achieve a good representation of the light field thousands of pictures are needed and how to acquire and compress the data is a difficult open issue. There exists many works on compression of the light fields (mainly based on vector quantization) and on how to randomly access them in an interactive browsing environment [12]. Another similar method, called *lumigraph*, and also based on the parametrization of the light rays over the surface of a cube has been introduced by Gortler et Al. [17].

Concentric mosaics

This representation, introduced by Shum and He [18] in 1999, is built by using a slit camera which moves on concentric circles and acquires a slit for every point on these circles. In this way we obtain a three dimensional representation (the dimensions are the radius, the rotation angle and the vertical elevation), in the form of a set of images, one for each value of the radius. Compared with light fields this is a more compact representation, and the acquisition phase is also simpler. At rendering time it is possible to reconstruct novel views by interpolating the different images obtained in this way. This representation allows the user to move continuously in a circular region with a correct representation of lighting and parallax effects. However in the rendered views there is always a vertical distortion, that is quite evident when the user moves forward and backward. It can be only partially alleviated by depth correction, as shown in [18].

Panoramic images

Panoramic images [19] offer a compact representation of what it is possible to see looking in every direction from a fixed viewpoint. It is a very common representation that has also been used in many commercial applications, like Apple's QuickTimeVR. This 2D representation of the plenoptic function is based on the idea that, unless the viewpoint changes, it is not possible to see the difference between a still scene surrounding the viewer and the interior of a closed shape textured with the same scene. The shapes used more frequently for panoramic image projections are cylinders, spheres and cubes.



Figure 2.2: Cylindrical panoramic image

Cylindrical panoramas are the most common solution. In these images the scene is projected on the interior of a cylinder with axis through the centre of projection and parallel to the imaging plane. The construction of this kind of panorama starts with the acquisition of a suitable set of photos (usually around 20) from a fixed nodal point. The images are then projected on the cylinder surface and merged together using ad-hoc software based on correlation methods. The resulting single image (for example the one in Fig. 2.2) covers all the 360° field of the view in the horizontal direction, but does not allow a complete freedom of observation in the vertical one (it is not possible to look directly up or down). When looking at panoramic images on a planar surface horizontal lines exhibit a characteristic distortion, and a warping procedure is necessary to reconstruct from them the required views. Cylindrical panorama viewers perform the rendering of panoramas simply by reprojecting pixel by pixel the section of panorama currently observed on a planar surface and perform some filtering operations to improve image quality.

Other projection types are also possible: the spherical projection allows a complete 360° field of view on the horizontal and vertical axis and a uniform sampling in every direction. It also requires more complex warping computations for the rendering and ad-hoc acquisition equipment and procedure. Cubic panoramas, present also in the last version of the QuickTime VR product, permit a complete look-around and are much faster to render than spherical panoramas due to the planar cube surfaces. In this approach the sampling is not uniform as in spherical panoramas, but the 6 faces of the cube represent a straightforward solution for the storage of the image. There are also other solutions, for example the panoramic visualization system we developed for the E.U. VITRA project (described in [20]) uses a prism projection to exploit hardware accelerated three dimensional rendering techniques in panoramic visualization. This solution is a trade-off between cylindrical and cubic panoramas that allows simple acquisition through the various cylindrical panorama authoring tools and fast visualization. In this approach and also in some visualization tools for cubic panoramas the shape where the panorama is projected on is considered as a 3D object with the panorama as its texture. This allows to perform the warping of the panoramic image as a standard 3D rendering, thus exploiting hardware acceleration of current 3D graphic cards.

2.2.3 Rendering with implicit geometry

In this set of approaches there is no explicit representation of the geometry. Instead they rely on a set of correspondences between samples in different views that allows to derive some information on the scene geometry.

View interpolation

This approach, introduced by Chen and Williams [21] allows to reconstruct arbitrary frames between a set of available views. It exploits morphing techniques to approximate the intermediate frames. However this techniques requires a set of correspondences between the two images to compute the morphing and works well only if the required image and the available ones are close. The correspondences can be usually computed using computer vision techniques such as stereo or feature correspondence, or from the depth values if they are available.

View morphing

Seitz and Dyer has developed a technique [22] based on image morphing that allows to reconstruct arbitrary views on the line joining the optical centres of two images. Their approach requires to know the projection matrices associated to the views and a set of correspondences between samples in the two images. In the first step the two images are pre-warped using the information on the projection matrices to form two parallel views. Then the morphed image is built by interpolating positions and colours of corresponding points. Finally the morphed image is warped to the required viewpoint.

2.2.4 Rendering with explicit geometry

In this family of methods (that includes also the traditional 3D rendering) the representation has also information about the geometry structure, in the form of depth associated to the samples in the views or of a standard 3D mesh representing the object surface.

3D warping

The information about the depth of the samples allows to reproject the points in an available view of the scene to nearby views. 3D warping techniques exploit this property to reconstruct the renderings from different viewpoints from a set of available images. In Section 2.3 the 3D warping process will be described in more detail with a particular focus on how it can be exploited in a remote visualization system. Another IBR technique based on 3D warping is represented by the multiple-center-of-projection (MCOP) images [23]. A MCOP image consists of a two dimensional image acquired with a slit camera moving along an arbitrary path. By combining a MCOP image with the camera trajectory information and the depth values for the samples it is possible to render novel views by warping the available data.

Layered depth images

A *layered depth image* is a view of the scene with multiple samples along each line of sight. For every pixel of the image multiple depth and colour values can be stored and this information can be exploited to warp the image to new, arbitrary, viewpoints. The work from Shade et al. [24] exploits also McMillan's warp ordering algorithm [25] to draw pixels in the output image in back-to-front order without using the z-buffer and splatting to address the resampling problem. This representation is compact (its size grows only linearly with the observed depth complexity in the scene), but requires the knowledge of the depth values for all the samples.

View dependent texture mapping

The standard texture mapped 3D models, though widely used, usually are not able to reproduce in a completely satisfactory way effects such as lighting and reflections. A possible solution, introduced for the first time in [26] and successively refined [27] is to replace the single texture applied to the model with a set of views of the scene. The proposed approach requires the availability of a representation of the geometry of the scene and of a set of calibrated photographs taken in the same lighting conditions and covering each surface in the scene from a few viewing angles. The geometry information is used to reproject the views to the target viewpoint and then the information from the different views is combined together.

The algorithm starts from a preprocessing stage. Firstly for each polygon the set of views in which it is visible is determined. Then a *hole filling* algorithm is used to assign a colour to the polygons that are not visible in any of the available views. The idea is to assign a colour to the nearby polygons by averaging the information from the different views and then assign a colour to each of the vertex of the hidden polygon by taking it from the adjacent mesh elements. The last preprocessing step is the construction of a view map for each polygon that stores the image with the closest viewing orientation for a set of directions regularly sampled on an hemisphere.

The rendering process draws all the polygons in 3 steps. In the first the hidden polygons are filled with the estimated vertex colours. Then the polygons visible in just one view are rendered using the texture from that view. Finally the polygons visible in more views are rendered. The position of the viewing direction of each of them on the hemisphere is computed and the view maps are used to determine the three closest views and their weights. Finally the three corresponding textures are applied to the polygon using projective texture mapping and *alpha blending* to combine them together.

2.3 Remote visualization by image transmission

One common approach to the remote visualization of 3D scenes, that can be considered part of the image-based rendering techniques, is to perform the rendering at server side and transmit to the client just the rendered views. Even if the key idea is quite simple there are many different versions of this framework and various predictive compression

schemes have been developed to make it feasible. The remote visualization scheme that is the main subject of this thesis represents an evolution of the previous system we built based on this scheme ([28, 29]), and also an answer to the limitations of this family of approaches. A brief overview of this approach and of its advantages and defects can be useful to understand some choices that will be made in the proposed visualization framework.

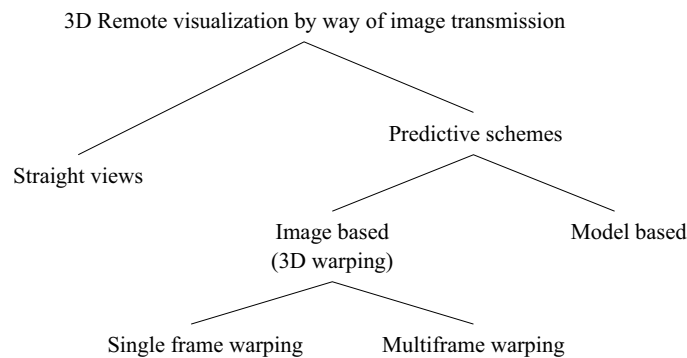


Figure 2.3: Remote visualization by image transmission

Figure 2.3 shows a general subdivision of the possible approaches to 3D visualization by image transmission. The most straightforward solution is to render the required view at server side and simply transmit it compressed using some suitable image compression standards [30, 31, 32, 33]. This approach is plagued by the high data rate required to achieve a good frame rate during the interactive browsing. Many proposed schemes try to solve this issue by predicting at client side the requested view and transmitting only the residual data. In the literature there are two major approaches to prediction: in the first [34] the client has a simplified version of the 3D scene description that uses during fast motion while the server holds the full model and sends the residual between the low and high resolution renderings when motion stops or slows down. The second predictive approach [28, 35, 36, 37] is based on 3D warping. It consists in using depth information to warp the already received views to the newly required viewpoint in order to obtain a prediction of the required view. Finally the residuals are sent. The various techniques differ in the way they built the prediction and compress the residual data.

2.3.1 Basic structure

Remote visualization of 3D models by image transmission basically splits into two parts what a standard 3D models browser does at client side by moving the rendering task (completely or partially) at server side (see Figure 2.4). The basic scheme is simple: the client sends the user's requests to the server over the network. At server's side the graphic engine renders the required views and sends them back to the client as compressed images. Finally the client decompresses and shows to the user the images received from the server. Various image compression techniques can be exploited to

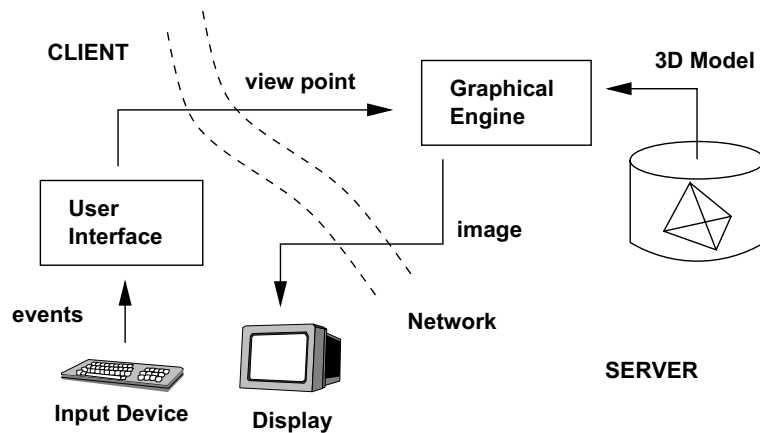


Figure 2.4: Application structure (remote visualization)

reduce the bandwidth usage, but video compression ones are not suitable because we do not know in advance the inspection path of the user. In such a remote visualization scheme the whole 3D model stored on the server should not be transmitted to the client.

The main advantages of this scheme are the following:

- The memory and computational requirements at client side are very limited: it has just to decompress and visualize images. Furthermore, the visualization's fluidity is independent of the 3D model's size.
- Visualization at client side can start just after the first view (a single compressed image) is received from the server without long initial downloads.
- Channel capacity is evenly used.
- The copyright control of the 3D models is fully ensured, since the end-user never receive the whole model, but its rendered views only.

Unfortunately this approach has also some drawbacks:

- The transmission of several images per second to support a fluid navigation requires a huge bandwidth. Predictive compression approaches can reduce the bandwidth requirement but the compression of the residuals remains tricky.
- A powerful server is required to render the views, specially if the scene is complex and there are several clients connected at the same time.
- These systems are very sensitive to network latencies and congestion.

If the clients have good computation and memory resources, it is fair to assume them capable of supporting non standard decompression and visualization tasks which may reduce the bit-rate of the transmitted datastream. In principle the heavier is the work at client's side the smaller can be the transmitted datastream.

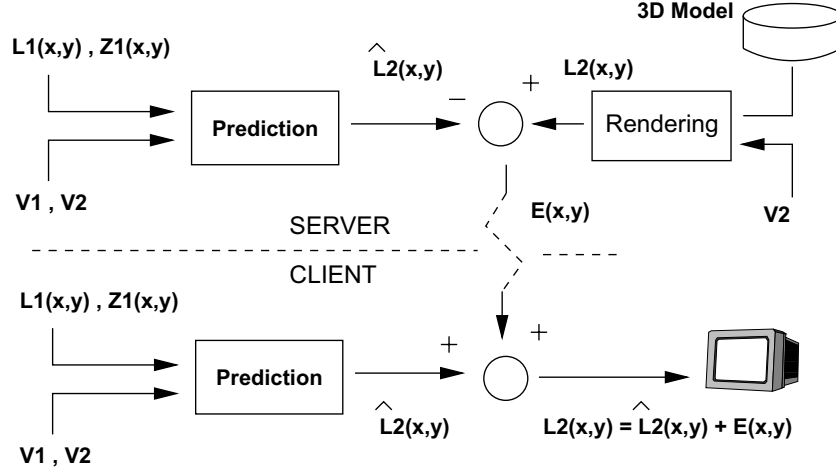


Figure 2.5: IBR predictive compression scheme

2.3.2 Prediction of the views

The most common approaches to reduce the amount of data that need to be transmitted from the server to the client are based on 3D warping. This is an image-based rendering technique where the required view is predicted by reprojecting the previously transmitted images to the new viewpoint using the z-buffer information. Figure 2.5 shows the general architecture of a client-server visualization system based on this approach. The prediction are compared with the actual rendering at server side and the residual difference is sent to the client that will use it to correct its prediction.

Entering in more detail we can represent with $V^1(\mathbf{x})$, $\mathbf{x} \in Q_1 = [0, W_1] \times [0, H_1]$ the available view of the 3D model corresponding to the position c^1 of the virtual camera. In the reference system of V^1 the 3D position of pixel $\mathbf{x} = [x, y]^T$ of $V^1(\mathbf{x})$, is $\mathbf{X} = [x, y, Z(x, y)]^T$ where $Z(x, y)$ is the z-buffer content at \mathbf{x} . Let $V^2(\mathbf{x}')$, $\mathbf{x}' \in Q_2 = [0, W_1] \times [0, H_1]$ denote the next view, associated to user virtual camera located at c^2 . The position \mathbf{X}' of \mathbf{X} in the reference system of V^2 is

$$\mathbf{X}' = [x', y', Z(x', y')]^t = T(\mathbf{X}) = T(x, y, Z(x, y)) \quad (2.3)$$

where T is a the projective transformation [38] between the two views, obtainable by simple matrix products in homogeneous coordinates.

Denote with $\widehat{V}^2(\mathbf{x}')$ the IBR prediction of $V^2(\mathbf{x}')$, i.e., the prediction of V^2 built from $V^1(\mathbf{x})$ and the corresponding depth information $Z(\mathbf{x})$ via 3D warping. Neglecting the Z -coordinate in (2.3) one may obtain

$$\mathbf{x}' = t(\mathbf{x}), \quad \mathbf{x} \in Q_1 \quad (2.4)$$

By defining with $I_p = Q_2 \cap t(Q_1)$ the samples of Q_2 that can be predicted from Q_1 and with $I_n = Q_2 - I_p$ its complement, the set of points that cannot be predicted with the warping. The relationship between \widehat{V}^2 and V^1 can be written as:

$$\widehat{V}^2(\mathbf{x}') = \begin{cases} V^1(t^{-1}(\mathbf{x}')) & \mathbf{x}' \in I_p \\ 0 & \mathbf{x}' \in I_n \end{cases} \quad (2.5)$$

Clearly, the computational complexity of the prediction depends only on the resolution of the images and not on the 3D model, making the system particularly suitable for very complex meshes. The reprojection gives an approximation of the required view, but has an important limitation: it is a procedure of forward mapping type [39] and so it does not define $\widehat{V}^2(\mathbf{x})$ on all the points of the new view. The unpredicted pixels (represented by I_n) can appear when parts of the scene not visible from V^1 enter in the region that is visible from V^2 . Furthermore inside I_p the prediction could be wrong due to occlusions, different lighting conditions or quantization issues.

Several extensions of the 3D warping scheme have been developed in order to overcome its drawbacks, for example combining the warping of multiple views, using Layered Depth Images (images taken from a single viewpoint but with multiple pixels along each line of sight [24]), or wavelet techniques [36].

Other extensions aim to fill the holes in the predicted view due to zooming operations. The simplest solution is the pixel splatting in which every pixel in the destination image is mapped to all the pixels inside a square of a fixed size. A more complex solution is to treat the reference image as a polygonal mesh and fill the warped mesh elements as in 3D rendering.

A completely different approach, presented in [34], is to store a simplified or untextured version of the mesh on the client and to use its rendering for the view prediction. The server renders both the simplified and the complete model, subtracts the two renderings and sends the difference to the client, which adds it to the prediction in order to reconstruct the actual rendering.

2.3.3 Residual selection strategies

The main issue of this remote visualization scheme is how to efficiently compress and transmit prediction errors. Since prediction errors become bigger when the predicted view moves farther from the starting view, it is common to periodically send a full view and the relative z-buffer in order to reset to zero the prediction error. The z-buffer can be transmitted just for the reference view or for every frame. The second solution allows better predictions but dramatically increases the amount of transmitted data.

The first and more obvious strategy is to transmit the difference between the correct rendering and the prediction (Figure 2.6). This solution allows to perfectly reconstruct the high quality rendering of the 3D model, but does not allow a very good compression. The error images contain many isolated pixels (high frequency content). Since most of the classic transform coding tools (JPEG, JPEG2000, etc.) exploit the fact that images essentially exhibit low frequency content, their performance on this kind of data is rather poor.

The compression performance associated to the previous strategy is not satisfactory and it is necessary to look for a trade off between the image quality and the bandwidth

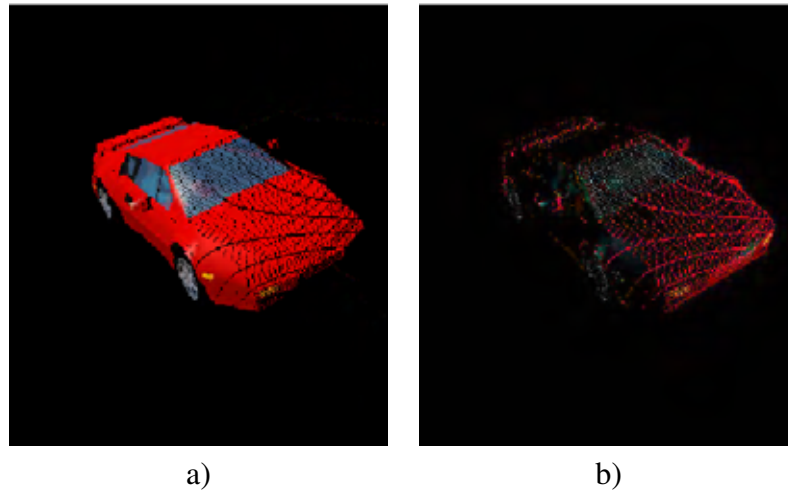


Figure 2.6: Prediction compensation: a) predicted image, b) error image

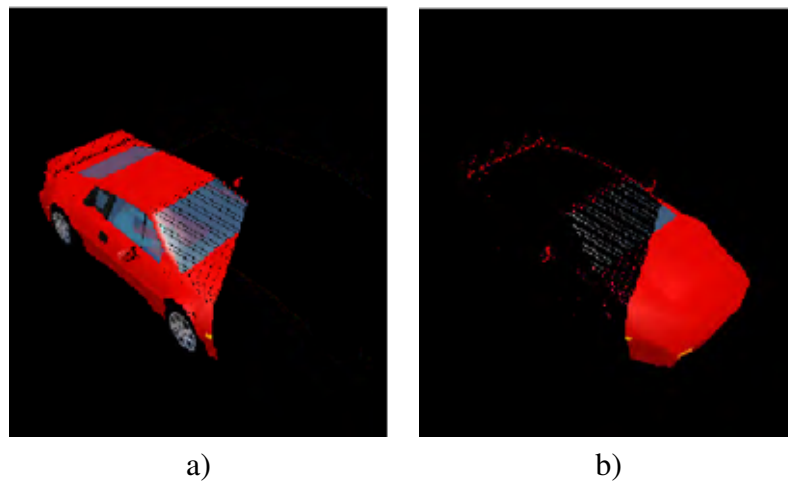


Figure 2.7: Transmission of the undefined samples: a) predicted image, b) undefined samples

requirements. As shown in Figure 2.7 the most visually conspicuous differences between the predicted and the actual view are the undefined pixels in the predicted view, $\widehat{V}^i(\mathbf{X})$, $x \in I_n$.

We can introduce a new function representing the set of undefined samples:

$$\Psi_{I_{n_i}}(\mathbf{X}) = \begin{cases} 1 & \text{if } \mathbf{X} \in I_{n_i} \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

by which the unpredicted part of the image can be written as

$$M_i(\mathbf{X}) = V^i(\mathbf{X})\Psi_{I_{n_i}}(\mathbf{X}) \quad (2.7)$$

This leads to a new strategy for the remote 3D visualization: firstly both the client and the server compute the prediction of the required view. Then the server compares it with the correct rendering. Instead of sending the complete difference between the prediction and the correct rendering as in the previous case, it just sends $M_i(\mathbf{X})$ i.e. the samples that cannot be predicted with the warping. Finally the client adds $M_i(\mathbf{X})$ to the prediction and shows the resulting image to the user. It is worth pointing out that while outside I_{n_i} , $M_i(\mathbf{X})$ is exactly equal to 0, $E_i(\mathbf{X})$ on I_{p_i} generally assumes small non-zero values, and the information associated to the complete difference is generally much higher than that associated to $M_i(\mathbf{X})$ because of high frequency contributions. The rendered image quality with the second approach is lower, but the visual difference between the two approaches is typically acceptable. However some artefacts remain in the images rendered with the second approach because it can not handle changes in the colour of the pixels due to reflections or lighting and occlusions from objects that were not visible in the previous image. Basically it assumes that the prediction, when available, is always correct, without compensating errors in the warped images.

A possible solution to the occlusions issue is presented in [29] and is based on the observation that in the samples affected by occlusions the predicted and real z-buffer values do not correspond. The idea is basically to make a comparison between the predicted and the actual z-buffer at the server and to send only the pixels for which the depth error is bigger than a fixed value and the unpredicted pixels.

Following the same approach of the previous case, we can define a new function representing the samples that need to be compensated:

$$\Omega_{I_{n_i}}(\mathbf{X}) = \begin{cases} 1 & \text{if } \mathbf{X} \in I_{n_i} \text{ or } |\widehat{Z}(\mathbf{X}) - Z(\mathbf{X})| > K \\ 0 & \text{otherwise} \end{cases} \quad (2.8)$$

where $\widehat{Z}(\mathbf{X})$ is the depth of pixel \mathbf{X} in the predicted view and $Z(\mathbf{X})$ its actual depth value (the comparison is done by the server). K is a constant representing the minimum error on the depth for which we expect an occlusion. It allows to find a trade-off between image quality and transmitted data (for small K more samples are transmitted, while for a very large K , $\Psi_{I_{n_i}}(\mathbf{X}) \simeq \Omega_{I_{n_i}}(\mathbf{X})$).

The residual can be written as

$$N_i(\mathbf{X}) = V^i(\mathbf{X})\Omega_{I_{n_i}}(\mathbf{X}) \quad (2.9)$$

The new algorithm is the same of the previous case, where residual $M_i(\mathbf{X})$ is replaced by $N_i(\mathbf{X})$. This strategy is a sort of compromise between the error and the complement image strategies and offers a reasonable image quality without sending all the pixels of the error image. Unfortunately it does not handle the colour problems due to lighting and is not suitable in the case of reflective objects. There are also other approaches to the selection of the information that needs to be transmitted, for example Yoon and Neumann [35] suggest a method in which the samples are selected using an heuristic ray intersection test.

2.3.4 Compression of the transmitted data

The data that needs to be transmitted in a predictive scheme for remote visualization of 3D models can be divided into three main categories: the images, the depth information and the prediction compensation data.

The complete images can be compressed by any image compression standard, for example JPEG2000 (see Section 2.4). The depth information seems easier to compress, but since the edges in the Z-buffer are fundamental for a correct reprojection, depth information needs to be compressed very carefully. Current video cards use 32 bits representations of the Z-buffer, but a smaller accuracy is sufficient for the computation of the warpings. Experimental tests showed an 8-bit quantization of the depth values offers a good trade-off. The data can be then compressed using lossless compression tools (such as the LZ77 algorithm) or lossy image compression standards such as JPEG2000. In this case it is necessary to use a high bitrate, because lossy compression artefacts on the edges can lead to wrong reprojections¹. Since either depth information and the residual need to be transmitted, it is not worth paying a data reduction in the depth maps with extra data for the residual and a trade-off must be found.

The compression of the residual remains the biggest issue. Standard waveform based image compression tools do not work very well with this kind of data. The bitstream associated to the difference images compressed with JPEG can be even bigger than the bitstream associated to the original full images JPEG compressed with a similar quality. JPEG2000 offers better performances than JPEG with the error images, but the data size remains too big for an interactive browsing system. It is also important to note that error images are differences and so require an additional bit for the sign. Possible solutions are to use a compression standard that supports higher bit-depths (such as JPEG2000) or to quantize the colour components by removing the less significant bit as in [34]. The images associated to the other two approaches can instead be compressed more efficiently using JPEG or JPEG2000, the plots of Figure 2.8 and 2.9 show an example of the transmitted data for a short sequence of views in the two cases². Another possible solution, is to send a list of the pixels that need to be

¹ A more sophisticated Z-buffer compression strategy based on the region of interest (ROI) feature of JPEG2000 and on a reshaping of the dynamic range of the depth is found in [40]. Other approaches are based on context modeling [41] and on the representation of the depth map as a triangular mesh [42].

²The plots show the data transmitted by the system described in [29] for a set of views of the 3D model shown in Figure 2.7.

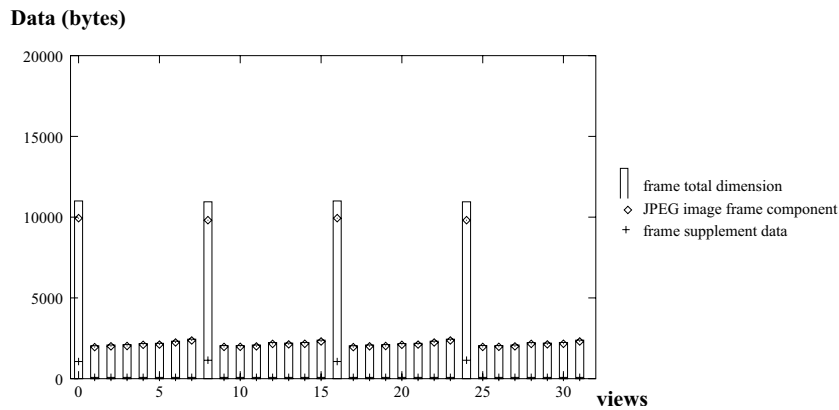


Figure 2.8: Example of transmitted data in remote visualization with correction of the undefined samples (320x240 pixels)

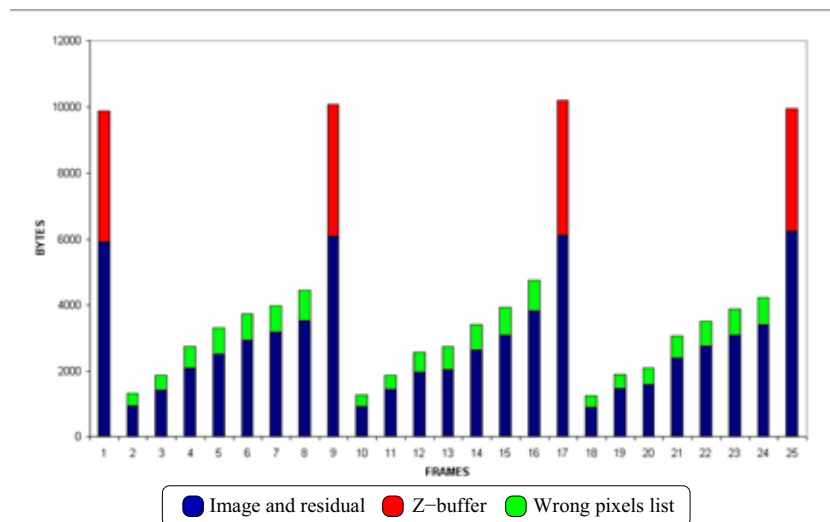


Figure 2.9: Example of transmitted data in remote visualization with depth comparison (300x260 pixels)

corrected, then pack them one after another in an array and lossless compress all the data.

Even if this remote visualization scheme has some useful features, it is not completely satisfactory. The most critical issue is the transmission of the prediction error for every frame, that is difficult to compress, forces the server to compute the rendering of all the required views for every client and requires a low network delay. The new approach that will be introduced in this thesis is still based on the transmission of images and depth information, but will introduce a way to exploit the information from several images and depth maps to reconstruct the new views at client side without the need to correct them by sending the prediction error data.

2.4 Compression and transmission of 2D images with JPEG2000 and JPIP

The remote visualization system that is described in this thesis is heavily based on the JPEG2000 image compression standard both for geometry and texture information. From a certain point of view it represents an attempt to extend some of the ideas behind this compression standard from the 2D image world to 3D scenes browsing. A complete description of the standard is out of the purposes of this thesis (a very comprehensive one is given in [8], while a good overview is [43]): instead in this section, after a brief overview of the basic ideas behind JPEG2000 we will focus on the scalable compression and transmission features that will be exploited in the proposed interactive visualization system.

2.4.1 Compression standard brief overview

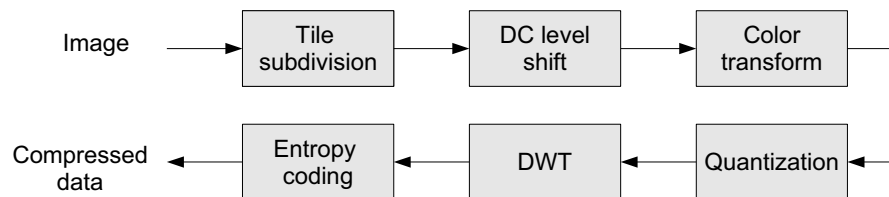


Figure 2.10: Overview of JPEG2000 compression

The procedure to compress an image in JPEG2000 is made of several steps, as shown in Figure 2.10. The images can be initially divided into non-overlapping blocks (*tiles*), then each tile is compressed as an independent image. The first step consists in applying a DC level shift to every sample in the source tile in order to transform the unsigned colour samples into signed values centred around the zero. JPEG2000 is able to compress images with an arbitrary number of components, in the case of standard RGB images (such as texture information), a colour transform can be applied to convert them into the YC_bC_R colour space. The standard supports two different types of colour transform, one reversible component transformation (RCT) that can be used both for lossy and lossless compression and one irreversible component transformation (ICT) that can be used only for lossy coding. This transform has three main targets: it decorrelates the colour components in order to achieve a better compression efficiency; it permits to exploit the bigger sensitivity of the Human Visual System to the luminance (Y) component in the quantization step; finally the reversible transformation allows a perfect reconstruction in the case of lossless compression.

The irreversible mapping from the RGB to the YC_bC_R is given by equation 2.10 :

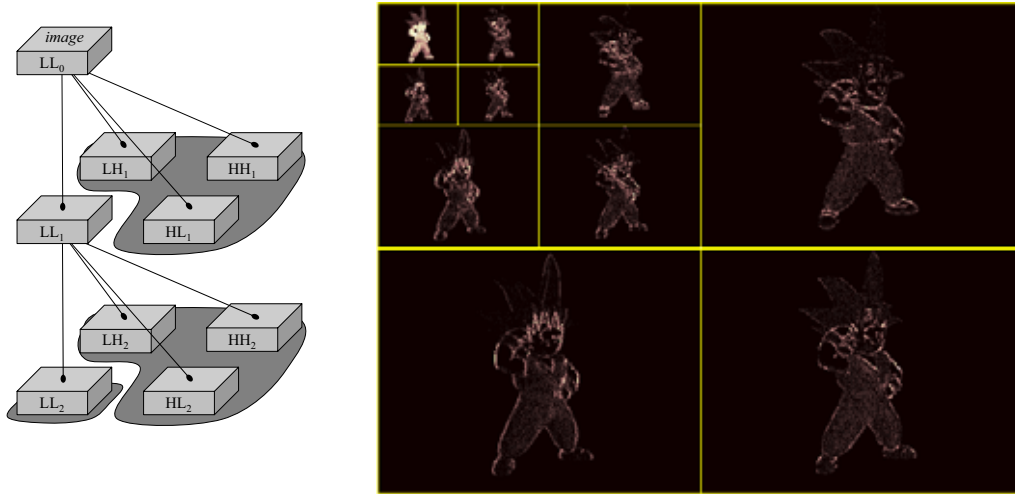


Figure 2.11: 2-level DWT decomposition

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (2.10)$$

This step is not applied to single component greyscale images (depth information can be represented as greyscale images, Section 3.2 will show the procedure used to compress the depth maps in JPEG2000).

The following step is the Discrete Wavelet Transform. The standard allows to use two different kind of transform, the (9,7) floating point Daubechies transform is usually used for lossy compression, while the (5,3) integer transform must be used if the target is to achieve lossless compression. As shown in Figure 2.11, every step of the transform divides the image into 4 different subbands, the low frequency subband (LL) correspond to a lower resolution version of the image, while the 3 high frequency bands (LH , HL , HH) contain the details. The LL subband is then iteratively decomposed with the same procedure. The number of levels is not fixed, but usually 5 levels are used. After a L level wavelet decomposition we obtain a low frequency subband at the deepest resolution level, that in the rest of this thesis will be denoted with LL_L , and a set of high frequency subbands LH_i , HL_i , HH_i for each resolution level $i = 1, \dots, L - 1$ (with this notation the image corresponds to the LL_0 subband). The wavelet decomposition offers a straightforward way to obtain the resolution scalability in JPEG2000 images. After the transform, a deadzone scalar quantization (where the central quantization interval is double the size of the others) is applied to the wavelet coefficients. A different quantization step size can be chosen for each subband. After the quantization the coefficients in each subband are divided into packets corresponding to rectangular non-overlapping regions. Then the three packets corresponding to the same region in the three high frequency subbands at the same resolution level are grouped together into *precincts*. The precinct subdivision allows an efficient spatial random access to the compressed data. Chapter 3 will show how this feature can be

exploited in an interactive browsing framework to transmit only of the region of every view required for the requested rendering. Finally the precinct are divided themselves into regular non-overlapping rectangles called *codeblocks* that will be the fundamental elements in entropy coding.

The next step is the entropy coding of the code-blocks. Every code-block is coded independently using a context-dependent binary arithmetic coding technique (the MQ coder) and the EBCOT paradigm [44]. This efficient fractional bit-plane coding process produces a finely embedded bit-stream that can then be truncated at any desired point, allowing a trade-off between image quality and coded length. A detailed description of the coding technique is out of the scope of this thesis (it can be found in [8]).

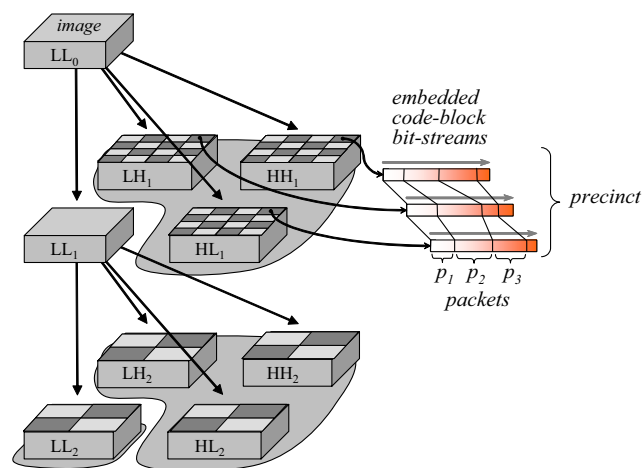


Figure 2.12: Basic JPEG2000 elements communicated by JPIP.

2.4.2 Scalability features in JPEG2000

As described in the previous section JPEG2000 is a highly scalable image compression standard, and the compressed representation contains numerous embedded subsets, each corresponding to an efficient representation of the original image at some reduced resolution, reduced quality, or over a particular spatial region of interest. Figure 2.12 shows the basic elements of this representation. As explained in the previous section, the subbands of the discrete wavelet transform are partitioned into *code-blocks*, typically measuring 32×32 samples each.

Code-blocks are organized into spatially coherent square regions in each resolution component, corresponding to the precincts. The code-block bit-streams associated with each precinct are then put into JPEG2000 packets. Each packet contains some incremental contributions (possibly empty) from each of the code-block inside the corresponding precinct and can be interpreted as a quality increment for a certain

region in a certain resolution level. The absence of one or more trailing packets is equivalent to truncation of the code-block bit-streams. The subdivision of the data in the packets for a JPEG2000 compressed image is usually arranged in order to make this truncation rate-distortion optimal, i.e. the best possible image quality for a certain amount of bytes is obtained by discarding the same number of trailing packets from all precincts. The collection of packets corresponding to a certain quality in the image is usually called *quality layer*. Table 2.2 summarizes the names used for the different data partitions in JPEG2000.

Data partition	Set of the 3 high frequency bands	Single band
Resolution	Resolution level	Subband
Resolution and position	Precinct	Code-block
Resolution, position and quality	Packet	-

Table 2.2: *Jpeg2000 data partitions*

2.4.3 The JPIP transmission protocol

A very efficient system to transmit JPEG2000 compressed images is described in the JPIP standard. This protocol allows to build interactive image browsing applications that fully exploit all the scalability features offered by the compression standard. In JPIP image browsing the user at client side can interactively browse an image that is stored at server side in a very efficient way by progressively transmitting only the part of the image that best fits the user's requests. To achieve this target JPIP does not deal with JPEG2000 code-blocks or packets directly. Instead, it starts by concatenating all the packets associated with a precinct to form a single *precinct data-bin*. Each precinct data-bin thus represents the information corresponding to a certain spatial region and resolution component. The data for each precinct data-bin can be transmitted progressively thus gradually increasing the image quality for that region. JPIP also defines other types of data-bins, for encapsulating compressed data headers and metadata. This allows JPIP to be used to communicate rich content in a progressive and selective manner.

The JPIP standard can be used also to browse multiple image at the same time, a very useful feature for the proposed remote visualization scheme. The Part 2 of the JPEG2000 standard [45] defines the JPX file format, that allows to include several compressed images into a single file together with a customizable metadata structure that can be exploited to store additional parameters of the 3D scene (see Section 3.2). JPX files can be represented in JPIP in terms of large collections of data-bins, thus allowing us to represent a full set of source views and depth maps within a single JPX file, which can then be transmitted dynamically to the client.

The Kakadu software tools [46] contains an efficient implementation of JPIP which is also able to handle JPX files. The client and server modules of this application have been integrated into the proposed 3D visualization system and are used for the transmission of both the geometry and texture information, as described in Chapter 3.

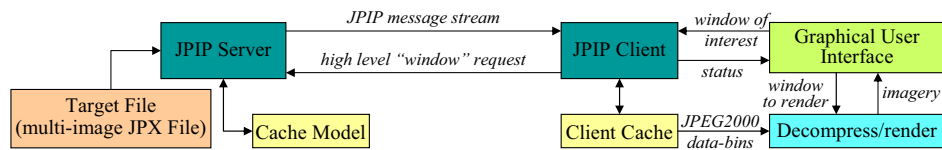


Figure 2.13: Client-server interaction in JPIP

2.4.4 Client-Server Interaction in JPIP

Figure 2.13 shows the general architecture of a client-server system based on the JPIP standard. The server holds the compressed image and progressively sends the information from the various precinct data-bins to satisfy the client requests. The user at client side can interactively browse the image and request the part of the image in which he is interested. A first important element is that the client does not explicitly request data-bins to the server. JPIP instead defines a request language which allows clients to express what they are interested in, using comparatively high level descriptors. For example in the case of standard image browsing the client could request a particular region of interest in the image at a particular resolution level. The client has a local cache system that holds a representation of the source data that is progressively improved with the information transmitted by the server and it may render the images at any time, based on its current cache contents. An important feature of JPEG2000 is that the image can be rendered from any arbitrary subset of the precinct data-bins which might be available at a certain time. The client can then continuously improve the rendering as soon as more data arrives from the server. This allows the rendering at the client to be completely asynchronous with server communications.

After the client has made a request, the server essentially streams JPIP messages to it, where each JPIP message consists of a single byte range from a single data-bin. The messages sent by the server are used to improve the contents of the data-bin in the client cache. The server attempts to send the most appropriate messages to satisfy the client request. However, when the client issues a new request, the server adjusts its transmission policy, but there are no strict requirements on when and how the change will happen. Usually JPIP servers maintain a model of the client's cache, in order to know which representation of each data-bin is held by the client.

These features are particularly good fit for 3D scene browsing. The freedom of the server to form its own decisions regarding the best way to improve a synthesized view at client side is particularly useful in such an environment, where there are many different views and depth maps from which the information to be transmitted need to be selected. In the proposed 3D browsing environment the client does not have a complete knowledge of the scene, it does not even know which views are available, and cannot efficiently decide which elements it needs. The server can also exploit additional information on the scene geometry or compression distortion that would add an high transmission overhead if sent to the client.

It is worth noting that JPIP does not currently provide a request syntax which can be used to explicitly represent the parameters the view V^* of interest for 3D browsing. However, in Section 3.3 we will introduce a possible extension to the *context* fields in order to handle this kind of requests.

Chapter 3

A new approach to remote 3D visualization

Before introducing the new remote visualization framework, let us have a brief recall of the biggest issues of the existing schemes: an efficient remote browsing system should try to overcome these problems. The standard approach to the remote visualization of 3D models, based on downloading the complete description of the scene and performing the rendering at client side, is well-suited to represent the complete 3D scene. However it presents some drawbacks in an interactive exploration over a bandlimited channel:

- High resolution models can easily require tens or hundreds of megabytes, and require a long initial download time before starting the exploration. The user is forced to download all the model description even if in the actual exploration involves only a small section of it.
- Smooth navigation of complex scenes requires very powerful computers. Both the amount of memory required to store them and the computational power required to perform the rendering at high frame rates could be huge. Dedicated graphics hardware can help to deal with this issue, but photorealistic representation of complex scenes are still challenging.
- The construction of a high quality 3D model can require months of work, but after users download it the complete description is stored at client side and the author loses control over its diffusion.

As described in Section 2.2 image-based rendering techniques try to deal with these issues by representing the 3D data as a collection of images and (eventually) depth information. They usually allow a photorealistic representation but they also introduce other problems:

- Some techniques (for example the light fields) require a huge amount of data to represent the 3D scene. They do not use geometric information to exploit the redundancy between close views and so the representation is not so compact as the traditional 3D representation based on mesh and texture. Unlike the classic

scheme they do not require to transmit to the client the complete dataset, but just a small subsection of it. This permits to use “light” clients, but a powerful server is needed to hold the data.

- Other techniques based on the transmission of views perform the rendering at server side. This strategy will allow to use very simple clients but also requires very powerful servers, specially if they need to handle a lot of clients.
- A common approach is to reduce the dimensionality of the plenoptic function by introducing constraints on the view parameters, thus reducing the freedom of exploration.
- Some techniques, like the rendered image transmission, require an interactive real-time response of the server to the user’s request at client side. Temporary network congestions or transmissions latencies can have a severe impact on their performances.

Feature	Model-based	Image-based
Random access	Difficult	Easy
Redundant information	No	Yes
Startup latency	High	Low
Sensitivity to network latency	Small	Big
Photorealistic rendering	Difficult	Yes
Server requirements	Low	Can be high
Client requirements	High	Low
Copyright protection	No	Yes

Table 3.1: Comparison of the image-based and model-based approach to interactive 3D browsing.

From this review (see Table 3.1) it is clear that in an interactive browsing framework both the model-based and image-based approaches have different drawbacks and a good solution should try to combine the best aspects of the two solutions. The new approach to the remote visualization of 3D scenes which will be presented lays in the middle between a model-based approach and an image-based one and tries to offer an answer to the many difficult issues of remote 3D browsing. Before introducing it, let us have a brief overview of the targets we will try to achieve.

- The system should be able to exploit the available bandwidth in the best possible way. This requires to transmit only the information really needed for the current view. At the same time it is also necessary to avoid the transmission of redundant information and to exploit the previously transmitted data to render the newly required views. Scalable geometry and image compression techniques should be exploited to achieve this target.
- An interactive visualization system should be able to follow the user’s real-time exploration by continuously improving the available description on the basis of

the user's point of interest. The system should continue to show to the user the views he requires exploiting the already received information. It should also improve them as soon as new data becomes available without stopping the browsing while waiting for additional data to be transmitted. The user should be able to continue the navigation also when the network is congested or the connection goes down even if with reduced image quality.

- Another target is to achieve a good frame-rate without requiring a powerful client. Complex scenes can require huge amounts of data and a powerful computer to render them in real-time. Thus to achieve this target is necessary to use an approach that does not depend on the complexity of the scene.
- A good image quality is of course another important requirement.
- In some cases it is also important to try to avoid the transmission of the complete scene description or other data that allow to reconstruct it to the client to avoid the loss of control on the distribution of the 3D data.
- Visualization at client side should start without delay, as soon as a limited amount of data is sent by the server.
- Data transmission should be based on the available bandwidth, in order to avoid navigation delay. In other words it should be possible to change the accuracy of the transmitted representation of the scene in order to find a trade-off between the amount of transmitted data and the rendering quality suitable for the current bandwidth availability and client capabilities.

Finally a last important aspect of remote 3D visualization must be underlined. The user at client side can interactively browse the scene and determine the particular views of interest, we do not know in advance the inspection path, differently from other problems such as video transmission. The user is expected to navigate between a variety of different views, but we do not know ahead of time which views will be of interest and how much time (transmission resources) the user will choose to devote to any particular view.

At one extreme, the user's interest may remain focused on a single view for a considerable period of time, waiting until a very high quality rendering has been recovered before moving on. At this extreme, the interactive retrieval problem is very similar to that of interactive image browsing, which is addressed most elegantly by progressive transmission of a single scalably compressed image.

At the opposite extreme, the user may select many different views in rapid succession, with the aim of understanding the scene's geometry. This phase might perhaps be a precursor to a detailed inspection of some particular view of interest. Since successive views are closely related, the most natural way to improve the efficiency of the browsing experience is to predict each new view from the ones that have already been transmitted, and then transmitting the prediction residuals. However, as previously described this approach suffers from a number of drawbacks.

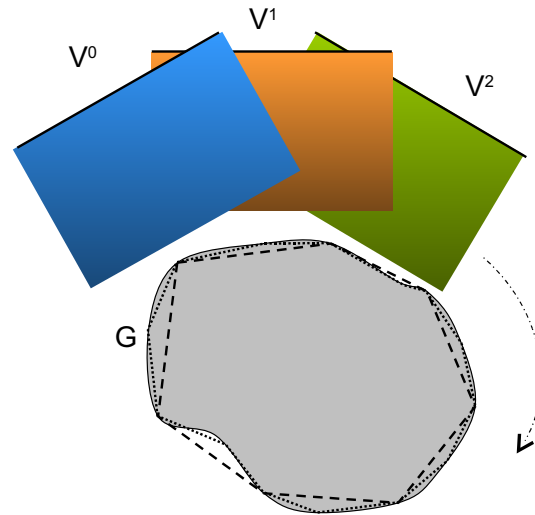


Figure 3.1: *Overview of the browsing environment. The server holds scalably compressed representations for a set of “original view images,” V^i and for the surface geometry, G .*

3.1 Proposed scheme

Considering the above arguments, let us introduce a framework for interactive scene browsing. The basic scheme is made by a server-client couple connected via a bandlimited channel. The server holds the scene description and delivers incremental contributions from two types of pre-existing data (Fig. 3.1):

- Scalably compressed images of the scene from a collection of pre-defined views, that will be denoted with V^i ;
- A scalably compressed representation of the scene surface geometry, G . This representation can be replaced by a set of scalably compressed depth maps Z^i , as will be described in Section 3.2.

The server does not generate new views or compress differential imagery. Instead, it determines and sends appropriate elements from a fixed set of scalable compressed bit-streams, so as to provide its clients with the most appropriate data from which to render their desired views. The client performs the rendering of the required views exploiting the information it has received from the server.

Figure 3.2 shows the complete system architecture. The server and the client are connected via a bandlimited channel, usually represented by the internet connection. The server stores the geometry description and a set of high quality views. At client

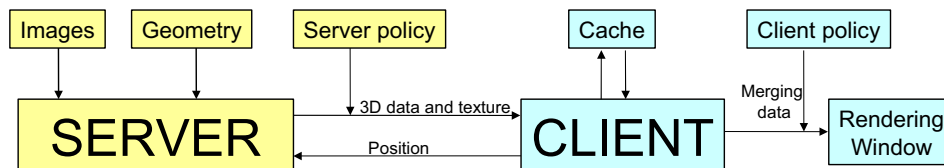


Figure 3.2: *Overview of the proposed system*

side, the user interactively determines the particular view of interest. The client application then communicates the parameters of the required view to the server. The server sends progressively compressed geometry information and texture, according to the user position and available bandwidth. The server does not send all its data, but only the part that best fits the user’s needs and the available bandwidth (the policy to determine which elements need to be sent will be the subject of Chapter 6). The client stores the 3D data and texture it receives from the server into a cache memory. The available geometry and texture information is combined into the rendering of the required view at client side according to a rate-distortion policy that will be introduced in the next chapters. During the interactive browsing the client tries to use the available data the best way it can while the server continue improving the scene representation at client side by sending additional texture and geometry data. The user can navigate the 3D environment also in the case of a congested network or even if the connection is down, since the client is asynchronous with the server, i.e. it will continue rendering with the data it has without waiting for the server communications.

Our proposed framework is particularly appropriate in view of the fact that 3D scene representations are usually generated from a collection of original 2D images; these are natural candidates for the set of available views V^i . If the client happens to request one of the original view images, it can be incrementally served directly from its scalably compressed representation. Interestingly, though, this might not always be the best policy. If the client has already received sufficient elements (sufficient quality) from one or more nearby original view images, V^{k_1}, V^{k_2}, \dots , it may be more efficient to send only the geometric information required for the client to synthesize the requested view, using the resulting bandwidth savings to further augment the quality of these nearby original view images. It follows that even if the server has a huge number of original view images, an efficient service policy would effectively subsample them based on the interactive user’s navigation patterns. More generally, the server may choose to send some elements from V^i , while expecting the client to derive other aspects of the view from the previously delivered, but less closely aligned original view images, V^{k_i} .

The proposed framework may thus be interpreted as fostering a greedy strategy for non-linear approximation of the plenoptic function, since it considers both view sub-sampling and rate-distortion criteria. The fact that efficient service policies can be expected to sub-sample the existing content automatically, brings the proposed approach into contrast with the predictive approach mentioned previously, where imagery

is delivered for every view requested by the user.

In the following sections we will describe the general architecture of our system, while the rest of the thesis will discuss the two fundamental issues of how to combine the information available at client side into the required rendering and how to decide which part of the texture and geometry description need to be transmitted. In particular chapters 4 and 5 try to give an answer to the first question, while the policy transmission for the server will be the subject of Chapter 6

3.2 Scene representation

The three dimensional scene is stored at server side and represented as a multitude of original image views together with the corresponding depth maps. This representation is particularly appropriate in view of the fact that 3D scene representations are usually generated from a collection of original 2D images, and most 3D acquisition systems such as laser scanners produce as output images with the corresponding depth. This approach allows to avoid to combine together the views into a unique three dimensional representation, a very challenging step which usually require a huge amount of work (see [47]). The proposed scheme does not rely on a particular arrangement of the images acquisition points and viewing directions, i.e. the views need not to be taken from a set of viewpoints with a certain geometrical distribution. The viewpoints can be freely chosen during the acquisition procedure, but the camera projection matrix associated with every view must be known (e.g. it is necessary to know the viewpoint, the view direction and the camera parameters for every view). The set of views should cover the whole scene, but the different viewpoints can be chosen without any particular constraint. For example it is possible to have more views in an area with a particularly complex geometry structure, or where are present interesting details (e.g. it is possible to have an evenly spaced set of images which cover all the scene and then some close-up views of the most interesting details). An example of the data available at server side is shown if Figure 3.3.

Procedure	Acquisition complexity	Hardware cost	Depth accuracy	Photorealistic renderings
Range camera views	Medium	High	High	Yes
Pictures and passive depth reconstruction	Medium	Medium	Low	Yes
3D model	High	(*)	(*)	No
3D model with real images	High	(*)	(*)	Yes
Synthetic models	Low(**)	None	High	No

(*) Depends on the acquisition technique used to build the 3D model

(**) Assuming model already available

Table 3.2: Data acquisition procedures

We envisage various data acquisition procedures, Table 3.2 shows some possible choices. The first is to use a range camera (see Figure 3.4) to acquire a set of images with the corresponding depth maps and then register the different views together. The

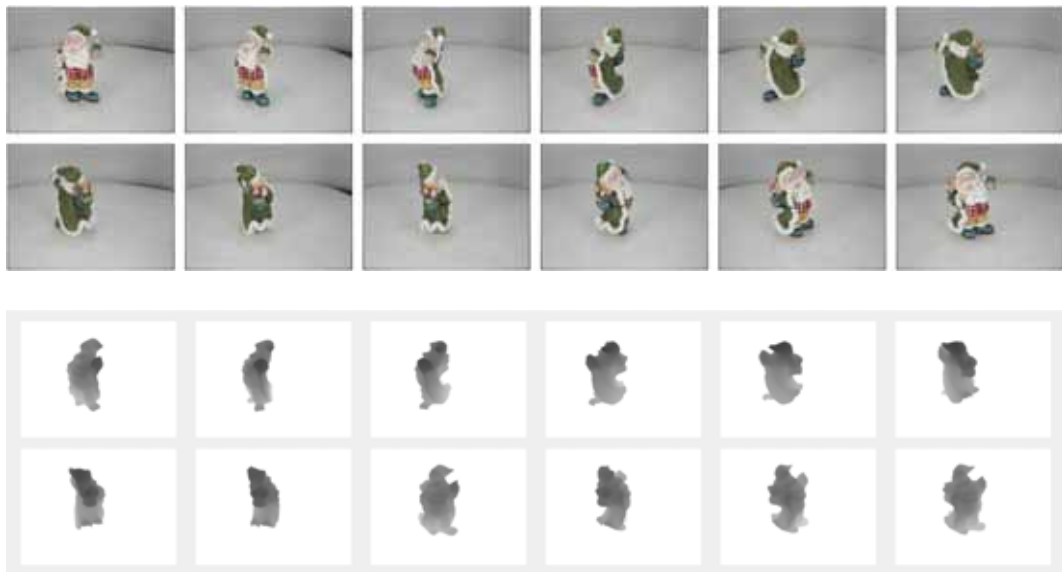


Figure 3.3: *Example dataset for a Santa Claus statuette*

proposed approach requires to know the camera parameters and so to perform the registration step after the acquisition but it allows to skip the integration of the 3D data, that is usually a challenging operation.



Figure 3.4: *3D acquisition with a range camera*

Another option is to take a set of pictures of the scene using a standard camera and then reconstruct the depth information using passive techniques such as stereo vision or space carving. A common way to use this acquisition system in the case of small objects is to place the object on a turntable and take a set of pictures by rotating the turntable of a fixed step after each photo (Figure 3.5). The depth maps can then be reconstructed using space carving, stereo vision or a combination of them. Figure 3.3 shows an example of images and depth maps acquired with this techniques; the dataset is made of twenty pictures spaced of 18 degrees each and the corresponding depth maps



Figure 3.5: *Turntable setup*

built using the method described in [48], which combines stereovision with silhouette information.

Otherwise it is possible to reconstruct a full 3D model of the scene using one of the various available techniques and render the views and depth maps from it with any 3D rendering software. This approach offers a greater freedom in the choice of the viewpoints and permits to render a huge number of views, but requires a lot of work to build up the 3D model. Another issue is that the use of a single texture applied to the model instead of a set of views implies a loss of information and a less photorealistic image quality. To avoid this problem it is also possible to use depth maps extracted from a 3D model combined with real views of the scene for which camera parameters registered with the model are available. The 3D rendering approach can also be used to exploit the proposed system in interactive visualization of synthetic 3D scenes built with CAD or 3D modeling software. Figure 3.6 shows an example dataset built from a textured 3D model of the *Goku* cartoon character. It includes 8 side views of the model spaced of 45 degrees, a top and a bottom view.

After the acquisition the views and depth information are stored at server side. A projection matrix is associated with every view. The server only uses the available views and does not need to compute new views during the interactive navigation. This approach permit to reduce the computational complexity at server side, allowing a not too powerful server to handle multiple clients at the same time. Another advantage is that the server transmits only views and depth maps (that are represented as greyscale images) and so a standard image server can be easily extended to an interactive visualization system. In our case the JPIP interactive server included in the Kakadu software tools will be used (see Section 2.4.3).

All the available views are scalably compressed using JPEG2000 and stored into a single JPX file on the server. The compression is done using the Kakadu coder [46]. The images are compressed exploiting the JPEG2000 scalability features to divide them in packets corresponding to the different resolution levels, spatial positions and quality layers. All these packets can be progressively transmitted. After the compression the different JPEG2000 images are packed together into a single JPX file (see



Figure 3.6: *Example dataset for a synthetic model of the Goku cartoon character*

```

kdu_compress.exe -i 1.bmp -o 1.jp2 -rate 0.8,0.4,0.2,0.1,
    0.05,0.025,0.0125 Cblk={32,32}
kdu_compress.exe -i 2.bmp -o 2.jp2 -rate 0.8,0.4,0.2,0.1,
    0.05,0.025,0.0125 Cblk={32,32}
kdu_compress.exe -i 3.bmp -o 3.jp2 -rate 0.8,0.4,0.2,0.1,
    0.05,0.025,0.0125 Cblk={32,32}
kdu_compress.exe -i 4.bmp -o 4.jp2 -rate 0.8,0.4,0.2,0.1,
    0.05,0.025,0.0125 Cblk={32,32}
kakadu\kdu_merge.exe -i 1.jp2,2.jp2,3.jp2,4.jp2
    -o object.jp2

```

Figure 3.7: Example script to compress a 4 images dataset

Section 2.4) using the `kdu_merge` tool. Figure 3.7 shows an example of the script used to compress the data for a 4 images dataset. In the datasets used for the system testing we used 7 different quality layers in the range from 0.0125 bpp to 0.8 bpp. Every quality layer has double the bitrate of the previous one. Using a resolution of 1024x768 pixels this choice leads to an average size of 77KB for every view at the maximum quality, while the first quality layer is just 1,2KB, corresponding to a poor image quality but still enough to allow the client to show something to the user before more data becomes available. Figure 3.8 shows an example of the image quality corresponding to the different quality layers, while Table 3.3 shows the size of some multiple images compressed files used to evaluate the system.

Name	Resolution	Number of views	Total size	Average image size
Santa Claus	2048x1536	60	19 MB	314 KB
Santa Claus	1024x768	60	4,6 MB	77 KB
Goku	1024x768	8	560 KB	70KB
Goku	640x480	8	241 KB	30 KB
GT car	1024x768	8	615 KB	77 KB

Table 3.3: *Example of image datasets*

The surface geometry is also available at the server in the form of scalably compressed depth maps, again corresponding to a multitude of original views. It is not strictly necessary that views and depth maps are taken exactly from the same viewpoints, but by matching every view with the corresponding depth map it is possible to ensure a better reprojection quality and also the acquisition procedure is easier. All the examples shown in this thesis refer to set of images and depth maps taken from the same viewpoints. The depth maps are also compressed in JPEG2000. This is a quite common choice for depth maps compression and even if ad-hoc compression techniques for depth information are available, usually they do not offer all the scalability features of JPEG2000. Another advantage of this representation is that the same transmission system can be used both for geometry and texture information.

Before compressing the depth maps it is necessary to represent the depth values as images. As shown in Figure 3.9, we assume that all the objects visible in a certain image and the corresponding depth values lays between two planes that in computer



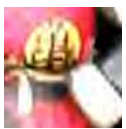




<i>Quality Layer</i>	<i>Bitrate</i>	<i>Image size</i>	<i>Image sample</i>
1	0,0125 bpp	1,3 Kb	
2	0,025 bpp	2,5 Kb	
3	0,05 bpp	5 Kb	
4	0,1 bpp	10 Kb	
5	0,2 bpp	19 Kb	
6	0,4 bpp	38 Kb	
7	0,8 bpp	77 Kb	

Figure 3.8: *Image samples for the different quality layers*

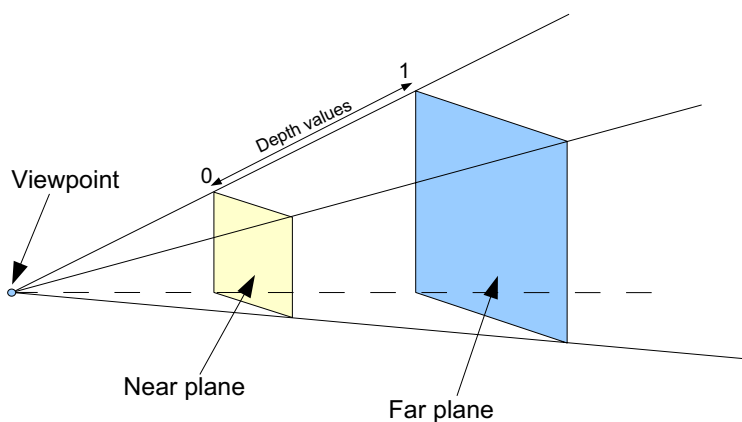


Figure 3.9: *Depth values represent positions between the near and far plane*

graphics are called *near plane* and *far plane*. To perform the conversion we firstly define the position of the near and far plane and then the depth values acquired from the range camera are converted in a set of values from 0 to 1 representing how far they are from the two planes. The values are distributed in a non-uniform way to achieve a better precision in proximity of the near plane. In the case of synthetic data, the values in the videocard framebuffer returned by 3D libraries such as *OpenGL* are already scaled in this way. The values are then converted in an integer representation by multiplying them for 2^{nbits} and rounding, where *nbits* is the number of bit per pixel in the image. Preliminary tests carried out using 8 bpp greyscale images have shown a poor accuracy in the depth values. We found that by representing the depth maps as 16 bit per pixel greyscale images it is possible to obtain a good trade-off between compression performance and depth accuracy.

```
kdu_compress.exe -i d1.raw1 -o d1.jp2 -rate 0.8,0.4,0.2,0.1,0.05,0.025,
  0.0125 Sprecision={16} Scomponents={1} Ssigned={no} Sdims={768,1024}
kdu_compress.exe -i d2.raw1 -o d2.jp2 -rate 0.8,0.4,0.2,0.1,0.05,0.025,
  0.0125 Sprecision={16} Scomponents={1} Ssigned={no} Sdims={768,1024}
kdu_compress.exe -i d3.raw1 -o d3.jp2 -rate 0.8,0.4,0.2,0.1,0.05,0.025,
  0.0125 Sprecision={16} Scomponents={1} Ssigned={no} Sdims={768,1024}
kdu_compress.exe -i d4.raw1 -o d4.jp2 -rate 0.8,0.4,0.2,0.1,0.05,0.025,
  0.0125 Sprecision={16} Scomponents={1} Ssigned={no} Sdims={768,1024}
kdu_merge.exe -i d1.jp2,d2.jp2,d3.jp2,d4.jp2 -o depth.jp2
```

Figure 3.10: Example script to compress a 4 depth maps dataset

Figure 3.10 shows the script to compress the depth information, we used the same bitrates as the images, corresponding to a much higher quality in the depth maps, but a high quality representation is needed for accurate reprojections and the progressive transmission system allows to send only a subset of the data if the complete information is not worth transmitting. The final step consists in including into the metadata field of the JPX file the information on the near and far planes.

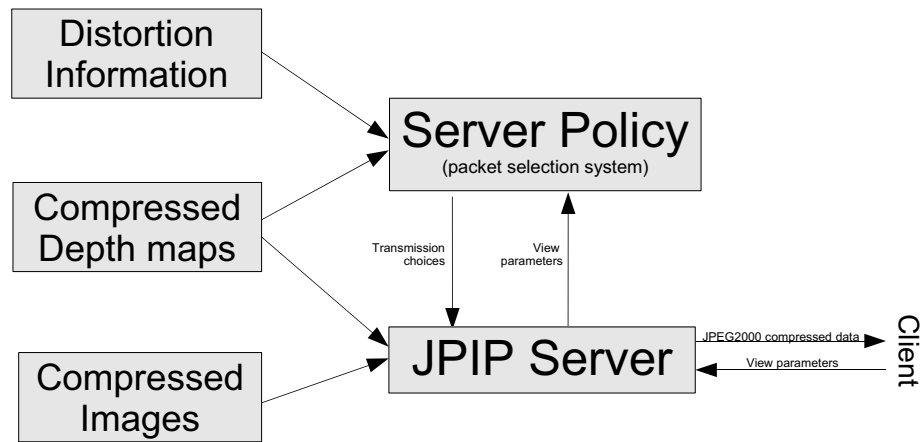


Figure 3.11: Overview of the server application

3.3 Server overview

In the proposed system the rendering is performed at client side and the server application has only the function of transmitting the compressed image and geometry information to the client. Figure 3.11 shows the basic architecture of the server application, that is divided into two main modules. The first one is a modified version of a JPIP server for interactive transmission of JPEG2000 images (see Section 2.4.3). We built it over the JPIP server included in the Kakadu software. This module has access to the two compressed files containing the depth and the image information and transmits in a progressive way the various packet corresponding to the different elements of the views and depth maps to the client on the basis of the requests received from the user at client side.

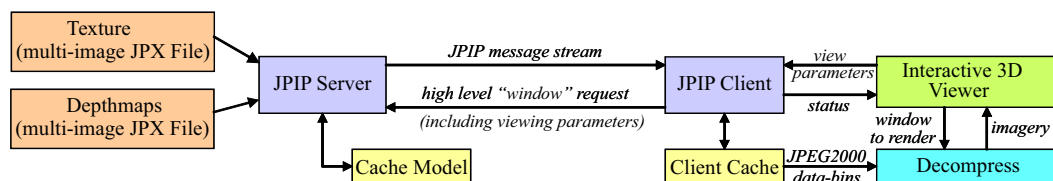


Figure 3.12: Modified JPIP architecture for 3D browsing

Figure 3.12 shows how the general structure of a JPIP system shown in Figure 2.13 can be extended to the case of 3D browsing. A key difference is that in standard JPIP application the requests are made in terms of spatial regions and resolution levels of the images, while in this particular case the user is asking for 3D views. Fortunately the JPIP standard allows to add *contexts* to the standard image requests to handle special needs. For the purpose of this project we added a new context type (that we identify

with the string “JP3D”) to the standard ones. The client request will contain the identifier of the new context followed by a parameter string. The syntax of this string is the following:

```
jp3d-geometry = rREQUESTvFLOATvFLOATvFLOAT ...
```

The “REQUEST” parameter identifies the type of request the client is going to make to the server, while the number of following floating point values depends on the type of request. In the current implementation we have included only two possible requests, each of them followed by 16 floating point values corresponding to the coefficients of the projection matrix corresponding to the viewpoint and gaze orientation selected by the user at client side (see Table 3.4). The first (called “KDU_JPIP_3D_WINDOW”) updates the viewing parameters with the new camera matrix. The second (called “KDU_JPIP_3D_UPDATE”) updates the parameters and also forces an immediate re-computation of the packets that are going to be transmitted. This difference is necessary because the current practical implementation is not fast enough to recompute the packets to be transmitted every time the user at client side changes the viewpoint, especially if he is moving very fast around the scene. A practical solution is that the server checks if the view parameters have been changed every t seconds and in this case it recomputes the packets that need to be transmitted. In the current implementation t is a few seconds but the software still has not been optimized and preliminary studies suggest that the system can perform the calculation much faster (at least 10 times faster). This procedure works fine combined with the first request type with whom the client can signal to the server that the viewing parameters have changed. Otherwise the client can force the server to recompute the packets immediately using the second request type. A very important aspect of this request procedure is that the client is going to transmit to the server only the camera parameters, i.e. the client is not asking for some elements of a particular view or depth map. It just sends the viewing parameters and it is the server that decides which elements from the various compressed codestreams corresponding to the different views and depth maps are the best to satisfy the client requests. Another key feature is that client and server are completely asynchronous. After making a request the client will not stop but will continue performing the rendering with the available data, and will improve it as soon as more data is received from the server.

Standard JPIP servers work in *epochs*, every one corresponding to a time interval usually between $\frac{1}{2}$ and 1 second. The amount of data that can be transmitted in each epoch depends on the available bandwidth. For every epoch, after receiving the requests from the client (usually corresponding to a region of the image at a certain resolution) the optimization procedure assigns a priority to every available packet based on rate-distortion computations and then builds a list of the packets that need to be sent to the client in that epoch. In our approach we introduce an external module (the *server policy* block in Figure 3.11) that receives from the JPIP server the view parameters and compute the priority for the available depth and texture packets. This module is based on an optimization algorithm, described in Chapter 6, that selects which contribution from the scalably compressed representations of the different views and depth maps

will be transmitted. The decision is made on the basis of the required viewpoint, of the already transmitted information and of distortion information on the compressed data. After completing the computation the priorities are sent back to the JPIP server that can use them to build the list of packets that will be transmitted in the current epoch. The data is then transmitted incrementally in the same way we do for standard JPEG2000 images.

Value	Request type	Description	Parameters
1	KDU_JPIP_3D_WINDOW	Update the projection matrix	16 floating point values
2	KDU_JPIP_3D_UPDATE	Update the projection matrix and the packet transmission choices	16 floating point values

Table 3.4: Request types for JP3D context

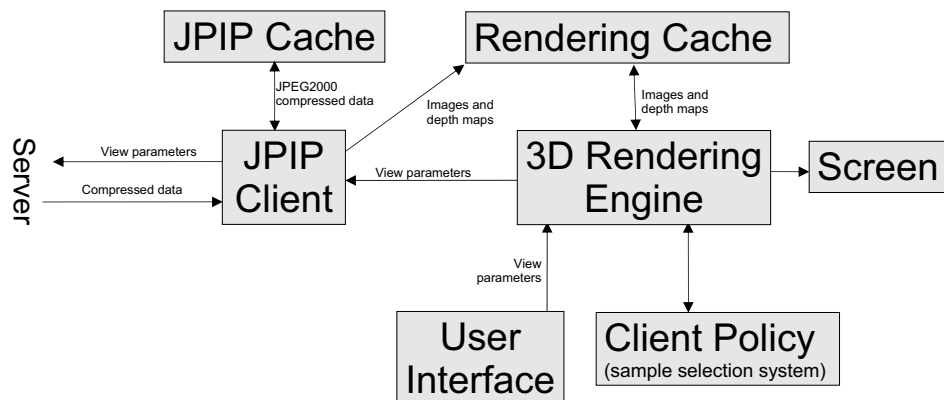


Figure 3.13: Overview of the client application

3.4 Client overview

The client application's target is to exploit all the image and depth information received from the server to show the rendering of the required views to the user. Figure 3.13 shows the architecture of the client. The user interface shown to the user is the same of a standard 3D browser. Like many other 3D viewers there is a rendering window showing the scene and the user can freely explore the scene using the mouse or the keyboard. He can look at the 3D world from any viewpoint and viewing direction and perform enlargements. The current implementation of this module exploits hardware accelerated rendering and is based on the *OpenGL* and *Glut*¹ libraries. After the user

¹ The *Glut* library [49] is an interface between the *OpenGL* graphics library and the window manager of the operating system (it is available both for Unix/Linux and Windows). It allows to build simple Graphical User Interfaces for 3D applications that exploits the *OpenGL* hardware accelerated rendering.

requires a particular viewpoint and orientation² the parameters of the view are sent both to the main application module (the “3D Rendering Engine” block) in Figure 3.13 and to the JPIP client. The first one then starts the rendering procedure using the data contained in the cache. The complete procedure will be described in Chapter 4, however the basic idea is to use the available geometry information to warp all the available views to the required viewpoint and then combine the various renderings together. The fusion of the different renderings is handled by the “Client Policy” module, that selects the source warped view for every element in the final rendered image. This module will also be described in detail in Chapter 4. At the same time the view parameters are also communicated to the JPIP client that will send a request to the server of the type described in Section 3.3. The server will then compute the packet that need to be transmitted and start sending them to the client that will put the data into the cache. By looking at Figure 3.13 it is possible to see that the cache is divided in two parts: the “JPIP Cache” holds JPEG2000 compressed data-bins, while the “Rendering Cache” holds uncompressed images and depth maps. This allows to find a trade-off between the performances (the data in the Rendering Cache has a very fast access time³) and the memory usage (compressed data in the JPIP Cache use much less space). The JPIP client will decompress on the fly the required information in its cache as soon as the rendering engine asks for data that is not available in the Rendering Cache.

A fundamental aspect of the client framework is that the rendering and transmission module are asynchronous. When the user asks for a view the rendering module will immediately display it using the data already available in the cache without waiting for more information from the server. At the same time a new request is made to the server and as soon as more data becomes available in the cache the rendering module can progressively improve the image quality. This approach permits to offer to the user an interactive browsing experience that is not affected from network delay or bandwidth congestion issues that usually represent a huge problem in systems based on rendered views transmission ([34, 36, 28]).

²In the final completely optimized version of the system this should happen continuously while the user is exploring the scene, in the current implementation when the user changes viewpoint the client shows a temporary rendering and then start the view synthesis procedure that still requires a few seconds.

³The data in the Rendering Cache can also be stored directly into the graphics card memory to be readily used in the hardware accelerated warping operations.

Chapter 4

Distortion-sensitive view synthesis

In the description of the general architecture of our remote visualization system we outlined two main issues: how to combine the information from the different views into the required rendering at client side and the selection of the information to be transmitted. We start from the first issue and then we will exploit the solution we found to develop the transmission strategy at server side. In this chapter we assume we already have a description of the geometry available at client side and we focus on how the required rendering can be reconstructed from the different views of the scene received from the server. The reconstruction of geometry information will be the subject of the next chapter.

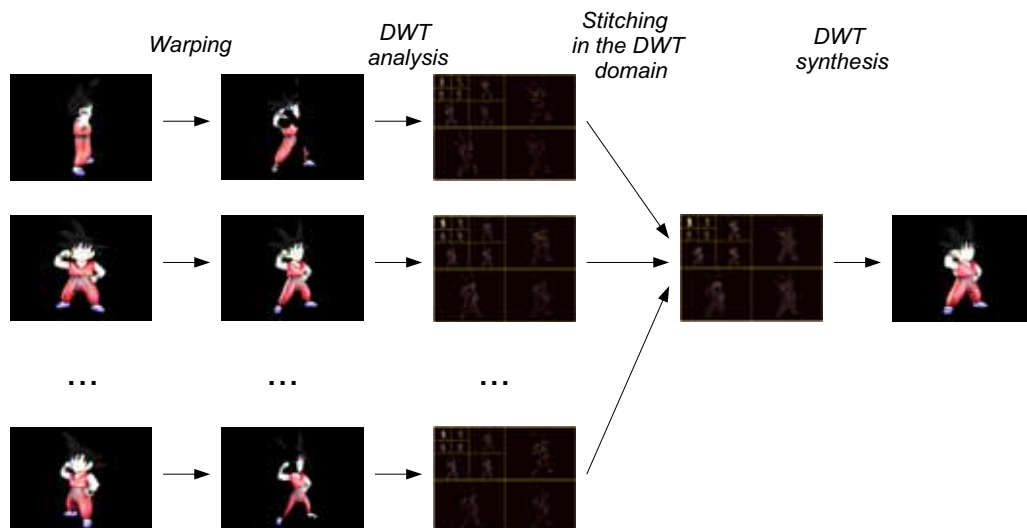


Figure 4.1: *Reconstruction of the required rendering from the different views and depth information.*

Our view reconstruction procedure is made of 4 steps (see Figure 4.1): In the first every view received from the server is warped to the required viewpoint using the geometry information, thus obtaining a set of predictions of the required rendering. Then all the warped views are decomposed using the wavelet transform. In the following

step the samples that will be used in the final rendering are selected from the different subbands of the decompositions of the various warped images on the basis of a minimum distortion policy. Finally wavelet synthesis will be applied to the selected samples to obtain the final rendered view.

Section 4.1 describes how the required view can be predicted by warping an available view to the required viewpoint, then we will introduce a multi-resolution stitching algorithm based on the wavelet transform that can be used to combine the different warpings. The minimum distortion framework used to select the samples will be described in the last part of the chapter. For the rest of the chapter we will assume that the client has already received a set of original view images $V^{i_0}, V^{i_1}, V^{i_2}, \dots$ and a description of the surface geometry G , represented as a triangular mesh. Later, we will see how to replace the geometry with a synthesized depth map obtained from the depth information received from the server. We will also denote with V^* the view required by the user.

4.1 Warping of a single view

This section describes the process of rendering the required view V^* from a single original view image, V^i and the geometry description, G . Let us call Δ_n the n -th triangle of the mesh G . By projecting the nodes of the mesh onto the image planes corresponding to V^* and V^i , we obtain two sets of corresponding triangles, denoted $\{\Delta_n^*\}$ and $\{\Delta_n^i\}$, respectively (see Figure 4.2).

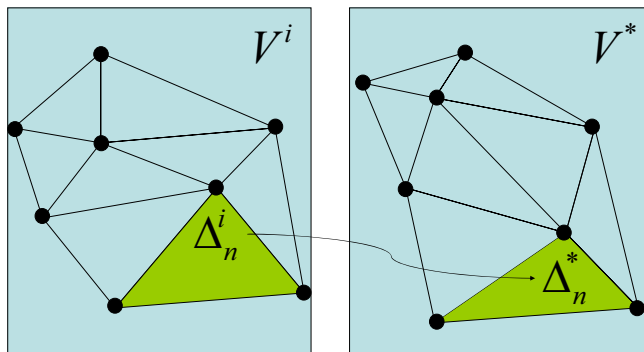
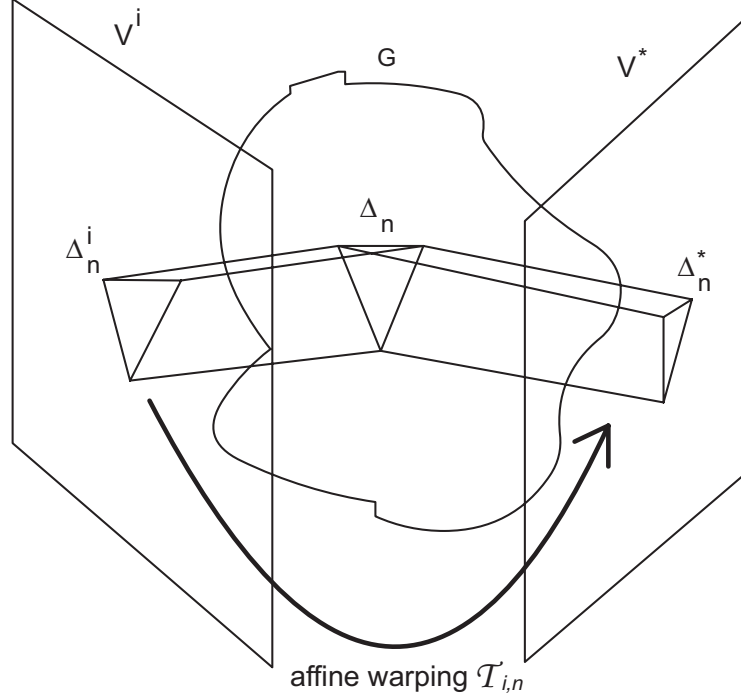


Figure 4.2: *Surface mesh projected onto an original view image V^i and onto V^* .*

Entering in more detail view V^* is associated to a projection matrix P^* that maps the vertex X_i in the 3D space to the point J_i on the 2D image plane of V^* , according to the following relationship in homogeneous coordinates [50]:

$$\begin{bmatrix} w\mathbf{J}_i \\ w \end{bmatrix} = P^* \begin{bmatrix} \mathbf{X}_i \\ 1 \end{bmatrix} \quad (4.1)$$

Figure 4.3: Rendering V^* from geometry G and view V^i

P^* has the structure of a standard projection matrix, given by the multiplication of an intrinsic part containing the camera parameters with a rototranslation matrix:

$$P^* = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R^T & t \\ 0_3^T & 1 \end{bmatrix} \quad (4.2)$$

where R and t are the rotation matrix and translation vector of the user reference system, with respect to the world reference system. Entries f_x , f_y , c_x , c_y and s are the intrinsic parameters of the virtual camera used by the client in order to render the world to the user. Let us denote by P^i the projection matrix associated with view V^i , Δ_n^* and Δ_n^i can be obtained by projecting the same triangle Δ_n of the 3D mesh G on V^* by P^* and on V^i by P^i (Figure 4.3).

It is important to underline that only some of the triangles of the mesh are visible in both the images. That is because some of the triangles could fall outside of the image area in one of the two images, others could be facing backwards and some could be hidden by other elements in the scene. Of course some of the projected triangles may be hidden in one image, but not in the other. If Δ_n^* is hidden, Δ_n^i is not involved in rendering, while if Δ_n^i is hidden, Δ_n^* cannot be rendered from V^i and a “hole” will remain in the part of V^* corresponding to Δ_n^* . Some of the triangles can be partially hidden, but this issue can be avoided by choosing a suitably fine mesh. Equivalently, it is sufficient to subdivide (i.e., remesh) those triangles which straddle object boundaries in V^* or V^i . We introduce the notation \mathcal{O}^i for the set of indexes n such that Δ_n^i is

visible in V^i – i.e.,

$$\begin{aligned}\mathcal{O}^i &= \{n \mid \Delta_n^i \text{ is observable in } V^i\}; \text{ and} \\ \mathcal{O}^* &= \{n \mid \Delta_n^* \text{ is observable in } V^*\}.\end{aligned}$$

Excluding the holes, all the visible triangles Δ_n^* , $n \in \mathcal{O}^*$, are rendered from Δ_n^i by affine warping using the texture mapping technique. This step can be performed very fast exploiting the 3D hardware acceleration features of the graphics card of modern desktop computers. We denote the complete warping operation from V^i to V^* by piecewise affine mapping as $V^* = \mathcal{W}^i(V^i)$. More formally, for every triangle Δ_n , we call \mathcal{W}_n^i the single linear affine image warping operator which aligns the imagery in V^i over Δ_n^i with that in V^* over Δ_n^i . Every different triangle has a different orientation in the 3D space and a different \mathcal{W}_n^i operator. Also, let $V|_{\Delta}$ denote the image obtained by restricting V to the support of Δ , i.e. leaving it unmodified in the region of triangle Δ_n and setting its samples to 0 elsewhere. Then the warping operation \mathcal{W}^i is defined by equation 4.3

$$V^* = \mathcal{W}^i(V^i) = \sum_{n \in \mathcal{O}^i \cap \mathcal{O}^*} \mathcal{W}_n^i(V^i)|_{\Delta_n^*} \quad (4.3)$$

It is important to underline that even if in theory the operator \mathcal{W}_n^i depends only on the pixels inside the triangle Δ_n^i , a good implementation of the affine texture mapping procedure requires a suitable interpolation filter. Several different interpolation filters can be used, from the simple bilinear interpolation filter to more complex cubic or spline filters. The size of the kernel depends on the chosen filter and for some of them (such as spline interpolators of quadratic or higher order) can be even infinite. A consequence of this is that the samples of the warped triangle depend also on pixels outside the source triangle and require us to regard \mathcal{W}_n^i as an operation on the entire domain of V^i , taking the result only over Δ_n^* . As previously said not all the triangles are visible in both the images and the individual affine warping operators \mathcal{W}_n^i are only defined for the pair of triangles visible on both images. These triangles are those whose index n belong to both \mathcal{O}^i and \mathcal{O}^* . For this reason some regions of V^* are undefined because they are not visible in both the views. We divided the undefined regions (“holes”) of V^* into two groups:

1. “Exterior holes” consists of those pixels in the support of V^* (typically a rectangular image) which fall outside the silhouette of the object as perceived from view V^* . This kind of holes is usually common when the scene is an object placed in front of a background that is not part of the scene. Instead in the case of the inspection of virtual places (e.g. a room) the scene usually covers all the view and there are no exterior holes. These holes represent samples that are not part of the scene and do not represent a big issue.

We denote with

$$\mathcal{R}^* = \bigcup_{n \in \mathcal{O}^*} \Delta_n^*,$$

the region of support of the object in view V^* (the “exterior holes” are so represented by the set of all samples that do not belong to \mathcal{R}^* , $(\mathcal{R}^*)^c$).

2. “Interior holes” instead represent triangles that are visible in V^* , but not in V^i . Using the previous notation this is the portion of \mathcal{R}^* which is not visible in V^i and cannot be predicted from the available view. More formally they are defined by the region

$$\mathcal{H}^{i \rightarrow *} = \bigcup_{n \in \mathcal{O}^* \setminus \mathcal{O}^i} \Delta_n^*.$$

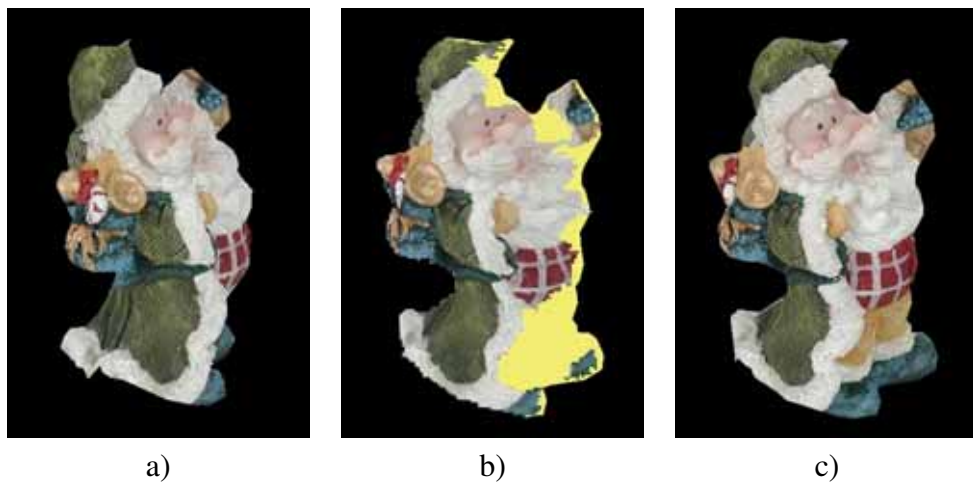


Figure 4.4: *Warping example: a) Available view, b) Prediction, c) Required view*

Figure 4.4b shows an example of the warping procedure. It shows the prediction of image 4.4c using the view in Figure 4.4a. From the figure is clear that only the part of the object that is visible in both images can be predicted. The triangles that are visible in the required view but not in the available one (the “interior holes”) are shown in yellow, while the region outside the object (“exterior holes”) is black.

A final remark is that affine warping does not exactly extend the behaviour of a perspective imaging model into the interior of the projected surface triangles Δ_n^* . The nodes of the mesh are reprojected using the correct perspective model and so are in the “right” position, while the samples inside each triangle are then calculated using texture mapping, that is based on an affine warping and do not fall exactly in the same position as if they were reprojected one after the other using the perspective model¹. However, this error can be made arbitrarily small by reducing the size of the triangles. A suitable remeshing is thus sufficient to solve the inaccuracy in our formulation due to the visibility and affine warping issues.

The proposed approach can be implemented in real systems by a standard warping procedure. The first step consists in calculating the perspective matrices that map the

¹Such an approach is much slower than common texture mapping techniques like the *scanline* algorithm

geometry nodes on the two views. The following one is to map the nodes using the new perspective matrices and the final operation is to warp all the samples using an optimized texture mapping algorithm (algorithms for fast warping of images are well known in computer graphics [51]). To obtain faster performance a good solution is to apply the source view V^i as a texture to the available surface description and then exploit the hardware acceleration features of the graphic card to perform the rendering from the point of view associated to V^* .

4.2 Warping from multiple views

As shown in the previous section in most cases a single view is not enough to obtain the correct rendering from a different viewpoint and it is necessary to combine the information from the reprojection of multiple original view images. To correctly exploit the information from multiple warpings two fundamental issue must be solved:

- Multiple warped views cover a bigger set of triangles in V^* but at the same time some triangles can be visible in different available views. A criterion to blend the information from different source views inside each triangle is needed.
- In theory the warping of the different views should be perfectly aligned but uncertainty in the geometry due to the acquisition procedure or to compression together with other issues like different lighting conditions or reflections can cause artefacts in the boundary between triangles taken from different views.

The simplest way to combine the information from multiple original view images, V^{i_0}, V^{i_1}, \dots , is to simply average the results obtained by mapping each of them onto the desired view. Equation 4.4 gives a formal representation of this procedure:

$$V^* = \sum_{n \in (\cup_i \mathcal{O}^i) \cap \mathcal{O}^*} \frac{1}{N(n)} \sum_k \mathcal{W}_n^{i_k} (V^{i_k}) \quad (4.4)$$

Where $N(n)$ represents the number of views in which the n^{th} triangle is visible and we set $\mathcal{W}_n^{i_k} (V^{i_k})$ to zero in the triangles not visible in V^{i_k} .

Unfortunately, as previously pointed out any imperfections in the surface geometry representation will produce misalignment amongst the separate renderings $\mathcal{W}^i (V^i)$, and averaging tends to blur high frequency spatial features. The simple average shows no preference for one possible rendering over another, a weighted one can provide better results but images still remain blurry, specially in the case of unreliable geometry (see Figure 4.5a).

Another solution is to select a single most appropriate original view image for every triangle. We refer to this as “stitching”. We will denote with i_n^* the “best stitching source” for the triangle Δ_n . The final synthesized view will be constructed by just placing side by side all the triangles rendered from the different best stitching sources, according to

$$V^* = \sum_{n \in (\cup_i \mathcal{O}^i) \cap \mathcal{O}^*} \mathcal{W}_n^{i_n^*} (V^{i_n^*})|_{\Delta_n^*} = \sum_{n \in \mathcal{O}^*} \mathcal{W}_n^{i_n^*} (V^{i_n^*})|_{\Delta_n^*} \quad (4.5)$$

Of course, i_n^* must have the property that $n \in \mathcal{O}^{i_n^*}$ so that $\Delta_n^{i_n^*}$ is visible in $V^{i_n^*}$. Also, note that the “interior holes” in V^* are now restricted to the portion of \mathcal{R}^* which is not visible from any of the available views V^i , i.e.,

$$\mathcal{H}^* = \bigcap_i \mathcal{H}^{i \rightarrow *} = \bigcup_{n \in \mathcal{O}^* \setminus (\cup_i \mathcal{O}^i)} \Delta_n^*$$

We will deal with the problem of how to select the best stitching source for each triangle in the next section. Figure 4.5b shows a rendering example obtained with this approach. It allows to avoid the blurring problem but it tends to produce visible discontinuities at the boundaries between adjacent triangles rendered from different original source views. This is due to the uncertainty in the geometry description that cause misalignment between samples rendered from different sources and also to illuminant-dependent shading effects. A possible extension of this approach is to perform some averaging in the vicinity of boundaries of triangles taken from different views, for example by forming a weighted average of the various candidates, $\mathcal{W}^i (V^i)$, in the vicinity of stitching boundaries. However this approach reintroduces the blurring problem in the high frequency due to averaging; moreover, it is unclear how much averaging is required to eliminate visible artefacts.

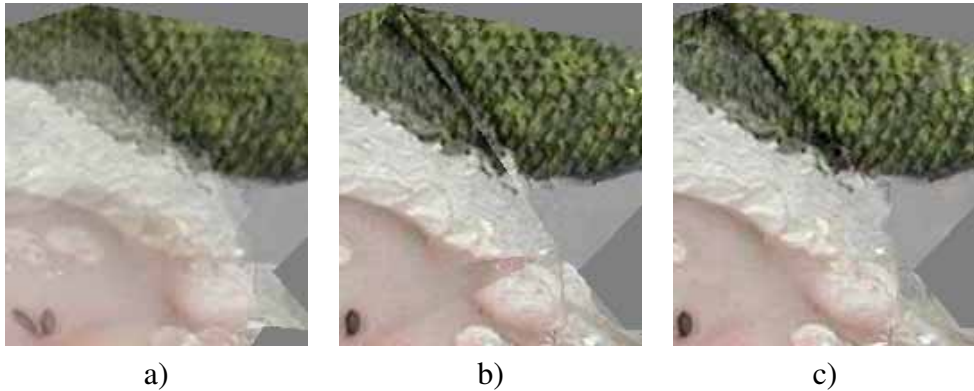


Figure 4.5: *Combining multiple views: a) Averaging, b) Stitching, c) Multi-resolution stitching based on DWT*

A possible solution to this dilemma, is to perform the stitching within a multi-resolution framework [52] such as the Laplacian pyramid or a Discrete Wavelet Transform (DWT). These approaches perform much more smoothing in the lower spatial frequency components than in the higher ones (the highest frequency details usually are not smoothed at all). Figure 4.5c shows how these approaches allows to avoid the discontinuities at the boundaries while passing higher frequency content unaltered, thus avoiding to destroy high frequency details.

In the present work we decided to use a stitching procedure based on the Discrete Wavelet Transform (DWT). One of the reason for such a choice is that our original view images are compressed using JPEG2000. This compression standard is based on

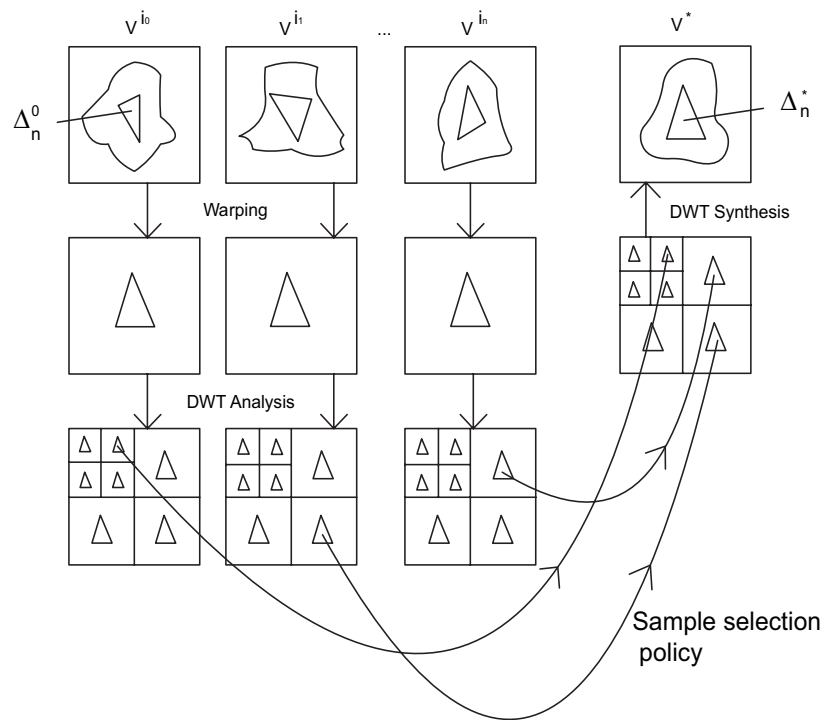


Figure 4.6: *Stitching in the DWT domain*

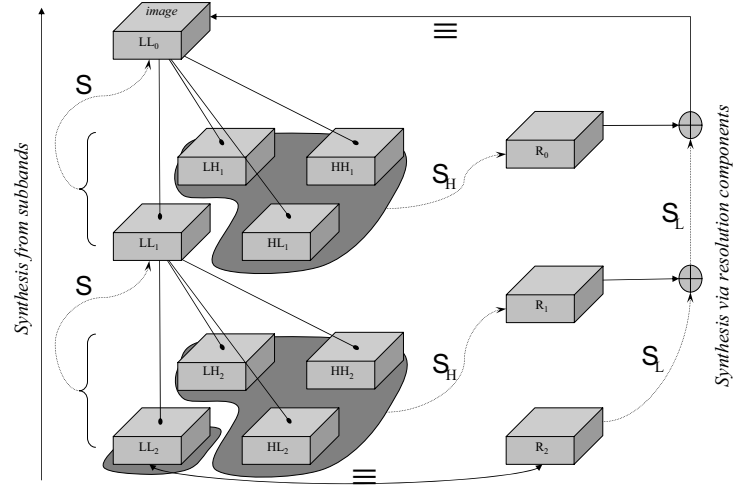


Figure 4.7: Decomposition of the image with a 2 level DWT

the DWT (see Section 2.4) and stores a compressed version of the coefficient of the wavelet decomposition of the image.

Figure 4.6 shows an overview of our multi-resolution stitching framework: in the proposed approach we will firstly reproject every original view image thus creating a set of warped images $\mathcal{W}^{i_k} (V^{i_k})$, $k = 0, 1, 2, \dots$ (where k represent the index of the source view). Then we will stitch all these warped images together using a DWT-based multi-resolution approach. To simplify the notation from this point we will call by $V^{i \rightarrow *}$ $\triangleq \mathcal{W}^i (V^i)$ the reprojection of V^i on the target viewpoint. After the warping we first decompose each $V^{i \rightarrow *}$ using the wavelet transform. We use a D level DWT decomposition based on the 9/7 Daubechies filter. Every resolution level d can be decomposed into a half resolution image LL_{d+1} and a set of high frequency subbands HL_{d+1} , LH_{d+1} and HH_{d+1} . After recursively applying the DWT d times, every image is decomposed into a low resolution base image $LL_D^{i \rightarrow *}$ and a collection of high-pass subbands $HL_d^{i \rightarrow *}$, $LH_d^{i \rightarrow *}$ and $HH_d^{i \rightarrow *}$, as shown in Figure 4.7.

To reconstruct the image $V^{i \rightarrow *} \equiv LL_0^{i \rightarrow *}$ we can recursively apply the DWT synthesis operator \mathcal{S} . Exploiting the fact that \mathcal{S} is a linear operator, every level can be expressed as:

$$\begin{aligned}
 LL_d &= \mathcal{S} (LL_{d+1}, HL_{d+1}, LH_{d+1}, HH_{d+1}) \\
 &= \mathcal{S} (LL_{d+1}, \mathbf{0}, \mathbf{0}, \mathbf{0}) + \mathcal{S} (\mathbf{0}, HL_{d+1}, LH_{d+1}, HH_{d+1}) \\
 &= \mathcal{S}_L (LL_{d+1}) + \underbrace{\mathcal{S}_H (HL_{d+1}, LH_{d+1}, HH_{d+1})}_{R_d}
 \end{aligned}$$

This decomposition divides the operations required for every single stage of the DWT synthesis into two groups, the low pass portion \mathcal{S}_L , and the high pass portion \mathcal{S}_H . We also represent with R_d the “detail” image formed from the three high-pass subbands at decomposition level $d + 1$. We also write R_D for LL_D so that $R_0^{i \rightarrow *}$, $R_1^{i \rightarrow *}$, ..., $R_D^{i \rightarrow *}$ together represent the complete set of resolution components for warped image $V^{i \rightarrow *}$. The idea is now to take the samples in the different subbands of every resolution level from the different decomposition of the warped views according to the choice we made for every triangle. The projection of the mesh on the warped image $V^{i \rightarrow *}$ can be mapped to the various subband by simply scaling it by 2^d . Every triangle Δ_n^* is so represented in the subbands at level d by its scaled version $\Delta_{n,d}^*$, reduced in size by factor 2^d . In the simple stitching algorithm we forced every triangle to be taken from a single view, but in this multi-resolution framework we have more freedom and we can take a triangle from a certain view in one subband and from a different one in another. We will use the notation $i_{n,d}^*$ for the different stitching sources of every triangle in every subband. This property is very useful because, as will become more clear in the following discussion, one source image might provide high quality details at some resolution levels but not at others, depending upon compression noise and the view orientation. Even if a different choice is theoretically possible for every single subband, in this work we will take each triangle in the three high frequency subbands HL_d , LH_d and HH_d at the same level from the same source view to avoid wavelet-related artefacts in the synthesized views. After selecting the source view for every sample in every subband the final view can be reconstructed by recursively applying the wavelet synthesis operator.

There are two possible options for the stitching of the data in a wavelet-based multi-resolution framework. The first, shown in Figure 4.8 for the case of two views consists in stitching in each subband the samples from the different source views and then perform the high frequency synthesis to get the resolution components. The various resolution component can be then synthesized up into the final image.

The second option, depicted in Figure 4.9, instead consists in synthesizing the three high frequency subbands into the corresponding resolution component independently for each source image. For every resolution component we obtain a different representation for each source view and then we can stitch together the data inside the resolution components (this is possible because we forced the same choices on all the three high frequency subbands at the same resolution). The stitched resolution components can then be synthesized up into the final image. Theoretically the second approach should avoid some artefacts related to misalignments in the high frequencies, but practical tests have shown only limited difference between the two approaches.

More formally the second approach proceeds by separately stitching each resolution component to form:

$$R_d^* = \sum_{n \in \mathcal{O}^*} R_d^{i_{n,d}^*} \Big|_{\Delta_{n,d}^*}, \quad d = 0, 1, 2, \dots, D \quad (4.6)$$

A big issue of this approach is that the formulation provided in equation (4.6) is not always able to work properly in proximity of “holes” in the various warped

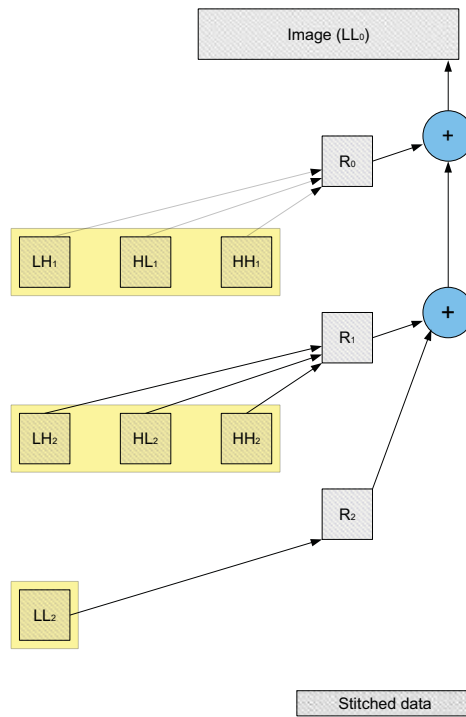


Figure 4.8: Multi-resolution stitching in the wavelet subbands

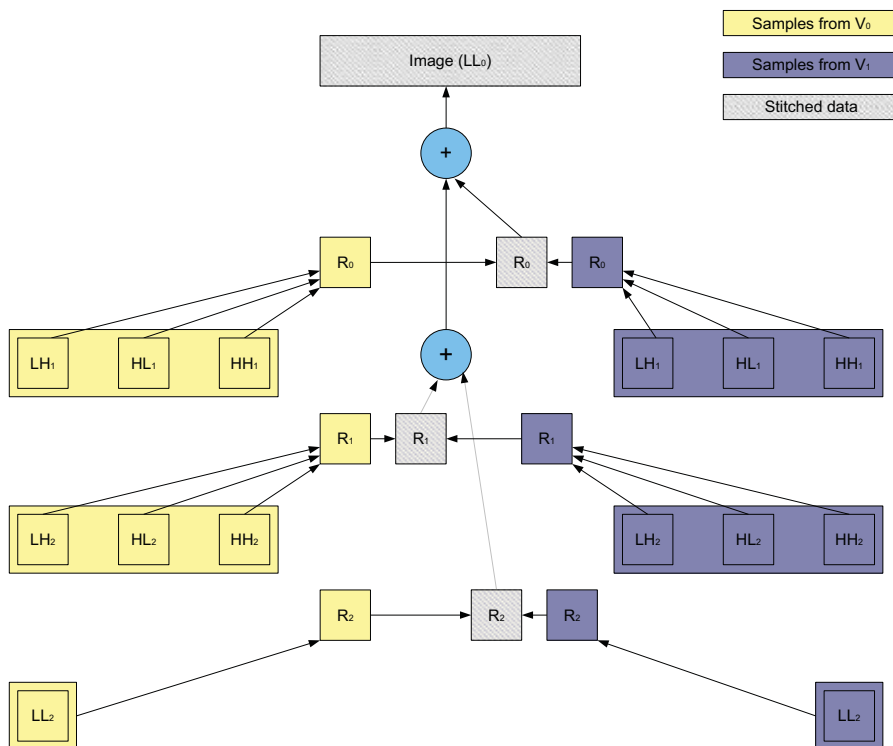


Figure 4.9: Multi-resolution stitching in the resolution components

views $V^{i \rightarrow *}$. The key problem is that the DWT transform involves overlapping basis functions, and resolution component samples which lie fully within a visible triangle may be influenced by the invalid data associated with holes. How to handle this issue will be the subject of Section 4.5.

4.3 Distortion-based selection of the stitching sources

The main issue in combining the information from the different views at client side is how to select the source view for the various triangles in the different subbands. In [10] we introduced a framework to estimate the distortion in the various subbands of the rendered views. A more refined version of the framework is presented in [11], in which we also removed the assumption that the geometry is represented by a triangular mesh and we estimate the distortion for every sample. Three main sources of distortion are taken into account, the quantization distortion due to compression in the source images, the translational error in the warped views due to geometry uncertainty and the color difference between corresponding pixels in different views due to lighting and reflection issues. An additional distortion term has also been included to take into account holes-related issues. The estimated distortion will be used to select the source views for the various samples in the wavelet subbands. In most situations we will simply choose for every sample the source image with the lower distortion, but in some particular cases (specially to handle “holes” related issues) a weighted average of more sources is also used.

4.4 Accounting for image compression distortion

The first source of distortion taken into account is the error introduced by lossy compression of the views. It is important to recall that in the proposed system the views are scalably compressed and progressively transmitted, so the compression distortion for a certain view continuously change. Just after moving to a new viewpoint, when only a small amount of data has probably been transmitted for a new view close to the required viewpoint, it can also be very high. We compressed all the images in JPEG2000, that performs the same wavelet decomposition we have used for the stitching and then quantize the subband samples. At server side it is possible to compare the DWT samples with the quantized version after compression and get the exact quantization distortion for every sample. If we apply the wavelet synthesis to reconstruct the image, the quantization error in the reconstructed view can be expressed as a function of the errors in the different subbands:

$$\delta V^i[\mathbf{n}] = \sum_b \sum_{\mathbf{k}} \delta B_b^i[\mathbf{k}] \cdot S_{\mathbf{k}}^b[\mathbf{n}] \quad (4.7)$$

In equation (4.7) B_b^i is subband b from image V^i , $\delta B_b^i[\mathbf{k}]$ is the error in the \mathbf{k}^{th} sample of this subband and $S_{\mathbf{k}}^b$ is the synthesis basis vector (itself an image) for that

sample².

The distortion for every sample can be easily computed and stored in a database at server side. Unfortunately storing the distortion for every sample in every subband of every view for each quality layers (in the current implementation there are 7 of them) leads to huge databases. In the mesh-based approach we are describing now a good solution is to store the distortion for every triangle, in order to match the fact that choices are made on the triangles. Other possible solutions, that we will use later when we will move to the depth map based representation of the geometry, are to store the distortion for every squared block of pixels (in our experimental tests we used 8x8 and 32x32 blocks) or for every codeblock of the JPEG2000 compressed image³.

The transmission of the distortion information to the client is not a good solution because it introduces a huge overhead in the amount of transmitted data. This issue can be partially solved by calculating the distortion on very large blocks, but in this way the distortion values we use could be not too accurate. A better solution is to estimate the distortion directly from the compressed image using the method described in [53]. This technique allows to obtain a reasonable estimate of the distortion (usually with an error smaller than 3 db) directly from the JPEG2000 compressed datastream with the aid of just a single extra parameter, provided globally for each subband.

4.4.1 Propagation of the distortion through DWT synthesis, warping and DWT analysis

After getting the distortion in the source images, the next step is to understand how the distortion in the subbands of the source images is mapped to the warped views through the DWT synthesis, the affine warping and the DWT analysis of the warped views.

The quantization error at location \mathbf{p} in resolution component $R_d^{i \rightarrow *}$ after the warping and the multi-resolution analysis is:

$$\delta R_d^{i \rightarrow *}[\mathbf{p}] = \langle \mathcal{W}^i(\delta V^i), A_{\mathbf{p}}^d \rangle = \sum_b \sum_{\mathbf{k}} \delta B_b^i[\mathbf{k}] \cdot \langle \mathcal{W}^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle \quad (4.8)$$

where $A_{\mathbf{p}}^d$ is the analysis basis vector (itself an image) associated with that sample, and $\langle \cdot, \cdot \rangle$ signifies the inner product between two images. As shown in Figure 4.10, an important observation is that the warping operation can spread out the distortion in a certain subband to all the others (for example a rotation can move distortion energy between the 3 high frequency bands HL, LH and HH and an enlargement can spread the distortion through different resolution levels).

To calculate the total quantization error energy inside a triangle $\Delta_{n,d}^*$ in resolution component $R_d^{i \rightarrow *}$ we can just sum up the error corresponding to the samples that fall into its region of support:

²We use vectors such as $\mathbf{k} = [k_1, k_2]$ and $\mathbf{n} = [n_1, n_2]$ to denote 2D coordinates.

³This last solution will be useful later in exploiting this framework for the server policy considering that the server must always transmit entire codeblocks to the client.

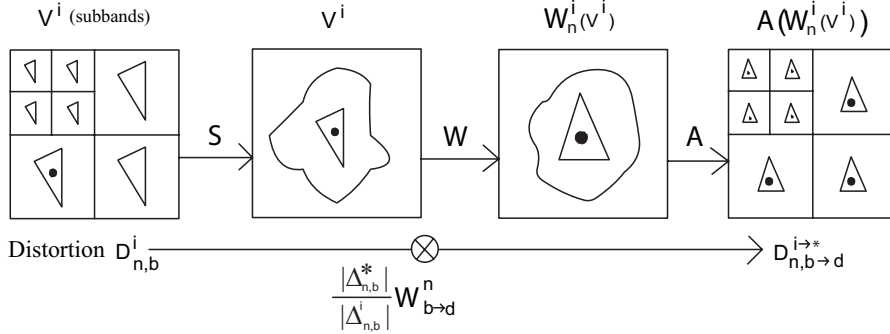


Figure 4.10: Propagation of the distortion

$$D_{n,d}^{i \to *} = \sum_{\mathbf{p} \in \Delta_{n,d}^*} |\delta R_d^{i \to *}[\mathbf{p}]|^2 \quad (4.9)$$

By substituting equation (4.8) into equation (4.9) we obtain:

$$D_{n,d}^{i \to *} = \sum_{\mathbf{p} \in \Delta_{n,d}^*} \left(\left| \sum_b \sum_{\mathbf{k}} \delta B_b^i[\mathbf{k}] \cdot \langle \mathcal{W}^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle \right|^2 \right) \quad (4.10)$$

A very common assumption in the distortion-directed decision literature is that the individual subband quantization errors $\delta B_b^i[\mathbf{k}]$ are approximately uncorrelated. With this assumption all the cross-terms of the form $\sum_{b_1, b_2} \sum_{\mathbf{k}_1, \mathbf{k}_2} \delta B_{b_1}^i[\mathbf{k}_1] \delta B_{b_2}^i[\mathbf{k}_2]$ in the quadratic expression evaluate to 0 and it is possible to rewrite the sum of equation (4.10) as:

$$D_{n,d}^{i \to *} \approx \underbrace{\sum_b \sum_{\mathbf{p} \in \Delta_{n,d}^*} \sum_{\mathbf{k}} |\delta B_b^i[\mathbf{k}]|^2 \cdot \langle \mathcal{W}^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2}_{D_{n,b \to d}^{i \to *}} \quad (4.11)$$

With the assumption of uncorrelated quantization error equation (4.11) can be taken as a strict equality. $D_{n,b \to d}^{i \to *}$ represent the fraction of the distortion in subband d of triangle n warped to the viewpoint of V^* coming from subband b . Due to the decay of the finite support operators in $\langle \mathcal{W}^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle$, $D_{n,b \to d}^{i \to *}$ depends principally on the distortion contributions $\delta B_b^i[\mathbf{k}]$ which are found inside $\Delta_{n,b}^i$, the projection of Δ_n^i into subband B_b^i . This allows us to make the simplifying approximation of a uniform quantization error power over the entire subband. By defining with $D_{n,b}^i$ the distortion into subband b of triangle n and by $\Delta_{n,b}^i$ the area of the triangle in subband b we can write:

$$|\delta B_b^i[\mathbf{k}]|^2 = D_{n,b}^i / |\Delta_{n,b}^i| \quad (4.12)$$

With the uniform error power assumption, by substituting (4.12) into the expression for $D_{n,b \to d}^{i \to *}$ we obtain

$$\begin{aligned}
D_{n,b \rightarrow d}^{i \rightarrow *} &= \frac{D_{n,b}^i}{|\Delta_{n,b}^i|} \cdot \sum_{\mathbf{p} \in \Delta_{n,d}^*} \sum_{\mathbf{k}} \langle \mathcal{W}^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2 \\
&\approx D_{n,b}^i \cdot \frac{|\Delta_{n,d}^*|}{|\Delta_{n,b}^i|} \cdot W_{b \rightarrow d}^n
\end{aligned} \tag{4.13}$$

Here, $W_{b \rightarrow d}^n$ measures the average value of the expression $\sum_{\mathbf{k}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2$ over a range of indices \mathbf{p} . This is reasonable, since $S_{\mathbf{k}}^b$ and $A_{\mathbf{p}}^d$ are both periodically shift invariant and the single affine operator, \mathcal{W}_n^i , captures the behaviour of \mathcal{W}^i locally over the triangle in which we are interested.

From a practical viewpoint, the formulation of equation (4.13) enables us to use a pre-computed table of weights, $W_{b \rightarrow d}^n$. This is very useful because the computation of the distortion directly from equation (4.11) for every triangle requires too much computational time to be feasible in a real-time system.

To understand how it is possible to build the table of weights, let us start from taking into account the effects of the affine warping. An affine transformation is a composition of 4 different operations: translation, rotation, scaling and shear. The actual energy after DWT analysis and synthesis slightly depend on the position of the samples in the image and in this context the effects of translation can be ignored with only a small loss in the accuracy of the weight values.

Therefore we will consider affine transformations mapping (x_1, y_1) to (x_2, y_2) according to the following expression:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{4.14}$$

where α is the rotation angle; s is the shear factor and c_1 and c_2 are the scaling factors in the direction of x_1 and x_2 . The first step in the construction of the table is the quantization of the affine parameters, in order to have a not too large number of possible affinities.

Angle α can be linearly quantized in the range between 0 and $\pi/2$ exploiting the symmetric properties of the wavelet transform to deal with angles of more than $\pi/2$: a rotation of 90 degrees will swap the HL and LH subband and a 180 degrees one will have no effect on the energy. The quantized α values for a N_α levels quantization are :

$$\alpha = \frac{\pi}{2} * \frac{i}{N_\alpha} \quad \text{with } i = 0 \dots (N_\alpha - 1)$$

Shear s usually is very close to zero because the reprojection operation usually does not have a great impact on this parameter. It can be quantized using a cubic approximation as:

$$s = K_s * \left(\left(\frac{2}{N_s} \right)^3 \cdot \left(i - \frac{N_s}{2} \right)^3 \right) \quad \text{with } K_s \text{ constant and } i = 0 \dots N_s$$

Where the number of levels is $N_s + 1$. This expression returns a set of values in the range from $-K_s$ to K_s with a dense quantization around the 0 and a coarse one

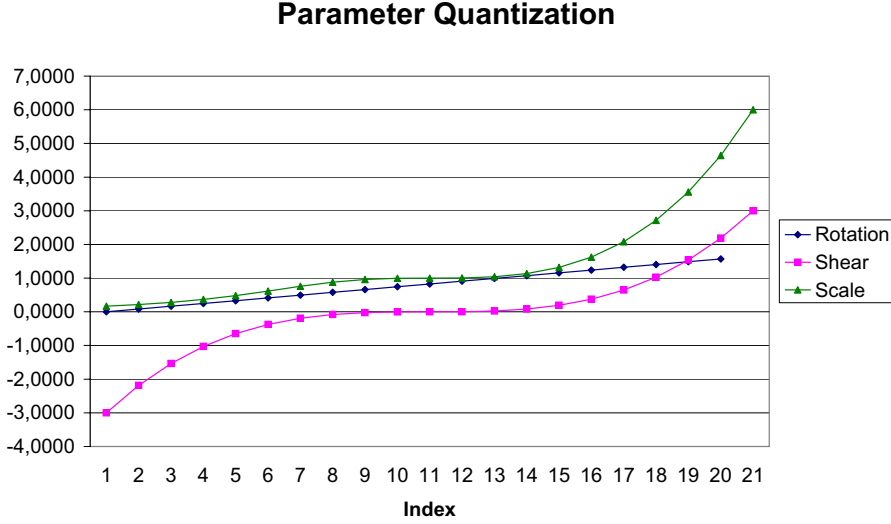


Figure 4.11: *Quantization of the affine parameters*

for larger values (the purple line in the plot of Figure 4.11 shows the shear values for $K_s = 3$ and $N_s = 20$). The shear factor in the most common affine warping operations is very close to 0 and it is also possible to avoid considering it in the table or use a very coarse quantization. Scaling factors c_1 and c_2 are frequently close to one, since usually most surface triangles are quite parallel to the viewer. Therefore it makes sense to sample c_1 and c_2 more densely near one (scaling quantization values are plotted in green in Figure 4.11). We used a cubic expression similar to the previous one, but this time centred around one:

$$\begin{aligned}
 c_1 &= \left(K_1 \cdot \left(\frac{2}{N_1}\right)^3 \cdot \left(\frac{N_1}{2} - i\right)^3 + 1\right)^{-1} && \text{with } i \text{ in the range } 0 \dots \left(\frac{N_1}{2} - 1\right) \\
 c_1 &= K_1 \cdot \left(\frac{2}{N_1}\right)^3 \cdot \left(i - \frac{N_1}{2}\right)^3 + 1 && \text{with } i \text{ in the range } \frac{N_1}{2} \dots N_1 \\
 c_2 &= \left(K_2 \cdot \left(\frac{2}{N_2}\right)^3 \cdot \left(\frac{N_2}{2} - i\right)^3 + 1\right)^{-1} && \text{with } i \text{ in the range } 0 \dots \left(\frac{N_2}{2} - 1\right) \\
 c_2 &= K_2 \cdot \left(\frac{2}{N_2}\right)^3 \cdot \left(i - \frac{N_2}{2}\right)^3 + 1 && \text{with } i \text{ in the range } \frac{N_2}{2} \dots N_2
 \end{aligned}$$

Where K_1 and K_2 represent the maximum scale factor represented in x_1 and x_2 respectively and N_1 and N_2 is the number of quantization steps in the two components. The scaling factors have a great impact on the energy in the various subbands and this elements usually need a more precise quantization than the shear and rotation.

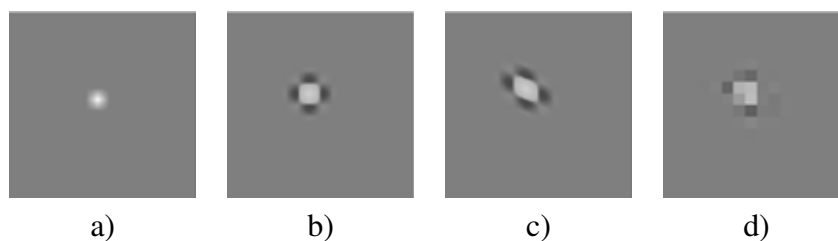


Figure 4.12: Computation of $W_{b \to d}^n$: a) impulsive error in LL subband at level $d = 4$ b) result of synthesis c) result of warping d) result of analysis in the same subband.

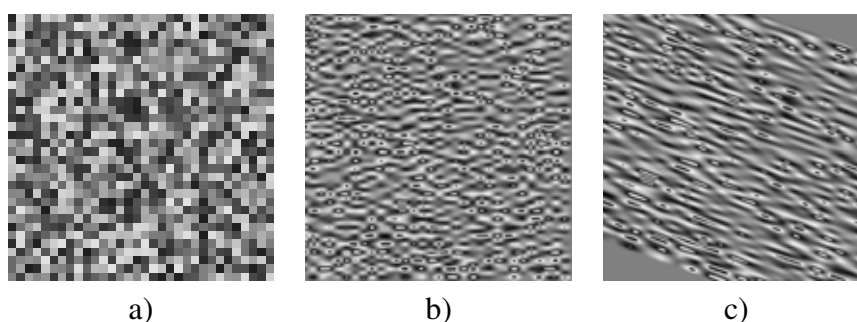


Figure 4.13: Computation of $W_{b \to d}^n$: a) White noise input in LL subband at level $d = 5$ b) result of synthesis c) result of warping.

A practical way to compute $W_{b \to d}^n$ is to place an impulsive signal⁴ in one of the subbands and compute synthesis, warping and analysis in order to get $W_{b \to d}^n$. Figure 4.12 shows an example of this procedure: Figure 4.12a shows the 16x16 subband at decomposition level 4 with a single sample at the center. After synthesis the sample expands in a blob on the 256x256 complete image of Figure 4.12b. The result of warping by an affine transformation are shown in Figure 4.12c. The result of the analysis in the LL subband at level 4, is shown by the 16x16 image of 4.12d. Its energy gives the value of the coefficient.

This approach does not offer fully satisfactory results. The position of the sample affects the result of analysis and synthesis in DWT and the impulse model does not match the shape of the quantization error in the subbands of real images. We switched to a more accurate procedure: firstly we fill up the source subband with random white noise (Figure 4.13a) and we measure its energy $E_{Subband}$. The reason for this is that the white noise offers a better representation of the quantization error in the images. Then we apply DWT synthesis, warping and DWT analysis (Figure 4.13b and 4.13c). To calculate the coefficient value we use equation 4.15 :

$$W_{b \to d}^n = E_{Region} \cdot \frac{A_{Subband}}{A_{Region}} \cdot \frac{1}{E_{Subband}} \cdot c_1 c_2 \quad (4.15)$$

Where E_{Region} is the energy in a rectangular region close to the center of the sub-

⁴We put a value of 1 at the centre of the subband and set it to 0 elsewhere.

band, while A_{Region} and $A_{Subband}$ are the area of the region and of the subband respectively. We take the energy in a region close to the centre to avoid boundary-related artefacts and we scale by the ratio of the areas to compensate for the smaller size of the region. We also multiply by the scale factor $c_1 \cdot c_2$ to account to the enlargement of the region with non-zero samples due to scaling. The coefficient is obviously the ratio of the computed energy and the energy in the input subband $E_{Subband}$. Experiments showed that the weights calculated with this procedure give a good approximation of the actual values given by (4.11).

To build the table is necessary to compute $W_{b \rightarrow d}^n$ for every combination of the 4 affine warping parameters and input/output subbands. Using 7 resolution levels and storing a 32 bit floating point representation of the coefficients for all the 4 subbands in each level the size of the table is then given by:

$$D = 4 \cdot 28^2 \cdot N_\alpha \cdot N_s \cdot N_1 \cdot N_2 \quad (4.16)$$

N_α	N_s	N_1	N_2	Size (MB)
20	21	21	21	554
14	4	20	20	67
10	10	10	10	30
10	1	20	20	12

Table 4.1: *Quantized weights table size*

This can lead to a very large table (see Table 4.1). The table is static and does not need to be transmitted, but too large tables can increase dramatically the memory usage of the application. The table corresponding to the quantized values in figure 4.11 requires 554 MB and is too large for practical purposes. By reducing by half the precision of all the warping parameters we can build a much smaller 30 MB table, but by taking into account the different impact of the 4 parameters on the subband energies is possible to obtain a better trade-off between the size of the table and the coefficients precision. For example it was experimentally verified that with $N_\alpha = 10$, $N_s = 1$ and $N_1 = N_2 = 20$ we can achieve a not too coarse approximation with just a 12 MB table. Further investigation on how to reduce the table size by exploiting additional symmetrical properties in the DWT or by decomposing the weights values in terms depending on the different warping parameters is still going on.

4.4.2 Effects of extra-band energy

Before going on, it is helpful to develop some intuition concerning the expected behaviour of our distortion formulation. The 9/7 biorthogonal wavelet transform used for our experiments is very nearly orthonormal, subject to appropriate normalization of the subband samples. We can suppose that the synthesis and analysis kernels in our transforms are mutually orthonormal, which must certainly be the case if our multi-resolution transform is derived from the orthonormal DWT, following the procedure outlined in the previous sub-section. Finally observe that the affine operator \mathcal{W}_n^i

stretches each $S_{\mathbf{k}}^b$ by an amount equal to the ratio of the area between the target and source triangles $|\Delta_n^*| / |\Delta_n^i|$, amplifying its energy by “roughly” the same amount (we will revisit this point shortly). From the orthonormality of the $A_{\mathbf{p}}^d$, it follows that

$$\sum_d \sum_{\mathbf{p}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2 = \|\mathcal{W}_n^i(S_{\mathbf{k}}^b)\|^2 = |\Delta_n^*| / |\Delta_n^i|, \forall \mathbf{k}$$

Now $W_{b \rightarrow d}^n$ is the average value of $\sum_{\mathbf{k}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2$ taken over \mathbf{p} . It can be shown that this is $|\Delta_{n,b}^i| / |\Delta_{n,d}^*|$ times the average value of $\sum_{\mathbf{p}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2$ taken over \mathbf{k} :

$$W_{b \rightarrow d}^n = \frac{\sum_{\mathbf{p}} \sum_{\mathbf{k}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2}{|p|} = \frac{|\Delta_{n,b}^i|}{|\Delta_{n,d}^*|} \cdot \frac{\sum_{\mathbf{k}} \sum_{\mathbf{p}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2}{|k|} \quad (4.17)$$

By substituting (4.17) into equation (4.13) we can conclude that

$$\begin{aligned} \sum_d D_{n,d}^{i \rightarrow *} &= \sum_d \sum_b D_{n,b}^i \cdot \frac{|\Delta_{n,d}^*|}{|\Delta_{n,b}^i|} \cdot W_{b \rightarrow d}^n \\ &= \sum_b D_{n,b}^i \sum_d \sum_{\mathbf{p}} \langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle^2 = \frac{|\Delta_n^*|}{|\Delta_n^i|} \sum_b D_{n,b}^i \end{aligned} \quad (4.18)$$

At first glance, if we assume that the total distortion in the source triangle $\sum_b D_{n,b}^i$ is roughly proportional to its area, $|\Delta_n^i|$, from equation (4.18) we could guess that the total distortion in the warped triangle $\sum_d D_{n,d}^{i \rightarrow *}$ should be roughly independent of the affine operator \mathcal{W}_n^i . However, this first impression is wrong, since a couple of important issues are missing from this picture.

The first important oversight in the above derivation is that \mathcal{W}_n^i must be a band-limited warping operator. In fact warping a spatially continuous image does indeed amplify its energy directly in proportion to $|\Delta_n^*| / |\Delta_n^i|$. This property is valid also for discrete images only when all spatial frequency components in the warped image can still be represented. If $|\Delta_n^*| / |\Delta_n^i| < 1$, the ideal continuous warping operator necessarily generates extra super-Nyquist spatial frequency components which must be suppressed in the discrete equivalent to avoid aliasing. This means that source views V^i for which $|\Delta_n^*| / |\Delta_n^i| < 1$ should yield less distortion power, making them more favourable selections for a single “best stitching source”, assuming that all the other parameters all the same. This observation also reminds us that care must be taken when warping both the real image samples and the synthesis basis images (for example in the computation of the weights $W_{b \rightarrow d}^n$), not to simply resample the image with a fixed interpolation function. A simple way to achieve the desired band-limiting behaviour is to perform the warping operation at a higher resolution and then sub-sample the result, using an appropriate anti-aliasing filter.

The second important oversight in the above derivation arises when the warping operation represented by \mathcal{W}_n^i is expanding the triangles in the source image, i.e.

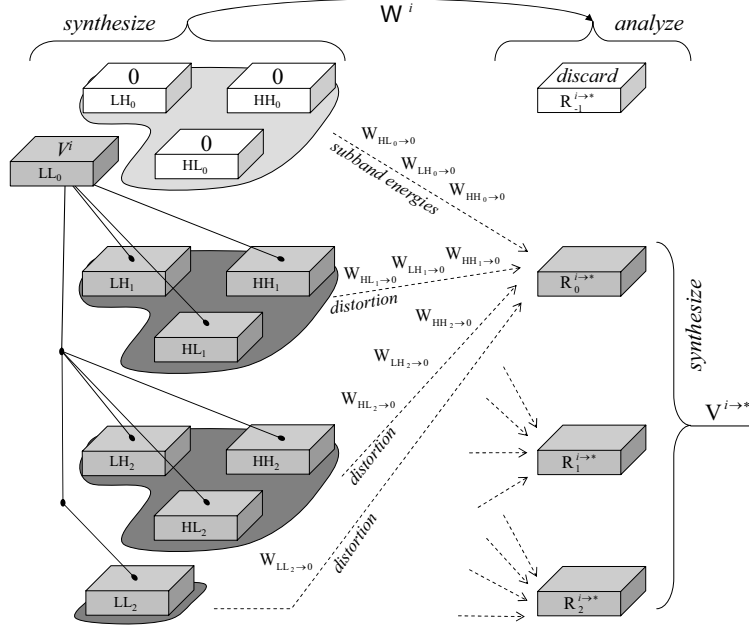


Figure 4.14: Procedure for mapping both imagery and distortion information from view V^i to warped view $V^{i \rightarrow *}$. Extra, hypothetical resolutions are shown lightly shaded.

$|\Delta_n^*| / |\Delta_n^i| > 1$. In this case, the highest resolution components of V^* cannot be recovered at all, since they depend upon high frequency components that does not exist in the source view. The absence of these components represents a form of additional distortion, in addition to that arising from quantization noise in the compressed source views. One way to take into account this effect is to include in the summation on the right hand side of equation (4.18) subbands from a set of hypothetical resolutions above the ones of the original images. These subbands are shown lightly shaded in Figure 4.14, and we will refer to them as “extra resolutions”. The figure also shows how the first issue described above may be addressed at the same time by performing the warping procedure at the higher resolution and then simply discarding the extra resolution components as a form of band-limited downsampling.

The main issue in the with the extra resolutions procedure is the estimation of the source distortion $D_{n,b}^i$ in these hypothetical subbands. Since these subbands are missing in the source images, their distortions are identical to their energies $E_{n,b}^i$. A possible way to obtain an estimate for these energies is to project each source image onto the other in turn and take the maximum of the energy produced by these warpings. The idea is that in some of the various views Δ_n should have a greater area and so more information on the higher frequencies. This, of course, gives only a conservative estimate of the energy in the extra resolutions. At client side this approach can be exploited

by projecting only the source images that are available at client side. By using only the available data we could underestimate the energy in the extrabands. However the energy is usually underestimate in the points where no available image is better aligned with the surface than the one we are analyzing. In this case the underestimation has no relevant impact on the distortion choices because there are not better views. If higher resolution versions of the source images than the ones used in the remote visualization are available, another obvious solution (only at server side) is to directly compute the extra resolution energy from them.

The practical consequence of these two effects is that original source views for which $|\Delta_n^*|/|\Delta_n^i|$ is smaller, are always preferred over those for which $|\Delta_n^*|/|\Delta_n^i|$ is larger, assuming that the source compression noise power is similar. This agrees with our intuition, that the source view whose focal plane is most parallel to the scene surface should provide the most information about its texture; this is the view for which the transformation \mathcal{W}^i is most contractive.

4.4.3 Distortion estimation with depth maps

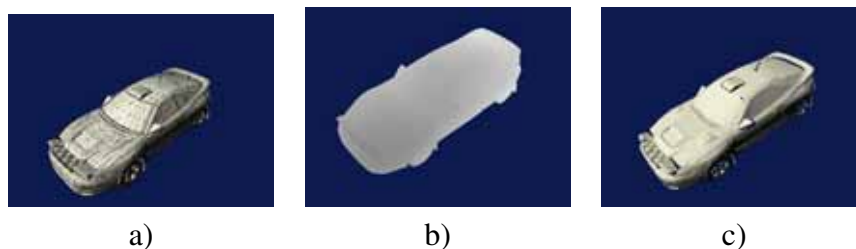


Figure 4.15: *Geometry representation: a) Triangular mesh, b) Depth map c) Geometry reconstructed from depth map*

In the previous sections we have been working with a complete description of the geometry G , in terms of triangular mesh elements. In the proposed approach however the only available description of the geometry is a set of depth maps. However to reproject the samples from the various views V^{i_k} to view V^* it is sufficient to hold a depth map $Z^*[\mathbf{n}]$, identifying the depth of each location \mathbf{n} in V^* . The depth information from Z^* allows to reproject all the pixels of V^* to the various views. A practical issue is that depth information is available only for the set of a available views at server side that usually does not include the required view V^* . How to exploit the information from the various available depth maps to reconstruct depth from the target viewpoint will be the subject of Chapter 5. We will see how the problem of estimating depth from view V^* can be considered as an analogous problem to that of synthesizing texture information. It is also possible to map the samples from the source views to V^* but this will require to have the depth information for every source view and introduces some sampling issues. If a complete mesh is available we can of course derive the depth map from it.

Shifting from a triangular mesh to depth maps has some important advantages. Firstly it allows to transmit only the part of the geometry that is relevant for the re-

quired viewpoint, by transmitting only the closest depthmaps and also by exploiting the scalable image compression features of JPEG2000 to transmit only the required part of every single depth map. This is particularly useful in remote visualization of very complex scenes where we cannot afford to transmit a complete representation of the scene’s geometry. Secondly, by moving to depth maps we can treat each pixel (or resolution component sample) separately. This permits to avoid the issues correlated to partially hidden triangles and to affine warping without the need to perform a remeshing. This allows also stitching decisions to follow meaningful scene features rather than being bounded by artificial mesh structures. All these problems can be theoretically solved by performing a fine remeshing to increase the number of triangles in the scene, but using too many triangles will have a severe impact on the performance of the system.

To adjust the previous formulation from the triangular mesh representation to the sample-based one it is necessary to take the limit of the previous equations as triangles Δ_n^* and Δ_n^i becomes arbitrarily small.

In the sample-based framework the multi-resolution stitching procedure now allows to select a source view for every single sample⁵. In this way the source selection is more flexible and is not anymore forced to follow the boundaries between triangles. The subband stitching formulation, previously given by equation (4.6) can be now expressed as:

$$R_d^*[\mathbf{p}] = \begin{cases} R_d^{i_d[\mathbf{p}] \rightarrow * }[\mathbf{p}] & 2^d \mathbf{p} \in \mathcal{R}^* \\ 0 & 2^d \mathbf{p} \notin \mathcal{R}^* \end{cases}, \quad d = 0, 1, 2, \dots, D; \quad (4.19)$$

The warping operator \mathcal{W}^i becomes a general geometric warping operator, that now has a value for every pixel in the image, instead of a single value for all the pixels in each triangle. It must be computed from the depth information in Z^* . A possible way of calculating the warping operator for a certain point \mathbf{p} , $\mathcal{W}^i(\mathbf{p})$ it is to take the normal to the surface in that point (that can be easily computed from the depth map), and generate a “virtual” triangle centered at p and with the orientation given by the surface normal. The three triangle vertexes can then be reprojected to V^* and finally we can compute the warping parameters.

If we divide both terms in equation (4.13) by $\Delta_{n,d}^*$ we obtain:

$$\frac{D_{n,b \rightarrow d}^{i \rightarrow *}}{|\Delta_{n,d}^*|} \approx \frac{D_{n,b}^i}{|\Delta_{n,b}^i|} \cdot W_{b \rightarrow d}^n, \quad (4.20)$$

The left hand side of equation (4.20) represent the average per-sample distortion over the extent of the n^{th} triangle. The meaning of equation (4.20) is that inside Δ_n the corresponding weight $W_{b \rightarrow d}^n$ maps per-sample distortion from source subband B_b^i in view V^i to the multi-resolution component R_d on view V^* .

Moving to the depth map based representation the weights can so be used as point-wise multipliers. We will also represent with $(\mathcal{W}_{b \rightarrow d}^i)^{-1}$ the operator which maps locations \mathbf{p} in the warped resolution component R_d , back to the corresponding location

⁵In practical implementations to achieve better performance and memory savings by computing the distortion and making the choices on small blocks of samples instead of single pixels.

$\mathbf{k} = (\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p})$ in subband B_b^i of view V^i (in other words is the operator who does the inverse mapping of the one provided by $\mathcal{W}_{b \rightarrow d}^i$). By replacing the weights and the distortion values associated with the triangles with the per-sample ones equation (4.13) can be rewritten as:

$$D_d^{i \rightarrow *}[p] = \sum_b D_{b \rightarrow d}^{i \rightarrow *}[p] = \sum_b W_{b \rightarrow d}[p] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right]. \quad (4.21)$$

It is important to underline that the source distortion must be taken from the sample of the source view that have been mapped to the considered sample \mathbf{p} , that is represented by $(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p})$.

As in the case of the triangular mesh after computing the distortion for every sample we can choose as the best stitching source the one which minimizes the distortion. The best source at location \mathbf{p} in R_d^* , $i_d^*[\mathbf{p}]$, is so given by:

$$i_d^*[\mathbf{p}] = \underset{i}{\operatorname{argmin}} D_d^{i \rightarrow *}[p]$$

There are, however, some obvious problems in using equation (4.21) to determine $D_d^{i \rightarrow *}[p]$ directly. The first problem is that $(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p})$ will generally reproject sample \mathbf{p} in the middle between some available distortion samples $D_b^i[\mathbf{k}]$ in subband B_b^i . This problem can be solved by simply selecting the sample closest to the computed position or by using some interpolation techniques such the bilinear filtering to average the value of the closest distortion samples.

The second is that our weighting formulation was developed based upon the assumption that the triangular mesh elements are large compared with the support of $\langle \mathcal{W}_n^i(S_{\mathbf{k}}^b), A_{\mathbf{p}}^d \rangle$ – this is certainly not true when we reduce all of the triangular mesh elements to individual samples.

In the practical implementation to reduce the memory usage and stabilize the results we do not use the distortion for every simple sample but we divide the image in a set of blocks and we store a single value for every block, that is obviously the average of the distortion value of samples inside the block. That solution also reduce the impact of both problems, the first because it is quite likely that the reprojected point will fall inside one of the blocks and the second because the size of the block is of course larger than a single sample. To reduce further the impact of these issues it is also possible to apply a low-pass smoothing filter to the distortion estimates $D_b^i[\mathbf{k}]$ before using them to compute equation (4.21). After applying the low-pass filter, specially if we are also using the distortion averaged on the blocks we can use nearest neighbour interpolation (i.e., just choose the sample or block closer to the reprojected point) in connection with $(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p})$ without any loss of fidelity.

To match the compressed data representation it is possible to compute the averaged distortion not on fixed square blocks but on the regions corresponding to the code-blocks used for JPEG2000 compression. The main difference is that by dividing the image in a set of block and repeating the same division on all the subbands we have the same number of blocks in all the subbands, with smaller blocks in low-frequency subband. Instead the JPEG2000 blocks have the same size in all the subbands, so there

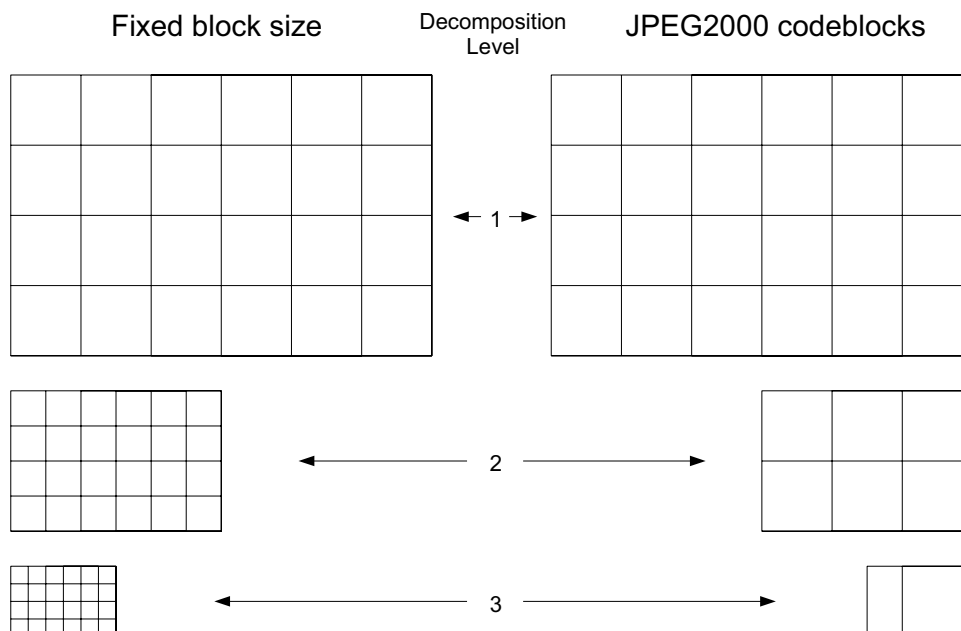


Figure 4.16: Division of the various subbands in squared blocks and in JPEG2000 codeblocks

are many of them in the higher subband and few (or just one) in the low frequency bands (see Figure 4.16).

Using the blocks the distortion field $D_b^i[\mathbf{k}]$ is thus already likely to be varying only slowly, and further low-pass filtering has little impact. Even if it seems that the block subdivision and the low-pass filtering will introduce an approximation that can reduce the accuracy of the sample selection procedure, practical experiments showed that the resulting image quality usually is improved. That is because by taking many closer samples from different views usually leads to a lot of stitching artefacts. Instead these approaches reduce the amount of spurious switching between different source views during stitching. Figure 4.17 shows an example of this issue, it represent an example of sample selection from two source views corresponding to the light and dark green colours. Fig. 4.17a correspond to a more accurate distortion representation but at the end leads to a worse image due to the many spurious switches. Fig. 4.17b, obtained with low-pass filtering and larger blocks leads to a better rendered image. Section 4.8 will show how it is possible to reduce further the amount of spurious switching by post-processing the distortion choices.

4.5 Holes related issues

As noted previously, some of the samples in the warped source views $V^{i \rightarrow *}$ have no corresponding points on the source images. We called them “holes.” We divided the

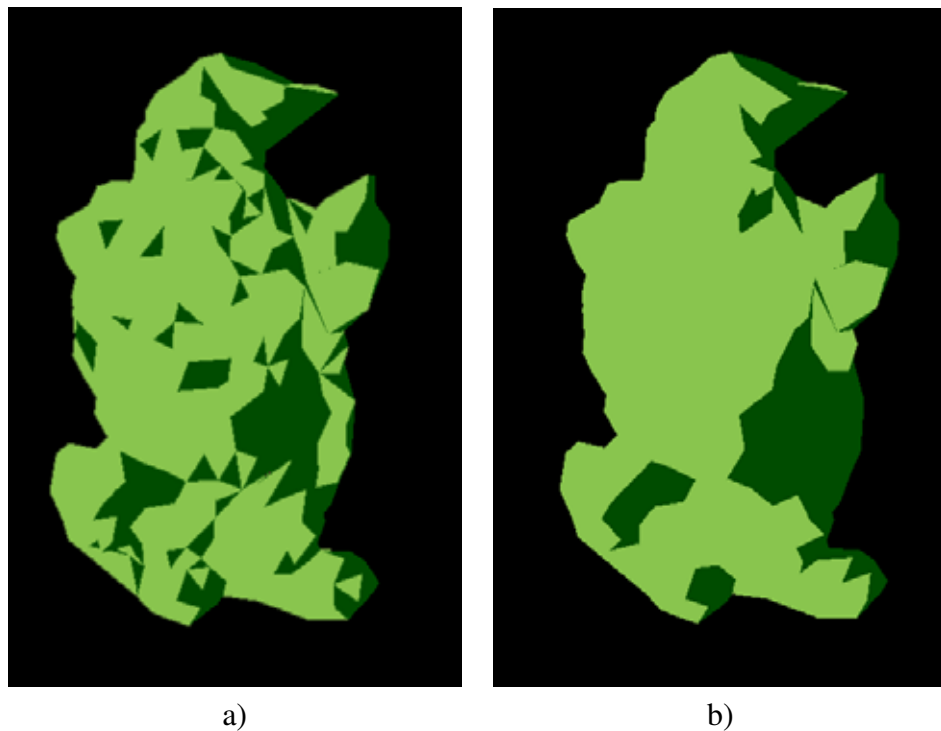


Figure 4.17: *Sample selection from 2 images: a) with per-sample distortion, b) with distortion averaged on blocks and low-pass filtering*

holes into two types: exterior holes and interior holes (see Figure 4.4).

Exterior holes are common to all the warped source images $V^{i \rightarrow *}$, since they correspond to pixel locations which do not belong to the silhouette \mathcal{R}^* of the object. Even though all $V^{i \rightarrow *}$ are zero outside \mathcal{R}^* , this does not mean that the resolution components $R_d^{i \rightarrow *}$ are zero outside the corresponding, scaled region. The reason is that the DWT (or any multi-resolution transform for that matter) involves overlapping basis functions, which arise as the translates of a set of analysis filter impulse responses. This can lead to artefacts in the rendered views: every sample in one of the subbands at resolution level l corresponds to a block of size $2^l \cdot 2^l$ in the image and this block can fall only partially inside the silhouette of the object. Setting the value to zero because most of the block is out of \mathcal{R}^* can cause errors in the samples inside the block. Another problem is that the wavelet synthesis in a certain point depends not only on the corresponding coefficient but also on the coefficients close to it. Again wrong values outside the silhouette can affect the samples on the boundary.

As a result, synthesizing V^* directly from the resolution components defined by equation (4.19) produces a result whose region of support is not limited to \mathcal{R}^* , with ringing near the boundaries of this support.

To eliminate this problem, it is sufficient to simply set $R_d^*[\mathbf{p}]$ equal to $R_d^{j^*[\mathbf{p}] \rightarrow *}[p]$ for all \mathbf{p} in the domain of R_d , i.e. we have to take from one of the views the samples outside the silhouette of the object. Even if no depth information is available for the point outside the silhouette and no reprojection is possible the result will be correct because all the samples outside \mathcal{R}^* in all the source images and in V^* will have the

same value (i.e. the background colour). The only missing element is how to select a suitable stitching source $i_d^*[\mathbf{p}]$ for each $\mathbf{p} \notin \mathcal{R}^*$. The distortion framework we used for the samples inside \mathcal{R}^* is useless because neither the warping operator $W_{b \rightarrow d}[\mathbf{p}]$ nor the distortion information $(W_{b \rightarrow d}^i)^{-1}(\mathbf{p})$ are available at these locations. We decided simply to take them from the same source view we used for the closest sample inside the silhouette. This effectively eliminates the problem of exterior holes, which we shall henceforth ignore. This problem however only arises in the case of single objects segmented from their background. If our original geometric description was sufficient to cover the real world, like in the case of a reconstruction of a room or another closed space, there would be no exterior holes, since \mathcal{R}^* will cover the entire image space of all the views.

Interior holes represent a more challenging problem, since they occupy different regions in each of the $V^{i \rightarrow *}$. While all the exterior holes samples are the same in all the warped views, some samples that are interior holes in a warped view are not holes in another. Also, it is not sufficient simply to ensure that the best stitching source $i_d^*[\mathbf{p}]$ corresponds to a view which has no holes at the corresponding location: in the samples close to the holes the extent of the overlapping wavelet basis function can lead to incorrect reconstructions. That is due to the fact that two corresponding points on different warped views can have completely different representation in the subbands because they depend also on the surrounding samples that can be holes in one view but not in the other.

More formally, in the wavelet decomposition of an image each sample in $R_d^{i \rightarrow *}$ is effectively formed from an inner product,

$$R_d^{i \rightarrow *}[\mathbf{p}] = \langle V^{i \rightarrow *}, A_{\mathbf{p}}^d \rangle, \quad (4.22)$$

The size of the analysis kernels $A_{\mathbf{p}}^d$ is important to understand which samples can be affected by the internal holes. The analysis kernels $A_{\mathbf{p}}^d$ can readily be found by iterative application of the relevant DWT filters. For the DWT-based transform outlined in Figure 4.7, we have

$$\begin{aligned} A_{\mathbf{p}}^D[\mathbf{n}] &= A_{\mathbf{L}}^D[\mathbf{n} - 2^D \mathbf{p}], \text{ and} \\ A_{\mathbf{p}}^d[\mathbf{n}] &= A_{\mathbf{H}, \mathbf{p} \bmod 2}^d[\mathbf{n} - 2^d(\mathbf{p} - \mathbf{p} \bmod 2)], \quad 0 \leq d < D \end{aligned}$$

where $\mathbf{p} \bmod 2$ is the vector $[p_1 \bmod 2, p_2 \bmod 2]$ and the fundamental low- and high-pass kernels are recursively defined by

$$\begin{aligned} A_{\mathbf{L}}^0[\mathbf{n}] &= \delta[\mathbf{n}] - \text{the unit impulse} \\ A_{\mathbf{L}}^{d+1}[\mathbf{n}] &= \sum_{\mathbf{k}} h_{\mathbf{L}}[\mathbf{k}] \cdot A_{\mathbf{L}}^d[\mathbf{n} + 2^d \mathbf{k}] \\ A_{\mathbf{H}, \mathbf{p}}^d[\mathbf{n}] &= -A_{\mathbf{L}}^d[\mathbf{n}] + \sum_{\mathbf{k}} g_{\mathbf{L}}[2\mathbf{k} + \mathbf{p}] \cdot A_{\mathbf{L}}^{d+1}[\mathbf{n} + 2^d(2\mathbf{k} + \mathbf{p})], \quad \mathbf{p} \in \{0, 1\}^2 \end{aligned}$$

In the previous equations $h_{\mathbf{L}}$ denote the low-pass DWT analysis filter impulse responses and $g_{\mathbf{L}}$ denote the synthesis one. With finite support filters, the dimensions

over which $A_{\mathbf{p}}^d$ is non-zero grow roughly as 2^d . While precise formulation of the region of support $\mathcal{R}_{\mathbf{p}}^d$ for $A_{\mathbf{p}}^d$ is not difficult, the following tight upper bound is convenient for the case of separable symmetric DWT filters, with lengths $2L_h + 1$ (analysis) and $2L_g + 1$ (synthesis). In our experiments, JPEG2000's 9/7 DWT is employed, for which $L_h = 4$ and $L_g = 3$.

$$\mathcal{R}_{\mathbf{p}}^d \subseteq [-l_d, +l_d]^2 + \mathbf{p}, \text{ with } l_d \triangleq \begin{cases} (2^D - 1) L_h, & d = D \\ (2^{d+1} - 1) L_h + 2^d L_g, & 0 \leq d < D \end{cases} .$$

d	l_d		
	D = 3	D = 4	D = 5
0	7	7	7
1	18	18	18
2	40	40	40
3	84	84	28
4	172	60	
5	124		

Table 4.2: Size of the region of support of $A_{\mathbf{p}}^d$

The values of l_d in the different resolution levels for a 4 level DWT are shown in Table 4.2. As it is possible to see from the table the region of support grows exponentially with the size of the filter and become very large for the lowest frequency bands. Figure 4.18 shows the region affected by internal holes in the example of Fig. 4.4 in the different subbands. It is clear that, excluding the lowest resolution level, the holes-affected region becomes larger after every decomposition step.

We can now exploit equation (4.22) to understand which samples are affected by interior holes. We note that $R_d^{i \rightarrow *}[p]$ is affected by interior holes in $V^{i \rightarrow *}$ whenever $\mathcal{R}_{\mathbf{p}}^d$ intersects $\mathcal{H}^{i \rightarrow *}$. The samples representing interior holes can have completely different values in the different warped views and if some of them are inside $\mathcal{R}_{\mathbf{p}}^d$ they will produce results which are generally inconsistent with any of the warped original views. We would like, therefore, to choose the best stitching source $i_d^*[p]$ only from those views V^i such that $\mathcal{R}_{\mathbf{p}}^d$ does not intersect any hole, i.e. is fully contained within the complement, $\overline{\mathcal{H}^{i \rightarrow *}}$ of $\mathcal{H}^{i \rightarrow *}$. This approach however arises some issues: The first is how to handle the case where in every view the region of support of the sample $\mathcal{R}_{\mathbf{p}}^d$ intersects with the interior holes $\mathcal{H}^{i \rightarrow *}$. Considering the large size of the region of support it is not a very unlikely case, specially for the low frequency subbands. Even if some of the views satisfies the constraint $\mathcal{R}_{\mathbf{p}}^d \subset \overline{\mathcal{H}^{i \rightarrow *}}$, they can have an unacceptably high level of distortion. Is it really a good solution to prefer a view for which only a small amount of information has been transmitted to one that has good quality and perhaps only a small overlapping region between the region of support of the sample and the holes? Instead of just excluding the views that are affected from the interior holes better results can be achieved by interpreting the overlap between the region

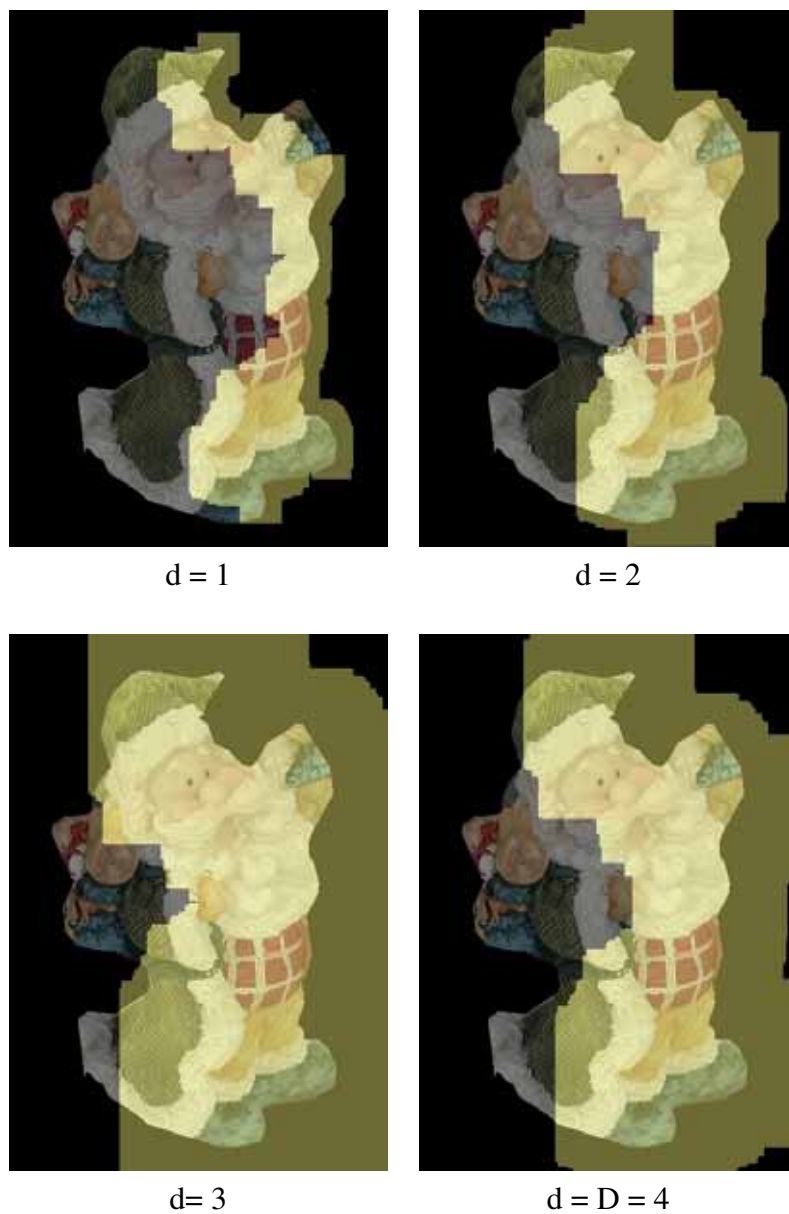


Figure 4.18: *Example of the expansion of the holes-affected region in the different subband of a 3 level DWT for the internal holes of Fig. 4.4*

of support of the sample \mathcal{R}_p^d and the interior holes $\mathcal{H}^{i \rightarrow *}$ as an additional source of distortion.

We can introduce the an additional term in equation (4.21) to take into account the effect of invalid samples in the region of support of \mathbf{p} . The idea is that the distortion is proportional to the extension of the fraction of the region of support covered by the holes. We can introduce an additional term of the form:

$$D_{holes,d}^{i \rightarrow *}[\mathbf{p}] = \frac{|\mathcal{R}_p^d \cap \mathcal{H}^{i \rightarrow *}|}{|\mathcal{R}_p^d|} \overline{E_d^{i \rightarrow *}[\mathbf{p}]} \quad (4.23)$$

Where $\overline{E_d^{i \rightarrow *}[\mathbf{p}]}$ is a local measure of the variance of the subband coefficients $R_d^{i \rightarrow *}$ in the vicinity of \mathbf{p} . In fact in all the resolution component except for the lowest resolution component ($d = D$), $R_d^{i \rightarrow *}$ has essentially zero mean, and so for every $d < D$, $\overline{E_d^{i \rightarrow *}[\mathbf{p}]}$ represents the local average of the power in the coefficients $R_d^{i \rightarrow *}[\mathbf{p}]$. This is reasonable, since the presence of holes tends to generate high energy coefficients in the detail components $R_d^{i \rightarrow *}$, and high variance in the low-pass component $R_D^{i \rightarrow *}$. The presence of artificial high energy coefficients generated by the transition between the valid samples and the holes is the main source of distortion because they are inconsistent across views with different holes positions. The amount of distortion is then given by $\overline{E_d^{i \rightarrow *}[\mathbf{p}]}$ multiplied by the ratio between the holes-affected area of the region of support and the area of the whole region of support. By adding equation (4.23) into the distortion model of equation (4.21) we obtain an augmented distortion formulation:

$$D_d^{i \rightarrow *}[\mathbf{p}] = \sum_b W_{b \rightarrow d}[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] + \frac{|\mathcal{R}_p^d \cap \mathcal{H}^{i \rightarrow *}|}{|\mathcal{R}_p^d|} \overline{E_d^{i \rightarrow *}[\mathbf{p}]} \quad (4.24)$$

Unfortunately this improved distortion model is still not enough to handle the holes issue. The problem is that selecting only one stitching source for each location \mathbf{p} is likely to produce spurious artefacts in the vicinity of holes, since the selected stitching source and the other sources we could have selected for other close samples are not consistent.

To handle this problem we decided to relax the assumption that every subband sample can have only a single stitching source and allow multiple weighted contributions from different views in the vicinity of interior holes. The stitching equation (4.19) becomes

$$R_d^*[\mathbf{p}] = \sum_i \rho_d^i[\mathbf{p}] \cdot R_d^{i \rightarrow *}[\mathbf{p}], \quad \text{with } \sum_i \rho_d^i[\mathbf{p}] = 1, \quad \forall \mathbf{p}, d = 0, \dots, D \quad (4.25)$$

Where ρ_d^i are the weights in the range 0 to 1 that indicates how much every single source view is used to generate the sample value. In the positions for which the region of support of the best stitching source⁶ i_d^* does not intersect any internal hole we will just set $\rho_d^{i_d^*}$ to 1 for the selected source and to 0 for all the other views. In the case

⁶ Computed with the augmented distortion model of equation (4.25)

where the region of support of the best stitching source intersect some of the internal holes we instead compute the sample value with a weighted average of all the warped views where the weights are in inverse proportion to the distortion corresponding to the single views. More formally, the stitching weights are given by:

$$\rho_d^i[\mathbf{p}] = \begin{cases} \delta(i - i_d^*[\mathbf{p}]), & \text{if } \mathcal{R}_{\mathbf{p}}^d \cap \mathcal{H}_d^{i_d^*[\mathbf{p}] \rightarrow *} = \emptyset \\ \frac{1}{D_d^{i_d^*[\mathbf{p}]}} & \text{otherwise} \\ \frac{1}{\sum_j \frac{1}{D_d^{j \rightarrow *[\mathbf{p}]}}} & \end{cases} \quad (4.26)$$

where $\delta()$ is the kronecker delta and $i_d^*[\mathbf{p}] = \operatorname{argmin}_i D_d^{i \rightarrow *[\mathbf{p}]}$, as before.

4.6 Accounting for Depth Uncertainty

If we use the distortion estimation based on the quantization error alone to determine the best stitching source on a set of views with the same image quality, what is likely to happen is that the selected source V^i for every sample (or triangle) will tend to be the one that for which the warping operator \mathcal{W}^i is most contractive (see section 4.4.2).

This is the view whose focal plane is most parallel to the 3D surface at the point in question. At a first glance it seems to be the best choice because in that view the considered region has the greatest area and so the representation of the texture is more detailed. Actually this impression would be correct if the geometry representation were extremely accurate. Unfortunately most of the acquisition procedures for 3D scenes (specially the passive ones) have a measurement error that is not negligible and depth maps sampling and compression further increase it. Even if the error on the geometry were the same on all the surface, the way in which it affects the reprojection of the views is not the same for all of them. In one extreme case if we reproject a view over itself the sample position will always be correct whatever is the error on the depth values. On the other side the reprojection of views that have a completely different orientation and viewpoint in respect with V^* will be heavily affected by the errors in the depth representation. This leads to the intuition that if the geometric model were highly unreliable we would expect to do better by selecting the original view image which is most closely aligned with the desired view V^* . Furthermore the error on the geometry is not evenly distributed in all the view samples (e.g. the component due to depth maps compression will be much higher in the proximity of the edges), and a careful modeling of the distortion geometry is required.

The effect of uncertainty on the surface geometry is to introduce an error in the local affine warping operator. This, in turn, introduces a translational uncertainty in the position of the reprojected points on the warped views. This effect has been studied previously in in [54].

To understand how uncertainty in depth δZ_n^* is translated into uncertainty in the position of the reprojected points let us have a look at Figure 4.19. In this example the user is looking at the 3D scene from position \mathbf{c}^* in the direction given by \mathbf{e}^* and a view V^i taken in direction \mathbf{e}^i from point \mathbf{c}^i is available. Let us consider the 3D point \mathbf{x}_n corresponding to location \mathbf{n} in V^* , with depth $Z_n^* = Z^*[\mathbf{n}]$. We are trying

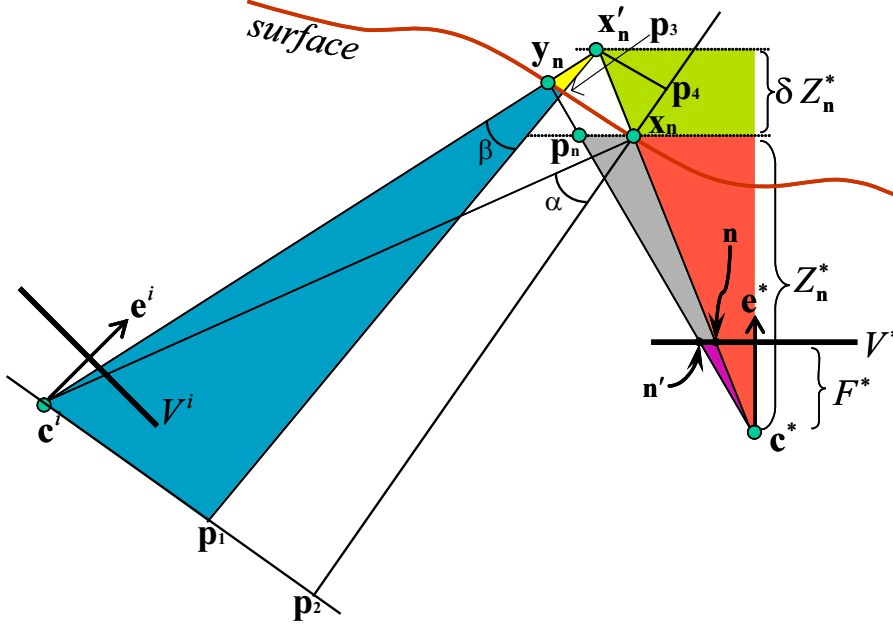


Figure 4.20: Highlight of some geometric relationships in the previous figure

From this relationship we can obtain the following expression for y_n :

$$y_n = c^i + \frac{\langle x_n - c^i, \nu_n \rangle}{\langle x'_n - c^i, \nu_n \rangle} (x'_n - c^i),$$

We can now define the error on the surface position as $\delta y_n \triangleq y_n - x_n$. For small $\delta Z_n^*/Z_n^*$, exploiting also the similarity between the red and green triangles in figure 4.20 we can obtain:

$$\delta y_n = \delta x_n - (x'_n - y_n) \simeq \delta x_n - \frac{\langle \delta x_n, \nu_n \rangle}{\langle x_n - c^i, \nu_n \rangle} (x_n - c^i) \quad (4.27)$$

$$= \frac{\delta Z_n^*}{Z_n^*} \left[(x_n - c^*) - (x_n - c^i) \frac{\langle x_n - c^*, \nu_n \rangle}{\langle x_n - c^i, \nu_n \rangle} \right] \quad (4.28)$$

We can denote with $\delta p_n \triangleq p_n - x_n$ the error on p_n . By looking at the purple and gray triangles in Figure 4.20 we can derive the following expression for it:

$$\delta p_n \triangleq p_n - x_n = (Z_n^*/F^*) \cdot (n - n') \quad (4.29)$$

By introducing the approximation $y_n - p_n \simeq p_5 - x_n$ and exploiting the similitude between the yellow and green triangles in Figure 4.21 we obtain the following expression for δp_n :

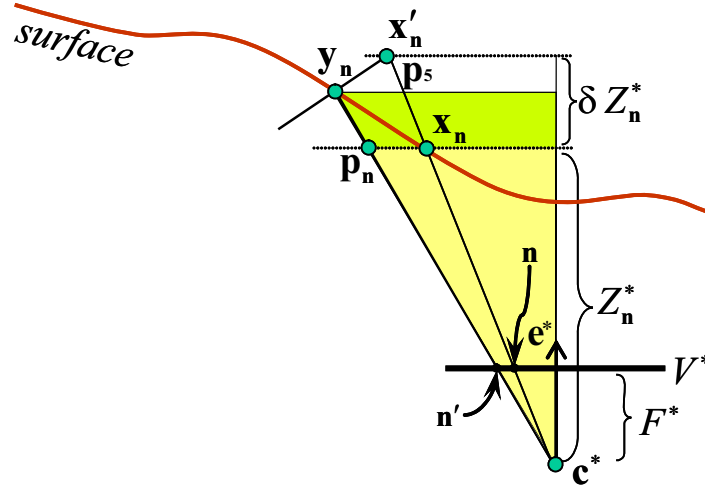


Figure 4.21: Again on the geometric relationships in depth estimation

$$\begin{aligned}
 \delta \mathbf{p}_n &= \mathbf{p}_n - \mathbf{x}_n \\
 &= \mathbf{p}_n - \mathbf{y}_n + \mathbf{y}_n - \mathbf{x}_n \\
 &= \mathbf{p}_n - \mathbf{y}_n + \delta \mathbf{y}_n \\
 &\simeq \delta \mathbf{y}_n - \frac{\langle \delta \mathbf{y}_n, \mathbf{e}^* \rangle}{\langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle} (\mathbf{x}_n - \mathbf{c}^*)
 \end{aligned} \tag{4.30}$$

where \mathbf{e}^* is the view direction for V^* . By substituting equation (4.28) into the above expression we obtain:

$$\delta \mathbf{p}_n = \frac{\delta Z_n^* \langle \mathbf{x}_n - \mathbf{c}^*, \nu_n \rangle}{Z_n^* \langle \mathbf{x}_n - \mathbf{c}^i, \nu_n \rangle} \left[(\mathbf{x}_n - \mathbf{c}^*) \cdot \frac{\langle \mathbf{x}_n - \mathbf{c}^i, \mathbf{e}^* \rangle}{\langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle} - (\mathbf{x}_n - \mathbf{c}^i) \right] \tag{4.31}$$

where F^* is the focal length for view V^* . Finally we can introduce a new variable $\delta \mathbf{n}$:

$$\delta \mathbf{n} = n - n' = \frac{\delta \mathbf{p}_n}{F^*/Z_n^*}$$

Putting all this together, we get the final representation of the mapping from depth uncertainty to geometry uncertainty:

$$|\delta \mathbf{n}|^2 = |\delta Z_n^*|^2 \cdot \underbrace{\frac{\langle \mathbf{x}_n - \mathbf{c}^*, \nu_n \rangle^2}{\langle \mathbf{x}_n - \mathbf{c}^i, \nu_n \rangle^2} \cdot \frac{(F^*)^2 \langle \mathbf{x}_n - \mathbf{c}^i, \mathbf{e}^* \rangle^2}{(Z_n^*)^2 \langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle^2}}_{g_n^{i \rightarrow *}} \cdot \left| \frac{\mathbf{x}_n - \mathbf{c}^*}{\langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle} - \frac{\mathbf{x}_n - \mathbf{c}^i}{\langle \mathbf{x}_n - \mathbf{c}^i, \mathbf{e}^* \rangle} \right|^2 \tag{4.32}$$

Equation 4.32 allows to map variance (error power) in the depth values Z_n^* to a corresponding variance (error power) in the position of the samples. It is important to underline that the factor $g_n^{i \rightarrow *}$ depends upon fixed viewing parameters, together with the surface position \mathbf{x}_n and normal ν_n seen at location \mathbf{n} in view V^* . At a first glance the computation of $g_n^{i \rightarrow *}$ looks complex, but can be made easier by exploiting some geometric relationship:

- The difference between $\langle \mathbf{x}_n - \mathbf{c}^i, \mathbf{e}^* \rangle$ and $\langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle$ is independent of \mathbf{n}
- $Z_n^* = \langle \mathbf{x}_n - \mathbf{c}^*, \mathbf{e}^* \rangle$
- \mathbf{x}_n can be expressed very simply in terms of Z_n^* and \mathbf{n} by suitable choice of reference coordinates.

The final step is the conversion of positional uncertainty into the amplitude distortion. A good solution to this issue is the method developed in [54]. Entering in more detail, we obtain an additional term in the distortion formulation for the warped resolution component $R_d^{i \rightarrow *}$, of the form

$$D_{geom,d}^{i \rightarrow *}[\mathbf{p}] = |\delta \mathbf{n}|^2 \cdot \frac{1}{(2\pi)^2} \int_{\mathcal{B}_d} \Gamma_{V^{i \rightarrow *}}(\omega) \cdot |\omega|^2 \cdot d\omega \quad (4.33)$$

where $\mathcal{B}_d \subset [-\pi, \pi]^2$ is the region occupied by resolution component R_d in the discrete space Fourier domain and $\Gamma_V(\omega)$ is the discrete space power density spectrum of image V . The $|\omega|^2$ term serves as a frequency-dependent weighting of signal power. It can be replaced by an assumed ‘‘average’’ weight $\overline{|\omega_d|^2}$ on the whole resolution component R_d . In practice, we use a simple mid-band approximation to get

$$\overline{|\omega_d|^2} = \left(\frac{3\pi}{4} 2^{-d} \right)^2$$

By substituting this approximation into equation (4.33) we obtain a new expression for the distortion term:

$$D_{geom,d}^{i \rightarrow *}[\mathbf{p}] = |\delta \mathbf{n}|^2 \cdot \overline{|\omega_d|^2} \cdot \overline{E_d^{i \rightarrow *}[\mathbf{p}]} \quad (4.34)$$

A variety of other methods to select $\overline{|\omega_d|^2}$ may be found in [54]. Considering that the value of $|\delta \mathbf{n}|^2$ varies locally, and we are also interested in the local per-sample distortion contributions $D_d^{i \rightarrow *}[\mathbf{p}]$, we can now augment equation (4.24) by introducing the new geometry-dependent term:

$$\begin{aligned} D_d^{i \rightarrow *}[\mathbf{p}] &= \sum_b W_{b \rightarrow d}[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] + \frac{|\mathcal{R}_p^d \cap \mathcal{H}^{i \rightarrow *}|}{|\mathcal{R}_p^d|} \cdot \overline{E_d^{i \rightarrow *}[\mathbf{p}]} \\ &+ |\delta Z_d^*|^2[\mathbf{p}] \cdot \overline{g_d^{i \rightarrow *}[\mathbf{p}]} \cdot \overline{|\omega_d|^2} \cdot \overline{E_d^{i \rightarrow *}[\mathbf{p}]} \end{aligned} \quad (4.35)$$

The depth uncertainty on the samples in the resolution component $|\overline{\delta Z_d^*}|^2[\mathbf{p}]$ is computed by reducing the resolution of the estimated local distortion field for $Z^*[\mathbf{n}]$ by the factor 2^d and applying a suitable low-pass smoothing filter. The filter is the same as the one used for $\overline{E_d^{i \rightarrow *}}[\mathbf{p}]$, the local average power in $R_d^{i \rightarrow *}$ at location \mathbf{p} . The same approach is used also for the coefficients $\overline{g_d^{i \rightarrow *}}[\mathbf{p}]$. They are obtained by reducing the resolution of the $g_n^{i \rightarrow *}$ field by factor 2^d and filtering the result.

An important remark on the geometric distortion model regards the handling of the boundaries between occluding objects. A true depth map Z^* should be discontinuous at the boundaries of objects which occlude other objects from view V^* . Unfortunately these discontinuities cannot be accurately represented using discrete samples $Z^*[\mathbf{n}]$ and the problem becomes even worse when depth information is derived from compressed data, because most lossy compression standards tend to smooth the edges, as discussed in Chapter 5. For this reason, it is reasonable to assign a depth uncertainty power, $|\delta Z_n^*|^2$ which is at least as large as the local variance in Z_n^* . In this way, we take into account the reduced accuracy of our geometric representation in the vicinity of occluding boundaries. Let us denote by $\sigma_p^2(w)$ the local variance around the sample (in our practical implementation is computed on a 7×7 window $w(\mathbf{p})$ centered on the sample). In equation 4.35 we can replace $|\delta Z_n^*|^2$ with the new augmented distortion term:

$$|\delta_{var} Z_n^*|^2 = \max(|\delta Z_n^*|^2, \sigma_p^2(w))$$

Finally, as expected, if all the other things (such as source view distortion) are equal, the geometry term forces the selection of views with smaller values of $g_n^{i \rightarrow *}$, that are the views which are closer to V^* .

4.7 Lighting related issues



Figure 4.22: *Example of lighting-related issues*

All the previous formulation starts from the assumption that the value of corresponding samples in different views is the same, i.e. if a certain point in the 3D space has a colour in a view it will have the same colour in all the other views. Unfortunately this assumption is not correct in many real situations, specially if the object

has a reflective surface, there are strong directional lights or shadows in the pictures. Figure 4.22 shows a clear example of this kind of issues, for example many points on the window and on the bonnet appears to be of a complete different color on the two views due to reflections. These colour differences causes the coefficient in the DWT decomposition of the various warped images to be different and leads to artefacts in the reconstructed renderings as we already saw in the case of the interior holes. If we look from the direction of V^* the position of the reflectances and shadows we expected to see is of course the one of that view: views closer to the required one will have similar reflections while the farther we move from the required viewpoint the more chances we have to find reflections and shadows in completely different positions from where they appear in V^* . There exist techniques to compute the behaviour of an object in respect to the lighting but they require to know additional information such as the position and characteristics of the lights and the reflective properties of the surface of the object. Instead in our approach we assume that the only available information is the set of views and depth maps.

Starting from these observations we introduced a new distortion contribution to take into account the effects of the shadows and reflection which grows with the deviation between the orientation of views V^* and V^i . We expect this distortion term to be proportional to the signal power, even if in the case of specularly issues this is not completely correct, and we give it the form:

$$D_{light,d}^{i \rightarrow *}[p] = \gamma \tan \left(\max \{0, \cos^{-1} \langle \mathbf{e}^i, \mathbf{e}^* \rangle\} \right) \cdot \overline{E_d^{i \rightarrow *}}[p]$$

Here, \mathbf{e}^i and \mathbf{e}^* are the view directions, as shown in Figure 4.19. If the angle between the two views is smaller than 90 degrees γ will be multiplied for the tangent of the angle between the two viewing directions, while for angles with more than 90 degrees we will just set the value to zero⁷. In the absence of careful lighting modeling, γ is a heuristically assigned quantity, which depends on the amount of reflectance and shadows we expect to find in the scene.

By introducing this new term into equation (4.35) we obtain a new augmented version of the distortion equation:

$$D_d^{i \rightarrow *}[p] = \sum_b W_{b \rightarrow d}[p] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] + \frac{|\mathcal{R}_p^d \cap \mathcal{H}^{i \rightarrow *}|}{|\mathcal{R}_p^d|} \cdot \overline{E_d^{i \rightarrow *}}[p] \quad (4.36)$$

$$+ |\overline{\delta Z_d^*}|^2[p] \cdot \overline{g_d^{i \rightarrow *}}[p] \cdot |\overline{\omega_d}|^2 \cdot \overline{E_d^{i \rightarrow *}}[p] + \gamma \tan \left(\max \{0, \cos^{-1} \langle \mathbf{e}^i, \mathbf{e}^* \rangle\} \right) \cdot \overline{E_d^{i \rightarrow *}}[p]$$

Note that both the distortion contributions from geometry and light vary with the local power $\overline{E_d^{i \rightarrow *}}$ in the relevant resolution component. The first term varies additionally with depth uncertainty, spatial frequency and properties of the surface normal (through $\overline{g_d^{i \rightarrow *}}$), whereas the second term is affected only by the viewing angles \mathbf{e}^i and \mathbf{e}^* .

⁷Samples from views more than 90 degrees apart are expected to be hidden in V^* or to be discarded due to the high distortion, the only reason because we set $D_{light,d}^{i \rightarrow *}[p]$ to zero is to avoid negative distortion contribution (specially for angles close to $-\pi/2$ where the tangent has high negative values) that can completely falsify the computed distortion values

4.8 Regularization of the selection choices

The target of the proposed approach is the minimization of the distortion in every single sample. Even if this approach is locally optimal it does not always lead to the best overall image quality. That is because geometry uncertainty and lighting issues cause position and color of samples warped from different views to have some mismatches. Many experimental tests suggest that taking many close samples from different views (see Fig. 4.17a) usually leads to lower image quality. For example the image obtained by taking a single sample (or a small group of samples) from a certain view $V^{i'}$ in the middle of a large region taken from another source V^i usually has a lower quality than the one we would have if we take all the samples from V^i . The reason is that even if the sample from $V^{i'}$ considered by itself has a lower distortion, the artefacts introduced by mixing it with the samples from the other sources can be worse than the distortion gain. As previously noted the filtering procedure associated with the wavelet domain stitching reduces the impact of this issues. Furthermore by computing the distortion estimates on block of samples and low-pass filtering their values it is possible to reduce the number of spurious transitions.

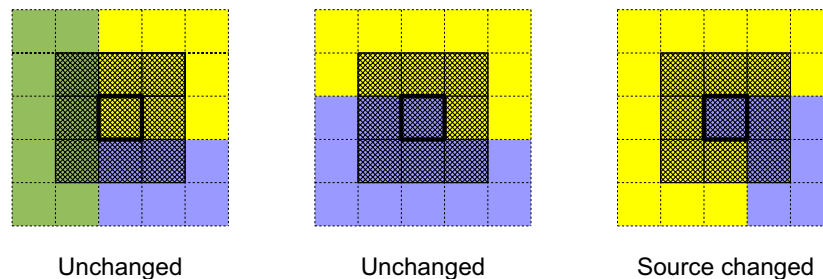


Figure 4.23: *Regularization algorithm: The source for the considered sample is changed only in the third example*

Even with all these precautions sometimes it is not possible to get completely rid of artefacts due to spurious switching so we introduced an optional post-processing procedure to reduce the number of transition in the distortion choices. The algorithm works on every subband singularly taken: it scans the distortion choices and for every sample it looks at the 8 pixels surrounding it (the pixels shaded in the examples of Figure 4.23). It counts how many of the surrounding samples are taken from any source view and if 5 samples or more are taken from a view that is not the current source view it is selected as the new source for the sample. This of course happens only if the sample in the new source is valid (it is not outside of the silhouette or a hole) and has an acceptable distortion. The procedure can be repeated more times to remove larger artefacts, specially in the high frequency subbands that have many more samples. However experimental tests shows that in most practical configurations it quickly (usually after 5-10 iterations) converges to a stable solution⁸.

⁸It is possible to build artificial shapes on which the algorithm can iterate many times without con-

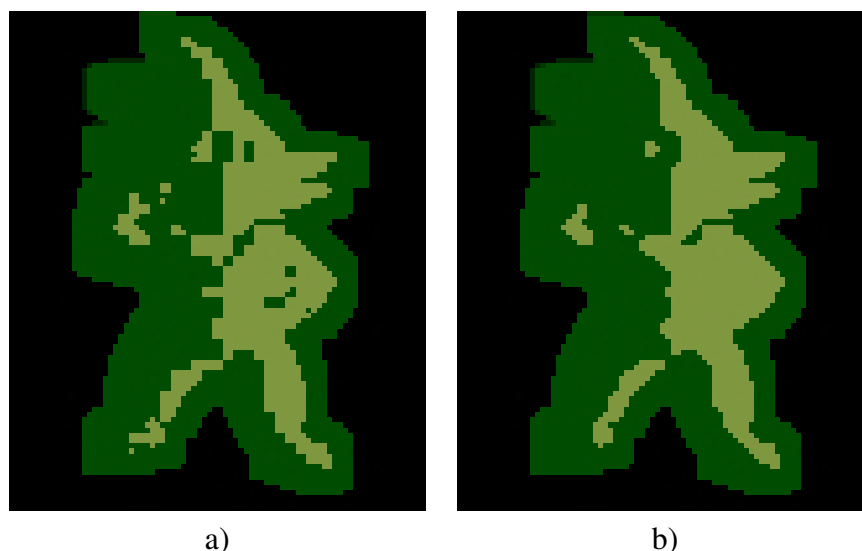


Figure 4.24: *Distortion choices: a) Before regularization, b) After regularization*

Figure 4.24 shows the distortion choices for a simple configuration with two source views, one on the right side and one on the left of the required rendering. Figure 4.24a shows the original distortion choice given by equation (4.36), while the output of the regularization algorithm is shown in Figure 4.24b, it is easy to see that most spurious samples have been removed. It is important to underline that the algorithm will clear in just a few iterations some kind of shapes (e.g. thin lines or 2x2 blocks of pixels) while bigger shapes like 4x4 or bigger blocks will not be cleared independently of the number of iterations.

4.9 Performance issues

At a first glance the computational complexity associated with the proposed framework could look too high for a real-time system that is able to work at interactive frame-rates. However some implementation precautions permit to achieve very good performances. The proposed framework is based on three main kind of operations: warping, wavelet transform and the distortion computation. The described procedure requires to warp all the available views to the required viewpoint, but this step can be performed very fast exploiting highly optimized mesh warping algorithms or, even better, the hardware acceleration features of the graphics card of most desktop computers. Differently from standard 3D renderings here the computational complexity depends only on the number of views and on the resolution of the depth maps, but not on the complexity of the scene. The procedure to reconstruct from the depth maps a mesh that can be used for high performance hardware accelerated renderings will be described in Section 5.1. Highly optimized algorithms based on the lifting framework [55] exist for the wavelet transform. They are already used in real time application such as wavelet-based video

verging to a solution but in practice they do not often happen.

decoders.

What seems to remain critical from the performance viewpoint is the distortion computation. A first observation is that it is possible to compute the distortion on undersampled versions of the subbands, i.e. divide the subbands into small blocks (usually 2x2 or 4x4 blocks), compute the distortion for only one sample inside every block and use the choice made for that sample on all the pixels of the block. Even if the distortion choices will be less accurate (but also more regular) the quality loss on the final rendered view is usually small.

Another simple but very useful trick is to exclude from the distortion computation all the views that are too far or with a too different viewing orientation from V^* . For example all the views with a viewing direction that forms an angle of more than 90 degrees with the required direction can be excluded because it is highly unlikely that any triangle will ever be taken from them. Far views instead could perhaps have useful information for low frequency subbands but will for sure have no data for high frequency (detail) subbands in a close-up viewing of some object.

Section 7.4 contains some examples of the performance of the current implementation of the systems on the sample datasets.

Chapter 5

Distortion-sensitive geometry synthesis

The previous chapter shows how the client can synthesize the required view V^* from a collection of available views and a single depth map Z^* corresponding to view V^* . A possible implementation of this strategy could be to compute Z^* at server side, compress it as an image and finally transmit the compressed data progressively to the client, which would estimate its distortion using the same methods it uses to estimate local distortion in the compressed views V^i . However, this approach has a couple of important drawbacks that makes it unfeasible: the first is that the server would need to compute and compress a new depth map from the 3D model for every frame. This operation would add a huge computational overhead at server side, specially for complex scenes, and also require the availability of a complete 3D model of the scene instead of just a set of depth maps. The second is that the server would have also to transmit a distinct depth map Z^* for each view the client may wish to render, thus widely increasing the bandwidth usage. This solution at the end has all the same fundamental drawbacks as having the server generate and compress each distinct view V^* which the client may be interested in, which is the problem we have been seeking to avoid. Another possibility could be to generate the depth maps from a scalably compressed 3D mesh which is incrementally transmitted by the server. There exists a substantial body of work on progressive compression of 3D meshes (e.g., [5, 6, 7]), which could be leveraged to this end. However we are trying to avoid the construction and transmission of a complete 3D representation of the scene.

5.1 Proposed approach

In our approach we assume the server holds only a set of depth maps taken from some different viewpoints (Fig. 5.1). The position of the viewpoints depends on the acquisition procedure used to build the depth information. Usually it corresponds to the taking position of the available views, but it is not necessary. As for the images also the depth maps are transmitted to the client in a scalable progressive way. At every time during the interactive browsing the client will have received only some of the available depth maps (or even only a certain region of a particular depth map) and all with different quality levels (see Figure 5.2).

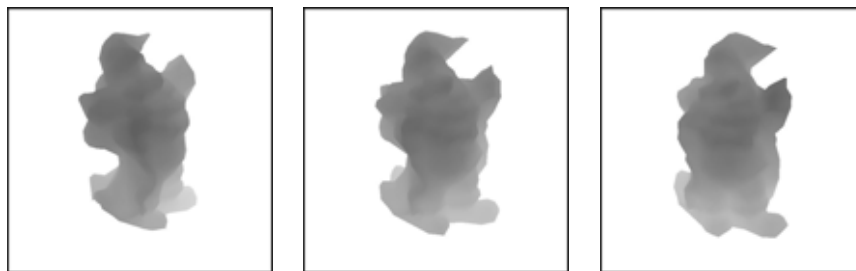


Figure 5.1: *Example set of depth maps for the Santa Claus statuette*



Figure 5.2: *Reconstructed 3D models using a depth map scalably compressed with different bitrates.*

This solution has some big advantages: firstly it matches perfectly the originally captured information, since usually 3D scanners deliver data in the form of images with depth information. The proposed scheme obviates the need to actually find a complete surface model, which is a hard task if the scene is complex or the content in the database grows as more information from different viewpoints is captured. Furthermore, this approach permits the delivery of only that part of the geometry which is relevant to the required view.

The procedure we use is basically the same we applied to the images. We aim at synthesizing the required depth map Z^* from a collection of available depth maps, Z^{i_1}, Z^{i_2}, \dots , which the server has already delivered with varying levels of fidelity and relevance.

The complete scenario is depicted in Figure 5.3, both the server and the client work with a collection of source images V^i and a collection of depth maps Z^i . Depth maps are also compressed as images: as explained in Section 3.2 they are all represented as 16 bit greyscale images and compressed using JPEG2000. Depth information is communicated incrementally, on demand. As for the images, it is possible to exploit all the quality, resolution and position scalability features of the JPEG2000 compression standard also for depth information. The viewpoints associated with the depth maps need not necessarily coincide with the viewpoints associated with the source images. The server must decide how to distribute its available bandwidth between the enhancement of views V^{i_k} which are already partially available at the client, the enhancement of existing depth maps Z^{i_j} which have also been partially transmitted to the client, the delivery of a new view, more closely aligned with V^* or the delivery of new depth

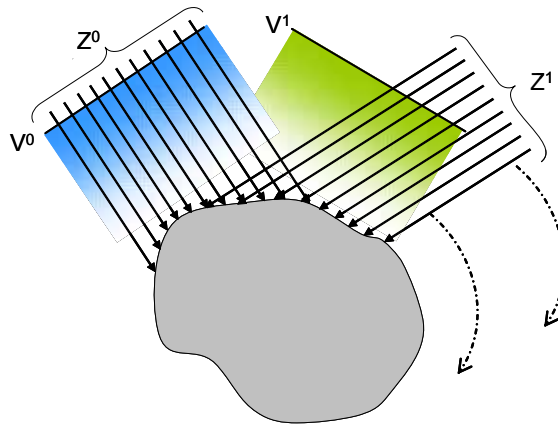


Figure 5.3: *Browsing environment with a fixed set of views, V^i , and depth maps Z^i . In this example, V^0 and Z^0 share the same view point, whereas V^1 and Z^1 do not.*

maps.

The synthesis procedure for geometry information follows a path similar to the image case: firstly each available depth map Z^i is turned into a corresponding estimate $Z^{i \rightarrow *}$ for Z^* . This is quite easy because every sample in every depth map is associated to a precise position in the three dimensional space. In our current practical implementation we build a triangular mesh for the surface described by every depth map Z^i and then we look at this mesh from the viewpoint of V^* , a task which may be accomplished using the graphic engine on many popular graphics cards¹.

A possible question at this point is if the construction of the mesh is really needed. In fact it is possible directly warp the samples from Z^i to Z^* but there are some drawbacks: it is not possible to warp the samples positions from Z^* to Z^i (we have no representation of Z^*), we must warp in the opposite direction with the consequent sampling issues, i.e. multiple samples of Z^i could be warped to the same position in Z^* and there could be no samples mapped to other positions. It is possible to deal with these issues by averaging and interpolating but the mesh approach gives better results. The triangular mesh also does not need to be rebuilt for each frame but only when new data become available for that particular depth map. Finally by loading the mesh into the graphics card memory and exploiting hardware accelerated rendering very good performance can be achieved.

The construction of the mesh seems very simple but should be performed carefully. For example a 1024x768 pixels depth map has almost 800000 samples and the mesh obtained by just using every sample as a vertex and joining the adjacent ones is quite big. To achieve good performance it is necessary to reduce the size of the meshes. A possible solution is to just downsample the depth maps, but this will cause a loss in

¹The most common graphics libraries offer a function that allow to extract from the hardware the contents of the depthbuffer of the videocard, that is exactly the depth map from the current viewpoint.

the precision of the three dimensional representation. Better results can be obtained by observing that a fine mesh representation is really useful only in the regions where the depth is changing fast.

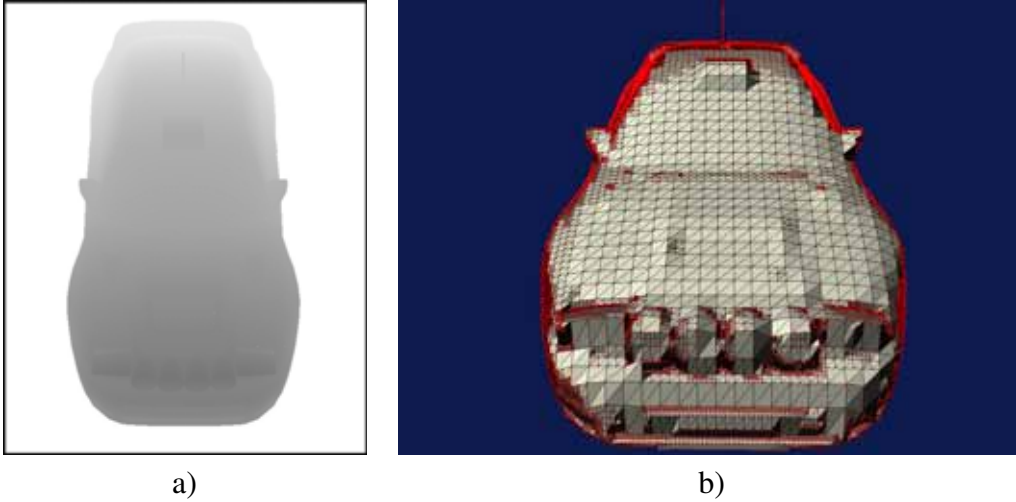
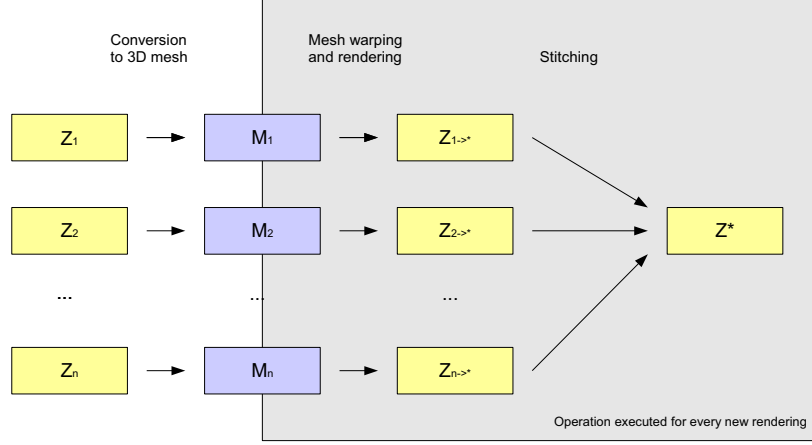


Figure 5.4: *Multi-resolution mesh construction algorithm: a) Source depth map b) Corresponding mesh*

A first observation is that in the points close to the edges of the object there is a very fast change in the depth values and a finer mesh representation is required. A possible solution is to build a mesh that has a precise representation in the proximity of the edges and a more coarse one everywhere else. The edges are not the only points in which the depth has big variance, to obtain a better representation we developed a multi-resolution algorithm to build a mesh that is finer where the depth has a higher variance. Figure 5.4 shows an example of mesh built using this technique, the vertices are highlighted in red and from the figure it is easy to see that they are more dense in proximity of the edges or of object features where there is a high depth variance such as the headlights.

The meshes built from the different depth maps are used to build a set of predictions of Z^* . As in the image cases, it is necessary to find a way to combine together the information coming from the different depth maps. Figure 5.5, shows an overall scheme of the procedure to reconstruct Z^* from a set of source depth maps. The multi-resolution stitching procedure we used for the images is not suitable for the depth information. The reason for this is that while the discontinuities in the images represent visual artefacts that we tried to avoid, we instead need to preserve the discontinuities in Z^* because they represent relevant scene features such as occlusions or edges. The filtering procedure used in multi-resolution stitching instead tend to destroy them. For the depth information we decided to simply perform the stitching directly in the full resolution pixel domain. The final depth map is so given by equation 5.1 :

$$Z^*[\mathbf{p}] = \sum_{\mathbf{p}} \alpha^i[\mathbf{p}] \cdot Z^{i \rightarrow *}[p] \quad (5.1)$$

Figure 5.5: *Depth reconstruction procedure*

Where $\alpha^i[p] = 1$ if sample p is going to be warped from the depth map Z^i and $\alpha^i[p] = 0$ otherwise. The final missing element is the computation of the $\alpha^i[p]$ coefficients, i.e. how to select which samples are going to be taken from every different source depth map.

The effect of the reprojection of the depth maps is similar to the image case, expansion or contraction in the local affine warping operators associated with this procedure affect the way in which distortion is mapped from subbands in the compressed Z^i images into the synthesized depth candidate, $Z^{i \rightarrow *}$.

This is governed by a set of weights, similar to the $W_{b \rightarrow d}$ developed in Section 4.4, except that we currently perform depth synthesis directly at full resolution – i.e., without the multi-resolution framework. The weights coefficients in this case depends only on the warping operator, we will denote them with $W_z[\mathbf{p}]$. Equation 5.2 gives the distortion on Z^* associated with every sample taken from depth map Z^i :

$$D_z^{i \rightarrow *}[p] = W_z[p] \cdot D_z^i[(\mathcal{W}_z^i)^{-1}(p)] \quad (5.2)$$

Local distortion estimates in each warped depth map $Z^{i \rightarrow *}$ are used to compute the α^i coefficients, by just selecting the depth map with the lower distortion in a way essentially identical to that described for images in Section 4.4:

$$\alpha_i[p] = \begin{cases} 1 & \text{if } i = \operatorname{argmin}_i D_z^{i \rightarrow *}[p] \\ 0 & \text{otherwise} \end{cases}$$

A useful feature of the distortion-sensitive geometry synthesis is that we also get the local distortion information in Z^* that is then available for inclusion in the view synthesis problem, in accordance with equation (4.36).

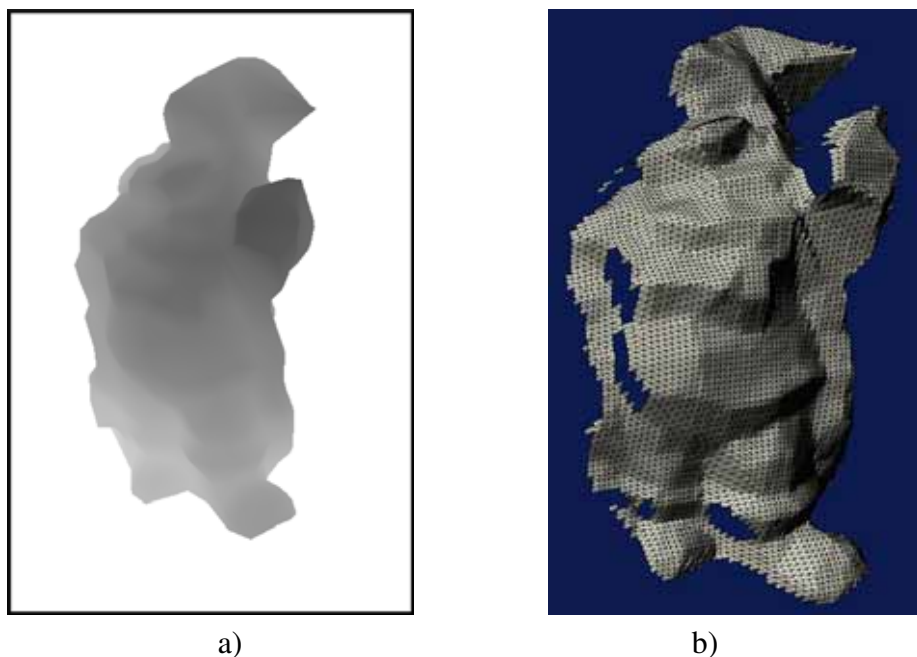


Figure 5.6: *Holes in depth estimation: a) Depth map Z^i , b) Estimation of Z^* from Z^i*

5.2 Holes related issues in depth estimation

Also in the depth map case the reprojection of Z^i on Z^* is not enough to predict the depth values and we have to deal with “holes” related issues in the depth synthesis problem. Figure 5.6b shows how the mesh reconstructed from the depth map in Figure 5.6a appears when observed from a different viewpoint. It is easy to see that some parts of the scene geometry cannot be derived from a single depth map and there are holes due to occlusions in the source view. At a first glance seems that the holes problem in the depth case should be much easier because we avoided to use the multi-resolution stitching. By performing just the simple sample stitching at full resolution every pixel depends only on itself and there are no issues related to large filter kernels. In this simpler framework the holes issue seems to be easy solvable by just avoiding to take samples that have been marked as holes.

Unfortunately it is necessary to take into account also the effects of lossy compression. Each individual depth map Z^i can generally be expected to exhibit strong discontinuities in regions of object occlusion and indeed these discontinuities do correspond to interior holes in the inferred depth map $Z^{i \rightarrow *}$. Compression using waveform coders such as JPEG2000 and JPEG, however, tends to produce ringing and considerable error in the vicinity of such discontinuities.

This phenomenon is illustrated schematically in Figure 5.7. In the figure, compressed depth maps Z^1 and Z^2 are each used to construct candidates $Z^{1 \rightarrow *}$ and $Z^{2 \rightarrow *}$ for Z^* , where Z^* corresponds to the vertical elevation of the scene surface in this example.

Evidently, the surface section between P_1 and P_3 is not visible in Z^1 and the depth values in this region should be extracted from another depth map. While from the

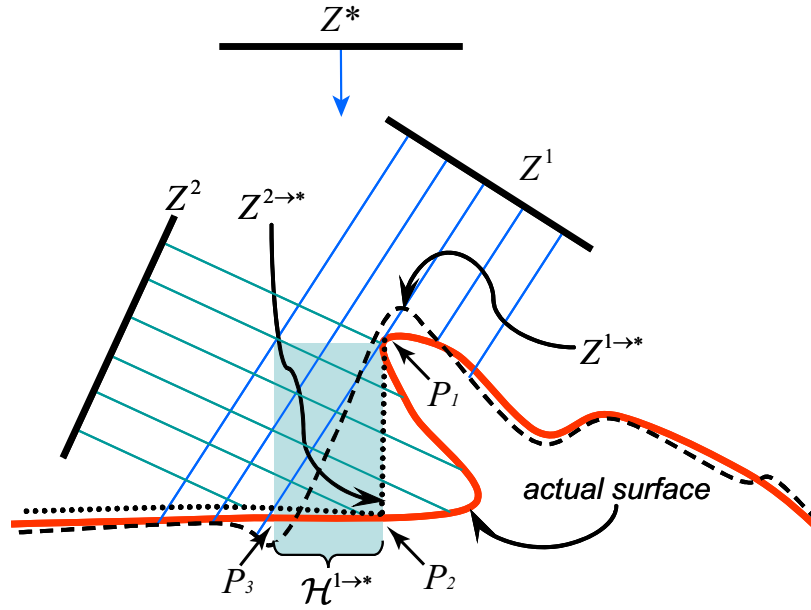


Figure 5.7: Illustration of the effect of discontinuities in individual depth maps Z^i on the synthesized depth candidates $Z^{i \rightarrow *}$. In the figure, the depth of interest Z^* which we wish to synthesize is the vertical height of the surface.

figure the presence of the hole is clear, it is quite hard to recognise the real situation from just Z^1 . We cannot get any information on the real shape of the surface between the two points and even if we can guess the presence of a hole from the fact that Z^1 should have a discontinuity at the point where the scene surface folds back upon itself, this discontinuity is usually corrupted by ringing and loss of high frequency details due to lossy compression. As a result, it is not possible to detect the hole $\mathcal{H}^{1 \rightarrow *}$ which should appear in $Z^{1 \rightarrow *}$. In fact, holes in $Z^{i \rightarrow *}$ are difficult if not impossible to detect, based on Z^i alone. The only way to detect the hole in Z^1 is by exploiting the information from another depth map. In the example in Figure 4.19 we can exploit Z^2 , for which the inferred depth $Z^{2 \rightarrow *}$ is shown as a dotted line.

Fortunately, our distortion-based stitching procedure should recognise that $Z^{1 \rightarrow *}$ is subject to a great deal of distortion due to the stretching which Z^1 undergoes in the vicinity of surface folding. To encourage this, we ensure that the distortion estimates used for each source depth map Z^i have the property that distortion is judged to be at least as large as the local variance of that depth map, in addition to any estimates of the underlying compression noise.

The source distortion is then given by the maximum of the compression distortion and the local variance around the sample, that in our practical implementation is computed on a 7×7 window $w(p)$ centered on the sample:

$$D_{var,z}^i[p] = \max(D_z^i[p], \sigma_p^2(w))$$

We can now replace the source distortion in equation 5.2 with the new augmented

version $D_{var,z}^i[p]$ that includes the variance, thus obtaining a new expression for the depth map distortion :

$$D_z^{i \rightarrow *}[p] = W_z [p] \cdot D_{var,z}^i[(\mathcal{W}_z^i)^{-1}(p)] \quad (5.3)$$

This is the same rule which we outlined at the end of Section 4.6 for Z^* . The idea behind the additional term in the source distortion is that regions in which depth map Z^i was originally discontinuous will be treated with distrust. The error is then further amplified by the stretching when the source depth map is mapped to $Z^{1 \rightarrow *}$, encouraging the selection of the much more reliable candidates from better aligned depth maps (such as $Z^{2 \rightarrow *}$ in the example). The final stitched depth map can have much less uncertainty than either of the candidates $Z^{i \rightarrow *}$ in isolation, allowing for high quality view synthesis via the methods of Chapter 4.

Chapter 6

Server policies

In the previous chapter we described how the information on the different images and depth maps available at client side can be exploited to reconstruct the views required by the user. In all the previous formulation we have assumed that the client had received from the server a certain subset of the information available at server side. The missing element in the proposed framework is how to select which information from the different views and depth maps need to be transmitted from the server in order to obtain the best rendering at client side.

The global architecture of the server application has been introduced in Section 3.3 and is based on the JPIP interactive protocol for JPEG2000 images. This choice allows a great flexibility in the selection of the information that is going to be sent. For every image and depth map it is possible to send just a particular spatial region with the preferred quality level and resolution. This wide set of choices allows an optimal exploitation of the available transmission resources but also increases the complexity of the decision problem on the information to be transmitted. The issue is made even more complex by the presence of two completely different kind of information , i.e. texture and geometry. Even if some work on this issue has been carried on (for example in [4] and [2]), how to distribute the bandwidth between these two elements remains an open issue.

The starting point to deal with the bandwidth allocation problem is of course the distortion framework we developed for the client. Not only the availability of an estimation of the distortion reduction associated to every element offers a straightforward way to select which packets are the best to be sent, but also by using the same model at client and server side we can ensure that the transmitted data will also be selected by the multi-resolution stitching procedure at client side. However some important issues needs to be considered. JPEG2000 offers many scalability features but while the selection procedure at client side allows to select a different source for every single sample, choices at server side must be done on the compressed data blocks, that correspond to larger regions. Furthermore the geometry dependent distortion term in Section 4.6 introduce a dependency between the choices made for the depth maps and the ones for the images. For example the transmission of more geometry information can force the transmission of more data for an already available view that can be warped with higher fidelity instead of transmitting a new closer view.

In the following sections we will assume that geometry has already been transmitted to the client and we will develop a distortion driven procedure to select which elements from the different compressed representations of the various views are the best to be sent based on the available geometry information. The extension of the proposed method to the distribution of bandwidth between geometry and view data is the subject of the current research activity and in Section 6.5 we will briefly describe how the proposed scheme will be exploited for geometry transmission.

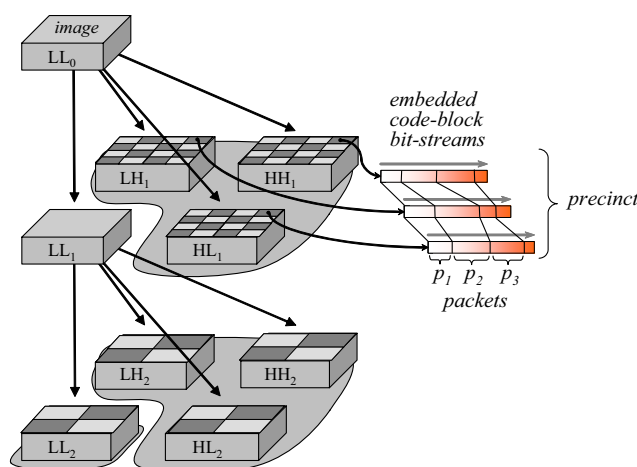


Figure 6.1: *Basic JPEG2000 elements transmitted by JPIP*

6.1 The server optimization problem

For simplicity we start from the problem of distributing transmission bandwidth amongst the various view images V^i , assuming that geometry is already available at client side. We assume that each view image has been compressed using JPEG2000 and is thus represented by a collection of code-blocks \mathcal{B}_β^i . As explained in Section 2.4 after performing the wavelet transform every subband is divided into a set of precincts corresponding to a subdivision of the subband into rectangular regions (in the current practical implementation 32x32 samples blocks). For efficient communication of localised regions of interest, JPIP servers often repacketize the content, as required, to ensure that precincts from the lowest resolution level (LL subband) of each image component contain only one code-block each, while precincts from all other resolution levels contain only one code-block from each of the LH, HL and HH subbands associated with that resolution level (i.e., three code-blocks in all). Each precinct is then itself divided into packets corresponding to the different quality layers.

JPIP servers typically work in epochs; in each epoch, a list of data-bin increments is compiled and delivered to the client. The amount of data sent in each epoch is controlled in order to avoid excessive amounts of in-transit traffic which could damage responsiveness to user navigation requests. This usually involves some form of channel

bandwidth estimation, with epochs sized for a transmission latency of say $\frac{1}{2}$ second to 1 second. These issues are discussed further in [9].

In the following discussion we will assume that the server transmits a whole number of packets from each precinct data-bin in each epoch. Every JPEG2000 packet contains the information required to improve the quality of one of the precincts as shown in Figure 6.1. Following the distortion-based approach we used until now, every packet contains the information required to reduce the distortion associated to the code blocks corresponding to the precinct. Let $D_{\beta,q}^i$ be the squared error distortion contributed to the decompressed image V^i by code-block \mathcal{B}_β^i when only the first q quality layers are available. These distortion values can be computed by just comparing the decompressed image with the full quality one and can be stored on a table at server side. Differently from the client image compression distortion estimation, in this case we have no transmission related issues and the size of the table is much smaller having to save the data only for each codeblock and not for every sample (or block of samples). So we can use the actual distortion values and there is no need to estimate them using the method described in [53]. The only issue related to this approach is that the actual values and the ones estimated at client side could have some mismatches leading to different choices on which information is going to be used. Also let $\{\mathcal{P}_\pi^i\}_\pi$ denote the collection of precincts from view V^i , indexed by π . Finally, we assume that the server keeps track of the number of quality layers q_π^i , which are available in the client's cache for all code-blocks in precinct \mathcal{P}_π^i .

At this point, since our spatial multi-resolution transform is based on an approximately orthonormal DWT, we can express the total distortion in V^* as a sum of the contributions coming from the distortion in the different subbands that are synthesized into V^* :

$$D^* \approx \sum_{d=0}^D \sum_{\mathbf{p}} D_d^*[\mathbf{p}] \quad (6.1)$$

In equation (6.1), $D_d^*[\mathbf{p}]$ represent the mean square error associated with the sample at location \mathbf{p} in resolution component $R_d^*[\mathbf{p}]$. The samples in the subbands used to reconstruct V^* are taken from the different source views and as described in Section 4.4 the distortion in the subbands of V^* can be estimated from the distortion in the source images. Assuming that the distortions associated with each source view are uncorrelated and exploiting equation (4.25) we can write:

$$D^* \approx \sum_d \sum_{\mathbf{p}} \sum_i (\rho_d^i[\mathbf{p}])^2 D_d^{i \rightarrow *}[\mathbf{p}] \quad (6.2)$$

In fact most of the terms in equation 6.2 are 0 because, except for the areas in proximity of the holes, the blending weights $\rho_d^i[\mathbf{p}]$ are just set to 1 for the best stitching source and to 0 for all the other views. If we look at equation (4.36), the distortion contribution coming from each sample can be expressed as the sum of two terms, one is the image compression distortion, that is a function of the amount of data transmitted for that particular codeblocks and the other is given by the sum of the geometry and

lighting terms that do not depend on the transmitted data¹. We can represent with $\Theta_d^{i \rightarrow *}[\mathbf{p}]$ the sum of the geometry and lighting contribution:

$$\Theta_d^{i \rightarrow *}[\mathbf{p}] = \overline{|\delta Z_d^*|^2}[\mathbf{p}] \cdot \overline{g_d^{i \rightarrow *}}[\mathbf{p}] \cdot \overline{|\omega_d|^2} \cdot \overline{E_d^{i \rightarrow *}}[\mathbf{p}] \\ + \gamma \tan(\max\{0, \cos^{-1}\langle \mathbf{e}^i, \mathbf{e}^* \rangle\}) \cdot \overline{E_d^{i \rightarrow *}}[\mathbf{p}]$$

If we substitute equation (4.36) into equation (6.2) we obtain the following representation of the distortion in the final rendered view at client side:

$$D^* \approx \sum_d \sum_{\mathbf{p}} \sum_i (\rho_d^i[\mathbf{p}])^2 \left(\Theta_d^{i \rightarrow *}[\mathbf{p}] + \sum_b W_{b \rightarrow d}^i[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] \right) \quad (6.3)$$

The server's target is of course to deliver the elements from the various compressed images that minimize the distortion D^* on the final rendered view subject to the constraint on the available bandwidth. If we denote with $L \leq L^{\text{epoch}}$ the amount of data that can be transmitted in every JPIP epoch, the problem can be reformulated in Lagrangian fashion as a family of unconstrained optimization objectives $J^*(\lambda)$, parametrized by $\lambda > 0$, where

$$J^*(\lambda) = D^*(\lambda) + \lambda L(\lambda) \quad (6.4)$$

The minimizing solution to each $J^*(\lambda)$ has the obvious property that $D^*(\lambda)$ is as small as it can be without increasing L beyond $L(\lambda)$. As a result, our global constrained optimization problem is solved once we find λ such that $L(\lambda) = L^{\text{epoch}}$. In practice the assumption that we are going to send whole packets made the problem discrete and it is quite unlikely that we can find such a λ . In the practical implementation, noting that $L(\lambda)$ is a decreasing function of λ , we select the smallest λ for which the constraint $L(\lambda) \leq L^{\text{epoch}}$ is satisfied. In common situations where we have several different source images each one made of many small code-blocks corresponding to the different subbands and spatial positions, we find that the resulting $L(\lambda)$ can be very close to L^{epoch} .

At a first glance the optimization problem seems similar to a standard image compression issue, specially considering that the geometry and lighting term does not depend on the transmitted data. Unfortunately it is complicated by the fact that the blending weights $\rho_d^i[\mathbf{p}]$ themselves depend upon the local distortion in the view images. In other words the blending weights depend on the transmitted data and by sending new data on one of the views we can cause a change in the stitching decision. This issue introduce a complex dependency between the two elements because by transmitting new data we can change the weights that in turn affect the decision on which data is going to be transmitted.

To address this difficulty, the optimization procedure has been divided into two steps. At server side two sets of blending weights are considered:

¹We are assuming that geometry is already available at client side, in the complete bandwidth allocation problem this term would depend on the choices on the transmission of geometry information.

- We denote with $\hat{\rho}_d^i[\mathbf{p}]$ the blending weights associated to the current available data, i.e. the values computed on the basis of the already received data, without taking into account the effect of the data that is going to be transmitted in the current epoch. The calculation procedure is the same that is used at client side, exploiting the fact the server keeps track of what it has sent to the client and knows the number of packets (quality layers) which the client has in its cache for each precinct data-bin.
- We will instead use $\vec{\rho}_d^i[\mathbf{p}]$ for the blending weights which would be used if all source views V^i were available with the maximum available quality – i.e., the weights corresponding to the blending choices that the client would make if it had received all the available data at server side. If the user stops on a particular view, while more and more data is received from the server the choices will progressively move towards $\vec{\rho}_d^i[\mathbf{p}]$ that represent the final stitching decisions. An useful property of this set of weights is that they do not depend on the amount of transmitted data and need to be recalculated only when the client's view of interest changes, rather than in each epoch.

A final remark is that the proposed scheme does not rely on the fact that the server is perfectly synchronized with the client. Even if they work with the same distortion model, we do not expect the blending choices to be exactly the same. This allows the server to use the more accurate distortion values obtained by comparison with the full quality image instead of the ones estimated from the compression parameters. The server can also work at reduced resolution, exploiting the fact that the distortion fields used to determine the best stitching source at client side are mostly smooth. This is also acceptable because we are only using this information at server side to make decisions regarding the amount of data to send for each code-block and code-blocks are reasonably large. Finally note that the server performs only distortion mapping; it does not actually synthesize any views.

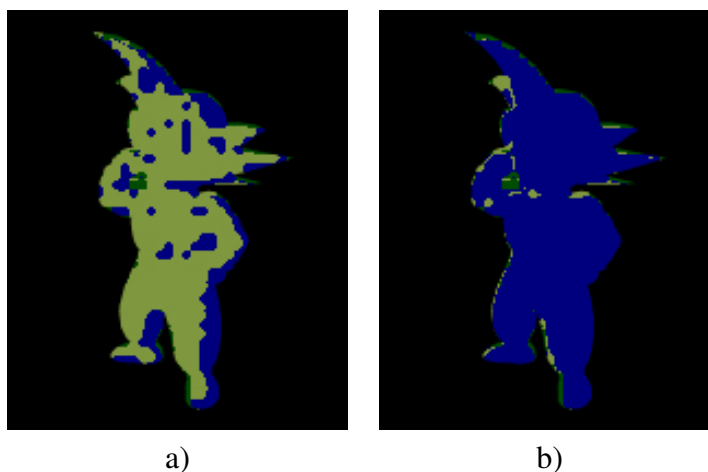


Figure 6.2: *Blending weights: a) Current choices $\hat{\rho}_d^i$, b) Choices with full quality images $\vec{\rho}_d^i$.*

For a better understanding of the difference between the two sets of weights, let's have a look at Figure 6.2. It refers to the case where 3 views are available at client side, the blue one is close to the required viewpoint but only a few bytes have been transmitted for it. The other two have already been completely transmitted but they are farther from the required viewpoint. Figure 6.2a shows $\hat{\rho}_d^i$, in this case the current version of the blue view has poor quality and most samples are taken from the other two views. Instead Figure 6.2b shows $\vec{\rho}_d^i$, the choices that we had if all the images were available at full quality. As expected most samples are taken from the closest view.

To deal with the optimization problem at server side we decided to consider two different particular cases :

- In the first case we solve the optimization problem using the assumption that $\rho_d^i[\mathbf{p}] = \hat{\rho}_d^i[\mathbf{p}]$, i.e. the blending weights will not change between this epoch and the next. This means that we are going to continue using the same choices we are using now also in the future. With this assumption the server will tend to send more information for those code-blocks which already contribute most strongly to the client's view synthesis process. This reduces the distortion associated with the source views (or part of them) that are already used to generate the required view and reinforces the decision of using them also in subsequent epochs. We call the set of packets that will be sent in this case "*reinforcing enhancements*".
- We instead call "*disruptive enhancements*" the set of packets that the server is going to send in the case where the blending weights will change from $\hat{\rho}_d^i[\mathbf{p}]$ to $\vec{\rho}_d^i[\mathbf{p}]$ once all data in this epoch has been transmitted, i.e. the choices that we would have made if all the images were available with maximum quality

By using reinforcing enhancements the system will continue to use the views that are currently available with good quality but will not be able to exploit the availability of closer views better aligned with the required rendering for which no data has still been transmitted. Moreover, since $\hat{\rho}_d^i[\mathbf{p}]$ may well be zero for some source views which are non-zero in $\vec{\rho}_d^i[\mathbf{p}]$, in many circumstances with reinforcing enhancements alone we will never reach the highest rendering quality. To deal with this issues we need the disruptive enhancements. On the other hand, the data that is going to be transmitted in the next epoch will not probably be enough to force the switch from the current blending weights to the final choices and part of the data we are transmitted will not be used because the choices at client side will be different. In the subsequent epochs after receiving further data the choices will change towards the final ones and the data will be used, but this will never happen if the user changes his view of interest before all the information has been transmitted. The two kind of reinforcements can be considered as the two extreme cases of the optimization problem and a solution that is able to combine the results from these two optimization procedures is needed. In the following sections we will start the description of the optimization algorithm from the simpler case of the reinforcing enhancements, then we will introduce the disruptive ones. Finally, in Section 6.4, we discuss the complete solution, in which the two optimization procedures are combined.

6.2 Optimization of Reinforcing Enhancements

The reinforcing enhancements correspond to the case where the blending weights remain constant and are not affected by the data that is going to be sent to the client. We can so replace the blending weights in equation (6.3) with the constant set of values $\rho_d^i[\mathbf{p}] = \overset{\circ}{\rho}_d^i[\mathbf{p}]$. In Section 6.1 we noted that the geometry-dependent terms $\Theta_d^{i \rightarrow *}[\mathbf{p}]$ do not depend on the amount of transmitted data. Now they are multiplied by the blending weights that are constant and the whole geometry dependent term represent a constant offset to D^* that can be excluded from the optimization problem. By substituting the distortion formulation into equation 6.4 we obtain the following optimization objective:

$$\overset{\circ}{J}^*(\lambda) = \lambda L(\lambda) + \sum_d \sum_{\mathbf{p}} \sum_i (\overset{\circ}{\rho}_d^i[\mathbf{p}])^2 \sum_b W_{b \rightarrow d}^i[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] \quad (6.5)$$

We can combine together the blending choices and the warping coefficients for all the samples which get mapped to a certain sample \mathbf{k} into a new operator $\overset{\circ}{\Psi}_b^i[\mathbf{k}]$:

$$\overset{\circ}{\Psi}_b^i[\mathbf{k}] = \sum_{d, \mathbf{p} \ni (\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) = \mathbf{k}} (\overset{\circ}{\rho}_d^i[\mathbf{p}])^2 W_{b \rightarrow d}^i[\mathbf{p}]$$

and by introducing it into equation 6.5 we obtain a new more compact representation of the minimization target:

$$\overset{\circ}{J}^*(\lambda) = \lambda L(\lambda) + \sum_i \sum_b \sum_{\mathbf{k}} \overset{\circ}{\Psi}_b^i[\mathbf{k}] \cdot D_b^i[\mathbf{k}] \quad (6.6)$$

To compute the distortion term in equation (6.6) we need to know the distortion on each single sample $D_b^i[k]$, but the server stores in the distortion tables only the total squared error $D_{\beta, q}^i$ in each code-block \mathcal{B}_β^i when we have received q quality layers for that block. In the absence of more precise information, therefore, we compute the distortion on each sample \mathbf{k} as the average of the distortion of the samples in the codeblocks \mathcal{B}_β^i that contains the sample. $D_b^i[k]$ is then given by the distortion in the codeblock divided by its size:

$$D_{b(\beta)}^i[\mathbf{k}] = D_{\beta, q}^i / |\mathcal{B}_\beta^i|$$

where $|\mathcal{B}_\beta^i|$ is the number of samples in the code-block and $b(\beta)$ denotes the sub-band to which the code-block belongs.

In the practical implementation the server can decide to transmit whole packets each one corresponding to a quality layer in one of the different precincts. Entering in more detail the data corresponding to the different precincts is divided into different quality layers and at the beginning of the current epoch the client holds a certain number of quality layers for each precinct. Every packet that the server can transmit permits to increase the number of quality layer in one of the precincts and the target is to find the ones that allow to obtain the best reduction in the total distortion subject to the bandwidth constraint.

Before going on, let us introduce some new notation symbols:

- $L_{\pi,q}^i$ represent the length in bytes corresponding to the first q packets in in precinct \mathcal{P}_π^i
- \hat{q}_π^i represent the number of quality layers (packets) that the client already has in its cache for precinct \mathcal{P}_π^i at the beginning of this epoch
- q_π^i is the total number of quality layers (packets) which the client will have in its cache for precinct \mathcal{P}_π^i after the end of the epoch current epoch. It is given by the sum of \hat{q}_π^i and the number of packets that the server will transmit for \mathcal{P}_π^i in the current epoch.
- $\hat{\Psi}_\beta^i$ represent the sum of the coefficients $\hat{\Psi}_b^i[\mathbf{k}]$ on all the samples that belong to the codeblock \mathcal{B}_β^i :

$$\hat{\Psi}_\beta^i = \sum_{\mathbf{k} \in \mathcal{B}_\beta^i} \hat{\Psi}_{b(\beta)}^i[\mathbf{k}]$$

With this notation the amount of transmitted data for each precinct is given by $(L_{\pi,q_\pi^i}^i - L_{\pi,\hat{q}_\pi^i}^i)$. We can now introduce a new expression of our optimization target in terms of precinct data-bin decisions :

$$\hat{J}^*(\lambda) = \sum_i \sum_\pi \left[\lambda \left(L_{\pi,q_\pi^i}^i - L_{\pi,\hat{q}_\pi^i}^i \right) + \sum_{\mathcal{B}_\beta^i \in \mathcal{P}_\pi^i} \hat{\Psi}_\beta^i \cdot D_{\beta,q_\pi^i}^i \right] \quad (6.7)$$

Equation (6.7) can be decomposed into a set of independent optimization objectives for each precinct, for which the optimal solution is given by

$$q_\pi^i(\lambda) = \underset{q_\pi^i \geq \hat{q}_\pi^i}{\operatorname{argmin}} \left[\lambda L_{\pi,q_\pi^i}^i + \underbrace{\sum_{\mathcal{B}_\beta^i \in \mathcal{P}_\pi^i} \hat{\Psi}_\beta^i \cdot D_{\beta,q_\pi^i}^i}_{\hat{D}_{\pi,q}^i} \right] \quad (6.8)$$

The server holds pre-computed tables with $D_{\beta,q_\pi^i}^i$ and $L_{\pi,q_\pi^i}^i$ and can exploit them to compute the solution of (6.8).

Figure 6.3 shows an example of the rate distortion curve and convex hull corresponding to a particular precinct. For each precinct the delivery of a new packet correspond to a couple in the set of distortion-length pairs $\left\{ \left(L_{\pi,q}^i, \hat{D}_{\pi,q}^i \right) \right\}_q$. Every pair correspond to a distortion-length “slope” $\hat{S}_{\pi,q}^i$. If the set of distortion-length points $\left(L_{\pi,q}^i, \hat{D}_{\pi,q}^i \right)$ is already convex, we can compute the slopes simply as the ratio of the distortion gain versus the data length associated to the packet:

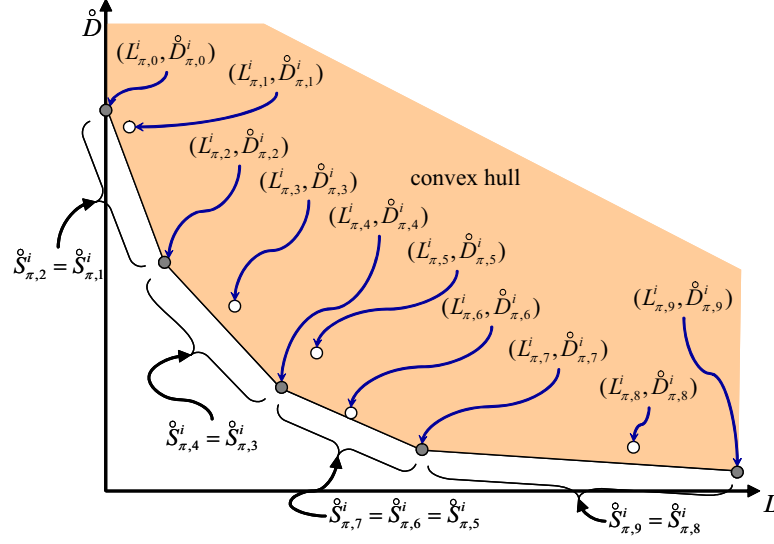


Figure 6.3: Convex hull and distortion-length slopes associated with a particular precinct P_π^i for the purpose of reinforcing enhancements. Only those points $h \in \mathcal{H}_\pi^i$ (shaded circles) will ever be selected as solutions to equation (6.8).

$$\dot{S}_{\pi,q}^i = \frac{\hat{D}_{\pi,q-1}^i - \hat{D}_{\pi,q}^i}{L_{\pi,q}^i - L_{\pi,q-1}^i} \text{ for } q > 0$$

Figure 6.3 instead shows the more complex case in which the set of distortion-length points is not convex. In this case we started from finding the lower convex hull \mathcal{H}_π^i of the $(L_{\pi,q}^i, \hat{D}_{\pi,q}^i)$ pairs, that in the figure is represented by the coloured area. Note that only some of the distortion-length pairs lies on \mathcal{H}_π^i and these points (represented by the shaded circles in the figures) will be the only ones that can be selected as solution to equation (6.8). We will use the notation h_i for the set of quality layers q that lies on the convex hull. The ordered set of quality layers on \mathcal{H}_π^i is so given by $0 = h_0 < h_1 < h_2 < \dots$.

The slopes are given by:

$$\dot{S}_{\pi,q}^i = \frac{\hat{D}_{\pi,h_{k-1}}^i - \hat{D}_{\pi,h_k}^i}{L_{\pi,h_k}^i - L_{\pi,h_{k-1}}^i} \quad \text{for } h_{k-1} < q \leq h_k, \quad k > 0.$$

The slopes are ordered in a decreasing way:

$$\infty = \dot{S}_{\pi,0}^i \geq \dot{S}_{\pi,1}^i \geq \dot{S}_{\pi,2}^i \geq \dots$$

Excluding the case where $q_\pi^i(\lambda) < \hat{q}_\pi^i$, i.e. the number of quality layers in the solution is less than the number of already transmitted quality layers², the solution to

²This case will never happen in practice

equation (6.8) is given by:

$$q_{\pi}^i(\lambda) = \max \left\{ q \mid \hat{S}_{\pi,q}^i > \lambda \right\},$$

Remember that only those $q \in \mathcal{H}_{\pi}^i$ can actually arise as solutions to equation (6.8). An important aspect of this slope-based formulation is that it provides some intuition about the meaning of λ . It tells us that λ represent a sort of threshold on the distortion-length slope. In each successive epoch, we expect λ to decrease, thus allowing additional quality layers q to pass the test $\hat{S}_{\pi,q}^i > \lambda$ and to be chosen for transmission to the client as reinforcing enhancements.

6.3 Optimization of Disruptive Enhancements

Disruptive enhancements correspond instead to the case where there is a switch from the current blending weights to the ones we would have if all the images had been received at maximum quality. The idea behind them is to handle the cases where instead of improving the quality of currently used images better results can be obtained by transmitting data for code-blocks of images that are not currently selected but are better aligned with the required rendering.

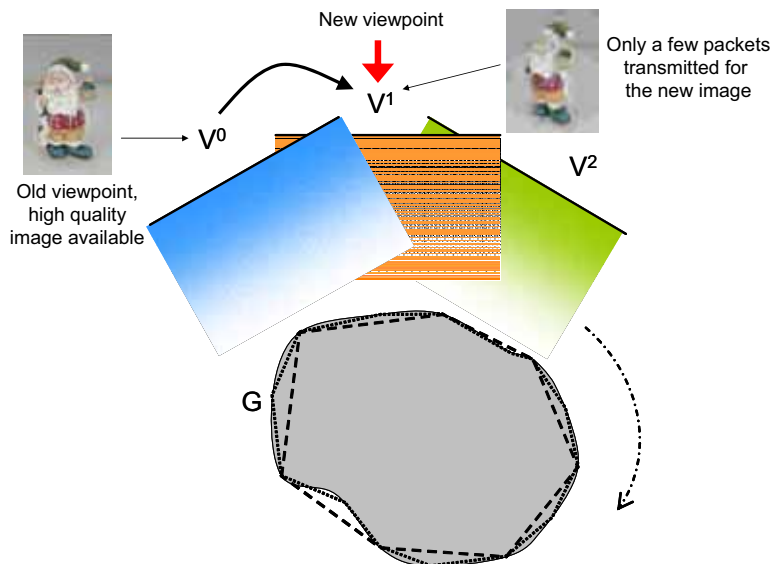


Figure 6.4: Example of interactive scene browsing requiring disruptive enhancements

For example let us consider the case depicted in Figure 6.4. In the example the user has been looking at the scene from the viewpoint of view V^0 for a while and a high quality version of V^0 is available at client side. Then the user moves to the viewpoint corresponding to view V^1 . Initially, no data (or only a small amount of data) is available for V^1 and it is most favourable to render the required view V^* by

warping the already received data for view V^0 . View V^1 has a very high distortion and the blending choices at the beginning force the system to take all the samples from the good quality view V^0 :

$$\begin{cases} \hat{\rho}_d^0[\mathbf{p}] = 1 & \forall d, \mathbf{p} \\ \hat{\rho}_d^1[\mathbf{p}] = 0 & \forall d, \mathbf{p} \end{cases}$$

It follows that $\hat{\Psi}_\beta^1 = 0, \forall \beta$ and from equation (6.8) we obtain that no reinforcing enhancements will ever be taken from image V^1 . At the beginning it looks acceptable because V^0 has a better quality, and the switch to V^1 will cause an increase in distortion (we will call the extra distortion associated to the change in the blending weights *switching penalty* in the following discussion). But V^1 is better aligned with V^* and the distortion associated to the samples taken from it goes down faster as soon as more data is received. At a certain point we expect the switching penalty to be compensated from the delivery of sufficient bytes from the code-blocks of V^1 . Consider also that the rate-distortion slopes in image compression become smaller and smaller as more data is received, thus reducing the gain obtainable from V^0 . Finally the highest image quality will probably be achievable only by using V^1 because some high frequency elements are just missing from V^0 (see Section 4.4.2). With the reinforcing enhancements alone the client will continue to render V^* from V^0 and it will never start to transmit data for the better aligned view V^1 .

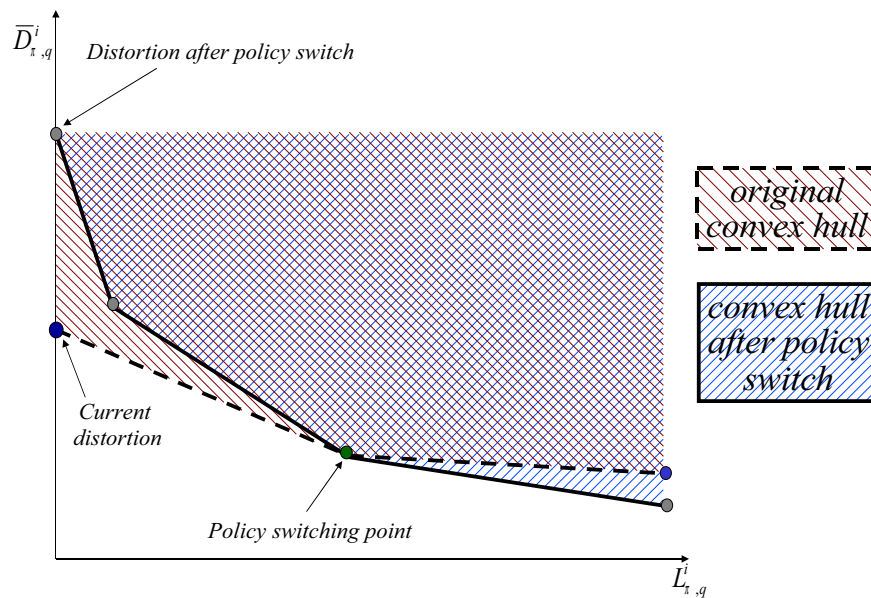


Figure 6.5: Convex hulls corresponding to the reinforcing and disruptive enhancements

Figure 6.5 show the convex hulls associated to the solution of the reinforcing and disruptive enhancements problems for one particular precinct. The dotted line bounds

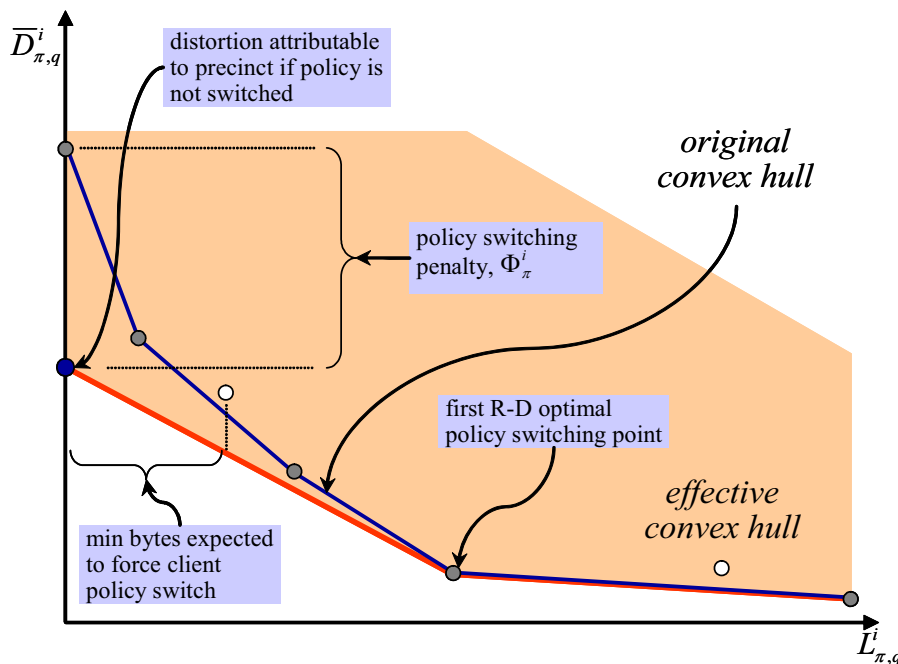


Figure 6.6: *Effective distortion-length slope properties associated with disruptive enhancements.*

the reinforcing solution convex hull. At the beginning it shows a lower distortion due to the already received data for V^0 , but due to the worse alignment the curve decreases slower than the disruptive one. Instead the disruptive slope (continuous line) starts from a higher distortion, but decrease also faster. From the figure it is clear that at the beginning it is better to send reinforcing enhancement but after a certain point (the *policy switching point* in the figure) disruptive enhancements offer better results and the change in the blending choices can actually happen.

In Figure 6.6 we underline how the policy switching penalty places an upper bound on the effective distortion-length slope associated to disruptive enhancements. Let us assume that the user's point of view remain fixed. At the beginning the *slope threshold* λ may be too large to favour disruptive enhancement, but the λ value decreases in each successive epoch. At a certain point most of the reinforcing enhancements with large distortion slopes $\hat{S}_{\pi,q}^i$ will have been sent to the client and λ would be small enough to include some disruptive enhancements. Another important observation is that the distortion associated to disruptive enhancements starts from a higher value and decreases faster, so there is a minimum amount of data to transmit on that particular code-block before the policy switch becomes worthy. Since the amount of data the server can transmit in every epoch is limited, it is also clear that we can introduce the disruptive enhancement only in a very small set of code-blocks in each epoch. A final remark about disruptive enhancement is that for making them really useful we expect

the client to made the same policy switch we have computed at server side, i.e. we expect the blending choices to change from $\hat{\rho}_d^i[\mathbf{p}]$ to $\bar{\rho}_d^i[\mathbf{p}]$ over the regions where we are going to transmit the disruptive enhancements. Unfortunately this is only partially true because the choices on the packets to be transmitted are associated with precinct regions in the source images, while blending decisions $\hat{\rho}_d^i[\mathbf{p}]$ are formed on the samples within each subband of the target view V^* . This mismatch cause the actual policy switching choices at client side to be different from the server's expectations. However while λ becomes smaller and smaller we expect to send more and more disruptive enhancements in successive epochs and probably the client policy will switch to $\bar{\rho}_d^i[\mathbf{p}]$, even if it does not do so immediately.

To solve the optimization problem in the case of disruptive enhancement we started from the computation of the overall policy switching penalty associated to a complete change of the blending choices from $\hat{\rho}_d^i[\mathbf{p}]$ to $\bar{\rho}_d^i[\mathbf{p}]$.

The distortion associated to the current blending choices , computed with the model in equation (6.3) is:

$$\Delta D^* = \sum_d \sum_{\mathbf{p}} \sum_i (\hat{\rho}_d^i[\mathbf{p}])^2 \cdot \left(\Theta_d^{i \rightarrow *}[\mathbf{p}] + \sum_b W_{b \rightarrow d}^i[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] \right) \quad (6.9)$$

while the (higher) distortion associated with the switched policy is:

$$\Delta D^* = \sum_d \sum_{\mathbf{p}} \sum_i (\bar{\rho}_d^i[\mathbf{p}])^2 \cdot \left(\Theta_d^{i \rightarrow *}[\mathbf{p}] + \sum_b W_{b \rightarrow d}^i[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] \right) \quad (6.10)$$

The policy switching penalty is given by the difference between (6.10) and (6.9). By defining :

$$\Delta \rho_d^i[\mathbf{p}] = (\bar{\rho}_d^i[\mathbf{p}])^2 - (\hat{\rho}_d^i[\mathbf{p}])^2$$

we can express the overall policy switching penalty associated with a wholesale change of all blending weights as:

$$\Delta D^* = \sum_d \sum_{\mathbf{p}} \underbrace{\sum_i \Delta \rho_d^i[\mathbf{p}] \cdot \left(\Theta_d^{i \rightarrow *}[\mathbf{p}] + \sum_b W_{b \rightarrow d}^i[\mathbf{p}] \cdot D_b^i \left[(\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) \right] \right)}_{\Phi_d[\mathbf{p}]} \quad (6.11)$$

The values of the subband distortions $D_b^i[\mathbf{k}]$ in equation (6.11) are the ones corresponding to the current number of layers \hat{q}_π^i for each precinct data-bin \mathcal{P}_π^i . The transmission choices at server side are made on whole packets corresponding to the quality layers of the precincts and we need to distribute the overall switching penalty between the different precincts in the source images. Unfortunately, there is no perfect way to do this, since blending weights are assigned on the samples in the subbands of V^* , rather than on the precincts of V^i . Nevertheless, a good distribution can be achieved

by distributing ΔD^* is on the basis of the distortion weights, $(\bar{\rho}_d^i[\mathbf{p}])^2 W_{b \rightarrow d}^i[\mathbf{p}]$, which would apply if the policy switch took place. This associates the policy switching penalty with those source precincts whose distortion impacts the synthesized view most, and these are the ones for which disruptive enhancements are likely to be sent, if at all.

The terms $\Phi_d[\mathbf{p}]$ in equation (6.11) represent the total distortion penalty on sample \mathbf{p} coming from all the subbands and the different source views. As previously discussed the distortion on \mathbf{p} coming only from a certain subband of one of the source views can be computed by multiplying $\Phi_d[\mathbf{p}]$ by the ratio between the distortion weight of that subband and the sum of all the distortion weights. In fact we distribute the distortion between the various source view subbands using the same weights that are used while calculating distortion, except with $(\bar{\rho}_d^i[\mathbf{p}])^2$ instead of $(\bar{\rho}_d^i[\mathbf{p}])^2 - (\hat{\rho}_d^i[\mathbf{p}])^2$

We will denote with $\Phi_{d,b}^i[\mathbf{p}]$ the distortion on sample \mathbf{p} coming from subband b of view i , that can be computed in the following way:

$$\Phi_{d,b}^i[\mathbf{p}] = \Phi_d[\mathbf{p}] \cdot \frac{(\bar{\rho}_d^i[\mathbf{p}])^2 W_{b \rightarrow d}^i[\mathbf{p}]}{\sum_{b',i'} (\bar{\rho}_{d'}^{i'}[\mathbf{p}])^2 W_{b' \rightarrow d}^{i'}[\mathbf{p}]}$$

The switching penalty coming from precinct \mathcal{P}_π^i of source view i can be computed by summing up on all the codeblocks belonging to that precinct the $\Phi_{d,b}^i$ corresponding to the samples that are mapped to these codeblocks.

$$\Phi_\pi^i = \sum_{\beta \in \mathcal{P}_\pi^i} \sum_{d, \mathbf{p} \ni (\mathcal{W}_{b(\beta) \rightarrow d}^i)^{-1}(\mathbf{p}) \in \mathcal{B}_\beta^i} \Phi_{d,b}^i[\mathbf{p}]$$

Using this new notation ΔD^* can be expressed as the sum of the switching penalties coming from the different precincts in the various source views :

$$\begin{aligned} \Delta D^* &= \sum_d \sum_{\mathbf{p}} \Phi_d[\mathbf{p}] \\ &= \sum_d \sum_{\mathbf{p}} \sum_i \sum_b \Phi_{d,b}^i[\mathbf{p}] \\ &= \sum_i \sum_{\pi} \Phi_\pi^i \end{aligned} \tag{6.12}$$

The additional distortion contribution given by Equation (6.12) must of course be included in the total distortion only if disruptive enhancements are selected for precinct \mathcal{P}_π^i . To compute the distortion gain associated with the disruptive enhancements for a particular precinct we can use the same method we used for the reinforcing ones. Following the discussion in Section 6.2 we can combine together the blending choices and the warping coefficients for all the samples which get mapped to a certain sample \mathbf{k} into the operator

$$\vec{\Psi}_b^i[\mathbf{k}] = \sum_{d, \mathbf{p} \ni (\mathcal{W}_{b \rightarrow d}^i)^{-1}(\mathbf{p}) = \mathbf{k}} (\bar{\rho}_d^i[\mathbf{p}])^2 W_{b \rightarrow d}^i[\mathbf{p}]$$

As in the reinforcing case we denote with $\vec{\Psi}_\beta^i$ the sum of $\vec{\Psi}_b^i[\mathbf{k}]$ over all the samples in the codeblock:

$$\vec{\Psi}_\beta^i = \sum_{\mathbf{k} \in \mathcal{B}_\beta^i} \vec{\Psi}_{b(\beta)}^i[\mathbf{k}]$$

The operator $\vec{\Psi}_\beta^i$ is the same as $\vec{\Psi}_\beta^i$ except for the fact that now we are using the switched blending weights $\vec{\rho}_d^i[\mathbf{p}]$ instead of the ones corresponding to the current choices $\rho_d^i[\mathbf{p}]$.

Let us consider the case of a disruptive enhancement that provides q quality layers for precinct \mathcal{P}_π^i . The amount of data corresponding to this enhancement is $L_{\pi,q}^i - L_{\pi,\hat{q}_\pi^i}^i$ bytes. We can compute the distortion after the disruptive enhancement using the same procedure we used for the reinforcing case:

$$\vec{D}_{\pi,q}^i = \sum_{\mathcal{B}_\beta^i \in \mathcal{P}_\pi^i} \vec{\Psi}_\beta^i \cdot D_{\beta,q}^i$$

Again the definition for $\vec{D}_{\pi,q}^i$ is identical to the one we give for $\vec{D}_{\pi,q}^i$ in (6.8) except that we replaced the blending choices with the switched ones.

The q quality layers transmitted for precinct \mathcal{P}_π^i will increase the number of available quality layers from \hat{q}_π^i to q_π^i and the distortion gain is so given by $\vec{D}_{\pi,q_\pi^i}^i - \vec{D}_{\pi,q}^i$. This gain is balanced by the switching penalty, and the total distortion change is given by:

$$\Delta D_\pi^i = \left(\vec{D}_{\pi,q_\pi^i}^i - \vec{D}_{\pi,q}^i \right) - \Phi_\pi^i \quad (6.13)$$

An important aspect of equation (6.13) is that the actual distortion gain is not necessary positive. If the number of transmitted quality layers is too small the distortion gain $\vec{D}_{\pi,q_\pi^i}^i - \vec{D}_{\pi,q}^i$ can be smaller than the policy switching penalty Φ_π^i and ΔD_π^i can be smaller than zero. In this case there is no point in transmitting disruptive enhancement for that precinct and we expect only reinforcing enhancements. In other words $\Delta D_\pi^i > 0$ represent the condition under which the disruptive enhancement is expected to cause policy switching at client side.

We can exploit equation (6.13) to find a new formulation for $J^*(\lambda)$ in the case of a disruptive enhancement which assigns q_π^i layers to precinct \mathcal{P}_π^i :

$$\vec{J}_{\pi,q_\pi^i}^i(\lambda) = \begin{cases} 0, & q_\pi^i = \hat{q}_\pi^i \\ \Phi_\pi^i + \left(\vec{D}_{\pi,q_\pi^i}^i - \vec{D}_{\pi,q}^i \right) + \lambda \left(L_{\pi,q_\pi^i}^i - L_{\pi,\hat{q}_\pi^i}^i \right), & q_\pi^i > \hat{q}_\pi^i \end{cases}$$

The first case correspond to the case where we decided not to transmit any disruptive enhancement, while the second one can be considered valid only when $q_\pi^i > \hat{q}_\pi^i$ is sufficiently large to ensure that $\Delta D_\pi^i > 0$.

It is instructive to consider solutions to the problem

$$q_\pi^i(\lambda) = \operatorname{argmin}_{q_\pi^i \geq \hat{q}_\pi^i} \vec{J}_{\pi,q_\pi^i}^i(\lambda) \quad (6.14)$$

This problem can also be formulated in the following way:

$$q_\pi^i(\lambda) = \operatorname{argmin}_{q_\pi^i \geq \hat{q}_\pi^i} \left[\lambda L_{\pi, q_\pi^i}^i(\lambda) + \bar{D}_{\pi, q_\pi^i}^i \right]$$

where

$$\bar{D}_{\pi, q}^i = \begin{cases} \bar{D}_{\pi, \hat{q}_\pi^i}^i - \Phi_\pi^i, & q = \hat{q}_\pi^i \\ \bar{D}_{\pi, q}^i, & q > \hat{q}_\pi^i \end{cases}$$

Figure 6.6 shows a plot of $\bar{D}_{\pi, q}^i$ vs. $L_{\pi, q}^i$. Let $\vec{\mathcal{H}}_\pi^i$ denote the convex hull of the $(L_{\pi, q}^i, \bar{D}_{\pi, q}^i)$ pairs. This is identified in Figure 6.6 as the *effective convex hull*. Following the discussion at the end of Section 6.2, only those q which belong to $\vec{\mathcal{H}}_\pi^i$ can arise as solutions to equation (6.14). All points $q > \hat{q}_\pi^i$ which belong to $\vec{\mathcal{H}}_\pi^i$ must necessarily satisfy the condition $\bar{D}_{\pi, q}^i < \bar{D}_{\pi, \hat{q}_\pi^i}^i$, which is equivalent to $\Phi_\pi^i + (\bar{D}_{\pi, q_\pi^i}^i - \bar{D}_{\pi, \hat{q}_\pi^i}^i) < 0$. As a result, any $q_\pi^i(\lambda) > \hat{q}_\pi^i$ which is solution of equation (6.14) already satisfies the condition required for the enhancement to be considered disruptive. As suggested by Figure 6.6, for large policy switching penalties Φ_π^i , the first $q > \hat{q}_\pi^i$ which belongs to $\vec{\mathcal{H}}_\pi^i$ can lie significantly beyond the point at which $\bar{D}_{\pi, q}^i < \bar{D}_{\pi, \hat{q}_\pi^i}^i$. This increase the probability that the assumption that the same policy switch will happen also at client side is right.

6.4 Complete Server Solution

In the two previous section we described the methods to compute disruptive and reinforcing enhancements independently. The final missing element for the server image transmission policy is how to combine together the two different types of enhancements.

A critical point is that reinforcing enhancements are based on the assumption that the blending weights $\rho_d^i[\mathbf{p}]$ will not change. This supposition means that there should be no disruptive enhancements. A possible solution to this problem could be to start from computing the disruptive enhancements, then recompute the blending weights taking into account the new data from disruptive enhancements and finally compute the reinforcing enhancements. These steps should be repeated for every value of λ until a solution which satisfies the bandwidth constraint L^{epoch} is found. Even if this approach is theoretically correct, the recomputation of the blending choices in each iteration of the Lagrangian optimization is impractical in a real time system as the one we are trying to build.

We instead follow a different approach: we simply take the maximum of the number of quality layers for each precinct $q_\pi^i(\lambda)$ computed with the two different procedures, i.e. the values yielded by equations (6.8) and (6.14). $q_\pi^i(\lambda)$ is so given by:

$$q_\pi^i(\lambda) = \max \left\{ \operatorname{argmin}_{q_\pi^i \geq \hat{q}_\pi^i} \left[\lambda L_{\pi, q_\pi^i}^i(\lambda) + \bar{D}_{\pi, q_\pi^i}^i \right], \operatorname{argmin}_{q_\pi^i \geq \hat{q}_\pi^i} \left[\lambda L_{\pi, q_\pi^i}^i + \bar{D}_{\pi, q_\pi^i}^i \right] \right\}.$$

We then adjust λ with an outer loop until the constraint on the epoch size $\sum_{i,\pi} L_{\pi}^i(\lambda) \approx L^{\text{epoch}}$ is satisfied. This is correct because both the solutions to (6.8) and (6.14) are non-increasing functions of λ , and so is also the maximum of the two.

This “max of solutions” policy tends to send somewhat more reinforcing enhancements than might be involved in a truly optimal solution, which may be regarded as a conservative position. The reason for this is that the reinforcing enhancements are computed using the assumption that the blending weights will not change. In a really optimal solution they should be recomputed after every disruptive step. In this case some reinforcing enhancements will probably be dropped because after the change in the blending weights due to the disruptive step the corresponding samples are no longer used.

From a certain point of view this is a good aspect because, as previously explained, we are not completely sure that the expected policy switch will really happen due to the mixing of many contributions in the policy decision represented by equation (4.26). Even if this policy does not give a truly optimal solution it is important to underline that any sub-optimality introduced by this approach is limited by the size of the epoch because the blending weights will be recomputed in the next epoch.

It is worth considering what happens when the client has very little or no cached precinct data from previous interaction with the server. In this case, it should not be relevant what initial blending weights $\hat{\rho}_d^i[\mathbf{p}]$ are assumed at the start of the epoch. To check that this is the case, suppose firstly that $\hat{\rho}_d^i[\mathbf{p}] = \bar{\rho}_d^i[\mathbf{p}]$. Then there will be no disruptive enhancements, but reinforcing enhancements will deliver the most relevant content to the client. If $\hat{\rho}_d^i[\mathbf{p}] \neq \bar{\rho}_d^i[\mathbf{p}]$, we expect to find that the policy switching penalties Φ_{π}^i of those precincts which contribute to the rendered view through the disruptive enhancements $\bar{\rho}_d^i[\mathbf{p}]$ will be negative due to the terms $\Theta_d^{i \rightarrow *}$ in equation (6.11). This means that the epoch length limit L^{epoch} will be achieved even with very large values for λ , for which disruptive enhancements can still arise from equation (6.14). Reinforcing enhancements, however, are unlikely to be produced when λ is very large. Again, therefore, the server will deliver the most relevant content to the client.

6.5 Geometry transmission policy

The previous sections present a solution to the allocation of transmission resources between the different contributions to the texture information. Geometry information is also represented as JPEG2000 scalable compressed greyscale images and the various depth maps are combined together using the same distortion framework. Thus geometry transmission can be optimized independently using the same procedure we used for the images. The final missing element is how to combine together the solutions found for geometry and texture. The idea is to exploit the geometry dependent component in the distortion formulation to understand how geometric distortion affects the rendered image and to optimally subdivide the bandwidth between texture and geometry. This solution does not take into account possible changes in the blending weights used for the images due to geometry improvements, but as in the image case the suboptimality

introduced is limited by the fact that weights are periodically recomputed. The current prototype application does not include this part of the server application, that is the subject of the current research activity.

Chapter 7

Experimental results and performance analysis

The experimental results obtained with the prototype application are presented in this chapter. The first section is dedicated to the multiple view fusion procedure at client side and will show how the distortion framework described in Chapter 4 is able to select the most appropriate samples from the different views in order to use them to reconstruct the final rendering. Section 7.2 will show some examples of reconstruction of the scene geometry from the depth information. Section 7.3 will discuss the transmission policy at server side and finally Section 7.4 contains a brief discussion of the performances of the current prototype.

Four different models have been used in the experiments. Two of them, the *Santa Claus* and *Frog* models have been obtained from real objects, while the other two (*Goku* and *GT Car*) are synthetic 3D models. The Santa Claus statuette (Fig. 7.1a) had been placed on the turntable of Figure 3.5 and 60 pictures of it spaced of 6 degrees each have been taken. The passive reconstruction method described in [48] has been used to reconstruct the geometry description from the pictures (Fig. 7.1b and 7.1c). Finally the depth maps have been extracted from the reconstructed 3D model. Passive reconstruction methods have a low accuracy compared with range cameras and this model is particularly interesting to evaluate the effects of the geometry dependent terms in the distortion framework. Note that the images used in the experiments are the real pictures of the object on the turntable, not renderings of a textured model or other synthetic data. This allows to obtain a photorealistic rendering even if the geometry description is poor. In most of the experiments we used only 12 of the 60 available views (see Figure 3.3), corresponding to a spacing of 30 degrees that is already enough to have a complete scene description. The *Frog* dataset (Fig. 7.2) has been built with the same procedure but compared to the Santa Claus it has a simpler geometry and less texture details. It represents an “easier” object.

The *Goku* model instead is a synthetic model made of about 9000 triangles. The views have been rendered at the resolution of 1024x768 pixels using the *3D Studio MAX* software. The dataset (shown in Figure 3.6) is made of 10 views, eight taken around the object, a top view and a bottom one. Depth maps have also been exported from the model for the same viewpoints as the images. This model has many details

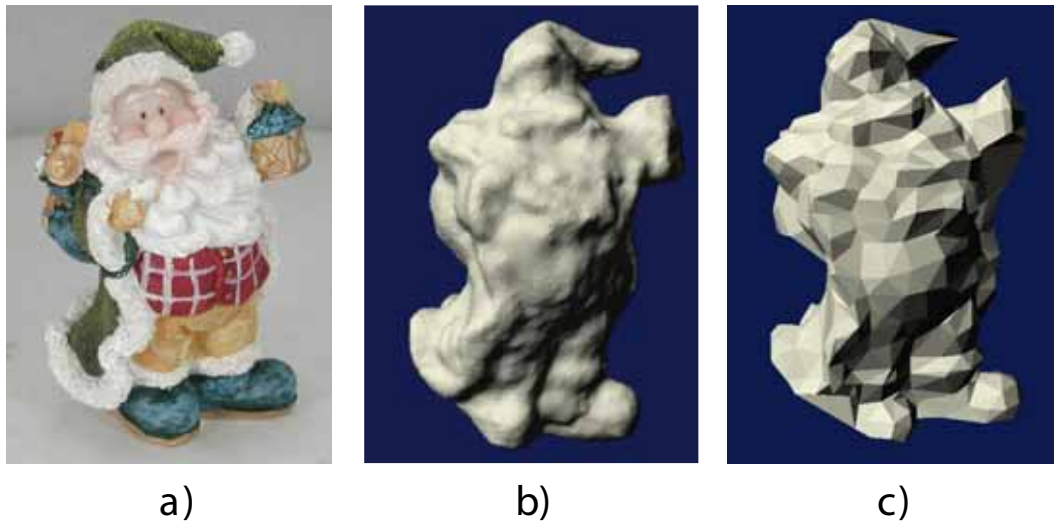


Figure 7.1: *Santa Claus: a) one view; b) the 3D model; c) simplified 3D model*

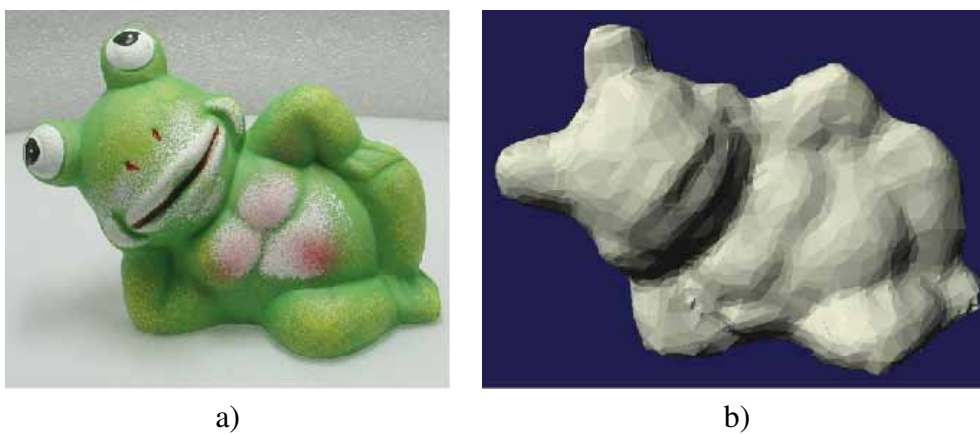


Figure 7.2: *Frog: a) one view; b) the 3D model.*

and the position of the two arms is interesting to evaluate the effects of holes and occlusions. Finally the *GT Car* model (Fig. 7.4) is another synthetic model of 8300 triangles. It has a relatively simple geometry but a lot of texture details. For this model we used only 8 views taken around it with the corresponding depth information.

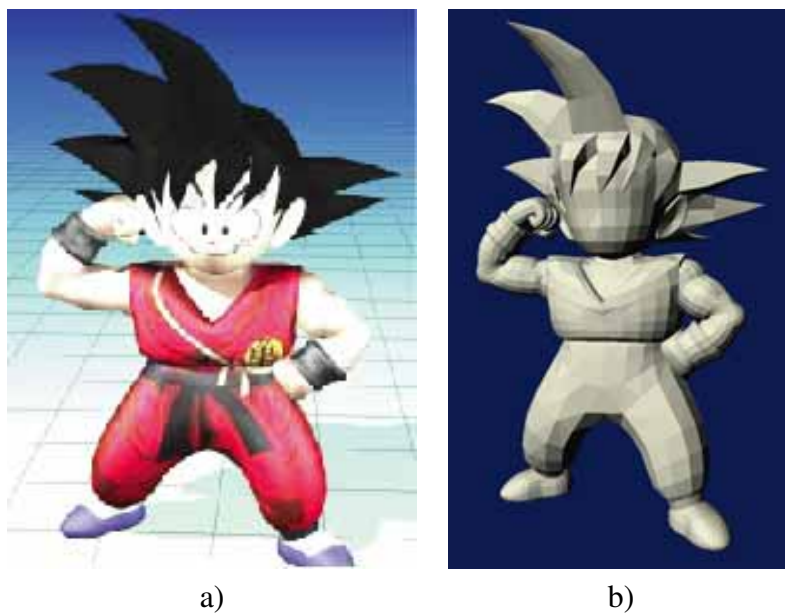


Figure 7.3: *Goku cartoon character: a) one view; b) the 3D model.*

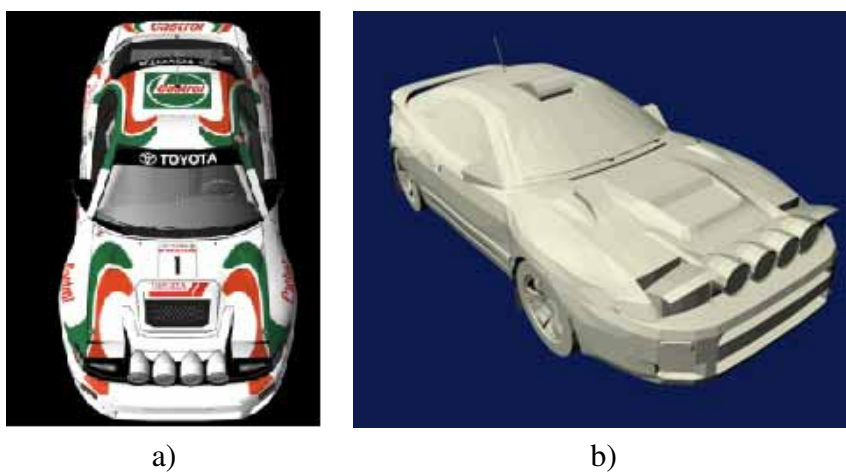


Figure 7.4: *GT Car synthetic model: a) one view; b) the 3D model.*



Figure 7.5: Santa Claus: pictures $V^i, i = 0 \dots 3$

7.1 View fusion experiments

Let us start from a simple case: the client holds four different views of the *Santa Claus* model corresponding to views separated by 90° . A high quality version of all the 4 images has been already transmitted to the client (the example pictures in Figure 7.5 shows the views at 0.8 bit per pixel corresponding to 78 KB for a 1024×768 image). To clarify the representation of the blending choices in the following discussion every image is also associated to a color, shown in the circle in the upper right corner of the picture.

The user asks for the rendered view from the viewpoint of Figure 7.6a. This view point does not correspond to any of the images available at client side. If we just try to stitch the warping of the two closest view in the image domain we obtain severe artefacts, as shown in Figure 7.6b. By moving to multi-resolution stitching we can obtain better results. Figure 7.6c and 7.6d shows the results obtained by multi-resolution stitching taking into account only the image distortion but not the geometry and lighting dependent terms, i.e. following equations (4.24), (4.25) and (4.26). Figure 7.6c shows the blending choices of this policy¹. Every sample on the shape of the Santa Claus is coloured with the color corresponding to the source view from which it has been taken. Multi-resolution stitching reduce the multiple views fusion artefacts; however, the Santa Claus model has a high geometry uncertainty due mostly to reconstruction errors (remember that it has been built using passive methods). This causes misalignments in the various view warpings that leads to visible artefacts. The complete distortion formulation of equation (4.36), including also the geometry-dependent terms, leads to the blending choices depicted in Figure 7.6e. The final rendered view is shown in Figure 7.6f. By comparing Figure 7.6c and Figure 7.6e it is possible to see that in the second one more triangles are taken from the purple view that is the most parallel to the user viewpoint. The resulting image quality is improved: the artefacts at the bottom of the cloak have disappeared and the face is also rendered more accurately.

Figure 7.7 shows another example of the effects of the geometry dependent term. This time we used the Frog model that is also a 3D reconstruction of a real object. As in the previous case four images are available at the same bit-rate, but this time

¹To clarify the figures, we select a single best stitching source for each triangle Δ_n^* , over all resolutions. The more general formulation allows for different decisions to be made in each resolution, but this would be very difficult to represent in the figures.

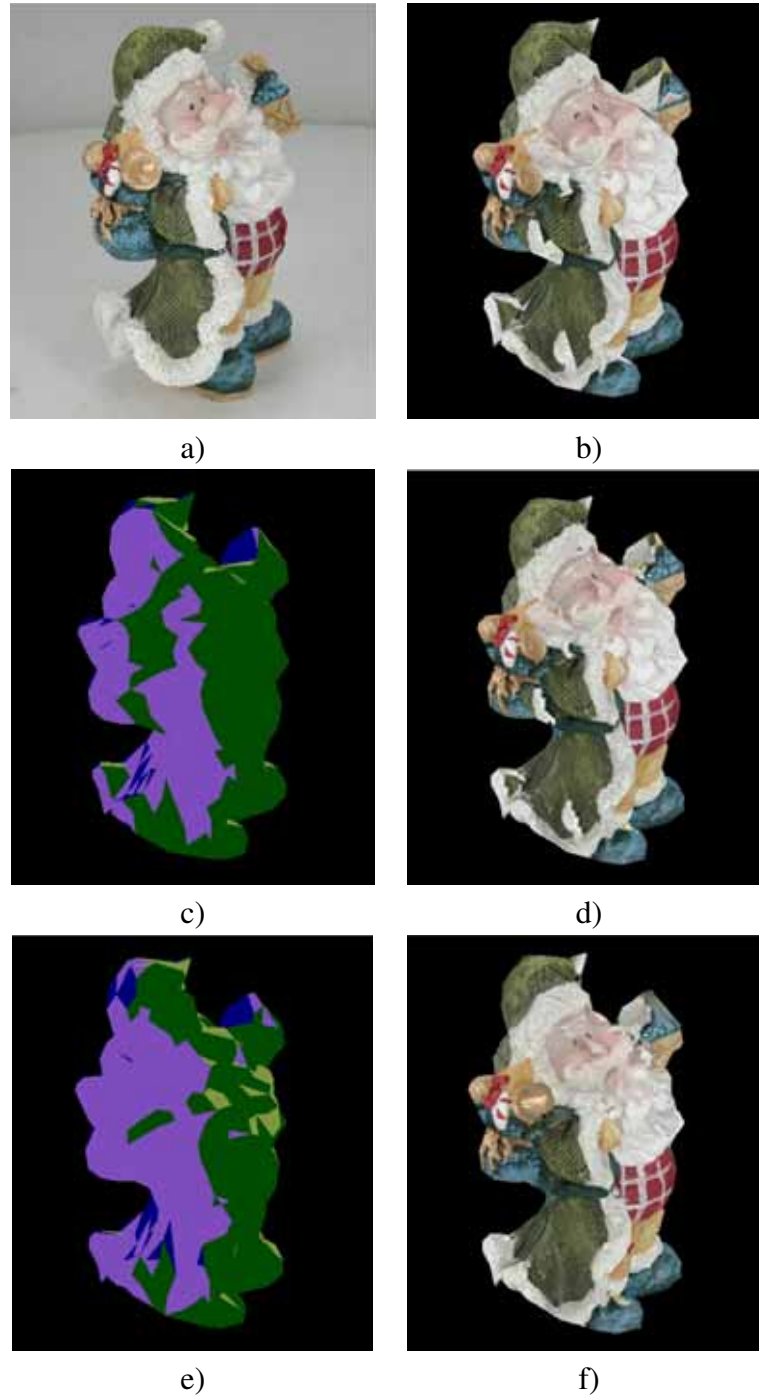


Figure 7.6: *Santa Claus*: a) reference image; b) rendering with image domain stitching; c) and d) triangle choice and rendering resulting from equation (4.24); e) and f) triangle choice and rendering resulting from equation (4.36).

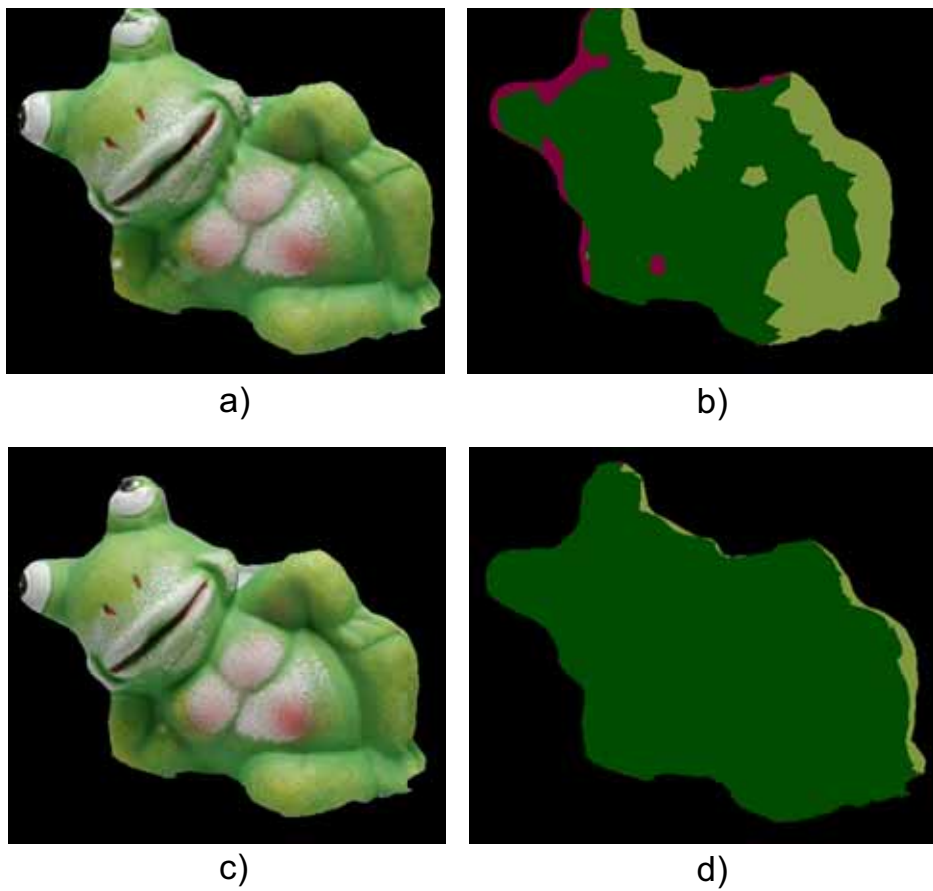


Figure 7.7: Frog: a) and b) rendering and triangle choice resulting from equation (4.24); c) and d) rendering and triangle choice resulting from equation (4.36).

one of them (represented by the dark green colour) is very close to the desired view. The stitching decisions based only on image compression distortion (using equation (4.24)) are shown in Figure 7.7b. The resulting rendering is the one of Figure 7.7a and shows some artefacts, for example on the right eye and in proximity of the neck. In particular by comparing Figures 7.7a and 7.7b it is possible to see that this second artefact lays exactly on the position corresponding to the transition between the light green and dark green views, and in fact is the typical artefact appearing on the edge between warpings that are not correctly aligned due to geometry inaccuracy. The high geometric error on the frog model increases the effect of the geometry term in the full distortion formulation of (4.36). This cause most of triangles to be taken from the view which is more parallel to the viewer. If we use the complete model we obtain the triangle selection shown in Figure 7.7d, where most of the samples are taken from the dark green view (almost all except for the ones close to the upper right edge, that correspond to the samples not visible in the dark green view). The resulting rendering, shown in Figure 7.7c is not affected by the artefacts on the neck and eye of the image of Figure 7.7a.

In most practical situation the user is randomly browsing the scene and when he requires a new rendering there are various views already available at client side, but while for some of them a good representation is available, for others only a few packet have been transmitted and the quality is poor. In the following example the user is looking at the *GT Car* model from the point of view of the images in the fourth column of Figure 7.8. Two views are available at client side, image V^1 , shown in the first column, and image V^2 , shown in the second column of the figure. At the beginning only a small amount of data is available for both the images, the low quality version (at 0.025 bpp) of the two images are denoted with $V^{1'}$ (shown in Fig. 7.8a) and $V^{2'}$ (Fig. 7.8b). Figure 7.8c shows the stitching decision, where the samples are taken almost in equal parts from $V^{1'}$ and $V^{2'}$ since the image distortions are similar. The final rendered image is shown in Figure 7.8d and as expected, having no good quality sources, the quality is not satisfactory. Then, the server sends more information for view V^2 , and a good quality (0.4 bpp) version of it becomes available, denoted with $V^{2''}$ and shown in Figure 7.8f. The distortion associated to the samples of V^2 is now smaller and many more samples in the final rendering are taken from it. The final rendered view, shown in Figure 7.8h, exhibits significantly improved quality, since most of the samples are taken from $V^{2''}$ and the data from the low quality source view $V^{1'}$ is used only in a few locations. Finally, if more data is received also for V^1 , and a high quality version of V^1 , $V^{1''}$ is available, the client computes a new set of stitching decisions, shown in Figure 7.8m , that again are balanced between the two views. The new choice leads to the rendered view shown in Figure 7.8n, with high quality details on the entire surface.

Another interesting example of the handling of images with different qualities is shown in Figure 7.9, 7.10 and 7.11. It refers to a very common browsing configuration, similar to the one depicted in Figure 6.4. The user has been looking at the scene from the viewpoint of view V^1 (shown in Figure 7.9a and associated to the light green color). Then he moves to a new viewpoint, corresponding to view V^2 (Figure 7.9b and 7.9c, dark green). At the beginning not enough data is available for V^2 and most samples are warped from V^1 , as it is possible to see from Figure 7.10a. Figure 7.11a shows

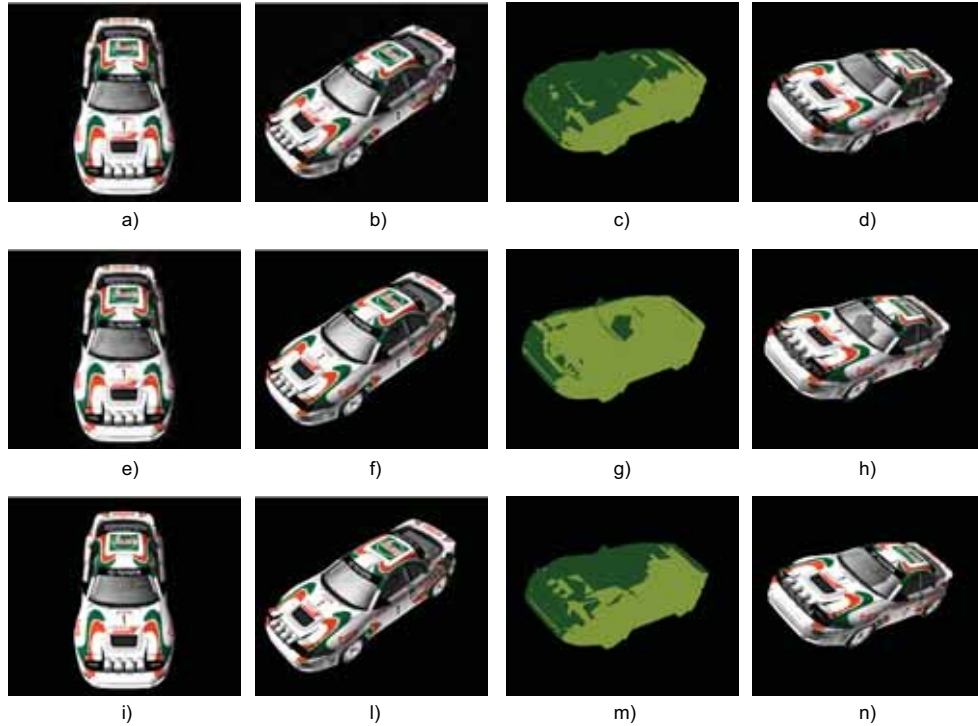


Figure 7.8: Rendering example with the GT car model: a) and b) $V^{1'}$ and $V^{2'}$, each at 0.025 bpp; c) and d) stitching decisions and rendered view based on $V^{1'}$ and $V^{2'}$; e) and f) $V^{1'}$ and $V^{2''}$ at 0.025 and 0.4 bpp, respectively; g) and h) stitching decisions and rendered view based on $V^{1'}$ and $V^{2''}$; i) and l) $V^{1''}$ and $V^{2''}$, each at 0.4 bpp; m) stitching decisions and rendered view based on $V^{1''}$ and $V^{2''}$.

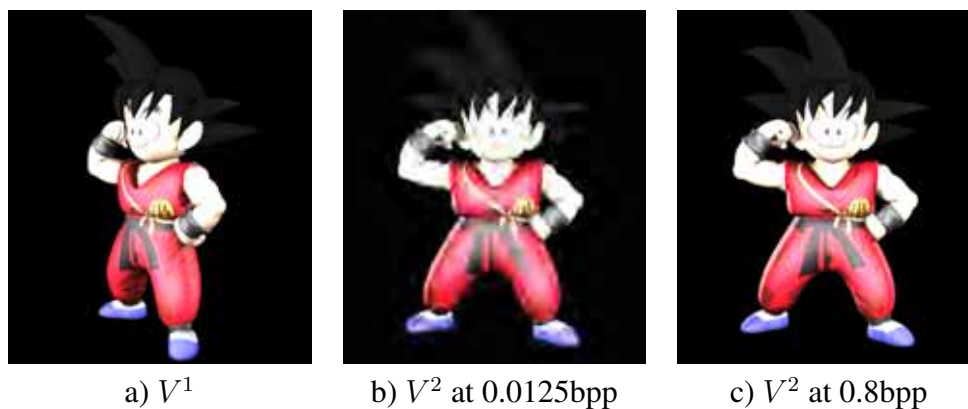
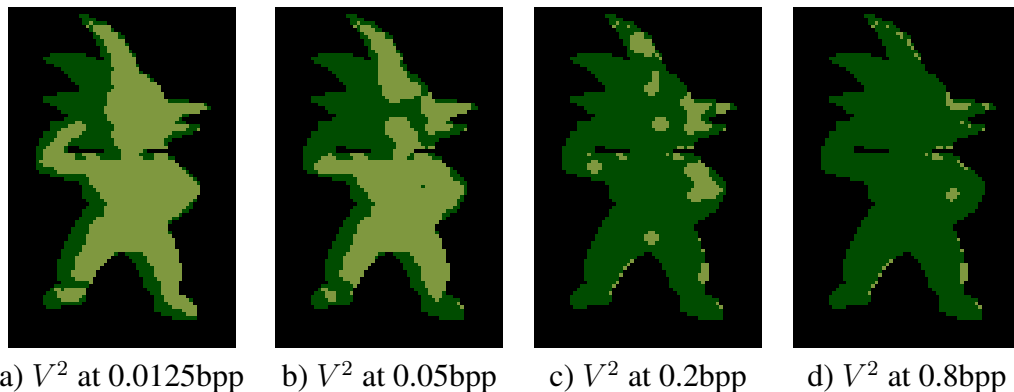


Figure 7.9: Views of the Goku model used in the experiment

the corresponding rendering, most of the image is warped from V^1 and has a good quality, only the leftmost samples that are not visible in V^1 are taken from the other view and show a poor quality. As soon as more data is received for V^2 it is distortion is reduced and more and more samples are taken from it. Figures 7.10b and 7.10c shows how the blending choices progressively moves to V^2 . The rendering corresponding to the choices in Fig. 7.10c has already a good quality on the whole image (see Figure 7.11b). Finally, when the complete description of V^2 is available as expected most samples are taken from it (Fig. 7.11d). This example shows clearly how the proposed system can handle a real-time browsing of the scene. When the user requires a new view the system firstly renders it by warping the previous ones and then progressively moves to the new one as more data becomes available for it. Two key aspects are worth being underlined. The first is that this approach allows to start rendering the required views without waiting for any information from the server, thus avoiding one of the biggest issues of the image transmission based systems described in Section 2.3. The second is that, as clearly shown in Figure 7.10, the switch from the prediction obtained from the already available view to the newly received one is a gradual process. The system does not just wait for the new image and then replace the prediction with it, but progressively takes more and more samples from the new view as soon as more data for it becomes available starting from the ones in which the estimation of the distortion in the warping is higher.



a) V^2 at 0.0125bpp b) V^2 at 0.05bpp c) V^2 at 0.2bpp d) V^2 at 0.8bpp

Figure 7.10: *Blending choices with V^1 at 0.8bpp and V^2 at different bitrates*

In all the previous examples we assumed that we were using the same blending choices for all the subbands (resolution levels) or at least that the choices were quite similar on the different subbands. In many cases the choices made in the different subbands are similar but sometimes they can be also quite different. Figure 7.12 shows a common browsing configuration with two views available at client side with the same quality. If we look at the scene just in the middle of the two views (from V^m), as expected, the choices are the same in all the subbands, roughly the left part is taken from V^1 and the right from V^2 . Let us have a look at what happens if instead we look at the scene from the viewpoint of V^* , that is between the two views, but much closer to V^2 .

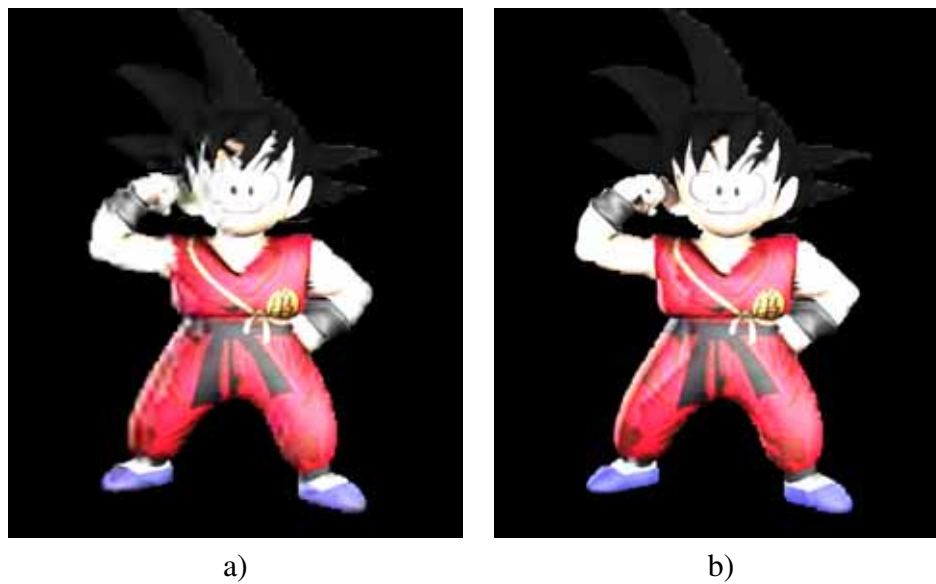


Figure 7.11: *Rendering: a) V^1 at 0.8bpp and V^2 at 0.0125bpp, b) V^1 and V^2 at 0.8bpp.*

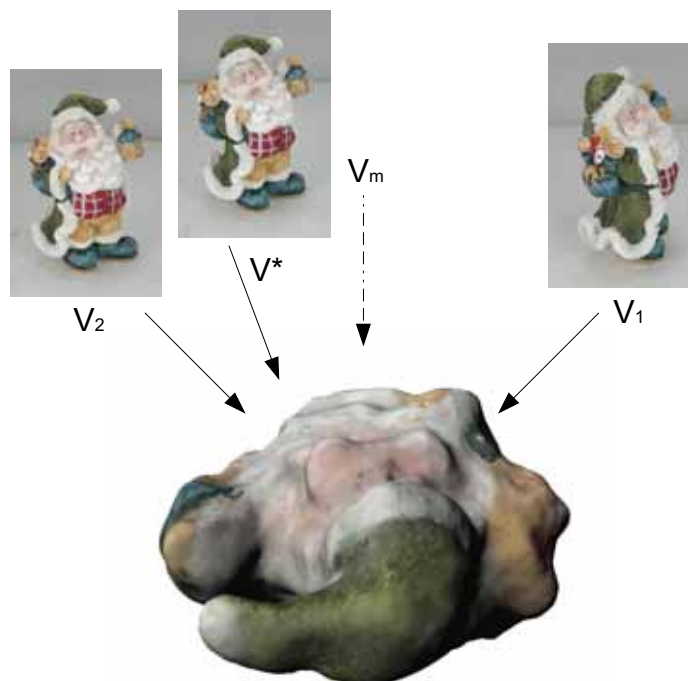


Figure 7.12: *Overview of the browsing configuration, the required view V^* is closer to V^2 .*

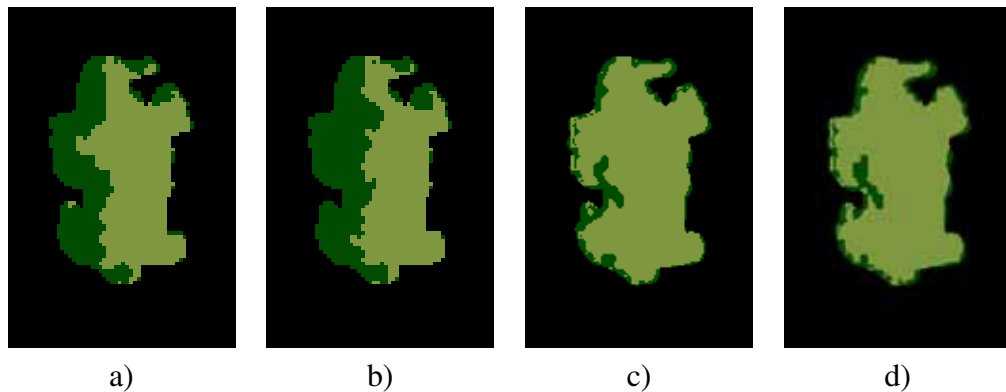


Figure 7.13: *Blending choices: a) LL at level 3, b) High frequencies at level 3, c) High frequencies at level 2, d) High frequencies at level 1*

Figure 7.13 shows the blending choices in the different subbands², the dark green samples are taken from V^1 and the light green ones from V^2 . As expected the client uses more samples from the closer view V^2 , but while in the low frequency bands (Fig. 7.13a and 7.13b) there are still many samples taken from the farther view (about one third), in the high frequency bands almost all the samples are taken from V^2 , as it is possible to see from Fig. 7.13c and 7.13d. This behaviour is justified by the expansive warping operator from V^1 to V^* . Small regions of V^1 are warped to larger ones in V^* . Low frequency information can be still warped from V^1 , while the highest resolution (frequency) components of V^* cannot be recovered at all from V^1 , since they depend upon high frequency components that does not exist in V^1 , as explained in Section 4.4.2. The higher distortion associated to the high frequency subbands in V^1 is mostly due to the hypothetical extra resolution subbands described in Section 4.4.2 and forces the system to use V^2 for these subbands.

7.2 Geometry synthesis experiments

In the proposed remote visualization scheme we decided to represent the geometry description as a set of depth maps. This allows a great flexibility in the interactive transmission of the data, as described in Chapter 5. Scalable compression features can be exploited to transmit only the information really relevant to the required rendering. At the same time the use of depth maps introduces additional issues at client side. During the interactive browsing different depth maps at different qualities representing overlapping regions of the object are available and the client must reconstruct from them a single geometry description. In this section some experimental results of this procedure are shown.

For the first experiment the *Goku* model is used. The dataset is made of 10 views V^i and 10 depth maps Z^i , eight of them are spaced 45° apart on a circle around the

² As explained in Section 4.2 we forced the system the system to made the same choices on the three high frequency subbands, so there is one map of the blending choices for the high frequencies at every resolution level and one for the LL band at the deepest level.

object, one is taken below and one above. Figure 3.6 shows the complete dataset at full quality, while Figure 7.14 shows a couple of them at different quality levels³.

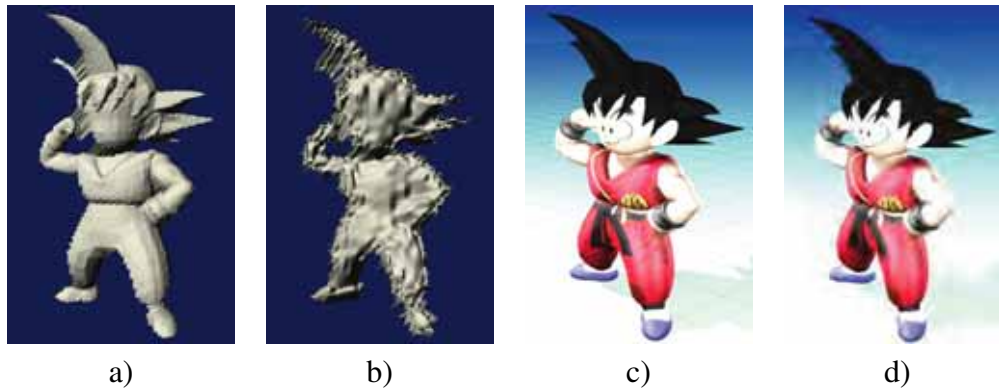


Figure 7.14: Example of Goku view and depth images at different quality levels: a) depth map Z^1 at 0.4 bpp b) depth map Z^1 at 0.025 bpp; c) view V^2 at 0.4 bpp; d) image V^2 at 0.025 bpp.

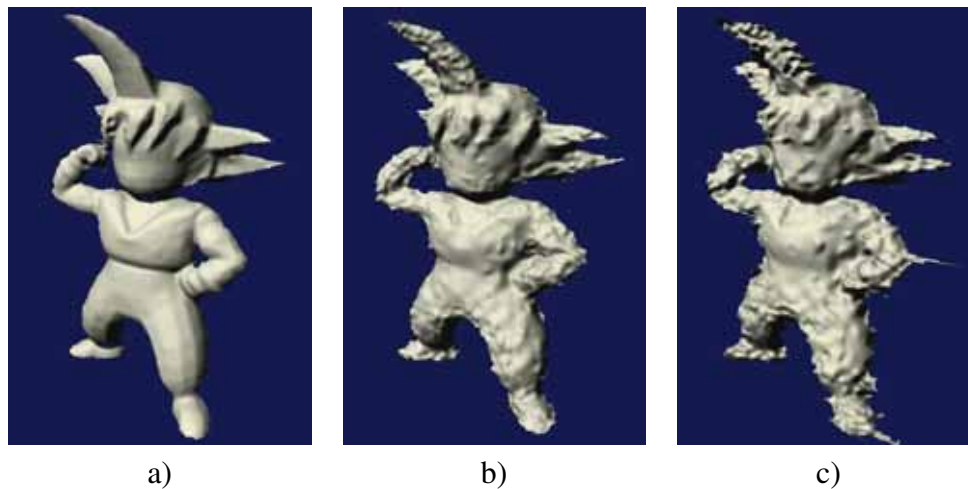


Figure 7.15: Synthesized depth map Z^* : a) from 0.4 bpp source maps; b) from 0.05 bpp source maps; and c) from 0.025 bpp depth maps.

The client should exploit the information from all the different depth maps to reconstruct the geometry. Figure 7.15 shows an example of the results of this process. Figure 7.15a shows the reconstructed depth map³ when all the Z^i are available at 0.4 bpp. Since the reconstructed Z^* takes contributions from multiple depth maps, its quality is generally higher than that of any of the individual source depth maps; when

³In order to represent more clearly the accuracy of the reconstructed shape, the images of the depth in this chapter shows the geometry reconstructed from the depth map and rendered without any texture. In Chapter 3 instead the depth information where shown directly with the brightness of the pixel corresponding to the depth value.

the single sources are available at full quality this gain is limited, but by combining more poor quality depth maps it is possible to obtain a reasonable quality one. This is clear if we compare Figure 7.14b with 7.15c. Both of them correspond to a bit-rate of 0.025 bpp, but while a single depth map at this bitrate is not enough for a good representation, by combining several poor quality depth maps we can obtain the better representation shown in Fig. 7.15c. That is because in many regions close to edges or occlusions where compression artefacts corrupt the depth values we can exploit the data from other depth maps where the same regions have a better representation (for example samples on the shape's edge in one Z^i can be in the middle of the object in other depth maps).

The depth maps fusion procedure is based on a minimum distortion criterion. Figure 7.16 shows how this framework is able to select the correct samples from the various sources by avoiding to take data with high distortion due to low bitrate compression or local expansion in the mapping from Z^i to Z^* . It refers to the case where all source depth maps are available at the same quality and the distortion-sensitive synthesis procedure is principally responding to local expansion in the mapping from different source depth maps to Z^* .

Figure 7.16a shows the result we would have if we just took for every sample the depth map $Z^{i \rightarrow *}$ closer to the viewpoint without any distortion considerations. Many small details are missing, specially around the hair of the cartoon figure; The geometry in Figure 7.16b, obtained using distortion information has a much better quality with less artefacts.



Figure 7.16: *Detail of the synthesized depth map : a) Without distortion information, b) Using distortion information*

How to optimally distribute the transmission resources between geometry and texture will be the subject of future work, however before concluding the discussion of the client view and geometry synthesis experiments it is interesting to have a look at how the two different type of data affect the final rendered image. Figure 7.17 shows the rendered images obtained using 8 views and 8 depth maps of the *Goku* model with different quality levels. Figure 7.17a shows the rendered view obtained using good quality views and depth maps (0.4 bpp for each V^i and Z^i). As expected the resulting



a)



b)



c)



d)

Figure 7.17: Rendering of V^* obtained with: a) 0.4 bpp depth maps images; b) 0.025 bpp depth maps and 0.4 bpp images; c) 0.4 bpp depth maps and 0.025 bpp images; and d) 0.025 bpp depth maps and images.

rendering is satisfactory. In Figure 7.17b the geometry information quality has been reduced and every Z^i is represented with only 0.025 bpp, while the images are still at 0.4bpp. The final rendered image is still fair even if some artefacts due to low quality geometry are visible. For example the Goku's face is stretched and he smiles more than he should. Figure 7.17c shows the opposite case, with good quality (0.4 bpp) geometry and highly compressed images (0.025 bpp). It corresponds to the same total amount of bytes of Figure 7.17b, but differently allocated. By comparing the two figures it is easy to see that in this case the results are worse and image compression artefacts are clearly visible. This suggest that, probably, it is better to allocate more bandwidth to the texture information. In Figure 7.17d the results with low quality images and geometry are shown. As expected the resulting rendering is not very satisfactory. However, by comparing it with Figure 7.17c the additional loss due to low quality geometry is limited. The comparison between the two figures in the upper row and the two in the lower suggests that the geometry impact is really relevant only when high quality images are already available. This example suggests that assuming the availability of coarse depth and view information at the client, a server should initially devote most of its bandwidth to refining the available view images. Geometry refinements might be sent only near the end of the progressive transmission. This conclusion agrees broadly with the results of other works in the field such as [1]. However the correct balance between geometry and texture depends on many different parameters and this simple example cannot give a satisfactory answer to such a complex problem.

7.3 Server policy experiments

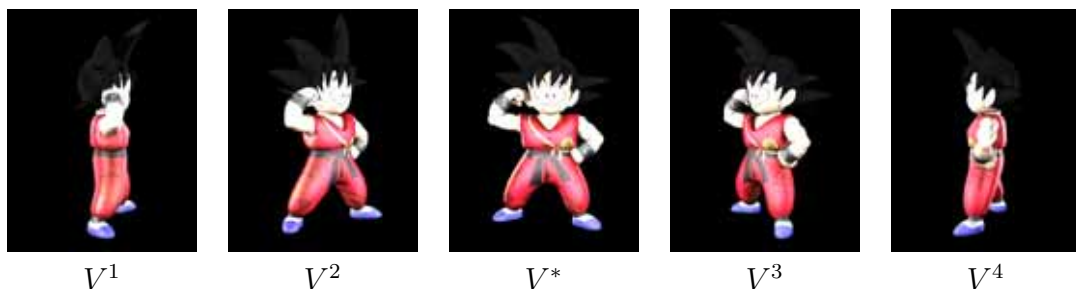


Figure 7.18: *Server policy experiment: available views V^i and required view V^**

To have an idea of how the server transmission system works let us start from a simple example. As shown in Figure 7.18, four different views V^1 , V^2 , V^3 and V^4 of the *Goku* model are available. The client has not still received any data from the server and the user requires view V^* , that is roughly in the middle between V^2 and V^3 . In this example the server is allowed to transmit up to 20 KB to the client. The plots of Figure 7.19 show how the server distributes the available transmission budget between the different views. As expected almost all the bandwidth is allocated to the two closer views V^2 and V^3 that receive 8,3 KB and 10,8 KB respectively. By looking at the sample selection choices at client side, it is possible to see that these two views are the

ones used to reconstruct the left and right part of the required rendering. The data is also quite balanced between the views on the left and right side as we can expect by looking at the pictures.

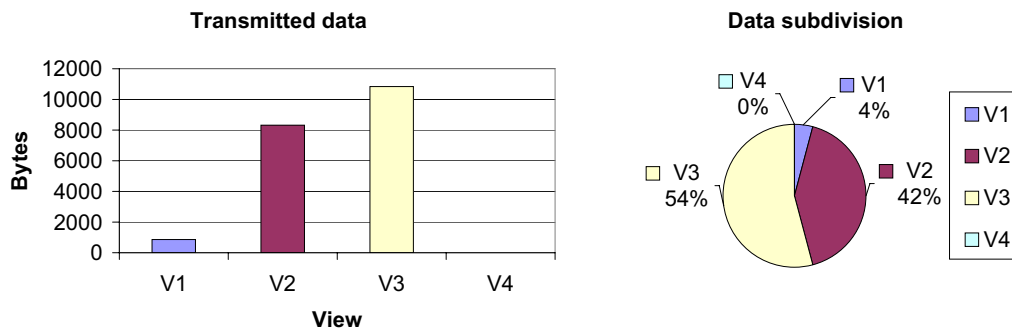


Figure 7.19: Allocation of the data between the different images

The previous example shows the allocation of the data between different images, but by using scalable compression techniques it is possible to select for transmission not only the more suitable images but also the elements in every single image that are more useful for the required views. The compressed datastreams corresponding to the various subbands of every image are divided into codeblocks corresponding to the different spatial regions. The server application is able to recognize which ones are more useful for the required rendering and to transmit more information for them. In the example of Figure 7.20a the user has required a close-up view of the face of the *Goku* cartoon character. Figure 7.20b shows the transmitted data for the front view relative to the three high frequency subbands at decomposition level 2. This subband is divided into 48 different codeblocks arranged on an 8 by 6 grid. Every codeblock corresponds to a particular spatial region as represented by the picture applied on the bottom of the plot. Every bar represents the amount of data transmitted for the codeblock that lays directly under it. From the plot it is clear that the server transmit information only for the codeblocks that are necessary for the required view and that the more a certain codeblocks affects the requested image the more information is transmitted for it.

It is easy to guess which will be the optimal solution when no data is available at client side. A more interesting problem is how the server should behave when some data has already been transmitted to the client and the user requires a new view of interest. The following example is a good starting point to understand the behaviour of the system in this case: at server side three images of the *Goku* 3D model are available. The first, denoted with V_f is taken in the front of the object, while the other two, V_l and V_r are taken at the left and right side of it (they roughly correspond to the three central views in Figure 7.18). The client has already received a small amount of data for the two images V_l and V_r . In the example it is around 2KB for each view, corresponding to 0.02 bpp. No data for view V_f is available at client side. The geometry transmission is still not included in the current server implementation, so we will assume that a good geometry description is already available at client side. In this example the server can

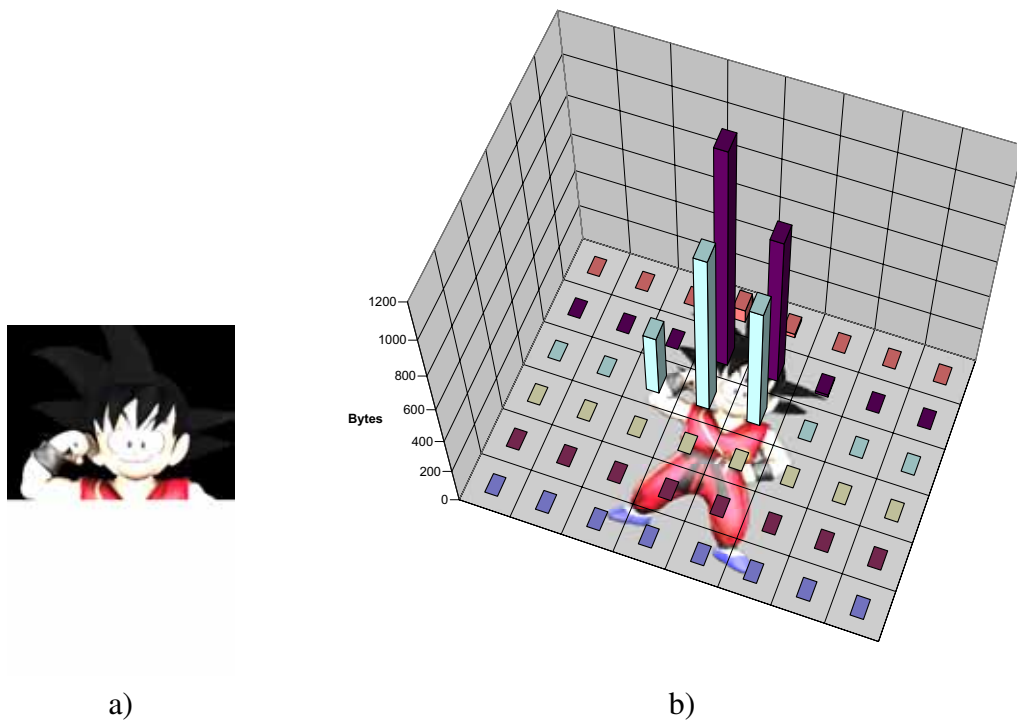


Figure 7.20: *Transmitted data in a close-up view: a) Required view, b) Transmitted data*

deliver up to 2048 bytes in each epoch.

The server is now facing the problem of how to distribute the available bandwidth between the three views. The plots of Figure 7.21 and 7.22 and show the experimental results obtained with the current prototype system. Figure 7.21 shows the subdivision of the data between the different views and compare the results with the simple transmission of V_f without exploiting the side views. The plot of 7.22 shows also the results of the reinforcing step alone, of the disruptive one and the combined solution for the first three epochs. To be more precise, the first group of columns of each epoch shows the bytes which would have been sent if only the enforcing decision was taken into account; the second one is about the disruptive decision only; the third one shows the combined solution. As it is possible to see from the first group of columns in Figure 7.22, in the first epoch reinforcing enhancements will force the transmission of data for the two side views because no data is available for the front one and so no samples are taken from it. If all the full quality images were available the optimal choice would be of course using just V_f that is perfectly aligned with the required rendering, and the disruptive enhancement will force the transmission of data only for that view. The plot confirms this intuition and in the second group of column most of the data is sent on V_f . From the third group (referring to the combined solution) it is possible to see that in the first two epochs disruptive enhancements turn larger than reinforcing ones, but the system will continue to send data also for the two side views. This is because in some precincts of V_f the additional data transmitted in this epoch will not decrease the distortion enough to compensate the policy switching penalty Φ_{π}^i . But as soon as

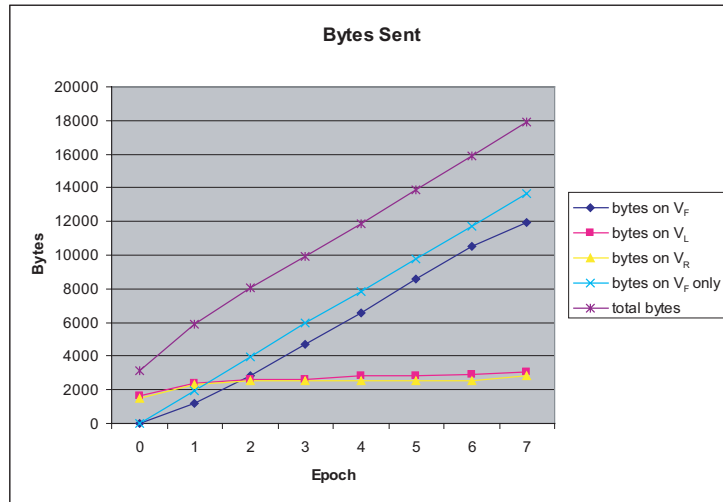


Figure 7.21: Bytes sent by the server in different epochs, shared among images.

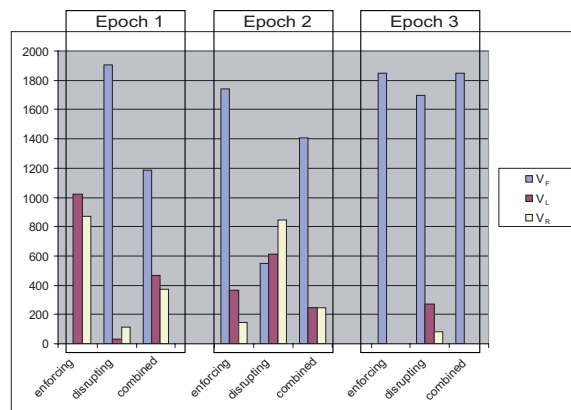


Figure 7.22: Bytes allocation between different views in the first three epochs.

more data is transmitted more and more samples are taken from the front view and from epoch 3 the policy completely switches to V_f , which receives almost the totality of the bytes spent by the server. For comparison purpose the plot of Figure 7.21 shows also the case when one only image V_f is sent (represented by the light blue line). By comparing the two curves it is clear that after some epochs the server delivers the same data in the two cases. The gap between the light blue and dark blue curves is due to the bytes which were sent at the beginning for V_l and V_r . Even if it seems that at the end the system will take a bit longer to receive the complete description, the bytes “wasted” at the beginning on the side views allow to obtain an acceptable rendering in epoch 1 and 2 even if not many bytes are available for V_f yet. This is clear from the renderings of the face in Figure 7.23, in the second epoch the results obtained with three views have a much higher quality.

Figure 7.23 shows a detail of the rendered images in the in different epochs. It compares the results obtained by just transmitting V_f (upper row) with the combined use of the three views V_l , V_r and V_f (in the lower row). It shows clearly that the biggest gain from the proposed approach can be obtained in the early epochs. In the first one it is possible to have an acceptable estimation of V_f even if no data has been transmitted for it. Also in the second epoch the visual result is much better with three views than with only one, since V_l and V_r provide useful information. From epoch 3, the server starts delivering almost all bytes for V_f in both cases, and more and more samples are taken from it. In the last epochs V_l and V_r become useless and are discarded. That example show clearly why the proposed approach is able to support an interactive browsing experience by continuously supporting the user navigation by exploiting the already transmitted data to render new views before enough information from closer views has been transmitted.

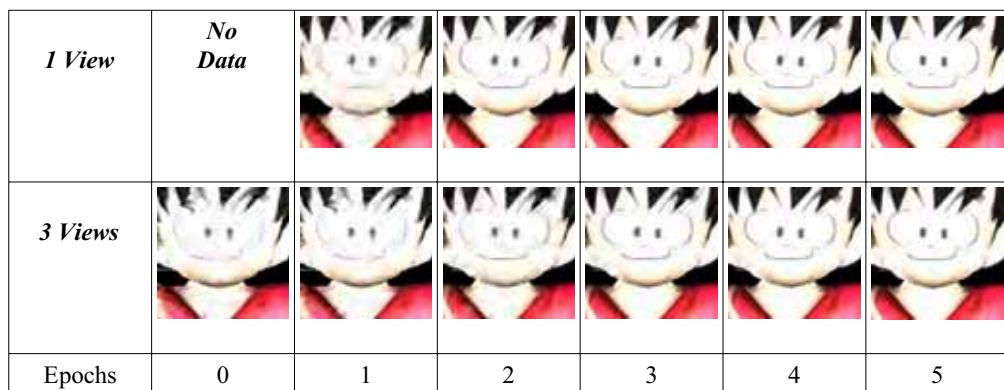


Figure 7.23: Detail of Goku in different epochs.

Before closing the experimental results section, a final remark needs to be done: in all the experiments the rendered images have been shown but no MSE or PSNR curves have been provided. That is because in this framework PSNR curves do not give a reasonable representation of the perceived image quality. The reason for this is that illumination issues and view warping artefacts (translational shifts) due to imperfections

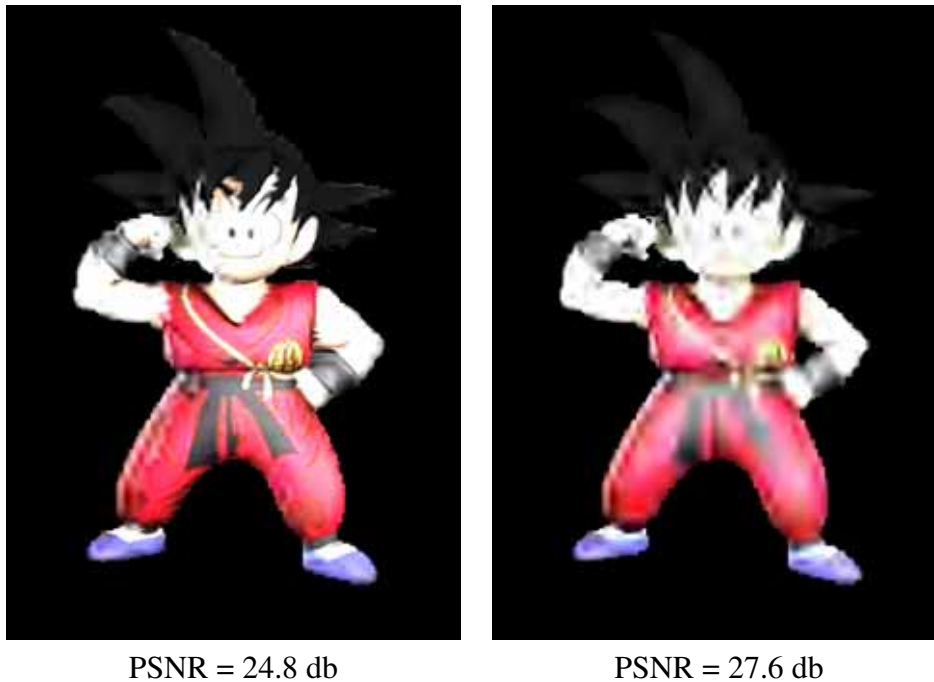


Figure 7.24: *Unreliability of the PSNR measures in view warping experiments*

in the geometry, introduce a large contribution to the MSE that does not correspond to the visual quality of the image, as described in [10]. Figure 7.24 shows two versions of the same view. The left one has been obtained by exploiting information from a nearby high quality view, while the right one is just a poor quality version of the required view. Even if by looking at the pictures the left one has clearly a better quality, the comparison with the original view gives a poor PSNR of just 24.8 db. The right view has a better PSNR of 27.6 db even if its quality is clearly worse.

7.4 Performance analysis

The target of the proposed framework is the construction of an interactive real-time browsing system for 3D scenes. To achieve this objective a very important requirement is to build an application that is able to perform all the required computations at several frames per second to follow the user interactive exploration of the scene. Even if the current implementation of the system still does not achieve the required performance, we will explain in this section why we think that a high performance implementation of the proposed approach is possible.

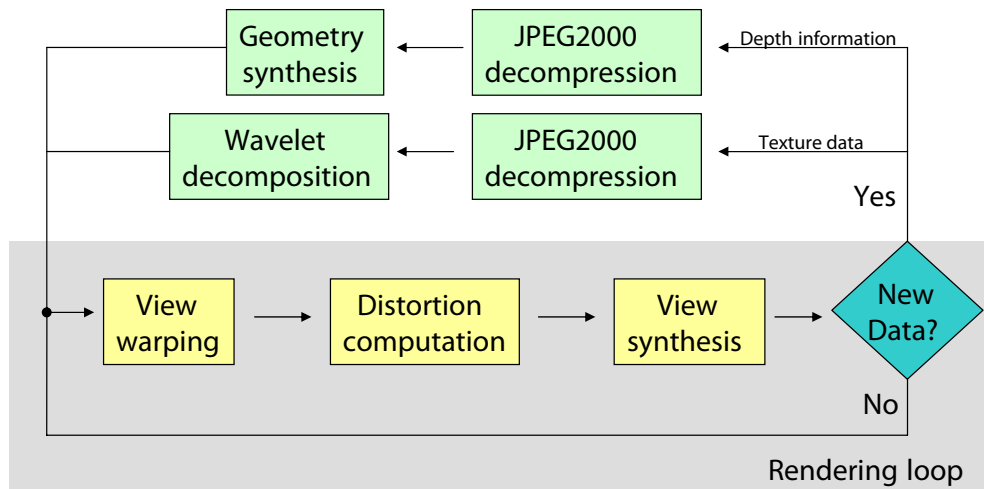


Figure 7.25: Overview of the client view synthesis procedure, only the steps in the shaded area must be repeated at every iteration of the rendering loop.

7.4.1 Client performance analysis

The rendering procedure at client side (shown in Figure 7.25) is basically split into three steps: the decompression and warping of the source views, the distortion computation and the synthesis of the final rendering.

The decompression of the source views needs to be performed only when new data is received for a particular view (and only for that view), not for every iteration of the rendering loop. As described in Section 3.4 the client application has a double caching system. It stores the decompressed data in the rendering cache (usually directly in the video card memory) and asks for the decompression only if the data is not present in the cache. The decompression system also works on a different thread, i.e. the JPIP module can fill up the cache while the main client module is performing the rendering of the required view⁴. In the current implementation it is based on the *Kakadu* JPEG2000 decoder that is an highly optimized application. For example it is able to decompress a 1024x768 pixel image in just 0,06 seconds on a standard 3Ghz desktop computer. Another useful feature, not exploited in the current implementation of the system, is that JPEG2000 is based on the same wavelet transform we used for the multi-resolution stitching, and it is possible to directly get the various wavelet sub-bands instead of the complete image, thus avoiding to perform the successive wavelet decomposition of the images. Anyway the wavelet decomposition of the source views is another operation that needs to be performed only after the decompression and not in every loop of the rendering procedure. JPEG2000 allows also to decompress the

⁴A synchronization system must be implemented to avoid that the rendering module reads invalid data from the cache while the decompression system is writing on it.

images at reduced resolution or only some regions of them, if we know that only a part (or a certain resolution level) of an image is needed, it is possible to exploit this functionality to reduce the computation time. However this opportunity is not exploited in the current version of the system.

In the warping stage all the available views must be warped to the required viewpoint and then decomposed using the wavelet transform into the resolution components. In Section 5.1 we explained how it is possible to reconstruct a triangular mesh from the available depth maps in order to exploit highly optimized texture mapping algorithms or the hardware acceleration of the graphic card to perform this step very fast. In fact it is not even necessary to warp all the information from the different views, but only the part of it that is visible from the required viewpoint.

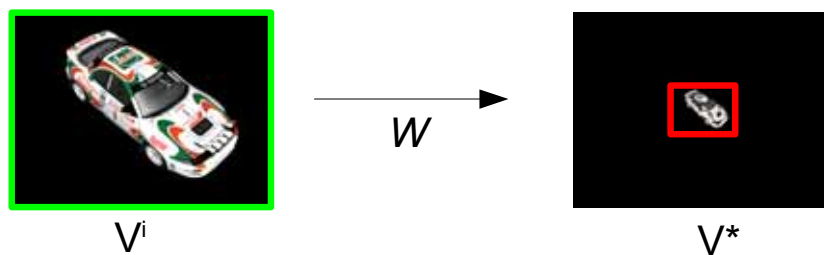


Figure 7.26: *Contractive warping: There is no need to decompress and warp V^i at full resolution.*

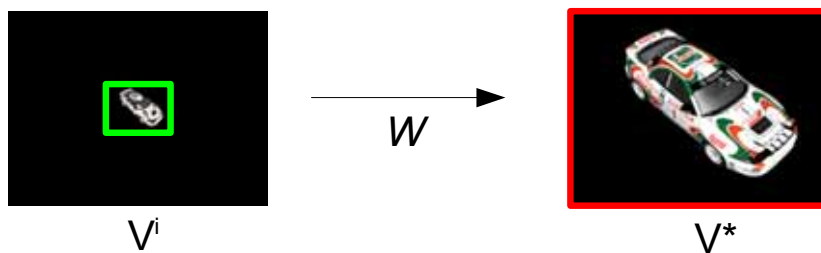


Figure 7.27: *Expansive warping: V^i has no information for the highest frequencies in V^* .*

Another important aspect is that there is no need to decompress the source view V^i at the highest available resolution if the warping operator which maps V^i to V^* is strongly contractive (see Figure 7.26), because the high frequency data associated to the highest resolutions is useless in the much smaller warped version of the image. Conversely, if the projection operator is strongly expansive (see Figure 7.27), the source image will not have information on the highest resolutions (a small region of the source image is mapped to a much larger one in the target rendering). In this case there is no need to generate the highest resolution components $R_d^{i \rightarrow *}$, since these should be close to 0. In other words it is possible to perform decompression and warping at reduced resolution. Since JPEG2000 offers the opportunity of performing the

decompression of an image limited to a given region and resolution of interest, both the decompression and warping stages can exploit this observations. More formally, the complexity of this stage is roughly proportional to:

$$\sum_i \min \{ \|\mathcal{R}^{i \rightarrow *}\|, \|\mathcal{R}^{* \rightarrow i}\| \} \quad (7.1)$$

where the first term $\|\mathcal{R}^{i \rightarrow *}\|$ represent the area of the portion of V^* which is visible from the source view V^i (expressed in number of pixels, the red square on V^* in the figures). When the warping operator is contractive this term is usually the smaller and determines the computational time. The second term $\|\mathcal{R}^{* \rightarrow i}\|$ denotes the number of pixels in V^i which are visible from V^* (the green square in the figures). This term is instead small for strongly expansive reprojection, in which $V^{i \rightarrow *}$ is synthesized only at reduced resolution.

Admittedly, the complexity expression provided above does not account for re-projections which contain both strongly expansive and strongly contractive elements. However, it does tell us that the complexity associated with synthesizing V^* from a large number of very close (and hence narrowly focused) source views is essentially the same as the complexity of synthesizing V^* from a small number of far away (and hence widely spread) source views, at least in the first stage. This means that the complexity depends only on the size of the reconstructed view V^* and the degree of overlap (or redundancy) between the relevant source views. In our present experimental implementation, we do not exploit the opportunity to perform reconstructions at reduced resolution, on which the above expression is based. To achieve even better performances it is also possible to directly discard all the views that are too far or with a too wide angle with the required viewing direction.

In our implementation, that does not exploit the opportunity to perform reconstructions at reduced resolution and is in no way optimized, currently the warping step consumes 2.7 seconds⁵ to reproject four source views at the resolution of 1024 x 768 onto a rendering window which also measures 1024 x 768. If instead we use an output window of 512 x 384 we can perform the same operation in 1.9 s. The fact that we are using just four source views derives from the exclusion of the farther views. For example the *GT Car* dataset has 8 views and if we exclude the ones on the back (that are useless for the current rendering), four views remain. These results looks quite unsatisfactory, specially considering the performances of modern graphic cards. In part the poor performances are due to the still unoptimized code, but another bottleneck is the transfer of the texture information from the graphics card memory to the RAM in order to use it later in the wavelet synthesis. There exist methods to perform the wavelet transform directly on the graphics hardware, for example the ones described in [56, 57]. By exploiting them it should be possible to avoid the continue transfer of the data in and out of the graphics card memory and to achieve higher performances.

⁵All the time measurements have been carried out using a standard desktop computer. The one used in these experiments is a 3,4 Ghz Pentium IV with 1 Gb of RAM and an ATI X300 video card.

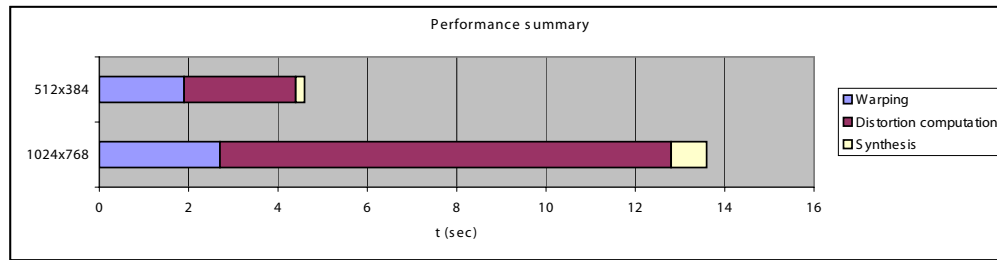


Figure 7.28: Overview of the computational time requirement of the different steps.

As it is possible to see from the plot of Figure 7.28 in the current implementation the second stage is the most critical from the computational point of view. However it is also the stage that has been less optimized and offers the wider set of opportunities to find a trade-off between the performances and the accuracy of the results. Basically in this step the distortion and energy fields associated with each source view subband are projected into the target resolution components using the weighting procedure described in Section 4.3. The same observations we made for the previous step are still valid: if the mapping is strongly contractive there is no need to include the contribution from high-frequency source view subbands, while under strongly expansive mappings no source view subband makes a significant contribution to the highest target resolution components. As in the previous case the overall complexity is dominated by the size of the required rendering and by the degree of overlap between relevant source views. Equation 7.1 is still valid also for this step: the total number of source views and their original sizes instead are not relevant to the computational complexity. A very important observation is that the smooth nature of the available subband distortion fields suggests that this stage could be performed on a subsampled version of the image. The computation of the distortion only on a subset of the image samples will of course lead to a small loss in the accuracy of the sample selection procedure but allows a great save in the computation time (the time is roughly proportional to the number of samples). A final option to improve the distortion computation times is to simplify or exclude some elements of the distortion model. For example if the object geometry is simple the impact of holes is limited and it is possible to avoid to use the holes-related term. However the subsampling offers a better trade-off between complexity and accuracy. Again, our current implementation does not exploit these various opportunities to reduce the computation time. After computing distortion information the blending weights are computed by just taking the source view with the minimum distortion for each sample (excluding the holes), a very simple step that has no relevant impact on the performances. Reprojection of distortion and energy fields and computation of blending weights currently consume 10.1 s to reproject four source views at a resolution of 1024 x 768 on a window of 1024 x 768, and 2.5 s on a window of 512 x 384. As expected the total time is proportional to the number of samples in the rendered view, and by subsampling the distortion field is possible to achieve very good performances (a 256 x 192 subsampled field can be computed in 0.7 s).

The regularization procedure is also based on simple calculations and very fast. On the subbands of the distortion field corresponding to a 1024x768 image every iteration takes about 13 ms. For example a regularization with 5 iterations (that in most cases is already satisfactory) can be performed in just 0.06 s and a 30 iterations one in just 0.32 s.

The final step is the synthesis of the stitched resolution components to form the required rendering. This step requires just to copy the data into the subbands and the computation of the wavelet transform. Highly optimized code based on the lifting strategy can be leveraged for this computation. In the current implementation the synthesis of the final rendering takes 0.8 s with four source views at a resolution of 1024 x 768 on a window with the same resolution and 0.2 s using a window with half the resolution (512 x 384). The synthesis is performed on the subbands of the final rendered image and depends only on the resolution of the output image, not on the resolution and number of the source ones. However the performances of other wavelet-based software, such as scalable video decoders suggest that much better performances can be achieved also in this step.

In the previous examples we assumed that geometry information were already available at client side. If the geometry representation is given as a set of depth maps it is also necessary to reconstruct the geometry from the received depth information. The depth reconstruction is based on the same distortion based framework we used for the images and all the complexity considerations previously made are still valid also for the depth maps fusion.

The analysis provided above suggests that a careful implementation could potentially operate at multiple frames per second, thus achieving the performances required by an interactive browsing application.

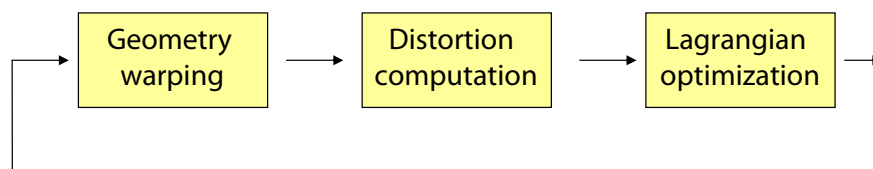


Figure 7.29: Overview of the server transmission policy computation.

7.4.2 Server performance analysis

Even if the distortion framework we used is the same, the computation procedure at server side is different from the client one. The server does not need to render any view, it just need to compute the distortion field and perform the optimization procedure. Another fundamental difference is that while the client needs to perform the rendering at interactive frame rates (several frames per second), the server just need to perform the computation once every epoch, that usually correspond to a much longer time interval (around 1 second). On the other side the server is also expected to be able

to handle several clients at the same time. That means that even if it is usually a more powerful computer compared to the client it must perform the computation in a much shorter time than the epoch size.

The server still need to compute the warping between the various source views and the required viewpoint: the distortion computation require the information on the warping both to compute the warping coefficients in the image distortion term and for the computation of the geometry-dependent term. The server instead does not need to reproject the texture information (it does not synthesize any image). A standard 3D warping is made of two steps, the geometry projection and the texture mapping, the server must perform only the first, that in the case of not too complex geometries is also the fastest. All the discussion about warping only the visible information and excluding the farther views made for the client remains valid also for the server.

The second step is the distortion field computation. As explained in Section 6.1 the server uses two different types of blending weights, one corresponding to the information transmitted to the client, used for the reinforcing enhancements and one corresponding to the full quality images, used to compute the disruptive enhancements. While the first set must be recomputed at every epoch the second one must be recomputed only when the required viewpoint changes. Every one of the two distortion computations works exactly in the same way as the client one. All the complexity consideration and the performance measurement made for the client can be extended to the server. The key difference is that now the server needs only to made choices on large compressed codeblocks instead of single samples and the subsampling of the distortion field can be exploited with only a very small loss. This means that is also possible to use higher sampling factors, working on blocks of 4x4 or even 8x8.

The final step at server side is instead the lagrangian optimization for the selection of the packets that need to be transmitted. The computational time required for this step depends not only on the resolution and number of views but also on compression parameters such as the size of the codeblocks, that determine the number of available packets. The current implementation of this procedure is still in an early stage and no optimization at all has been carried out.

Even if the actual performances of the client and server prototype application are not completely satisfactory, the discussion of the many possible optimization and approximation options leads to the firm belief that it is possible to build an optimized version of the system that is able to operate at interactive frame rates.

Chapter 8

Conclusions

This thesis presents a completely new approach to the remote visualization of three dimensional scenes that lays in the middle between image-based rendering techniques and a standard 3D visualization system. The proposed scheme is based on the representation of the scene as a set of views with the corresponding depth and allows to exploit scalable compression and transmission techniques already used for remote browsing of standard images in a three dimensional framework. In particular the scalability features of JPEG2000 and the JPIP interactive transmission protocol have been effectively exploited in order to transmit only the subset of the scene description really necessary for the required renderings. The great flexibility provided by these tools has permitted to overcome some of the drawbacks of other remote browsing schemes, such as the lack of random access to subsets of the data or the robustness to network congestions and latencies.

In particular the proposed architecture, following the approach used in JPIP image browsing, completely decouples the rendering at client side and the server transmission policy: the client can perform the rendering at any time exploiting the already received data (JPEG2000 allows to decompress the scene description from any subset of the compressed datastream) without waiting for the information from the server. This approach made the system particularly robust to network issues. The proposed scheme also allows the server to make his own decision on the information that needs to be transmitted, exploiting the knowledge on the data that have not been transmitted to the client.

An effective solution to the very challenging issue of how to combine a set of different views and depth maps into the rendering of a new, unknown view has been proposed. The proposed approach starts by building a set of predictions of the required view using 3D warping, then the problem of combining regions taken from the different warped views has been addressed using a multi-resolution stitching algorithm based on the wavelet transform. This procedure allows to preserve the high frequency content and at the same time to avoid visible discontinuities between the regions coming from different sources. The issue of selecting the best sources for the required rendering has been solved by introducing a completely new distortion estimation framework. The framework described in chapters 4 and 5 allows to estimate how the different sources of distortion, such as data compression, geometry uncertainty and lighting issues affect

the rendered image quality. It offers a straightforward way to select both the best source to be used for every sample in the final rendering and the information that needs to be transmitted from the server. Experimental tests has shown how this approach is able to correctly select the source view for the various samples in the final rendering in many different circumstances. The distortion-based stitching framework has been extended to the depth information and effectively used to combine the different depth maps together into a single three-dimensional representation of the scene.

The distortion framework has been used also to develop an optimization procedure to select the information that need to be transmitted. In particular a two steps optimization procedure has been developed. The server performs an *enhancing* step to decide which elements provide the better improvement to the rendering obtained using the current blending choices. At the same time a *disruptive* step is performed to decide which elements from better aligned but still not used views can be transmitted. Finally the server combines the two solutions together. Experimental results have shown how this approach, even theoretically sub-optimal, allows to allocate the available bandwidth between the many different contributions to the scalably compressed views in a very effective way.

A prototype of the system has been built and used to experimentally validate the proposed approach. The application is still under development, but the performance of the current prototype and a careful analysis of the computational requirements of the different steps suggest that a real-time browsing system based on the proposed approach is really feasible.

The proposed distortion framework allows to estimate the impact of the different elements of the texture and geometry description on the rendered views and opens the way to find a solution to the fundamental issue of the allocation of the available bandwidth between texture and geometry description. A more complete version of the server application is currently under development and will extend the optimization algorithm including also geometry information, thus offering an interesting answer to the fundamental issue of how to allocate the available bandwidth between texture and geometry description.

Concluding, a novel approach to interactive remote visualization of 3D scenes has been proposed. It extends the achievements obtained in the field of remote visualization of standard images to remote 3D browsing.

Bibliography

- [1] I. Cheng and A. Basu, “Reliability and judging fatigue reduction in 3d perceptual quality,” in *Proc. of 2nd Int. symposium on 3D Data Processing, Visualization and Transmission, 3DPVT2004*, IEEE, September 2004.
- [2] D. Tian and G. AlRegib, “Fqm: a fast quality measure for efficient transmission of textured 3d models,” in *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, (New York, NY, USA), pp. 684–691, ACM Press, 2004.
- [3] L. Balmelli, *Rate-distortion optimal mesh simplification for communications*. PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, 2001.
- [4] Y. Pan, I. Cheng, and A. Basu, “Quality metric for approximating subjective evaluation of 3-d objects,” *IEEE Transactions on Multimedia*, vol. 7, pp. 269–279, April 2005.
- [5] H. Hoppe, “Progressive meshes,” *Computer Graphics*, vol. 30, pp. 99–108, 1996.
- [6] S. Rusinkiewicz and M. Levoy, “QSplat: A multiresolution point rendering system for large meshes,” in *Siggraph 2000, Computer Graphics Proceedings* (K. Akeley, ed.), pp. 343–352, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [7] A. Khodakovsky, P. Schröder, and W. Sweldens, “Progressive geometry compression,” in *Siggraph 2000, Computer Graphics Proceedings* (K. Akeley, ed.), pp. 271–278, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [8] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image compression fundamentals, standards and practice*. Cambridge, United Kingdom: Kluwer Academic Publisher, 2002.
- [9] D. Taubman and R. Prandolini, “Architecture, philosophy and performance of jpeg: internet protocol standard for JPEG 2000,” in *Int. Symp. Visual Comm. and Image Proc.*, vol. 5150, pp. 649–663, IEEE, July 2003.
- [10] P. Zanuttigh, N. Brusco, D. Taubman, and G. Cortelazzo, “Greedy non-linear approximation of the plenoptic function for interactive transmission of 3d scenes,” in *International Conference of Image Processing, ICIP05*, (Genova, Italy), September 2005.

- [11] P. Zanuttigh, N. Brusco, D. Taubman, and G. Cortelazzo, "A novel framework for the interactive transmission of 3d scenes," *Signal Processing: Image Communication*, vol. 21, pp. 787–811, October 2006.
- [12] P. Ramanathan, M. Kalman, and B. Girod, "Rate-distortion optimized streaming of compressed light fields," in *Proc. of IEEE International Conference on Image Processing, 2003 (ICIP'03)*, vol. 3, 2003.
- [13] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH*, pp. 209–216, August 1997.
- [14] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," *M. Landy and J. A. Movshon, (eds) Computational Models of Visual Processing*, 1991.
- [15] H.-Y. Shum and S. B. Kang, "A review of image-based rendering techniques," in *IEEE/SPIE Visual Communications and Image Processing (VCIP)*, (Perth), pp. 2–13, June 2000.
- [16] M. Levoy and P. Hanrahan, "Light field rendering," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 31–42, ACM Press, 1996.
- [17] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 43–54, ACM Press, 1996.
- [18] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 299–306, ACM Press/Addison-Wesley Publishing Co., 1999.
- [19] S. E. Chen, "QuickTime VR — an image-based approach to virtual environment navigation," *Computer Graphics*, vol. 29, pp. 29–38, 1995.
- [20] G. M. C. F. Aru, P. Zanuttigh, "Visualization of panoramic images over the internet," in *Digital Heritage* (L. W. MacDonald, ed.), ch. 17, pp. 467–488, Berlin, Germany: Springer, May 2006.
- [21] S. E. Chen and L. Williams, "View interpolation for image synthesis," in *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 279–288, ACM Press, 1993.
- [22] S. M. Seitz and C. R. Dyer, "View morphing," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 21–30, ACM Press, 1996.

- [23] P. Rademacher and G. Bishop, "Multiple-center-of-projection images," in *Proc. ACM Annu. Computer Graphics Conf.*, (Orlando, FL), pp. 199–206, July 1998.
- [24] J. Shade, S. Gortler, L. wei He, and R. Szeliski, "Layered depth images," in *SIG-GRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 231–242, ACM Press, 1998.
- [25] L. McMillan, "Computing visibility without depth," tech. rep., University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.
- [26] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach," *Computer Graphics*, vol. 30, pp. 11–20, 1996.
- [27] P. E. Debevec, G. Borshukov, and Y. Yu, "Efficient view-dependent image-based rendering with projective texture-mapping," in *Proc. 9th Eurographics Rendering Workshop*, (Vienna, Austria), p. 14, June 1998.
- [28] G. Cortelazzo and P. Zanuttigh, "Predictive image compression for interactive remote visualization," in *Proceedings of the 3rd International Symposium on Image and Signal Processing and Analysis (ISPA2003)*, (Rome), pp. 168–173, 2003.
- [29] P. Zanuttigh, G. Michieletto, and G. M. Cortelazzo, "A predictive compression method for remote visualization of 3d models," in *Proceedings of International Workshop VLBV05*, (Cagliari, Italy), September 2005.
- [30] K. Engel, O. Sommer, and T. Ertl, "A framework for interactive hardware accelerated remote 3d-visualization," in *Proc. VisSym Joint Eurographics-IEEE TCVG Symposium on visualization*, pp. 167–177, 2000.
- [31] M. Feißt and A. Christ, "Dynamically optimised 3d (virtual reality) data transmission for mobile devices," in *Proceedings 3DPVT 2004*, pp. 270–274, 2004.
- [32] K. Tsoi and E. Gröller, "Adaptive visualization over the internet." TR-186-2-00-21, November 2000.
- [33] M. Bailey and C. Michaels, "VizWiz: a Java applet for interactive 3d scientific visualization on the web," in *Proceedings IEEE Visualization '97*, pp. 261–267, 1997.
- [34] M. Levoy, "Polygon-Assisted JPEG and MPEG compression of synthetic images," in *Proceedings of SIG-GRAPH 95*, pp. 21–28, 1995.
- [35] I. Yoon and U. Neumann, "Web-based remote rendering with IBRAC (image-based acceleration and compression)," in *Proceedings of EUROGRAPHICS 2000*, vol. 19, (Oxford (UK)), Blackell Publishers, 2000.

- [36] P. Bao and D. Gourley, "Real-time rendering of 3d scenes using subband 3d warping," *IEEE Transactions on Multimedia*, vol. 6, 2004.
- [37] W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3d warping," in *Proceedings of the 1997 symposium on Interactive 3D graphics*, 1997.
- [38] A. Watt, *3D Computer Graphics*. Reading (MA): Addison-Wesley, 2000.
- [39] G. Wolberg, *Digital Image Warping*. Los Alamitos (CA): IEEE Computer Society Press, 1990.
- [40] R. Krishnamurthy, B. B. Chai, H. Tao, and S. Sethuraman, "Compression and transmission of depth maps for image-based rendering," in *Proceedings IEEE Int. Conf. on Image Processing (ICIP01)*, 2001.
- [41] P. Bao, D. Gourlay, and Y. Li, "Context modeling based depth image compression for distributed virtual environment," in *Proceedings International Conference on Cyberworlds*, vol. 00, 2003.
- [42] B. B. Chai, S. Sethuraman, and H. S. Sawhney, "A depth map representation for real-time transmission and view-based rendering of a dynamic 3d scene," in *Proceedings First International Symposium on 3D Data Processing Visualization and Transmission*, (Padova (Italy)), pp. 107–114, June 2002.
- [43] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, "An overview of JPEG-2000," in *Data Compression Conference*, pp. 523–544, 2000.
- [44] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. Image Proc.*, vol. 9, pp. 1158–1170, July 2000.
- [45] ISO/IEC 15444-2, "Information technology – JPEG 2000 image coding system – Part 2: Extensions," 2002.
- [46] "<http://www.kakadusoftware.com>."
- [47] N. Brusco, *Wide-range reconstruction of 3d scenes with passive and active methods*. Università degli studi di Padova: PhD thesis, 2005.
- [48] N. Brusco, L. Ballan, and G. M. Cortelazzo, "Passive reconstruction of high quality textured 3d models of works of art," in *6th international symposium on virtual reality, archeology and cultural heritage VAST05*, (Pisa, Italy), November 2005.
- [49] "<http://www.opengl.org/resources/libraries/glut/>"
- [50] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, United Kingdom: Cambridge University Press, 2000.
- [51] J. D. Foley, A. V. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics Principles and Practice*. Addison-Wesley, 1995.

-
- [52] P. J. Burt and E. H. Adelson, “A Multiresolution Spline With Application to Image Mosaics,” *ACM Transactions on Graphics*, vol. 2, pp. 217–236, October 1983.
- [53] D. S. Taubman, “Localized distortion estimation from already compressed jpeg2000 images,” in *International Conference of Image Processing, ICIP06*, (Atlanta, USA), October 2006.
- [54] A. Secker and D. Taubman, “Highly scalable video compression with scalable motion coding,” *IEEE Transactions on Image Processing*, vol. 13, pp. 1029–1041, Aug 2004.
- [55] W. Sweldens, “The lifting scheme: A custom-design construction of biorthogonal wavelets,” *Appl. Comput. Harmon. Anal.*, vol. 3, no. 2, pp. 186–200, 1996.
- [56] J. Wang, T. T. Wong, P. A. Heng, and C. S. Leung, “Discrete wavelet transform on gpu,” in *Proceedings of ACM Workshop on General Purpose Computing on Graphics Processors*, (Los Angeles, USA), pp. C–41, August 2004.
- [57] M. Hopf and T. Ertl, “Hardware accelerated wavelet transformations,” in *Proc. of Symposium on Visualization VisSym '00. EG/IEEE TCVG*, May 2000.