



Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA



ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

MASTER THESIS

Design of a new 3D multi-view representation

Degree:

Telecommunications Engineering in Audivisual Systems

Author:

Pere FARRERO i ROGER

Under the supervision of:

Prof. Pascal FROSSARD, LTS 4, EPFL

Dr. Thomas MAUGEY, LTS 4, EPFL

Prof. Ferran MARQUÈS ACOSTA, TSC, UPC

Year:

Spring 2013

Abstract

3D video is increasing in the current State-of-the-Art of audio-visual technologies. This make researchers invest lot of resources to this field. Thus, new options in this scenario are appearing day by day. This technology has to deal with very different aspects like scene capture, data representation, compression, transmission, rendering or display. All these issues, despite their differences, have to be usually managed together and a complete system requires to be improved for each of its possible faces. The representation of the data is a main point to have in mind. Once the scene has been captured, it is important to decide in which way has to be coded all the data. One important issue is the resolution of the image. Resolution variability is very important for some applications and it is a good way of allowing some advantages as higher compression and a decrease in the computational load. Multi-resolution variability and adaptability are important facts that will be studied.

Acknowledgements

First of all, give a lot of thanks to Professor Pascal Frossard, Doctor Thomas Maugey and Professor Ferran Marquès for making this thesis possible. It has been a pleasure to see LTS4 from inside and to learn how a good team like this works. Thanks also for all the things that you have made me learn. Six months of work focused in one topic are translated into a lot of new knowledges.

Secondly, I would like to thank also all the Erasmus students with whom I have shared this experience in Lausanne. There is not a small number of people inside this group, but everybody have made my days in Lausanne a bit more special. However, special thanks to Dani Aldeguer, Ignasi Canales, Dimitrios Manolas, Bel Mias, Ana Olasolo, Xavier Riera and Pep Soldevila. I am thinking, of course, about all these lunches at Esplanade, dinners at FMEL, trips around Switzerland or even Italy and all the other big amount of unforgettable moments!

It is very important for me to thank Alba Puig the most for making me feel so special since I heard a voice saying "I am" and also for the never ending future that is waiting for us. Because everything has begun in Lausanne but it is going to take place wherever we go... together.

Finally, of course, hundreds of thanks to my parents and to my sister for making me who I am today. You have built my entire life and you still doing it, so thank you very much for making my possible dreams bigger and the unreachable ones smaller.

Contents

1	Introduction	1
2	State of the art	3
2.1	How to process the images? From 3D to 2D	4
2.1.1	3D Mesh	4
2.1.2	Ray Space Representation	5
2.1.3	Splitting the viewing possible positions	6
2.1.4	Layered Depth Images	7
2.2	2D Coding for 3D application	8
2.2.1	Discrete Cosine Transform	9
2.2.2	Discrete Wavelet Transform	10
3	A new adaptive representation. Design.	15
3.1	Formulation of the problem	15
3.1.1	Process	17
3.1.2	How to represent the data? Definition of the structures	19
3.1.3	Compression of the LDI image: which coefficients should be added or subtracted?	20
3.2	Needs of the system	22
3.3	Organization of the system	23
3.4	Compression of the tree	25
3.4.1	JPEG-2000	25
4	A new adaptive representation. Implementation.	27
4.1	Software	27
4.1.1	Classes	27
4.1.2	Functions	30
4.1.3	Libraries	31
4.2	Algorithm	32

4.2.1	Coder	32
4.2.2	Decoder	35
5	Experimental results	37
5.1	The state before this system	37
5.2	Multi-resolution Layered Depth Image	38
5.3	Redistributing the resolution	40
5.3.1	Fixed bit rate	42
5.4	The effects of the JPEG-2000 compression	44
5.5	The movement forwards and backwards	45
6	Conclusions and future work	49
6.1	Conclusions	49
6.2	Future steps	50
	Bibliography	51

Chapter 1

Introduction

In the last years, 3D video has suffered a high technological development in all its faces. Different versions of 3D video imaging TVs and cinemas can be today found in the market. But this market needs to be previously studied and created. This creation, thus, is done by academic researchers and companies I+D departments. And the fact of having these new technologies currently out in the market means that they have been studied for several years. The process is slow.

Nowadays, however, what research studies is the future trends of the market. The future products and technologies that are going to appear in an academic and commercial way. For us, one important field of the future trends is the Multi-view 3D. A concrete scene is captured by several cameras and the system allows to see it from any of the possible points of view. Thus, the 3D scene is captured and processed in order to allow the visualization from all the possible view and also different options like step forward or backward to the user in relation to the scene.

Why to design a new 3D multi-view representation system? Which data has to be code in order to optimize the trade off between quality of imagery and amount of information? It is an intuitive idea to say that not all the data is needed. There are going to be some views which are not going to be usually seen in front of others that will be more typical. Also it is not needed to relate each view with each other possible view of the scene. Hence, in order to improve the computational and stability results of the system, it is possible to define viewing segments of the image: the viewing segment. Each segment for separate can be coded and visualized without any needed data of the other segments that can complete the

scene. By reducing from the whole scene to a smaller segment, some other techniques can be developed, like for example the Layered Depth Image.

Other way to reduce the amount of information is by reducing the amount of bits related to a concrete view. This means to reduce in any of the possible ways the information of the image. One of this ways is the resolution. Not all the parts of the scene require the same resolution. For example, a flat wall has got a more uniform color despite the fact that a picture on this wall is less uniform and require more resolution for maintaining a quality level.

From this point, it is immediate to think in a resolution adaptive system. In other words, it is a system that can decrease the resolution in some parts of the image in order to increase it in any other part of the image.

This is very useful for example when applications include for example distance variations or zooming, that require a change in the size of the image. In this case, when increasing the size of each pixel, probably an increment on the resolution in some parts will help on maintain the quality in this closer view of the image.

The idea that holds this project is to try to improve the results in quality terms related to an amount of bits using an adaptive resolution system. We are going to see how the system has to be defined, designed, implemented and, finally, we are going to analyze the results in order to see its effectivity.

A design has been designed and it is explained how it works in detail in the next chapter. Even though, as a briefly explanation, it is based on an iterative algorithm that looks for the optimal distribution of the resolution for a fixed bit rate.

In the chapter of the experiments, it is explained how the results of the system are and its proofs of success. The builded idea is good and it is demonstrated in this mentioned chapter.

Chapter 2

State of the art

More than one way for coding multi-view image data have been proposed [11]. On the one hand, there is the *geometrically based representation* approach, which was an early way of modeling three-dimensional data. The idea is that it renders the 2D image from a composition of different geometrical figures. With that, all the effects of the light and colors are simulated. Despite of the big amount of research made in this field, geometrically based representation do not offer enough image quality [7].

On the other hand, with the improvement of the technology and the computational capability of the machines, another approach to the problem appeared [7]. In order to have a better image quality, for making the render look like a picture, the basis of the rendering should be images. With that, appeared the first *image based representation* techniques. The idea was to build a 3D representation departing from a set of 2D images. That new technique is based on having a set of pictures as origin and to get the definitive rendering by computing modifications on this set of images. Because of the realistic input, that methods led to a more perceptually realistic results than the geometrically based systems.

It was tested also a mixture of both kind of image representations as it is illustrated in [4]. One example of that is modeling the shapes with a geometrical information and then add the textures that are extracted from real pictures. However, the results were not satisfying because of the high complexity of the reality. This model of the reality leads to a non-good merging of the two techniques.

At its time, inside this image modeling tool, there are different ways of

acting also. Lots of investigations have been done over that new idea and it has been the focus for many of the researchers. In a general level, this thesis is englobed by *Image Based Representation* research.

By the moment, thus, there is defined a set of images. But what to do with that? There are lot of different ways of acting.

2.1 How to process the images? From 3D to 2D

It is important to define how to handle the scene. There are some different ways of doing it even though we are going to reach the conclusion that adapting the 3D scene to 2D are the best set of techniques [8], [1].

2.1.1 3D Mesh

By one side, we have the possibility of building a tridimensional mesh of the scene. That means that we need a complete set of images that gives us information about all the faces of the scene. With that, we can build a 3D representation that contains all the visible details of the scene from any position. This is, for sure, the best option in terms of intuitive representation because it is like an extension of a 2D image into a three-dimensional domain; we add a depth dimension varying just a bit the concept of pixel. From a 3D Mesh, by spotting the representation by any side, which in fact is projecting the data, an image is seen. And this is the main advantage of meshes, that the image is directly available from the data and do not require any kind of process.

However, the bad point is that for some applications a 3D mesh contains a lot of not needed data. This data is translated into not-coding-efficiency in terms of compression and time. Thus, the need of a selective coding of the 3D mesh appears. It is more interesting to try to code only the data that we need and that is useful for the final target than all the data including this big amount of useless information.

2.1.2 Ray Space Representation

As it comes explained in [14], Ray-Space is a representation of the light rays in the 3-dimensional space with photo-realistic results. It is assumed that all the rays radiating the same position are of the same color, what means that cameras in different positions receive the same colour from a particular point of the scene.

For describing a light ray, it is needed the origin position (x, y, z) and the direction of propagation, represented by the angles $(\Theta$ and $\Phi)$. It can be also simplified assuming that the rays go through a focal plane, with which we can obviate the coordinate z that corresponds to the current plane.

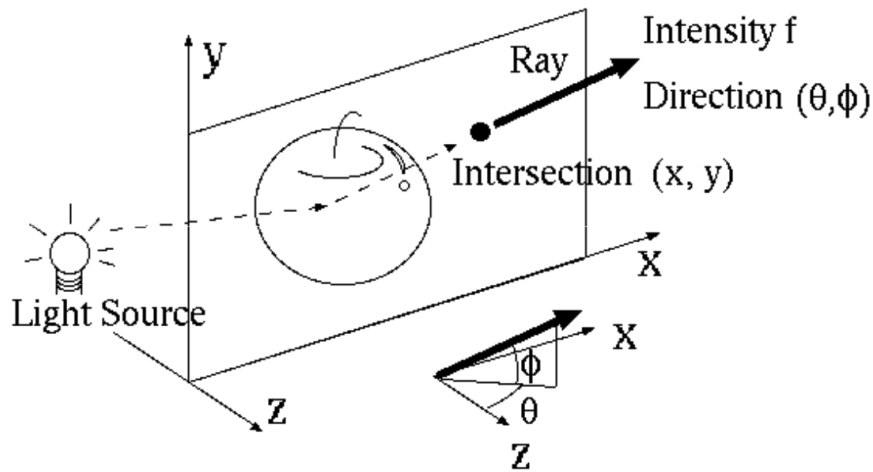


Figure 2.1: Ray description

With the information of the light traces, we are building up a parametrization of the light rays. With the whole set of the cameras, we can construct a complete representation of the scene.

The acquisition is not exhaustive of all the points-of-view because of cameras and economical reasons. However, we can fill all the holes where there are no cameras using interpolation. With that, we complete all the viewable points-of-view imaging.

By cutting this cube, each section with different angulation returns different interaction between figures and different point-of-view projections. With that, the desired position of the user defines the angle with which the cube has to be cut and then, that gives back the photo-realistic representation of the scene for that concrete position.

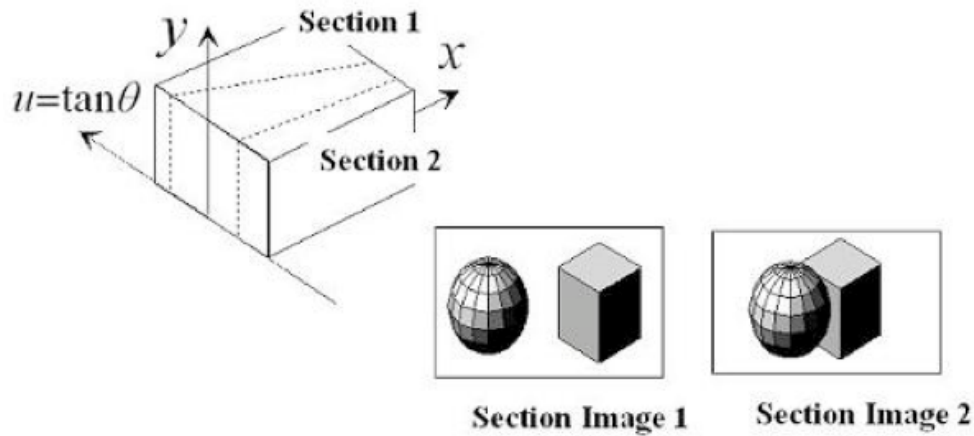


Figure 2.2: Representation of the cube and its projection cuts

2.1.3 Splitting the viewing possible positions

One assumption that casts doubt on the previous representation systems is that the user is not probably going to use all the possible views in the space. Thus, we define a viewer segment, which is the set of possible positions where the observer can be located. Usually, that is a continuous path around a part of the scene instead of around all the scene. That allows the representation system just to code a small part of the data. One idea is, for example, to define a basic image like the view from the center of the viewer segment and to code it. To move from the point where this image is taken, we just need to code those amount of data which is not possible to see from the basic image. It is important the fact that all the other information is useless so we don't code it. With that, we have the basic image and the hidden but necessary objects coded.

We can now, intuitively with some ease, render the image in any of the points of the viewer segment. We just need to project the main image and fill the holes that appear with the back objects' data.

There are some ways, at the same time, to code the depth based information. One of the important points of that is the depth map. It is a representation that gives the depth for each position of the image. With that, it is easy to know how deep is an object or the spacial relations between different objects at the rendering time.

In any case, however, usually is needed a set of cameras. With only one

camera, it is not possible to represent many viewer positions because of the occlusion of objects.

Lots of ways for representing that 3D data have been proposed. One typical is to use the image as a classical 2D picture and Time-of-Flight Camera for getting the depth. With that, the system is able to get a depth mapping for each pixel of the image. That is useful for some applications but especially for recognition.

On the other hand, it is possible to get a group of images from different cameras of the same scene. With the possession of some points of view, it is possible to compute the hidden objects, the distances between objects and, also, to get the depth of the objects. For that reason, it is needed a good system that matches all the objects in the different images and works out the geometry of the scene.

2.1.4 Layered Depth Images

The previous ideas of splitting the viewer segment in order to reduce the amount of data needed to code, match perfectly with the idea of Layered Depth Image (LDI). That representation needs some specifications. First of all, a viewer segment has to be defined. The results of the algorithm are better by splitting the viewer range in some independent segments. With that, we define a specific set of continuous positions where the viewer can be watching the scene from. That technique is well defined in [13], [12] and [2]. The next idea is to get a main image, which could be defined as the central position of the viewer segment, which is the first layer of the scene. Now, the back objects that are hidden have to be represented. The next step is to code all the objects that are possible to be seen from any viewpoint and the way of code them is in layers. Each layer represents a different number of objects that hide the current object of the specific layer.

For example, the main view builds the layer one. That layer will probably include some objects that are in the foreground and some that are in the background. Thus, the objects of the foreground are hiding other objects. That hidden object can be another thing of the foreground or of the background. It is represented in the second layer. If it is foreground, the object hidden behind is coded in the next layer. The idea is to iterate

this process until have all the possible viewable objects covered.

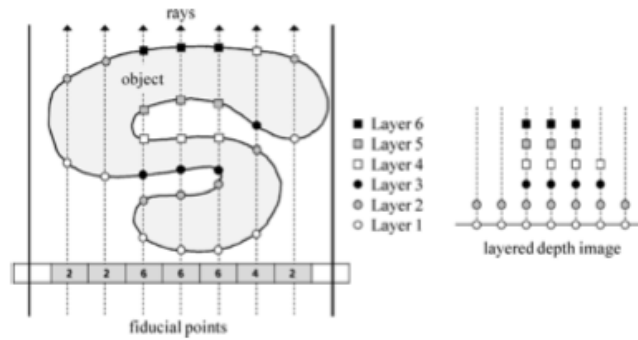


Figure 2.3: Layered Depth Representation with the different layers

Nevertheless, that gives us the relation between objects but not the depth. Then, for each object it is computed also the depth and it is coded in each pixel of each layer. That means that not all the layers are flat in the depth dimension. The depth of each layer is particular and can have a very different range of values.

Knowing the depth, the idea is that you can make projections of the objects that are already existing in the current image that the user is viewing. That projections does not cover all the image space so, knowing the back layers and its objects, the holes can be filled [2], [3].

2.2 2D Coding for 3D application

Since the moment of the jump from 3D data to a set of 2D it is possible to find a coding solution within the two-dimensions tools. A very important tool for coding nowadays is the Discrete Cosine Transform (DCT). This DCT allows to compress very efficiently the image with a relative low loss of quality. Even though, there is another technique that we are going to prefer. The Discrete Wavelet Transform compress also very efficiently the data. In addition, the DWT has got some properties related with the resolution that are very interesting for the current project.

2.2.1 Discrete Cosine Transform

The Discrete Cosine Transformation is probably the most commonly used approximation of the Fourier Transform in signal processing. It is a linear transform with a set of cosinoidal functions as bases instead of complex exponentials. That is mostly an advantage because it allows the system to work only with real numbers instead of complex. The transform, so, represents the signal in terms of a lineal combination of cosinus of all the possible different frequencies.



Figure 2.4: 8x8 pixels Discrete Cosinus Transform basis

Another good point for DCT is that encloses the big part of the big coefficients near the origin of the transformed domain. That is very useful for data compression because it is very easy to discard lot of high frequencies low-valued coefficients in order to reduce the amount of data to code. Obviously, this is mainly useful for lossy compression and because of this fact, the DCT is the main idea of JPEG compression.

The transformation is not applied to the whole image because the big amount of different frequency variations in the complete picture would not allow a good results at putting together the highest energy coef-

ficients. For that reason, the image is segmented in small blocks of typically 8x8 or 16x16 pixels. Each of these blocks can be considered almost as stationary data, so the transformation is applied to each of those blocks.

One of the main pros is the ease in vary the quality of the resulting image. That quality variation allows to work with a very flexible data bit rate, which is very important at transmission when there is not for sure a stable / high quality channel.


One of the contras of DCT is that it does not allow multi-resolution variation. The addition or subtraction of coefficients of the transform just improve or degrade the quality of the image, but not its resolution. For that reason it is interesting to analyze another tools like wavelet applications.

2.2.2 Discrete Wavelet Transform


Meanwhile DCT is just a frequency analysis of the signal, it is possible to define a multiresolutional analysis of the frequency domain of the signal. One way is the Discrete Wavelet Transform and it is implemented as a meeting point between transformations and filter-bank analysis [6]. The function of the basis of the transformation are defined in 2.5:

$$\psi_{k,n_0}[n] = \frac{1}{\sqrt{2^k}} \psi\left(\frac{n-2^k n_0}{2^k}\right)$$

High scale k



Low scale k



- The **DWT** can be expressed as:

$$y[k, n_0] = \sum_{n=-\infty}^{\infty} x[n] \frac{1}{\sqrt{2^k}} \psi\left(\frac{n-2^k n_0}{2^k}\right)$$

Figure 2.5: Discrete Wavelet Transform formulation

Depending of the value of k , which is the index of the transformed domain, the basis function is scaled in order to change the resolution of the analysis.

However, it is easier to understand the DWT if we talk about a filter bank. The analysis only consists on a set of high and low pass filters

placed with decimators between them. That decimation is the step that makes possible, in the implementation, the multiresolution analysis.

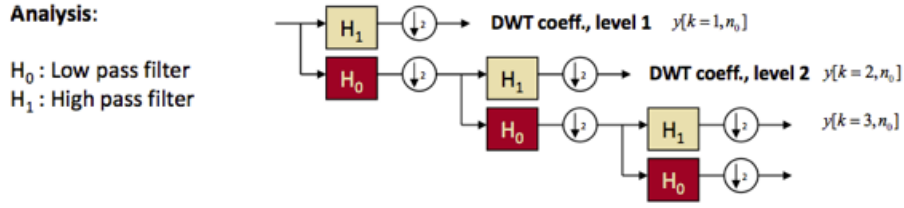


Figure 2.6: Discrete Wavelet Transform analysis scheme

We can see the scheme of the analysis process in the figure 5.1. Each output of the high pass filter and the last one of the low pass are the DWT coefficients. After that analysis, for reconstructing the initial signal only is needed to do the opposite process. The treatment is only to interpolate and apply the reconstruction filter at each level as it can be seen in the figure 2.7.

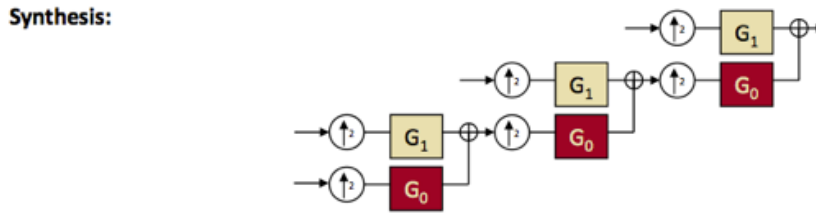


Figure 2.7: Discrete Wavelet Transform synthesis scheme

Even though one of the most exciting things about the transformation using wavelets is the relation between the filters. The only thing needed about the filters is perfect reconstruction, which is not usually difficult to find. It is not needed a perfect filter. The illustration of this fact is done in the figures 2.8 and 2.9

In the analysis, after the filtering and the decimation, because of the non-requirement of ideal filters, probably there will be superposition of alias, ergo, aliasing. However, the only fact of perfect reconstruction of the filters leads the system to make alias disappear as it is shown in the figure 2.9.

The extension of DWT from one dimension to two dimensions is immediate, the only thing to do is to apply that transformation in one direction

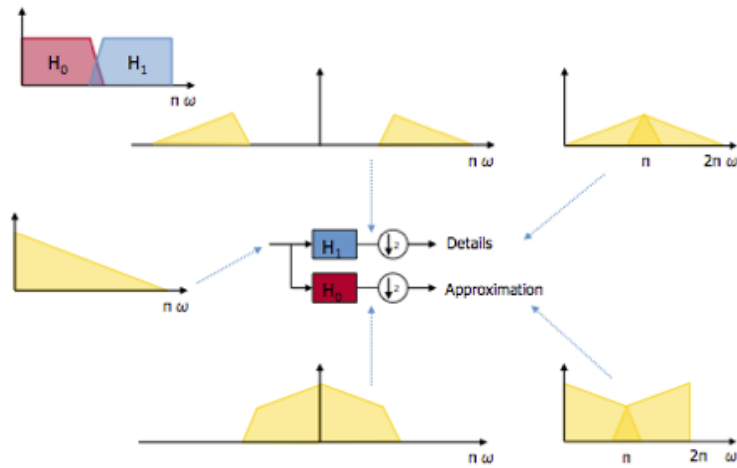


Figure 2.8: Discrete Wavelet Transform analysis process

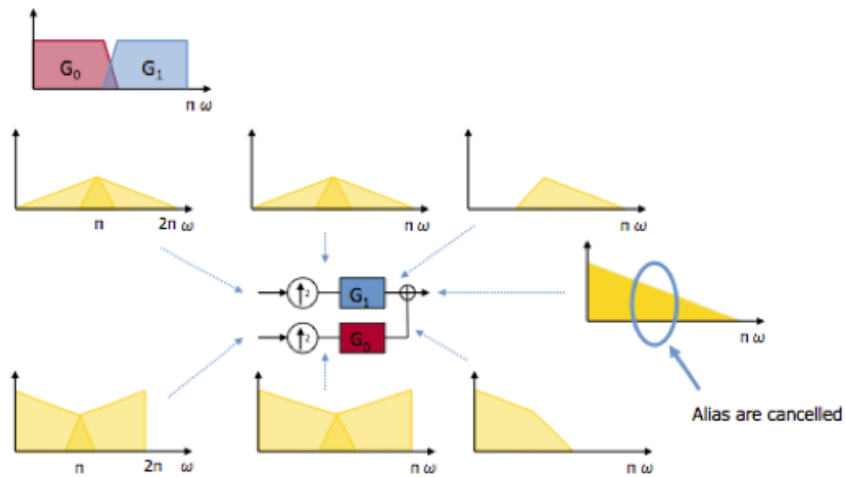


Figure 2.9: Discrete Wavelet Transform synthesis process

and then in the other. With that, each level has got 4 coefficients as a combination of the four filters (high and low pass horizontal and high and low pass vertical).

That makes DWT a not very difficult to implement system and it gives more advantages than DCT in some particular applications. It is possible to define and choose the wavelet basis function. In the filter bank, this corresponds to just define the filter. That makes one kind of analysis or another to the signal.

The number of coefficients that results from the DW transformation is the same than the original amount of data. Thus, the DWT transform also keep the number of samples of the signal domain in the transformed domain. It is clearly showed in the figure 2.10.

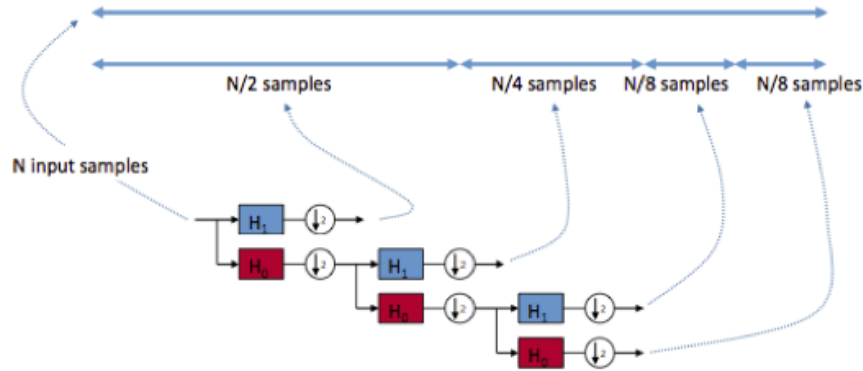


Figure 2.10: The DWT has got the same size than the input

One of the most interesting points of DWT, which has been exploited in JPEG-2000 [5], [15], is the high versatility of the transformation with scalabilities. One to be considered is, as opposite of DCT, the easily scalable resolution. The number of levels of the DWT gives a multi-resolutional representation of the image. If it is not needed the entire resolution for the application, it is possible to not decode all the levels and give less resolution to a particular point. As the opposite, it is possible to increment the resolution by adding coefficients of the DWT if it is interesting for the application [9]. That gives a very useful tool, but it will be seen with more attention later in this thesis.

The interesting point for this project is the fact of allowing multi-resolutionality in some interest regions [10]. It is possible by using DWT in the image and controlling the coefficients for each place of the image. It is actually something similar to the analogy of DCT in JPEG but with DWT. That makes possible to increase or decrease the resolution to a set of points of the image, which is one of the targets of this thesis.

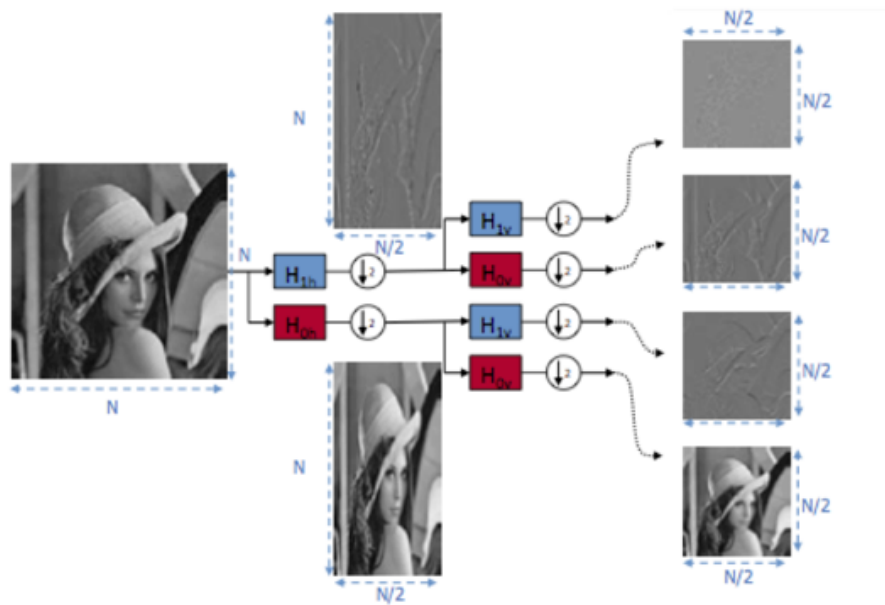


Figure 2.11: Discrete Wavelet Transform results

Chapter 3

A new adaptive representation. Design.

After thinking about how to build the system, it is time to design its implementation. And this is what is exposed in this chapter. We are going to see how the system works in detail and, also, to analyze the functions and the algorithm used.

3.1 Formulation of the problem

Once it has been analyzed the State of the Art, it is possible to build a precise Formulation of the Problem. In other words, we can now explain the main purpose of the project and, step by step, all the parts and actions planned as well as all the problems that are imaginable to appear.

The idea of the project is to design a representation system for multi-view 3D data improving some of the current system features. For that, we have set some targets. One is to code the minimum of data possible in order to compress the amount of information to transmit offering the best quality possible. That means to keep the lowest distortion with the lowest number of bits possible. Other target is to allow a multi-resolution system taking into account different Regions of Interest (ROI). We do not want a static-resolution system, but a variable one. There are some parts of the image that have more important details than some others and, for that, it is important the variability of the resolution for a specific object. The idea is that if we are looking inside a museum, for example,

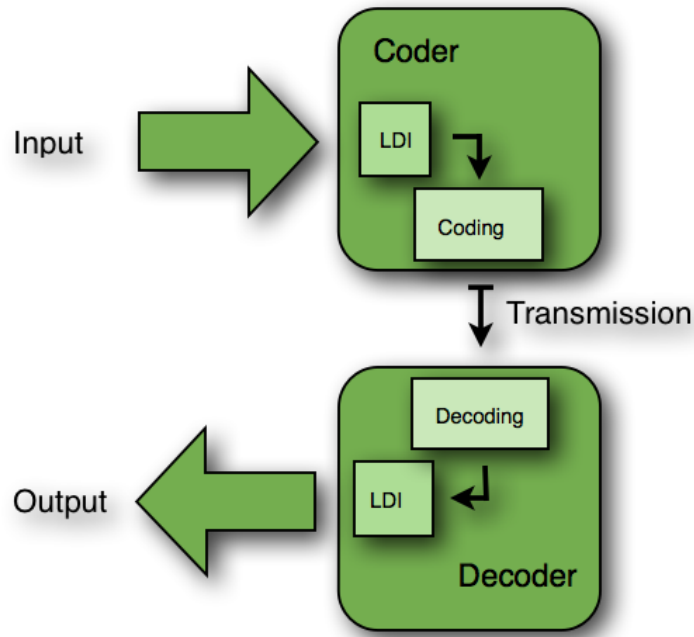


Figure 3.1: Scheme of the system

it is interesting to add more resolution to the part of the picture that contains a painting that to a flat white wall. It is for that reason that the system proposed includes a layered resolution stream with which it can be improved the resolution in relation of a region of interest. That resolution variability has also to allow not only movements in one side or the other, but movements backward and forward as well.

The main purpose of the project is the following. We start start having a 3D Mesh and the position of the cameras that we will call

$$P = p_1, \dots, p_k \quad (3.1)$$

being k the total number of cameras. The main target is to get a rendered view of that scene again with the minimum distortion possible, transmitting the lowest bit rate possible and also having modified the resolution in function of the needs of each region.

Being x_i the image that the particular i camera gets (and are the same that the projection of the 3D mesh on the camera position p_i),

$$X = x_1, \dots, x_n \quad (3.2)$$

and \hat{x}_i the rendered image (the output of the system) for the i position,

$$\hat{X} = \hat{x}_1, \dots, \hat{x}_n \quad (3.3)$$

then, the main purpose of the project is to minimize the Distortion between those two sets of images is:

$$\operatorname{argmin} \left(D(X, \hat{X}) \right) \quad (3.4)$$

being D a criteria of distortion and taking into account the bit rate to transmit.

The minimum distortion between the input and output pictures will be when it is represented the same image in the rendering output that in the input. However, the bit rate is going to be maximum. For that, there will be a trade off with the bit rate produced that will be mandatory to have in mind.

With all of that, the project consists in an input of a 3D mesh from which has to be builded the representation of the 3D scene. After that, when the 3D image has been transformed in a set of layers that represent one image with all the depths, all this data has to be coded in some way for getting a good representation bitstream.

Thus, it has to be defined the way of sending the packets of the coded image in order to represent all the image with the definitions improvements. And finally, it is needed the inverse system that decodes the image and represents right the navigation over the scene.

3.1.1 Process

Departure

The starting point of the system is a Layered Depth Image. This is using the output of a previous currently working system available described in [13]. This previous step generates the LDI from a set of images from several cameras getting the info of one scene from different points of view.

From LDI to the multi-resolution friendly bitstream

To code the bitstream from the LDI, we study the wavelet techniques and the Discrete Wavelet Transform. The standard JPEG-2000 is using them and they give known good results.

This system codes the input in a set of packets that allow different resolution scalability. That is done by packing first the basic layers and then enhancements of this basic image by the other resolution layers.

For providing multi-resolution to different segments of the image independently, JPEG-2000 uses Embedded Block Coding with Optimal Truncation (EBCOT). That is based on coding the image in independent $N \times N$ blocks in order to be able to give different resolutions to any of this blocks if needed. The current project will be based on that in order to allow the variable resolution system.

In order to define well the resolution for each part, it is important to take a look on the depth of each point. That is because of perspectives and movement. For example, if the user move a step forward to the scene, two objects in different distances will increase their size for the user, but not in the same measure. The increasing of the pixels' size of the nearer object will be higher than the size of the backwards' object pixel. Then, we will need more resolution scalability in the nearer objects than in the more distant ones. In addition, as the system doesn't know which will be the user's point-of-view, the ROI has to be defined in a general way.

Going back: From the bitstream to an image

At the moment we have all the data coded and transmitted, it is needed to undo all the process and to get again an image. For that, we have to anti-transformate the DWT. The transmission system will code only those required coefficients on enhancement layers according to the previously defined criteria. Hence, it is needed to have a good control of where do each coefficient belong.

Rendering: From LDI to the final image

It's time now to compute the final image that it is going to be the output of the system.

We have the complete image with all the resolution coefficients. What is needed to do now is to compute what appears in the image and what not and, after that, to project it. All the projections are computed for each pixel and, then, the image is redistributed. The holes are filled with the projection of the back layers, also, iteratively, until having the image complete.

Of course, all that is done for multi-resolution images. In the end, it is the same idea but having different resolutions in the different layer's images.

Once it is done, the image is recovered and able to be shown.

3.1.2 How to represent the data? Definition of the structures

Another part of the problem is to decide and design how to represent and to code all the needed data at the implementation.

We start from a 3D mesh and we finish with a complex coded structure. But what and how will this structure contain all the information needed of the 3D mesh? First of all, we have to code the different LDI layers L_i from the 3D mesh.

Each of these LDI layers is a set of pixels $p(i, j)$ with its color information Col . It is necessary to code also the depth D , but it can be coded in a parallel depth map. The basic layer, however, will have the minimum resolution needed for the whole image. That resolution is not enough for some parts, so the system is going to improve the resolution by adding wavelet coefficients c_i in the corresponding part of the image. With that, the structure of the data will be like the following:

LDI image:

- LDI layers
 - Color information (tridimensional data like $RGB, YCbCr...$) of the basic layer.
 - Position (j, k) of each pixel and coefficient.
 - Enhancement coefficients $c_1, c_2...$ where $c_i = c_{i,1}, c_{i,2}, c_{i,3}$ is composed for the three DWT coefficients. These coefficients

are located in each possible pixel.

- Depth maps

It is correct to say that LDI is a group of some set of vectors. And it is possible to say that:

$$L_i = c_{i,0}(j, k), c_{i,1}(j, k), c_{i,2}(j, k), c_{i,3}(j, k) \quad (3.5)$$

However, there are two more important points. The depth map will be coded completely for separate, so it will not be included on the data. Hence, the data is easier to code using a tree structure. The idea is to have a root level which is the low resolution LDI representation. The next level of the tree is filled with the different enhancement coefficients of each LDI layer. As a resume, first of all are coded the lowest resolution LDI layers and, after that, the enhancement information is added to the tree.

3.1.3 Compression of the LDI image: which coefficients should be added or subtracted?

How to decide how many coefficients transmit, for example, for each image? And how to decide which of these coefficients should be the elected? We need to build a more precise idea about it.

First of all, it is important to define a distortion parameter between images. We could work with a perceptual distortion measure or with something that does not take into account the human visual system as the Peak Signal-to-Noise Ratio (PSNR). Instead a perceptual measure may give a bit better results when the image is only for human viewing, PSNR needs less complexity for being computed and gives general imaging better results. For that reasons, the distortion will be given by the PSNR between the images, and it is defined in 3.6:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \quad (3.6)$$

Another point is how to compute that PSNR. We have got the original image in the emitter and the recovered image in the receiver, so we can not compute the distortion measurement directly. In order to solve that,

there are needed two receivers instead of one. One in the decoding point and the other one in the sender. We can rebuild the image before sending it and compute the PSNR for any of the cases.

After the introduction to the distortion measurement, we can define L as the LDI representation data and L_i each layer of it,

$$L = L_1, \dots, L_m \quad (3.7)$$

where m is the number of layers of the 3D representation.

We can assume that the channel capacity will be something fixed and the target is to put all the needed information inside this channel with the less distortion possible. We assume that because it is the common situation. So we have to compress in order to keep the bit rate inside the possibilities of the transmission channel.

Actually, the process during the implementation is not going to be to compress but the opposite: to add coefficients to the lowest resolution data in order to increase its quality and resolution while the bit rate of that data is inside of the capacity of the channel.

Algorithm 1 Basic algorithm

```

while bitrate(code(L)) < channel capacity do
    addCoefficient(L)
end while

```

The function $code(L)$ is the process that we use to go from the LDI image to the bitstream. The $addCoefficient(L)$ function is based on a lossy compression system where there is defined a basic quality layer that includes the lowest quality required in the image. Then, it is improved the quality by adding those coefficients to the data that are going to minimize the distortion coefficient. With that, we add coefficients until reach the maximum number of bits possible.

The system will add to the bitstream the coefficient that maximizes the PSNR of the recovered image. We need to have in mind that the possible targets to maximize the PSNR are the different end leaves of the tree and not the nodes of the same level, for example. For that, the system is going to look for on different pixels sizes for the enhancement. So, being T the

set of possible leaves of the tree (targets) and \tilde{T} the optimal one:

$$\tilde{T} = \arg \max_T \left(PSNR(X, \hat{X}) \right) \quad (3.8)$$

Then, the previous algorithm is like follows:

Algorithm 2 Algorithm that follows the equation 3.8

```

while bitrate(code(L)) < channel capacity do
   $\tilde{T} = \arg \max_T PSNR(X, \hat{X})$ 
  addToTree( $\tilde{T}$ )
end while

```

3.2 Needs of the system

For this system, we have some concrete needs to cover and restrictions. The system is required to be resolution adaptable. This is in order of improve the quality of the image when not only horizontal or vertical movements in relation of the scene but also going closer or further. When the viewer wants to get closer to the scene, the current image becomes larger and, because of that, the perception of the image resolution changes. Thus, if the system is displaying the same image with a larger size, an important decrease of the resolution takes place. The figure 3.2 illustrates this phenomena.

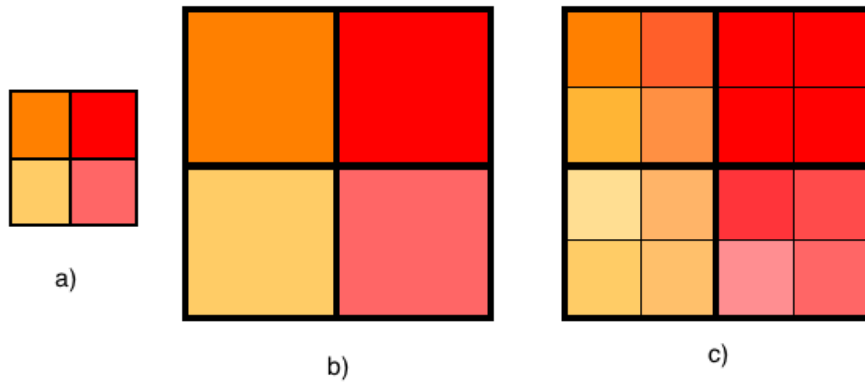


Figure 3.2: Small size set of pixels (a) whose resolution decreases when we enlarge them (b). However, the result is the same when increasing resolution.

Hence, a variability on the resolution of the rendered image makes possible also to step in or step out of the scene without a change on the image perceptible resolution.

One important point to have always in mind is that what we want is the lowest distortion in the final image related with the image that it should optimal be without the compression and with highest resolution. For that reason, the system needs a decoder in the encoder also working in an iterate process that optimizes the image until a condition. We are going to see this point later, but there are two important conclusions inferred from this paragraph. The first one is that we actually only have to take care of the output's results and not on the intermediate steps. That means that, for example, some layers are going to be processed in a extreme way, but we don't mind meanwhile the output is optimal. The other idea to extract is that the system is not going to be symmetric. Hence, the computational time and charge on the coder and decoder are not equal.

3.3 Organization of the system

How all the joint process are executed? In this section it is going to be described exactly how the system manages all the data from the input to the output. The diagram 3.3 illustrates the resumed scheme of the whole system.

When the LDI images are computed, the first step of the system is to code all the LDI images into a tree structure. We are going to see later into more detail how this structure is done, but the main idea is that a wavelet is coded and, while the lowest resolution layer is defined, all the enhancement coefficients are related with its low resolution image's corresponding pixel. Thus, we have got the same number of trees than layers have the LDI.

In a parallel way, it is computed the rendering of the final view with the original LDI Input. This means that we need to know the target position of the viewer. With this process, we obtain the result to which our system has to produce the most similar result.

When both the set of trees and the rendering are computed, is the time

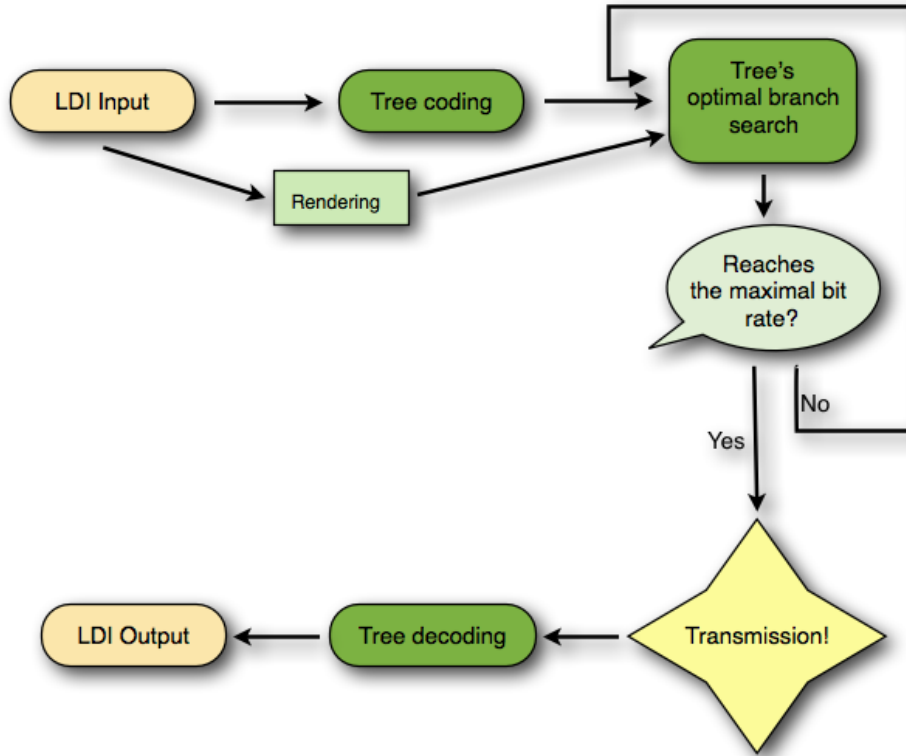


Figure 3.3: System's scheme

of modifying the trees in order to minimize the distortion for a concrete size of the bitstream. This process is made by an iterative system, which analyzes the Peak Signal to Noise Ratio of each of the target trees in relation to the original rendered view (the reference image).

The target trees are built by adding the branch of the original tree that maximizes the PSNR at each iteration. But the algorithm will be described with more detail later.

This is done until a limit is reached. And this limit is the maximum size defined as a parameter of the system. When the target trees reach the maximum bit stream length, the system stops to adding more branches to the target trees and prepare them for the hypothetical transmission. This is the output of the system!

Finally, in the other side of the channel, there is the decoder. It is based on a tree decoder and, from here, the output is the LDI images with the resolution adaptation done and prepared to be rendered.

3.4 Compression of the tree

The problem of the system is that it is not useful at all if there exists the possibility of coding the whole image without any kind of loss with a less amount of bits. That is why it has been designed a way for compressing the images in order to accomplish the purposes.

It is not easy to compress the tree with some basic entropy coding tools because the current compression techniques can reach impressive compression ratios. There is the need, then, to compress the tree with one of these current techniques. And because of the common points with JPEG-2000, this is the best technique to use.

3.4.1 JPEG-2000

JPEG 2000 is an image compression standard and coding system created by the Joint Photographic Experts Group (JPEG) committee in 2000. The purpose was to try to get a better system than their Discrete Cosine Transform-based JPEG standard (created in 1992) with a newly wavelet-based method.

Why JPEG-2000?

The main properties of this compression tool interesting for the current project are the Embedded Block Coding for Optimal Truncation (EBCOT) and the fact that JPEG-2000 uses the wavelet transform. In concrete, the wavelet implementation is Cohen-Daubechies-Feauveau 9/7. That is why the wavelet transform applied in this project is the same. In order to get a higher compression, both wavelet techniques had to be the same.

When we subtract a coefficient from the tree, we are decreasing the resolution in one area. When the wavelets coincide, the tract of the area is the same than with the wavelet and we are going to set zeros in those parts of the transformation wavelet of the JPEG-2000. Then, the EBCOT system will manage how to optimally compress this transform. Even though we will have had helped it by adding zeros in some known and related parts of the image. These coefficients to zero are reducing the entropy of

the image and also allowing to code it with a less amount of bits.

How JPEG-2000?

JPEG-2000 is implemented in an easy way. In order to compress a tree into an image format, it is needed to decode this tree to an image before. Then, the process is that, just after the coder, a decoder is set.

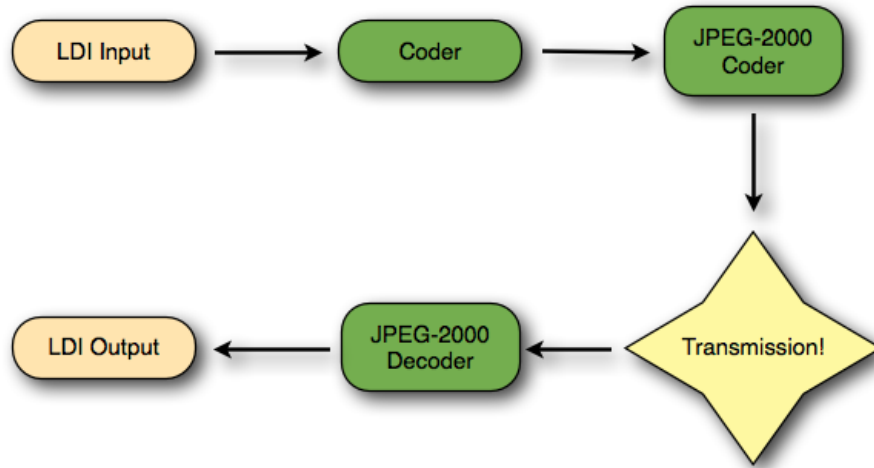


Figure 3.4: New system's scheme

After this decoder, there is a JPEG-2000 coder. Then, the output of the system to be transmitted is a JPEG-2000 bitstream containing the information of the decoded tree.

This change in the system also has got impact on the receiver, who only has to know how to read JPEG-2000 bitstreams. There is no need of reading the system's trees or any kind of personalized format. This is a great advantage due to the increasing versatility of the system. Only is needed a coder and not a decoder. Hence, by producing one file in a single coder, all the minimum-modern (understanding JPEG-2000) computers in the world will be able to read this file.

Chapter 4

A new adaptive representation. Implementation.

We have seen the main ideas of the system. But now is time to see how to implement them. Here, we have described the classes and also the functions of the system in order to build it up.

The system has been programmed using C++ language. Hence, in this chapter both the set of classes and of functions will be deeply explained.

After that, we are going to see and explain also the algorithm and its details.

4.1 Software

In this section we are going to see how the software is divided into classes and functions. While the classes are only two, there is a bigger amount of functions that are important for the system.

4.1.1 Classes

The first thing explained are the classes. There are only two classes and they are both for supporting the tree structure. One class is for the whole tree and the other class is for each node. The name of both classes is using a prefix in order to recognize them quickly and to match them

with this project because its implementation is specifically done for this project.

***APCNode*. The nodes of the tree**

This class has been programed for simulating the behavior of a node of the tree. Each *APCNode* contains some properties that the tree is able to manage and also a set of functions that help with the coding into and decoding from these trees. The main attributes that this class possesses are:

- **Coefficients.** Each node contains inside it a set of four coefficients (c_0, c_1, c_2, c_3). These coefficients are a link to another *APCNode* and they represent the next coefficients of the multiresolution scalability. While c_1, c_2, c_3 are exactly this idea of coefficients, c_0 is a bit a different case. This coefficient in particular is only existing as an implementing method for making the tree work. The three first coefficients are related to the enhancement layers while this coefficient only links the low resolution image at each level.
- **Value.** Each node is not only a link to the other nodes. It also contains the value of the pixel related to it. Thus, there is a vector that contains the different color information like the *RGB*, *YC_bC_r* or only the luminance if it is a greyscale image.
- **Control variables.** For building up the tree and also in order to prune this tree in the selective search, there are needed some booleans and variables that control for example if the node is a leaf, if it is a target for the pruning process or that also contains the level of the node inside the tree.

On the other hand, the main functions implemented in this class are:

- *deleteNode()* and *clearNode()*. These two methods are useful for deleting all the contents that are under the current node. The difference between both is that deleting a node, all the information is erased while clearing it, the links of the heritage nodes are not destroyed.
- *developNode()*. This method is basic for building the tree. It starts from one node and build, recursively until reaching the boundaries

of the image, each branch that is down each node.

- *reconstructNode()*. This method works as exactly the opposite of the previous one. Its function is to build again the image from a root node by scanning all the hanging nodes and by projecting them to its image's position.
- *copyBranch()* and *pruneBranch()*. These both methods are implemented for the recursive system that builds the optimal tree. The first one returns the entire branch hanging on the current node and the second method prunes the tree in the number of levels specified.

***APCTree*. The tree itself**

This class represents, as its name shows, the tree. It is builded as a set of *APCNodes*. This class is, as usual, mainly composed for a set of attributes and methods. The attributes are:

- Root map. This is a set of *APCNodes* that are the roots of the whole tree. Its size is the same than the lowest possible resolution.
- Size control values. There is a set of values designed for control the size of the original image and also of the subparts of the tree as the Root map.
- Number of levels of the wavelet. This is one important parameter at the time of building itself or reconstructing the image.

And the main methods are:

- *imageToTree()*. When a tree is referenced and this function is called, an image is the parameter needed. This image is a wavelet transform of the image that we want to transform and, using this function, the tree of this image is created and stored.
- *treeToImage()*. This method is designed for doing exactly the opposite than the previous one. Its use is to build the image from the tree.
- Set of auxiliary methods implemented for building the optimized tree in the iterative process.

4.1.2 Functions

Meanwhile the explained classes are important for the tree coding, everything else in the system is managed by functions. There are two important functions: *coder* and *decoder*. The other functions are mainly for doing what those two functions need to do.

Coder

This function's purpose is to build the set of optimal *APCTrees* in order to be transmitted to the decoder. It first of all computes the set of original trees from the original LDI and also the original rendering. From this point, it starts the iterative process that computes the optimal tree. The computational charge of this function is the highest of the system, because it includes all the tree codings and the image renderings, which require lot of operations and, thus, a several time.

Working under this one there are some important functions that have to be explained.

- *simpleCoder()*. This function's purpose is to code into a tree a single image. When the input is any normal image, this functions computes the Discrete Wavelet Transform of the image and, then, moves this transformed image into a tree structure using the *APC-Tree*'s methods. The return parameter is this tree.
- *apcdwt_2d()*. This function is an implementation of the Cohen-Daubechies-Feauveau 9/7 wavelet transformation. It is very important to specify that the wavelet is this one because is the same that the JPEG-2000 uses. Always that is needed, then, the DWT of one image is computed with this function.
- *ldirender()*. This image is an adaptation from [13] system. This adaptation is done in order to compute the output image for a particular point of view when a Layered Depth Image is the input.
- Distortion functions. There is also a file that includes some ways to compute the distortion between two images and also between two pixels. This is used for computing the optimal tree.

Decoder

While the coder computes the optimal trees, this functions goes back to the Layered Depth Images. Its idea is simple, but it also has got some functions working inside. The main idea is that it receives a set of trees and it returns the optimal LD images.

- *simpleDecoder()*. As *simpleCoder()*'s function was to code only one image, this functions purpose is to decode a single image. Its first step is to move the input tree into a image and, after that, to compute the Inverse Discret Wavelet Transform in order to get the modified image.
- *apcidwt_2d()*. This function computes the inverse Cohen-Daubechies-Feauveau 9/7 wavelet transformation. Just to remind that this kind of transform is the used in JPEG-2000.

4.1.3 Libraries

This system uses some different libraries. Some of them are open source libraries and others are specifically implemented in order to adapt other systems.

OpenCV

This library is important mainly for handling images and graphic data. It makes easier to work with images because it has got its own types designed and also some functions for work with these images.

From this library, it has been extensively used the *IplImage* class as an image container. It groups all the image info and makes easy to access and modify it.

LDI

This is a set of classes and functions originals from [13] modified in order to adapt LDI's system to the current one. Only a few classes and functions of LDI's project are needed, but some had to be several modified for being adapted. Thus, they are added inside the project as a library.

OpenJPEG

This library is used for coding the images into JPEG-2000 with different parameters. Despite the *OpenCV* library allows to save images with JPEG-2000 format, this library allows a high amount of different settings in order to personalize how to code the JPEG-2000 bitstream.

Miscellaneous

Other functions with general purposes are also inside this group. They have functions like, for example, to easily display an image through the screen or to compute the size of a file in the computer's *filesystem*.

4.2 Algorithm

The analysis of the algorithm as an entire system can be long and complicated. Thus, it will be easier to split it a bit and organize it in order to better understand how the system works.

4.2.1 Coder

First of all, we have the encoding part. It is shown in the algorithm 3. At its turn, it depends also of other subfunctions. The algorithm of these functions is also explained below.

In the coder, the first step is to build the needed material for the process: the rendered image with the original LDI, which is going to be the reference for the Peak Signal to Noise Ratio (PSNR) measure, and original set of trees from the LDI, which are the reference from whom the new trees are going to be created. Then, what is done is to treat each node of the tree as a possible target. Between all of these possible targets, it is computed which one improves the most the PSNR. This is done in an iterative way in order to find the lowest distortion within a stablished maximum file size of the output. It is tested any branch of the tree and, if its PSNR is greater than the previous tested, this branch is recorded. If we do this for each branch of the whole LDI, we are going to find the

optimal branch. However, we have to do it iteratively in order to find all the optimal branches before reaching the maximum size.

Algorithm 3 Algorithm of the Coder

Require: Set of Layered Depth Images in the *filesystem*

```

img ← loadLDI()
rendOri ← renderLDI(img)
oriTrees ← simpleCoder(img)
Set the minimum resolution image to targetTrees
continueBool ← true
while continueBool is true do
  for each level of each tree do
    for each branch of the level do
      prevAuxTree ← auxTree
      auxTree.setBranch(setBranch(oriTrees.getBranch()))
      tmpImage = simpleDecoder(auxTree)
      rend ← renderLDI(tmpImage, img)
      if PSNR(rend, rendOri) > previousPSNR then
        if (prevSize ≤ maxSize) and (size(rend) > maxSize) then
          targetTrees ← prevAuxTree
          prevSize ← size(rend)
          continueBool ← false
        end if
      else
        auxTree ← prevAuxTree
      end if
    end for
  end for
end while
return targetTrees
  
```

Inside this main coding algorithm, lower level functions are being executed. The algorithm for the *renderLDI()* function is illustrated in the algorithm 4. Its purpose is to build the final view from the Layered Depth Images. For this, firstly is built a huge matrix - known as *first2* - that contains all the values of the pixels, their position and their depth. After that, all the elements of this matrix are projected into the final rendered view image space. If two pixels coincide in the projection position, the system uses the closest to the view (the smallest depth) because this one has to be occluding the one behind it.

Another function defined and which is important to take into account is *simpleCoder()*. This functions moves an image into a tree through the Discrete Wavelet Transform (DWT). First of all, this transformation of

Algorithm 4 Algorithm of the function *renderLDI()*

Require: Set of Layered Depth Images in the input

Load the configuration file from the *filesystem*

Compute the sizes of the images from a first projection

for each pixel of each layer **do**

if the pixel is not a hole in the LDI **then**

$first2 \leftarrow$ add the pixel

end if

end for

for each element of *first2* **do**

$pixel \leftarrow projectPixelInTheRenderedView(first2 \text{ element})$

if (pixel position empty) **or** (pixel depth < existing depth) **then**

$renderedImage \leftarrow$ add the pixel

end if

end for

return $renderedImage$

the image is done and, after that, this resultant image with the transform of the original one is coded into the tree. The algorithm of the DWT and the tree coding are not going to be explained here because it is not necessary. They are already existing systems despite the fact that for this project were specifically implemented. The algorithm of this *simpleCoder()* function is briefly seen in the algorithm 5.

Algorithm 5 Algorithm of the function *simpleCoder()*

Require: Image, number of coding levels

$wavelet \leftarrow discreteWaveletTransform(image, codeLevels)$

$tree \leftarrow imageToTree(waveletImage)$

return $tree$

4.2.2 Decoder

Despite the fact that the big amount of process is inside the Coder, a Decoder part is also needed to be implemented. The idea of this part is to do the opposite of the coder in order to get the modified LDI.

Its structure is mainly based in the same than the coder. However, is much simpler than the previous one: the decoder does not need to work with iterative system. It is described in the algorithm 6.

Algorithm 6 Algorithm of the Decoder

Require: LDI coded into a set of trees

```

for each tree do
  image  $\leftarrow$  simpleDecoder(tree)
end for
return LDI as a set of images
  
```

Below this Decoder function works the *simpleDecoder()*. Its mission is only to move a tree into an image, computing the Inverse Discrete Wavelet Transform. It is seen in the algorithm 7.

Algorithm 7 Algorithm of the function *simpleDecoder()*

Require: Tree (contains the number of coding levels)

```

waveletImage  $\leftarrow$  TreeToImage(tree)
image  $\leftarrow$  inverseDiscreteWaveletTransform(image, codeLevels)
return image
  
```

Chapter 5

Experimental results

After explaining how the system is designed and implemented, it still remaining the most important part. Are the results good? Does the system work? Is it useful? All this information will be included in this chapter.

First of all, it is very important to do a clarification. Due to implementing problems and to a time lack in the final steps of the project, a part of the algorithm is missed. There were problems with the handle of the branches in order to find the optimal trees in an automated way. This led to a temporary lack of results that was solved by managing manually this step. Hence, the results showed in the following part of the thesis are not the ones produced by the system but a simulation of them. In other words, the results are not the optimal ones but yes suboptimal. This point is not critical to the results because anyway it is possible to proof that the idea works and it is useful.

5.1 The state before this system

First of all, it is interesting to see how were the results of the LDI before this system to have a reference idea. This images are shown in the figure [5.1](#). We can see that the system allows to see the same scene from a variety of different points of view. The main 4 points of view are illustrated in the already mentioned figure. The black holes of the image are due to numerical errors in the system. The projection of all the pixels of the original images to its final position produces some parts

where some pixels are overlapped and occluding the others.



Figure 5.1: Views of the original LDI output

We can also see that the parts with highest errors are the graphic boundaries between objects located at different distances. This is normal because is where there is a lack on the pixels of the first layer that are complemented with the other. Thus, it is the most sensible area.

The next visual experiments, in order to have a strong reference, are going to be based in the view upper in the right of the figure 5.1. In this way, all the comparisons will be made on the same basis.

5.2 Multi-resolution Layered Depth Image

The main characteristic of the system is to change the resolution depending on the part of the image. The system that computes this resolution automatically, as was said in the introduction, is not currently working due to a lack of time in its implementation. Thus, the multi-resolution of the layers have been changed manually using a Matlab script that defines the different regions. There has been tested different options relating each of these zones to one different resolution layer. The different

areas selected for having each particular resolutions are highlighted in the figure 5.2. We can see that this highlighted areas are those that have a highest components of highest details and that seems to require highest resolutions.



Figure 5.2: The highlighted regions correspond to the objects were the resolution is varied

The figure 5.2 is the layer 1 of the LDI, which is the main layer. This means that the biggest amount of information about the scene is in this image and the other layers only contain the pixels for covering the holes when this layer is projected. The black areas in there are those that because of geometrically reasons can not never be seen from any viewpoint.

The most important elements of the scene could be the Lena's picture in the wall and the man. After these two elements, there are the two chairs and the other pictures in the wall. We have also the shelf, the drawer, the cupboard and the table. We are going to define each different quality as a level, and we are going to locate each object to a map for this level. For example, a way of organizing the objects is as follows:

- Level 1: Lena, man and other pictures of the wall.
- Level 2: The chairs and the table and the cupboard.
- Level 3: The drawer and the shelf.

In terms of bit rate, the mean number of bits needed when decreasing one level of resolution is a 15% less. This has been computed by comparing

the whole layer 1 with its equivalent where all the pixels have been coded with 1 coefficient less and so on. The other layers, because of their big black zones due to the lack of coefficients, are less compressed by using this system. However, the significant point is that the main layer can be well adapted in terms of resolution with less bits coded.

5.3 Redistributing the resolution

If we set all the resolution coefficients for the Level 1, we reduce 1 coefficient for the Level 2 and we reduce 2 coefficients for the Level 3, the result is what figure 5.3 shows. What happens in terms of quality and bit rates? Peak Signal to Noise Ratio (PSNR) allows us to measure how much the quality of the image is changed. The PSNR between this new rendered image with the LDI resolution modified and the original view projected with the full resolution LDI is of $29.16dB$. That means that a good quality is well reached. This happens because the parts that require highest resolutions are kept in these resolutions, the original ones. With the automated system the general PSNR would be higher because its target is to optimize in terms of this distortion measure.



Figure 5.3: Case with all the coefficients in the most important areas

The problem with the previous method is that if we step forward into the scene, when enlarging the image, the resolution is not going to be enhanced in any way because there are not more coefficients available.

One option is to reserve all these coefficients for closer views and not for farther views. This means to decrease the maximal resolution but to allow a good detail when moving into the image. The results with this method are illustrated in the figure 5.4.



Figure 5.4: Reduction of the coefficients in order to enhance resolution when step forward

In this case, the PSNR is of $27.61dB$. It is a low loss in terms of general quality if it is not perceptually seen. It is important to have in mind that the resolution has varied and also the information of the image. Some artifacts can be observed when we are applying this change in the image. However, with the full system working this artifacts would be clearly reduced because the resolution variation is not done for large areas but for small groups of pixels and would optimize the quality.

5.3.1 Fixed bit rate

The most interesting application of this system is that for a certain bit-rate we can distribute the coefficients per zones in order to enhance the resolution in the particular areas with more details where it is needed. This allows to decrease the amount of information coded for a flat wall while increasing the amount of information for a more detailed part of the image. This leads to a better quality with the same bit rate.

The background wall, for example, is the flattest surface of the image. Its coding process may not need lot of coefficients if they are well managed. Hence, these non needed coefficients can go designated to the Lena's picture where a higher resolution is needed. For comparing the results, we can try the experiment by keeping constant the amount of bits and varying the allocation of coefficients.



Figure 5.5: Rendered view a low rate without reorganizing the coefficients

By keeping constant the bit rate, we can notice that the system works because the difference between results is significant. In the figure 5.5 we can see the rendered view computed from a low resolution layer with the same number of coefficients at the whole image. We can observe that despite a PSNR of $26.11dB$, closer than the expected to the previous one, for example, the perceptual quality of the image is not very good.

However, by adding more coefficients in a particular set of areas and by subtracting them from the less needed areas, we can improve a lot the quality of the image. The PSNR now is 3dB over the previous one being $29.16dB$. Even though, the image has got lot of better results to human eye because the shapes are better represented and all the objects have got better shapes. The result can be seen in figure 5.6.



Figure 5.6: Rendered view with the reorganized coefficients

For the human eye, the results have got a very good look and seem much like the original rendered view. However, it has less resolution in some parts and keeps the bit rate of the figure 5.5.

For a better comparison, a zoom into the image can help to see how are the variations. In the image of the left, we can observe that all the objects at any layer have got the same resolution while in the right the most detailed objects have got the most resolution related. We can see than in the image of the left the whole image has got a constant decrement in the resolution. Thus, all the parts of the image are borrowed in the same measure. In the image of the right, while the floor has got a worse quality than in the previous image, the detailed parts as the chair of the Lena's picture have got more noticeable resolution.

To sum up, the important point of this section is that by reducing the



Figure 5.7: Zoom in of the figure 5.5 (left) and 5.6 (right)

resolution in some parts of the image and by increasing it in some others, we can find a much higher perceptual quality keeping constant the amount of bits used.

5.4 The effects of the JPEG-2000 compression

Despite JPEG-2000 has got a lossless mode to compress de data using only entropy coding, the most interesting is to try to compress the image as much as possible without any perceptible changes in the result. This means using lossy coding.

Despite the fact that the lossless mode leads to a compression of a 17.5% from the original PNG lossless image, the lossy mode allows to compress the original image an 88.22%.

If the JPEG-2000 lossless system is less than the previous one, we can set it as the default lossless image compression mode. Even though in this case the reduction with the lossy algorithm is of an 85%.

These data has been extracted by coding the Layer 1 (the most significant) of the LDI with different resolution compositions and with the three systems and computing its average compression ratio. The numbers are shown in the figure 5.8.

In terms of quality, the distortion using the compressed files is not no-

Original PNG	JPEG-2000 Lossless	JPEG-2000 Lossy
1 120 521 bytes	799 272 bytes (71.33%)	184 454 bytes (16.46%)
869 865 bytes	765 479 bytes (87.99%)	90 075 bytes (10.35%)
862 715 bytes	763 987 bytes (88.55%)	90 504 bytes (10.49%)
841 409 bytes	689 953 bytes (81.99%)	82 909 bytes (9.85%)
	Mean: 82.46%	Mean: 11.78%

Figure 5.8: Each row represents the same image coded in the different styles. The percentile represents the size respect the original lossless image in PNG

ticeable with a simple eye. The mean PSNR between the lossy coded and the original of the 4 previous cases is $34,40dB$. With this results, we can use lossy JPEG-2000 compression without problems because the results are good enough.

5.5 The movement forwards and backwards

The user's movement in the viewing direction, in other words, the movement produced by a step forward or backward, can be simplified like a zooming. These kind of movements are not actually as simple as zooming, because a zoom represents a straight view into the image making larger a specified area. By the other hand, a step forward is not exactly an enlargement of the image from the same point of view. The difference is to take into account that the position varies in the horizontal way but not on the vertical with the displacement. It is easily to understand by seeing the figure 5.9.

This difference is a reprojection of the image in order to simulate the different points of view when the movement in direction of the scene has been produced. Even though the difference between both movements has to be taken into account when describing the viewing system, for us is not very important. The behavior related with the image resolution of both approaches is the same. Thus, for not needing the re-projection of the image, we are going to use a zoom in order to see how the system works.

In this section, we are going to work with the picture of lena hanging on the wall. First of all, say that we have defined a different mask that allows

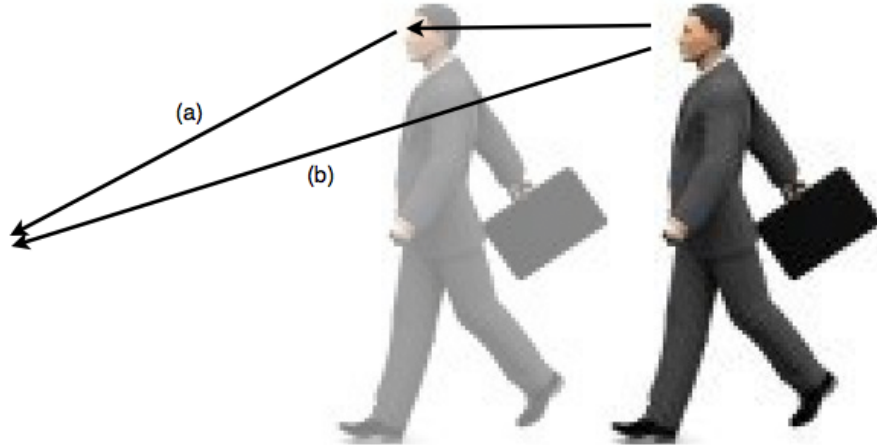


Figure 5.9: Graphic difference between a step forward (a) and a zoom in (b)

a finer resolution control on the different areas of the lena picture. This new resolution mask is shown highlighted in the figure 5.10 as well as the related result image when we combine different layers of resolutions.



Figure 5.10: Result image in the left and the related mask highlighted in the right

This model is a bit more closer to the optimal one than the shown with the whole image because the resolution is defined as it is needed in a small local way.

With the whole system, each pixel is analyzed to see if it is who needs the enhancement on the resolution the most. Hence, it is good to see how is the behavior in a smaller area than the whole image with a finest definition of resolution. The results with this finest resolution is the shown in the figure 5.11. The image's resolution in general is not very



Figure 5.11: Results of the addition of coefficients in particular areas of Lena's picture

good, but that is because we are enlarging in an extreme way a very small part of the scene. The overall resolution of the image is as desired. In figure 5.12 it is possible to see how keeping the bit rate constant of the file it is possible to improve the resolution of the local areas without a big perceptible difference in the more stationary parts of the image. The PSNR is also higher in the second image ($47.80dB$) compared with the first image ($46.89dB$).

There is another very important point to be taken into account. Now, we are zooming into the smallest parts of the image so it is very important to execute all the available resolution resources. All the coefficients are set in a part of this image in order to show the best resolution possible. As we said, we can reserve all this highest resolution levels for these zooming movements. They do may not give a noticeable quality enhancement with the global view and they are very useful now, when enlarging the image pixel size.

With all these exposed results, we can observe than the research field has



Figure 5.12: Comparison between constant (left) and variable resolution (right) for a fixed bit rate

given positive results. We have proved that the idea works despite of the fact that some improvements could be done.

Chapter 6

Conclusions and future work

After all the results, it is needed to extract some conclusions of the work done. Has the system worked as desired? What should be improved? What we realized that was less useful than the expected? And, also very important, what would be the next steps from this project?

6.1 Conclusions

The most important conclusion is that the system is based in a good idea. The fact of being able to generate a resolution variable representation system that allows to compress the image with ratios as the exposed maintaining the quality level proves that the basis of the system are good. Thus, we are in a right position to affirm that this branch of study can give lots of satisfying results if well exploded.

It is very interesting to have an adaptive system because it gives flexibility to both the coder and the decoder. Not all the information has to be coded in the same moment because it is possible to transmit the parts of the image while they are needed. For example, it is possible to send first the main optimized view and, if the user requests a zoom in, send also the enhancement coefficients.

It permits also a simpler decoder. The requirements of this system for the decoder are minimum - just being able to read JPEG-2000 files - so it is possible to implement a simple receiver for this system in concrete. All the computations are done in the coder and this also allows a best

real time application usage if the images on the coder are previously computed and generated.

The main problem of the system is the computing time. It would be a good idea to adapt the algorithm in any way in order to allow if not a real time implementation something closer to it. The proposed algorithm requires a lot of different operations with the trees and the final renderings that lead to a high time of execution.

6.2 Future steps

The main future step would be to finish the right implementation of the branch handling system in order to see how the totally optimized results are. The prediction and the intuition say that they should be better than the current results, but this should be really proved.

Another point is the computational load and the large execution time. The system using this algorithm requires a lot of computational load because it has to project several times all the pixels. There are different ways of how this problem could be attacked. One would be, for example, by projecting only smaller parts of the image and then a very important smaller amount of pixels. Other way would be to try to reach the same results by analyzing only the LDI and by weighting it in order to simulate the rendered view.

It is also a possible way of acting the fact of removing the tree structure and to code the image directly from the LDI by defining different searching areas. This option could be also studied and it seems that would lead to the same results.

Bibliography

- [1] Jin-Sung Lee ; Kwang-Yeon Rhee ; Sang-Young Park ; Seong-Dae Kim. Efficient three-dimensional object representation and reconstruction using depth and texture maps. In *Opt. Eng.* 47(1), 017204 (January 16, 2008). doi:10.1117/1.2832372, 2008.
- [2] Sang-Young Park ; Seong-Dae Kim. Depth compression of 3d object represented by layered depth image. In *Data Compression Conference (DCC), 2010*. Dept. of EE, KAIST, Daejeon, South Korea, 2010.
- [3] Sang-Young Park ; Seong-Dae Kim. Efficient depth compression based on partial surface for 3-d object represented by layered depth image. In *Dept. of Electr. Eng., KAIST, Daejeon, South Korea*. Dept. of Electr. Eng., KAIST, Daejeon, South Korea, 2010.
- [4] Paul E. Debevec ; Camillo J. Taylor ; Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.
- [5] Ferran Marques. Jpeg-2000. In *Coding of Audiovisual Contents*, 2012.
- [6] Ferran Marques. Wavelets. In *Coding of Audiovisual Contents*, 2012.
- [7] Leonard (Jr.) McMillan. *An Image-Based Approach to Three-dimensional Computer Graphics*. PhD thesis, University of North Carolina, 1997.
- [8] A. ; Muller K. ; Wiegand T. Merkle, P. ; Smolic. Multi-view video plus depth representation and coding. In *Image Processing, 2007*.

- ICIP 2007. IEEE International Conference on*, volume 1, pages I – 201– I – 204. Fraunhofer Inst. for Telecommun., Berlin, 2007.
- [9] Christopher M. Brislawn ; Brendt E. Wohlberg ; Allon G. Percus. Resolution scalability for arbitrary wavelet transforms in the jpeg-2000 standard. In *Visual Communications and Image Processing 2003*, 2003.
 - [10] David Taubman ; Rene Rosenbaum. Rate-distortion optimized interactive browsing of jpeg2000 images. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, 2003.
 - [11] K. ; Merkle P. ; Fehn C. ; Kauff-P. ; Eisert P. ; Wiegand T. Smolic, A. ; Mueller. 3d video and free viewpoint video - technologies, applications and mpeg standards. In *Multimedia and Expo, 2006 IEEE International Conference on*, pages 2161– 2164. Fraunhofer Inst. for Telecommun., Heinrich-Hertz-Inst., Berlin, 2006.
 - [12] Jonathan Shade ; Steven Gortler ; Liwei Hey ; Richard Szeliskiz. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998.
 - [13] Uday Takyar. 3d representation for multiview video. Master's thesis, School of Computer Science and Communication Systems IC (EPFL), 2013.
 - [14] Masayuki Tanimoto. Ray space. <http://www.tanimoto.nuee.nagoya-u.ac.jp/study/FTV/ray-e1.html>.
 - [15] M.W. Taubman, D.S. ; Marcellin. Jpeg2000: standard for interactive imaging. *Proceedings of the IEEE*, 90:1336 – 1357, 2002.