University of Southern California
Universitat Politècnica de Catalunya
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

EDGE-PRESERVING DEPTH-MAP CODING USING GRAPH-BASED WAVELETS

Diploma Thesis

by Alfonso Sánchez

Advisor:
Antonio Ortega

Barcelona, Spain, June 2010

Abstract

This thesis presents a new wavelet transform specifically designed for the coding of depth images which are used in view synthesis operations. Two basic properties of these images can be leveraged: first, errors in pixels located near the edges of objects have a greater perceptual impact on the synthesized view; second, they can be approximated as piece-wise planar signals. We make use of these facts to define a discrete wavelet transform using lifting that avoids filtering across edges. The filters are designed to fit the planar shape of the signal. This leads to an efficient representation of the image while preserving the sharpness of the edges. By preserving the edge information, we are able to improve the quality of the synthesized views, as compared to existing methods.

Acknowledgments

I would like to express my greatest appreciation to my advisor, Antonio Ortega, for his support, stimulating suggestions and encouragement throughout the course of this research work.

I would also like to express the most sincere gratitude to my colleague Godwin Shen, for without his contribution this work would have not been possible. Moreover, I am very grateful to my family and friends for their unconditional support.

Lastly, I would like to thank *Vodafone España, Fundación Bancaja and AGAUR* for their financial support. This work was also supported by NASA under grant AIST-05-0081.

Contents

List of Figures

Chapter 1

Introduction

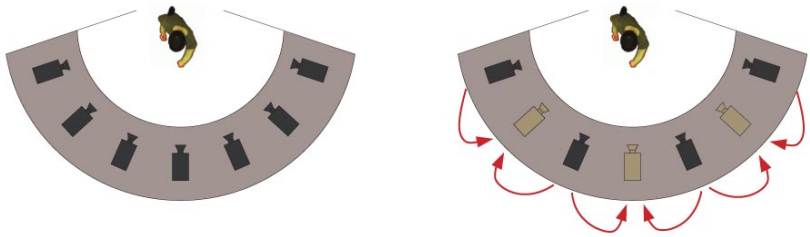
Recent advances in multi-view video (MVV) allow applications that not so long were only possible in science fiction, such as three-dimensional television (3DTV) [SMS+07] or free view-point video (FVV) [Tan04], to be getting closer to becoming a part of our daily lives. These techniques have a very promising future in services such as broadcasting television, video games, virtual reality, surveillance, as well as video conferencing. Nevertheless, challenges remain that are preventing these technologies from reaching the market.

When capturing a certain scene for MVV applications, an array of multiple cameras which record the stage from various positions, or view points, (as shown in Fig. 1.1(a)) is used. Since each camera generates its own sequence of frames, the total amount of data produced by the system is proportional to the number of cameras, and can easily become excessive for existing storage and transmission technologies. Moreover, since all frames across multiple cameras are representing the same scene, the information is very redundant.

Many researchers have developed techniques that take advantage of this redundancy (or correlation between views) in order to reduce the total amount of data. Some of the most important approaches are: interview prediction [XGZ05], 3D motion estimation [YKS07], motion vector reuse [SHLY08] and multi-view video plus depth (MVV+D) [YKK+07].

Our work can be seen as an example of a MVV+D approach, which is also referred to

as depth-image-based rendering (DIBR). As the name indicates, this approach makes use of the depth information, i.e., the distance from the viewer (camera) to objects in the image, in order to compute the position of these objects in 3D space. Using this information, it is possible to artificially generate the images that would be seen from different points of view. MVV+D uses this property in order to reduce the number of cameras that record an image. Using DIBR techniques, frames that certain cameras would generate can be artificially produced by manipulating the information provided by the rest of the cameras, so that those intermediate cameras can be eliminated from the array without loss of information. Fig. 1.1 illustrates this idea. The technique used for the synthesis of images using information provided by other cameras is called *view interpolation*.



(a) Original MVV camera array.

(b) Camera array using DIBR and view intepolation.

Figure 1.1: Thanks to view interpolation, the frames from some cameras can be generated using the information provided by others.

It has been stated that DIBR applications require the location of objects in 3D space. However, standard video frames only provide information about positions in two dimensions, thus the third spatial component must be given by depth images, also referred to depth maps (DM). A DM is a gray scale image that provides information about the position of objects in the three spatial dimensions, but not about colors, therefore, it is used in combination with a standard image (referred to as the DM's parent) that represents the colors and textures of the objects. In a DM, the value of each pixel represents the distance that separates the viewer (camera) from the object represented by that same pixel (same location) in the DM's parent. Fig. 1.2 shows an example.

Thanks to DIBR the number of cameras can be dramatically reduced, and, consequently, so can be the quantity of data generated. Nevertheless, the number of cameras can still be considerably high, and, in addition, now each camera does not

(a) Video frame. (b) Corresponding DM

Figure 1.2: Example of an image and depth map pair. Objects closer to the viewer have a higher intensity in the DM.

only produce sequences of frames, but also sequences of DMs. All this implies that the total information generated by the system requires excessive storage and/or transmitting capacities. Therefore, compression techniques are necessary.

In our work we have focused on the compression of DMs as isolated images (not as parts of a sequence). In order to find the most optimal compression approach it is first necessary to perform an analysis of the characteristics of such images. Also, since DMs are not directly seen by the viewers, but only used during the interpolation process, it is necessary to pay special attention to the effect that errors in DMs have in the resulting synthesized image. Bearing these ideas in mind a coding algorithm, which is efficient for this kind of signals, is built. Assuming that DMs follow a piece-wise planar (PWP) signal model and using an innovative edge-avoiding filtering technique specially designed for this kind of images, we have developed a discrete wavelet transform (DWT) [Mal08] which is able to compress the DMs while minimizing the distortion around the edges of the objects. This improves considerably the perceived quality of the synthesized images.

Others have tackled this same problem in different ways:

> Platelets [WN03], which also assumes that the images follow a PWP model, segment the image until a point where inside each block there is only one edge (which must allow parameterization using a straight line). Then, the parameters of the planes at each side of the edge are computed and sent along with the block segmentation overhead. At the decoder, each pixel's value is computed using the corresponding plane parameters. This method's main limitation is that

it does not allow perfect reconstruction (it is not completely invertible), since the error generated when approximating the image values for planes cannot be corrected at the decoder.

Directionlets [VBLVD06] and bandelets [PM05], apply a similar block segmentation as Platelets. In this case, inside each block a dominant directional pixel intensity flow (parallel to the edge) is defined. Then, a DWT algorithm is applied, where separable filtering along a grid aligned with each directional flow is used. These methods are discussed in more depth in Section 2.2.

In tree-based wavelets [SO09], a DWT is applied along trees, which connect the pixels in the image. These trees are built so that two pixels from different sides of an edge are never filtered together. However, since no assumptions are made about the signal's model, the filter coefficients cannot be adapted to the position of the pixels in the image. This method does not use block segmentation, but requires that both the encoder and decoder share the edge information, which is sent as overhead.

Shape adaptive wavelets [MD08], use a technique similar to the mirroring of the pixel at the boundaries of an image. In this case, when an edge is reached, the value of the pixels on the other side is obtained by extrapolating the function that the image follows on the one side of it, forcing the input of the filter to be continuous. This transform uses 1D filters; hence, the extrapolation is done using pixels along a certain 1D direction. Like tree-based wavelets, this algorithm requires the sharing of the edge information by encoder and decoder.

Our transform is based on a similar filtering approach to the one used by shape-adaptive wavelets. The main difference between both algorithms is that we allow the encoder to select from a more general set of filtering kernels and develop the necessary locally adaptive kernel selection algorithms. By assuming that DMs follow a PWP model, the position of pixels distributed in 2D space is used for computing each plane's parameters. These are used for designing optimal filters. On the contrary, shape adaptive wavelets only consider 1D filtering operations. Based on this filtering approach, we have developed a completely invertible wavelet transform, that, without directional filtering or block segmentation, is capable of efficiently compressing DMs to be used in view interpolation operations. Like in the tree-based and shape-adaptive wavelets, the edge information needs to be sent as side information.

The rest of this work is organized as follows. First, the bases and main elements of our algorithm are introduced in Chapter 2. Chapter 3 describes in detail the adaptive filtering technique used in our system. For a simpler understanding of the algorithm, the operations detailed in both chapters are explained for the first level of decomposition; the changes that the algorithm suffers as the rest of the steps are taken are detailed in Chapter 4. Some experimental results are presented in Chapter 5. Conclusions and future lines of work are detailed in Chapter 6.

Chapter 2

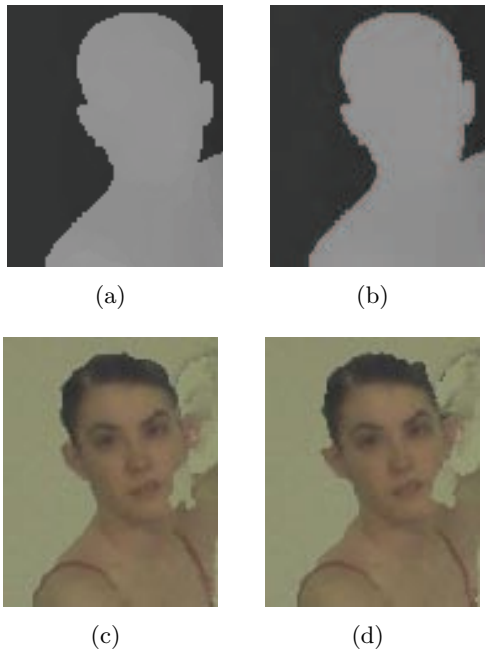
Proposed Algorithm

The core of our work is the design of a coding technique for DMs which efficiently compresses the information while minimizing the errors of the interpolated images. This algorithm is based on wavelets and designed using the lifting procedure [DS97]. The two properties of the DMs that are used by the algorithm in order to adapt the elements of the procedure are the following:

First, DMs are assumed to be piece-wise planar signals, which means that they are formed by planes with arbitrary shapes. The separation between two adjacent planes is named *edge*. In most DMs, the distance from the viewer to every part of a small object is practically the same; thus, all the pixels that represent those objects in a DM will have approximately equal value. As for the value of the pixels that represent bigger objects, such as walls, tend to vary very smoothly; this variation can be well approximated by planar functions.

Second, the information of pixels located near the edges has to be preserved. This is motivated by the fact that, as shown in Fig. 2.1, errors on those pixels are more likely to generate severe artifacts in the synthesized images than the errors on pixels located far from edges. This fact has been discussed and analyzed in depth in [LOD+09] and [KOL+09]. Moreover, when applying coding techniques based on standard filtering operations, such as standard DWT, large high frequency coefficients tend to appear in those areas. Due to the spatial distribution of these high frequency coefficients (they tend to be isolated), their

coding usually requires a large number of bits, as compared to low frequency coefficients. Consequently, when the images are compressed at very low bit rates, more severe errors appear around edges than in other areas of the image, which is, clearly, not optimal for view synthesis applications.



(a) (b)

(c) (d)

Figure 2.1: Example of artifacts in a synthesized image. (a) Original DM; (b) compressed DM using standard DWT; (c) synthesized image using (a); (d) synthesized image using (b). Although the compressed depth maps may be perceptually lossless, the interpolated view shows artifacts.

Bearing this in mind, we have developed a new wavelet algorithm which is based on an innovative adaptive filtering technique. By avoiding filtering across edges, our transform is able to reduce the intensity of the high frequency coefficients (especially around edges), which allows high compression rates while minimizing the errors in the synthesized images, as compared to existing methods.

The proposed transform is used as part of a coding system (refer to Fig. 2.2) which also includes other components such as an *edge detector*, whose output is used by our transform, and a supplementary coding component which quantifies and codes the transform coefficients. While for coding we use an existing algorithm (for our experiments we have used SPIHT [KP97, SP]), a new *edge detector* has been especially designed for our transform, as explained in detail in Appendix A. Since the

edge information must be shared with the decoder, it is also encoded (using JBIG [jbi93] in this case) and sent as side information. The influence of these elements in the performance of our transform is studied in Chapter 5.

In this chapter only the wavelet transform is discussed in detail. Section 2.1 reviews the most important ideas behind the wavelet transforms and the lifting procedure. How each operation in the lifting scheme is adapted to the scenario is explained in Sections 2.2, 2.3, 2.4 and 2.5.

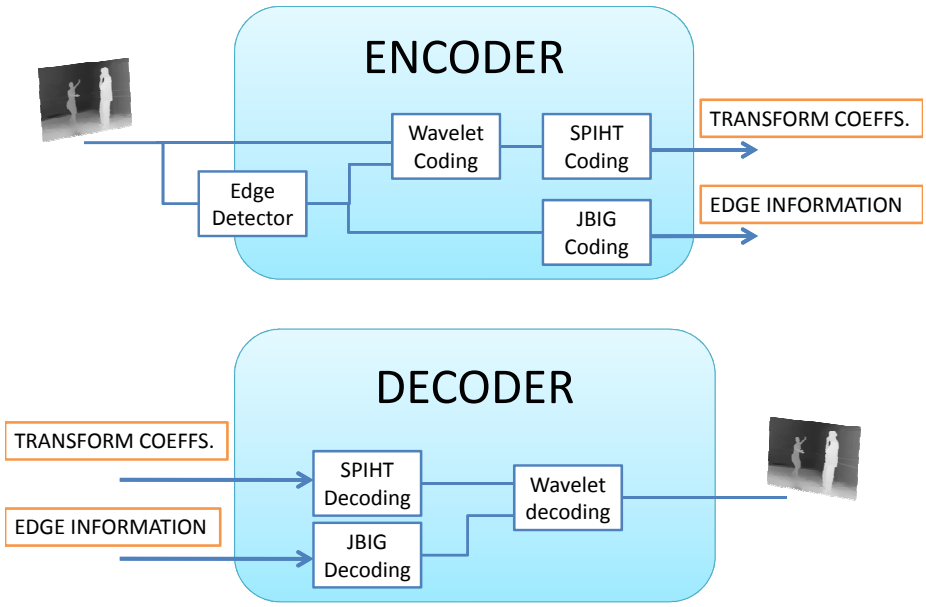


Figure 2.2: Block diagram of Encoder and Decoder

2.1 Wavelets and Lifting

Contrary to other frequency analysis tools, such as the Fourier transform [Tol76], wavelet transforms render the information of a signal in ways that are localized in both space and frequency. When applying a wavelet algorithm to an image, the frequency components are separated in 2D space as shown in Fig. 2.3. In the original

image, the position and intensity of each pixel is completely specified but no frequency information is available. Thanks to the wavelet transforms it is possible to estimate locally the relative weight of different frequency components. However, the resolution in the spatial domain has decreased as compared with the original image. This last phenomenon is a consequence of the Heisenberg uncertainty principle [Hei49].

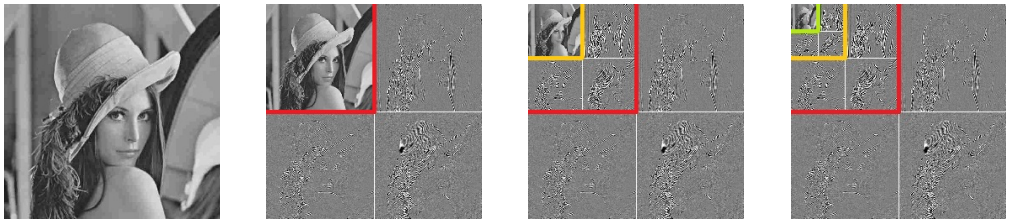
(a) Original image

(b) Image transformed using DWT

Figure 2.3: Example of the application of DWT to an image. (a) Original Image; (b) Resulting image after the application of DWT. In (b), frequency bands ordered: Lh-Lv(top-left), Lh-Hv(bottom-lft), Hh-Lv(top-right) and Hh-Hv(bottom-right), where L and H represent Low and High frequency band respectively, and h and v stand for horizontal and vertical direction respectively.

Since for typical images most of the information is located in the lowest frequency bands, the part of the output corresponding to those bands is considered an *approximation* to the original image. Therefore, in order to perform a more detailed analysis, the transform can be applied iteratively on that *approximation* until the desired spatial/frequency resolution is reached. This is done by simply cascading the transform blocks as many times as levels of decomposition are desired; the result of such operation is shown in Fig. 2.4. As the information of the signal is not changed after applying a wavelet algorithm but only expressed in a different space; the process can be inverted in order to return to the previous space (original image).

Wavelets are used in image processing for many purposes such as de-noising, eliminating textures or simply analyzing of the characteristics of a particular image. Since, the information of most signals (images) is not uniformly distributed along all the frequency bands but rather condensed in just a few ones (normally the lowest frequency bands) this kind of algorithms are commonly used for compression.



(a) (b) (c) (d)

Figure 2.4: Different levels of decomposition. (a) Original image. (b) After 1 level of decomposition, (c) 2 levels and (d) 3 levels.

One common way for designing wavelet transforms is by the use of the *lifting procedure* which is explained in Section 2.1.1. Section 2.1.2 explains how the procedure can be manipulated so that the whole transform is adapted to a certain scenario.

2.1.1 Introduction to the Lifting Procedure

Every wavelet transform can be expressed as a combination of lifting stages. The process of transforming a discrete signal using lifting is explained next:

First of all, given a discrete signal, its samples must be split into two disjoint sets, namely the sets of the even ($\mathcal{E}$) and odd ($\mathcal{O}$) samples, also referred to as evens and odds for short. This operation is often called *samples split* or *parity assignment*. Let us illustrate this operation with a simple example. Let $\mathbf{x} = (x_k)_{k\in\mathbb{Z}}$ be an infinite length 1D signal. We can define the even and odd sets using the parity of the samples: $\mathbf{x}_e = (x_{2k})_{k\in\mathbb{Z}}$ and $\mathbf{x}_o = (x_{2k+1})_{k\in\mathbb{Z}}$.

Then an approximation of the values of the odd set ($\mathcal{O}$) is constructed using only the samples of the even set ($\mathcal{E}$) and a prediction operator P. This operation is called *prediction step*, and the samples corresponding to the error of such approximation are named *detail coefficients*.

$$\mathbf{d} = \mathbf{x}_o - P(\mathbf{x}_e). \tag{2.1}$$

For example, a prediction of every odd value x_{2k+1} can be performed by averaging x_{2k} and x_{2k+2} (as shown in Fig. 2.5(a)):

$$d_k = x_{2k+1} - \frac{1}{2}(x_{2k} + x_{2k+2}). \tag{2.2}$$

If $\mathcal{E}$ and $\mathcal{O}$ are sufficiently correlated, the approximation error will have very low energy (if $\mathbf{x}$ in our example was locally linear, all detail coefficients would have no energy ($d_k = 0\ \forall k$)). As proved in the context of DPCM methods [Cut52], it can be more efficient in terms of bit consumption to retain the detail values ($\mathbf{d}$) than the ones of the original signal ($\mathbf{x}_o$).

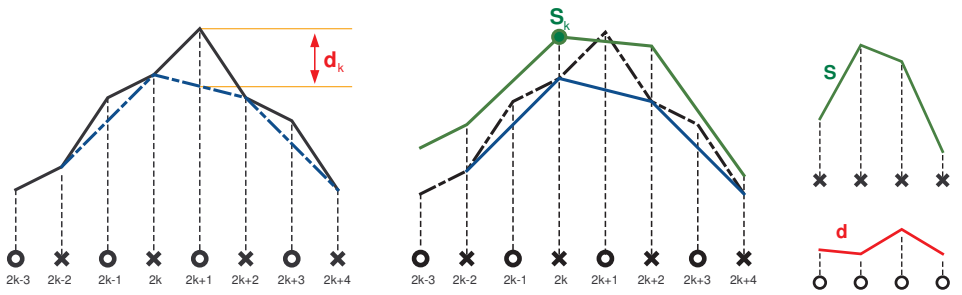
At this point the signal has been transformed from a pair of sequences $(\mathbf{x}_e, \mathbf{x}_o)$ to another $(\mathbf{x}_e, \mathbf{d})$; since $\mathbf{x}_e$ is obtained by a simple sub-sampling of the original signal, aliasing occurs. To correct this, the values of the evens are updated using an operation that ensures that the running average of the signal is preserved. This operation, which is named *updating step*, uses the detail values obtained in the previous step. The resulting samples are called *smooth coefficients*. This operation can be written as:

$$\mathbf{s} = \mathbf{x}_e + U(\mathbf{d}). \tag{2.3}$$

For example, the update operator could be defined as:

$$s_k = x_{2k} + \frac{1}{4}(d_{k-1} + d_k). \tag{2.4}$$

Once this step is completed the signal has been transformed to a pair of sequences (**s**,**d**), in which **s** is a sparse approximation of the original signal **x**, while **d** represents the error of such approximation. Fig. 2.5 illustrates the whole process.



(a) Splitting and Prediction (b) Updating (c) Separation

Figure 2.5: Example of lifting transform on a 1D signal

Since both the predicting and the updating steps are invertible, the whole transform can be reversed by applying an inverted procedure:

$$\begin{aligned} \mathbf{x}_e &= \mathbf{s} - U(\mathbf{d}), \\ \mathbf{x}_o &= \mathbf{d} + P(\mathbf{x}_e). \end{aligned} \tag{2.5}$$

Fig. 2.6 shows the block diagram of the forward transform operation. Similarly to wavelets, the operation can be repeated iteratively on the smooth coefficients **s**, which can lead to greater compaction of the information of the signal.

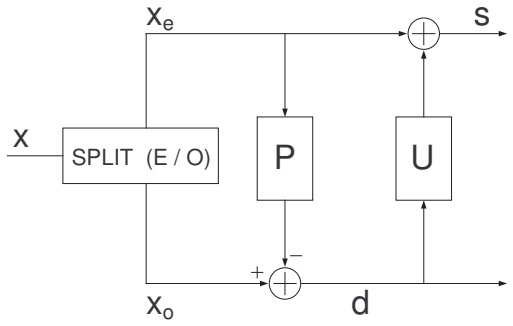


Figure 2.6: Lifting Procedure block diagram.

2.1.2 Design of a Lifting Transform for Images

Every lifting based approach is fully specified by the choice of the sets $\mathcal{E}$ and $\mathcal{O}$ and the operators P and U. Regarding the even/odd sets selection, the only requirement is that both sets be strongly correlated, for, this way, the intensity of **d** can be minimized, and thus high rates of compression can be achieved. In order to separate the frequency components of the image into 4 bands after each level of decomposition (as in Fig. 2.3), the algorithm is applied as a separable 2D transform (see Fig. 2.7). This means applying the transform in one direction (which implies a downsampling along that direction), and next, applying another operation in a different direction. The choice of these directions can have important repercussions on the performance of the whole transform.

As for the predicting and updating operations, the expressions that will be used in this work are:

$$d(i,j) = X(i,j) - \sum_{(k',l')\in\mathcal{N}_{i,j}} \mathbf{p}_{i,j}(k',l') \cdot X(k',l'), \tag{2.6}$$

$$s(k,l) = X(k,l) + \sum_{(i',j')\in\mathcal{N}_{k,l}} \mathbf{u}_{k,l}(i',j') \cdot d(i',j'). \tag{2.7}$$

Regarding the prediction of (2.6), the filter $\mathbf{p}_{i,j}$ weights the contribution of each

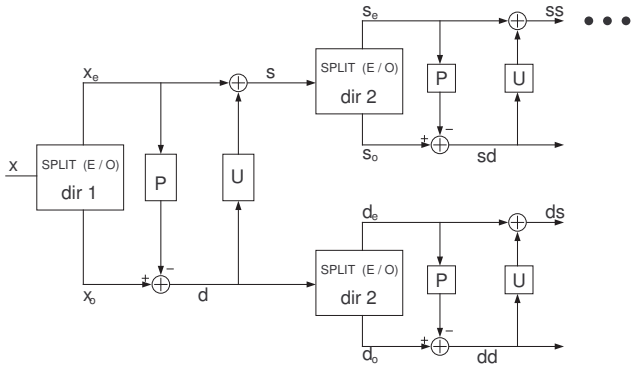


Figure 2.7: Block diagram of a separable 2D lifting transform. If more levels of decomposition are needed, the operation can be repeated on the SS signal.

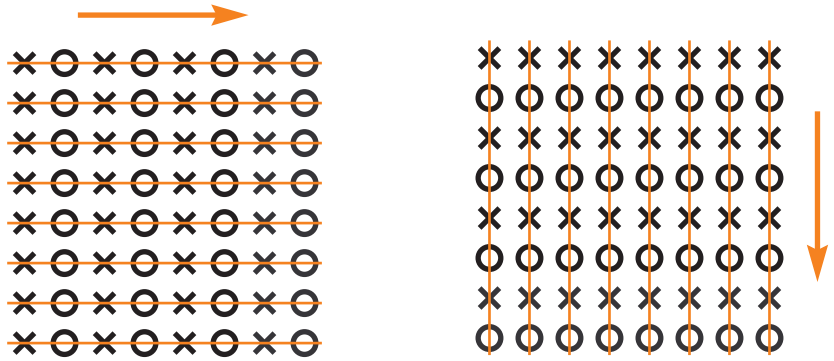
$(k', l') \in \mathcal{N}_{i,j} \subset \mathcal{E}$ to the approximation of the value $X(i, j)$. As for the updating operation of (2.7), it is clearly another example of filtering; where $\mathcal{N}_{k,l} \subset \mathcal{O}$.
Two lifting transforms with small differences in the way that the filters $\mathbf{p}_{i,j}$ and $\mathbf{u}_{k,l}$ are obtained and in the structure of the sets $\mathcal{N}_{u,v}$ can have completely different performances. Therefore, the way that these elements are defined needs to be adapted to the scenario in which the transform is applied.

The following sections and chapters describe the choice of all these elements in detail. Note that for a simpler understanding of the algorithm, the detailed operation of all the blocks is explained for the first level of decomposition (before any downsampling has been done). How the operations vary as the level of decomposition increases is explained in Chapter 4.

2.2 Proposed Coefficient Split

The first step in the application of a lifting algorithm is the separation of the pixels of the image into the even and odd subsets ($\mathcal{E}$ and $\mathcal{O}$). The most common approach is the use of separable schemes with alternating parities, as in Fig. 2.8. This means that, as shown in Section 2.1.2 two directions in the 2D space are chosen. Then, for the first lifting step, the pixel parity is alternated along one direction (which we name *transform direction* (TD)), i.e., the two adjacent pixels of an odd would be even, and vice versa, along TD; while it is forced to be constant along the other direction (named *alignment direction* (AD)), i.e., the pixels are either all odd or all even along

AD. Once the smooth and detail coefficients have been separated, the operation is repeated over the two signals (**s** and **d**) as shown in Fig. 2.7. This time, the TD and AD are switched. We call *standard separable splitting* the method that, as shown in Fig. 2.8, chooses the vertical and horizontal directions (rows and columns) as the TD and AD with the parity distribution explained above.

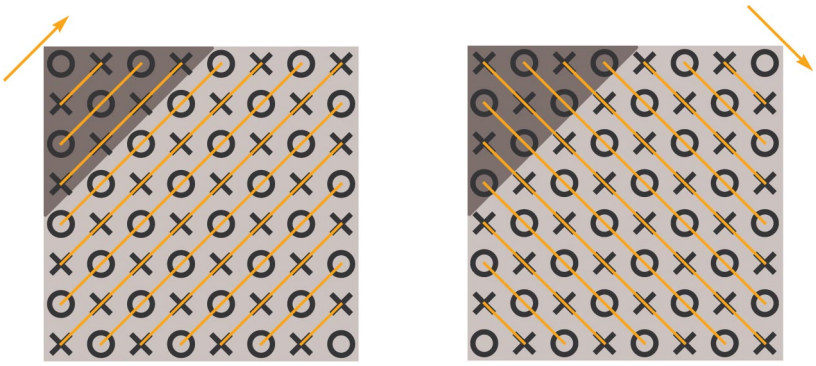


(a) Horizontal direction as TD. (b) Vertical direction as TD.

Figure 2.8: Example of standard separable splitting.

Many have seen in the splitting a way for adapting the wavelet algorithm to the characteristics of the image. Regarding DM coding, some approaches, such as *directionlets* [VBLVD06] and *bandelets* [PM05], base their algorithms on the optimal selection of these directions in order to avoid filtering across edges. These methods segment the image until a point where there is only one main edge inside each block, which must allow parameterization using a certain function (a straight line in the case of directionlets and a curve in bandelets). Then they define the TD as the one parallel to the edge and the AD as the one orthogonal to it. The separable wavelet algorithm is then performed along these directions as in Fig. 2.7 (with parity alternation along TD and constant along AD). Fig. 2.9 shows an example.

The main drawback of these methods is that the set of possible directions is finite whereas the edges may adopt arbitrary shapes. Consequently, it cannot be guaranteed that the edges will not be crossed while filtering, and thus the energy of the detail coefficients may not be minimized. Moreover, large HP coefficients always appear when filtering along the direction orthogonal to the edge (as in Fig. 2.9(b)). Our method deals with the avoidance of edges during the *prediction step*, hence the filtering



(a) TD parallel to edge. (b) TD orthogonal to edge.

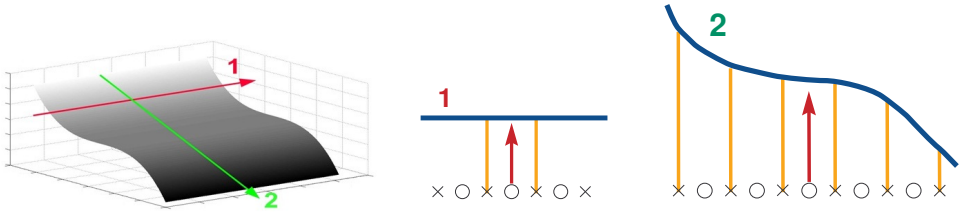
Figure 2.9: Directionlet transform inside a block with a diagonal edge.

direction is chosen only based on our signal model.

We say that a wavelet transform has filters of p vanishing moments (VM) when the transform coefficients expressed as z-transform have p zeros in π. This means that any polynomial signal up to the order $p-1$ can be represented completely in scaling space, and thus the detail coefficients have zero intensity for that signal. In a general case, the VM that the filters would need in order to correctly approximate a signal would vary depending on the filtering directions. Fig. 2.10 shows how a particular 2D signal can be filtered in many directions. For the vertical direction (number 2 in the image), the filters would need many VM in order to provide a good approximation to the signal; however, along the horizontal direction (number 1 in the image) the number of VM needed is minimized. Nevertheless, it may not be always possible to find this direction in a real image, and even if it were possible, it may happen that the shape of the signal in the orthogonal direction (along which the transform also has to be performed) requires filters with more VM. In such cases large HP coefficients would still appear.

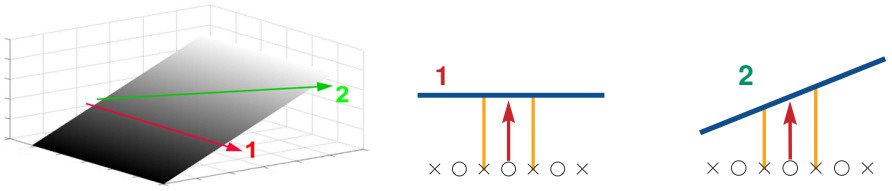
Yet, in our work, DMs are assumed to be planar functions (except around edges), hence, as can be observed in Fig. 2.11, regardless of the direction chosen, the signal intensity varies linearly and thus, filters with two VM will always be able to approximate the signal correctly.

Since the transform direction has no influence on the filtering result (along every direction the signal can be perfectly approximated by using 2 VM filters), in our case directional filtering is not necessary. Hence, selecting a *standard separable splitting*



(a) Signal (b) Few VM needed (c) Many VM needed

Figure 2.10: Example of approximation of a signal along different directions



(a) Plane (b) 1 VM needed (c) 2 VM needed

Figure 2.11: Example of approximation of a planar signal along different directions

approach, such as the one shown in Fig. 2.8, while using specific edge-avoiding filtering operations (with 2 VM filters) is sufficient for our purposes. Thus, our choice of parity is not adaptive to the location of the edges (while the filtering operations will be adaptive).

2.3 Proposed Prediction Step

Once the pixels of the image are divided into the even and odd sets, the next step of the transform is the prediction of the odd pixels. The prediction step is the most important part of our algorithm for it is only during this operation that the energy of the detail coefficients can be minimized. If this is done, then high compression rates can be achieved with minimum distortion on the resulting synthesized image.
As has been introduced in Section 2.1, for each odd pixel $(i, j) \in \mathcal{O}$ from the image $X(i, j)$ an approximation is built by combining pixels from the even set ($\mathcal{E}$). Then the corresponding detail coefficient is computed as the error of such an approximation.

It can be inferred from (2.6) that for each odd (i, j) there are two elements that need to be specified: $\mathcal{N}_{i,j}$ and $\mathbf{p}_{i,j}$. The set of pixels $\mathcal{N}_{i,j}$ is the *neighborhood* of

the odd pixel (i, j), and $\mathbf{p}_{i,j}$, which contains the coefficients that weight the relative importance of each $(k, l) \in \mathcal{N}_{i,j}$ on the prediction of (i, j), can be seen as a filter. Also, as discussed in previous sections, we have assumed that DMs are *piece-wise planar* signals, i.e., images formed by planes that are separated by arbitrarily shaped edges. Our algorithm bears these two concepts in mind (**Edges** and **Planes**) in order to make an accurate prediction. In our work the selection of each neighborhood $\mathcal{N}_{i,j} \subset \mathcal{E}$ addresses the edge avoidance issue (all the evens in $\mathcal{N}_{i,j}$ must belong to the same plane as (i, j)), while the filters $\mathbf{p}_{i,j}$ are specifically designed for planar signals.

Non-adaptive techniques use the same filtering strategies for all odd pixels, i.e., $\mathbf{p}_{i,j} = \mathbf{p}\ \forall (i, j) \in \mathcal{O}$ and $\mathcal{N}_{i,j}$ has the same structure around every pixel (i, j). For example, during the prediction operation in a 5/3 DWT [Mal08] (using the horizontal direction as TD) $\mathcal{N}_{i,j} = \{(i, j-1), (i, j+1)\}\ \forall (i, j) \in \mathcal{O}$ and $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}\ \forall (i, j) \in \mathcal{O}$. On the contrary, in our algorithm we take advantage of the flexibility of the lifting procedure to allow different $\mathbf{p}_{i,j}$ and $\mathcal{N}_{i,j}$ to be chosen for each (i, j). This makes it possible to avoid situations where the intensity of the detail coefficients would be high if a non-adaptive technique were used. However, while non-adaptive transforms may be able to simplify the prediction step into a few operations, the fact that our algorithm processes each odd node separately implies the use of more computational resources. Two other important drawbacks of adaptive techniques are that the transform loses its orthogonality, and that side information must be sent, along with the transform coefficients, to the decoder. How these drawbacks affect the final result is thoroughly studied in Chapter 5.

The steps taken prior to the application of (2.6) are the selection of $\mathcal{N}_{i,j}$ and the computation of $\mathbf{p}_{i,j}$. These steps are followed in this order for each odd node individually. As will be seen in the following sections, the coefficients of the filter $\mathbf{p}_{i,j}$ depend directly on the position of the pixels that form $\mathcal{N}_{i,j}$, and the selection of these depends directly on the location of the edges in the image with respect to the odd (i, j). Before getting into the details of how to design these two elements it is important to discuss some characteristics of the distribution of the edges in the image in order to find the best procedure.

The following subsection explains the blocks inside the *prediction step* based in what is considered an efficient strategy in terms of computational resources consumption.

2.3.1 Predicting Operation Blocks

Since, clearly, the vast majority of the pixels will not be located around edges but deep inside the planes, it may be useful to see which $\mathcal{N}_{i,j}$ structure would be beneficial in such cases. By describing a straightforward operation that processes these odds separately and rapidly, while ensuring that the neighborhood structures are not changed (with respect to the ones that would be selected in case that these pixels were not separated from the rest), the transform can be computed considerably faster.

Let (i, j) be an odd pixel located far away from any edge, so that all the even pixels around it belong to the same plane in the image (i.e., all could be included in $\mathcal{N}_{i,j}$). As illustrated by Fig. 2.11, most combinations of these evens, together with the appropriate filters, can provide a perfect approximation of the odd's value, i.e., so that $d(i, j) = 0$. This means that $\mathcal{N}_{i,j}$ can be formed in a great variety of ways without altering the value of the detail coefficient. Nevertheless, the choice of evens does have an impact on the performance of the overall system depending on how close to orthogonality the resulting transform is. As explained in Chapter 5, most of the encoders used for coding the transform coefficients, introduce less errors for transforms that are closer to orthogonality. Therefore, the algorithm will try to imitate, as far as possible, the neighborhood structures of the 5/3 DWT, which is known to be close to orthogonal. Hence, the combination of evens that is chosen in these cases is the one that is formed by the two immediate neighbors along TD.
This means that when computing the transform along rows (TD = horizontal), the odd (i, j) will be predicted by using the value of the pixels in $(i-1, j)$ and $(i+1, j)$ (i.e., $\mathcal{N}_{i,j} = \{(i-1, j), (i+1, j)\}$); and when TD corresponds to the vertical direction, by using $\mathcal{N}_{i,j} = \{(i, j-1), (i, j+1)\}$. For this combination of evens, the filter expressions given in Section 2.3.3 imply that $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}$. We refer to the two immediate neighbors of (i, j) along TD as its *cousins*. This result means that if the transform was applied to an infinite 2D planar signal (no edges or image boundaries) the algorithm would be equivalent to the 5/3 DWT.

These neighborhood structures and filter coefficients will be chosen for any odd node that has no edges between it and its two *cousins*. Other combinations will be chosen only when edges have to be avoided and when the boundaries of the image are reached. In these cases, the procedure is adapted to the shape of the planes on the image, elsewhere the transform behaves as a 5/3 DWT. Fig. 2.12 shows an example of the

structure of the neighborhoods for pixels in the image, where some are near and some are far from edges. Similarly to *tree-based wavelets* [SO09], since in our case the connections between pixels (which are later used for filtering) are defined using a graph topology, we say that the algorithm computes a *graph-based* wavelet transform.

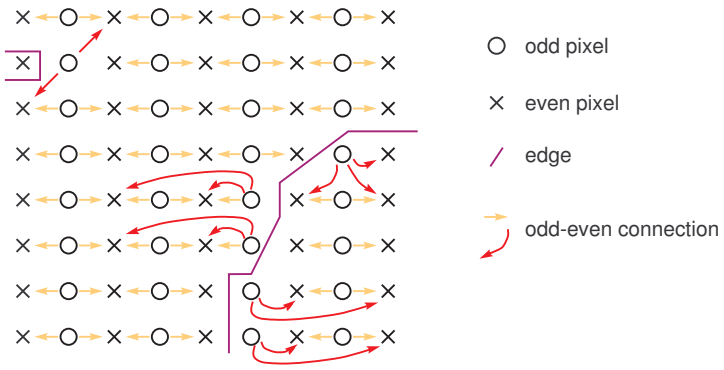


Figure 2.12: Most odds use the 5/3 DWT neighborhood structures (orange arrows). Only when edges need to be avoided other structures are used (red arrows).

We refer to the pixels whose predictions are computed as in the 5/3 DWT as *non-extrapolated*, while the pixels that are separated from any of their *cousins* by an edge are called *extrapolated* pixels. Fig. 2.13 shows an example.

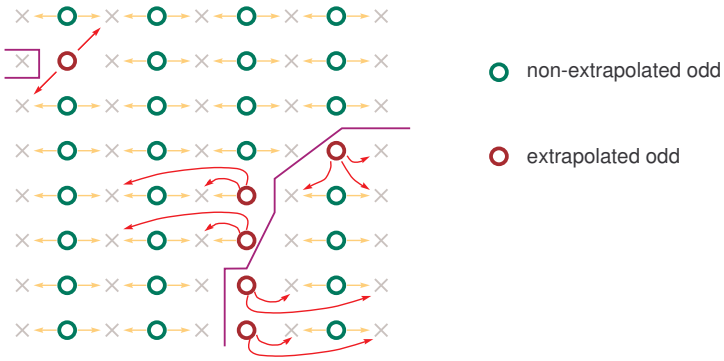


Figure 2.13: Division into extrapolated and non-extrapolated odd pixels

This procedure is very similar to the one presented in [MD08]. In that technique edges are avoided by extrapolating the value of the pixels at the other side of the edge, making the signal continuous along the transform direction. However, this method uses only evens in the same row/column (depending on TD) as the odd, whereas ours uses the information provided by all the evens in the 2D space. This makes a

difference in cases where there are not enough evens along the 1D direction to build an extrapolation of the function. This phenomenon is especially common when high levels of decomposition have been reached. In such cases, [MD08] fails to provide an accurate prediction of the odds, while our method is still able to minimize the corresponding detail coefficients by using evens in other rows/columns. Nevertheless, [MD08] can be extended to transforms that use filters with more than 2 VM, such as the 9/7 DWT [Mal08], whereas our method cannot. However, since our algorithm is designed specially for DMs, which have been assumed to follow planar functions (only require filters with 2VM), it needs not be adapted to other types of wavelet algorithms, aside from the 5/3 DWT.

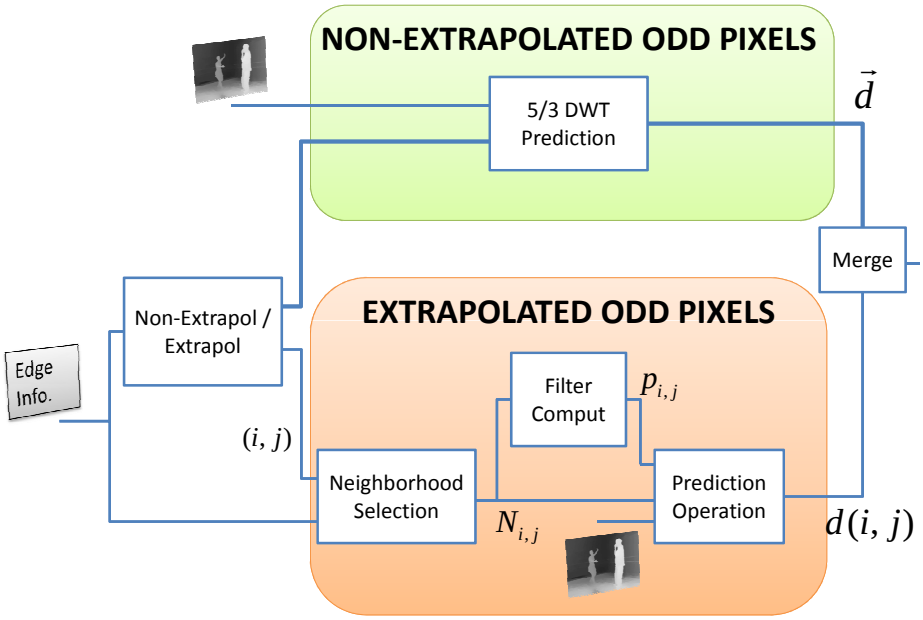


Figure 2.14: Block Diagram of the Prediction Step.

The complete prediction step works as follows (see Fig. 2.14): First the *extrapolated* pixels are pulled apart from the rest (the *non-extrapolated* ones) using the edge information. Then, for each pixel (i, j) belonging to the first group the neighborhood $\mathcal{N}_{i,j}$ is computed using the edge information. Next, with $\mathcal{N}_{i,j}$ as an input, the coefficients of filter $\mathbf{p}_{i,j}$ are calculated. Lastly, (2.6) is applied using the $\mathcal{N}_{i,j}$ and $\mathbf{p}_{i,j}$ obtained.

As for the second group of pixels, composed by the *non-extrapolated* odds, they are treated in a different block that computes the prediction rapidly by using vector operations. Finally, the values obtained for all the pixels are grouped together. The output of this operation is a signal formed by all the detail coefficients obtained. Note that the resulting transform coefficients would have equal value if all the pixels were treated as *extrapolated* pixels, since the choice of $\mathcal{N}_{i,j}$ and $\mathbf{p}_{i,j}$ for *non-extrapolated* pixels would not vary.

In the following subsections and chapter each block is described in detail. The block dedicated to the *non-extrapolated* odds is explained in Section 2.3.2. Section 2.3.3 presents the equations used for the computation of the coefficients of $\mathbf{p}_{i,j}$. Finally, due to the sophistication of the procedure that defines each $\mathcal{N}_{i,j}$, this is described separately in Chapter 3. Even though the selection of $\mathcal{N}_{i,j}$ must be done prior to the computation of the filter $\mathbf{p}_{i,j}$, the latter is explained first for a simpler comprehension of the algorithm.

2.3.2 Non-Extrapolated Pixels Prediction

For all the odd pixels whose prediction does not require the avoidance of edges a standard procedure equivalent to the one computed in 5/3 DWT is done. Due to the simplicity of the process, the prediction consists in a simple vector operation. Let the vector $\mathbf{v}(k)$, for $k = 1 \dots M$, contain the value of the M *non-extrapolated* odd pixels. Then, given $\mathbf{v}(k) = X(i,j)$, we define the vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ as follows: in the case of horizontal filtering, $\mathbf{n}_1(k) = X(i-1,j)$ and $\mathbf{n}_2(k) = X(i+1,j)$. In the case of vertical filtering, $\mathbf{n}_1(k) = X(i,j-1)$ and $\mathbf{n}_2(k) = X(i,j+1)$. This way $\mathbf{n}_1(k)$ and $\mathbf{n}_2(k)$ contain the intensity of the *cousin* pixels of $\mathbf{v}(k)$.

Since both *cousins* are equidistant to the odd (which is located exactly between them) the weight of each pixel in the predicting operation must be the same. Therefore $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}$. This is consistent with the result that the equations for the computation of filters (explained in detail in Section 2.3.3) would provide. It is also equal to the 5/3 DWT filters. The predicting operation is then computed as:

$$\mathbf{d} = \mathbf{v} - \frac{1}{2}(\mathbf{n}_1 + \mathbf{n}_2). \tag{2.8}$$

The resulting vector is formed by the detail coefficients corresponding to the pixels in $\mathbf{v}$, i.e., $\mathbf{d}(k) = d(i,j)$.

2.3.3 Prediction Filters

In our algorithm, and given the piece-wise planar assumption about DMs, filters $\mathbf{p}_{i,j}$ are designed to approximate planar 2D signals with no error (i.e., they have 2 VMs).

Chapter 3 explains in detail how $\mathcal{N}_{i,j}$ is formed for each odd pixel (i,j) so that there is no filtering across edges. At this point it is only necessary to know that, given an odd node (i,j), once the subset $\mathcal{N}_{i,j}$ is defined, all the pixels involved in the prediction operation ((i,j) and the evens that form $\mathcal{N}_{i,j}$) are assumed to belong to the same plane in the image. Knowing that, one can use the position and intensity of the pixels in order to calculate the coefficients of such plane. Then an approximation to the odd's value can be built by extrapolating the intensity that the obtained plane would have in the odd's position. The only requirement to make this possible is that $\mathcal{N}_{i,j}$ must be formed by either two even pixels collinear with (i,j) or three non-collinear even pixels. Refer to Appendix B for a detailed proof of this fact. We say that three nodes are collinear if and only if a straight line can be defined that passes through their positions in 2D space. Fig. 2.15 illustrates the idea, while this can be mathematically expressed as follows:

$$collinear\{(k_1,l_1),(k_2,l_2),(k_3,l_3)\} \Leftrightarrow \exists A,B,C \mid A\cdot k_i + B\cdot l_i + C = 0,\ \forall i = 1,2,3. \tag{2.9}$$

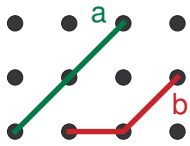


Figure 2.15: The three pixels connected by the green line (a) are collinear, whereas the pixels connected by the red line (b) are not.

Since $\mathcal{N}_{i,j}$ can have different configurations (they can be formed by either 2 or 3 pixels), there must be also different ways for computing the filter $\mathbf{p}_{i,j}$. We consider two cases.

Firstly, if $\mathcal{N}_{i,j} = \{(k_1,l_1),(k_2,l_2)\}$, where (k_1,l_1), (k_2,l_2) and (i,j) are **collinear**, the filter can be designed so that it computes the parameters of the line formed by the two even nodes and extrapolates its value in the position (i,j). The resulting value

is an approximation of $X(i,j)$. Note that this operation is valid both when the odd pixel is located between the two evens and when both evens are located at one side of the odd. Fig. 2.16 illustrates the idea.

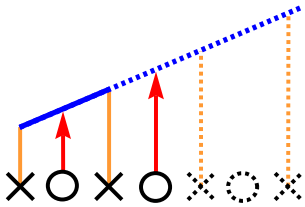


Figure 2.16: Approximation of odd values using two collinear evens.

Mathematically, the filter's expression will be the following:

$$\mathbf{p}_{i,j}(\mathcal{N}_{i,j}) = \begin{bmatrix} i & 1 \end{bmatrix} \cdot \begin{bmatrix} k_1 & 1 \\ k_2 & 1 \end{bmatrix}^{-1}. \tag{2.10}$$

In case $k_1 = k_2$, the matrix cannot be inverted; however the filter can still be computed using ordinate coordinates (j, l_1 and l_2) instead of the abscissa ones (i, k_1 and k_2). The resulting equation is the following:

$$\mathbf{p}_{i,j}(\mathcal{N}_{i,j}) = \begin{bmatrix} j & 1 \end{bmatrix} \cdot \begin{bmatrix} l_1 & 1 \\ l_2 & 1 \end{bmatrix}^{-1}. \tag{2.11}$$

Note that the resulting filter is the same as that in [MD08] when computing a transform equivalent to 5/3 DWT. However, in [MD08], the three pixels involved in the operation (odd and two evens) need to be located in the same row/column (depending on TD), whereas, in our algorithm, the distribution of these three pixels in the 2D space is not fixed, as long as they are collinear. For example, in the case of having three pixels that define a diagonal line (with respect to TD), the filter coefficients can still be computed using the coordinates of their positions and (2.10).

Secondly, in the case that $\mathcal{N}_{i,j}$ is formed by three **non-collinear** evens ($\mathcal{N}_{i,j} = \{(k_1,l_1),(k_2,l_2),(k_3,l_3)\}$) a similar operation can be done. This time, the parameters of the plane formed by the three evens are computed, and the approximation consists of an extrapolation of that plane's equation in the odd's position. Fig. 2.17 illustrates the idea.

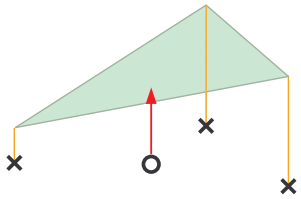


Figure 2.17: Approximation of an odd value using three non-collinear evens.

Mathematically, the filter's expression will be the following:

$$\mathbf{p}_{i,j}(\mathcal{N}_{i,j}) = \begin{bmatrix} i & j & 1 \end{bmatrix} \cdot \begin{bmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \\ k_3 & l_3 & 1 \end{bmatrix}^{-1} . \tag{2.12}$$

The method in [MD08] does not provide filtering strategies that use non-collinear pixels, and so, this kind of filtering cannot be performed using their algorithm. Both (2.10) and (2.12) provide filters with two vanishing moments. Therefore, if the signal was completely planar the approximation would have no error, and hence the energy of the detail values would be minimized. Also, note that if $\mathcal{N}_{i,j}$ was formed by the two *cousins* of (i,j), the filter obtained from (2.10) will be $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}$, which is consistent with the method used for the non-extrapolated pixels.

2.4 Updating Step

The updating operation is applied to all the pixels in the even set ($\mathcal{E}$) in order to correct potential aliasing effects introduced by the sub-sampling. For each even pixel (k,l) in the image the updating operation is computed using the expression (2.7).

Contrary to what has been said about the prediction step, in this case neither $\mathbf{u}_{k,l}$ nor $\mathcal{N}_{k,l}$ are freely chosen. Instead, they are determined by the selection of the filters $\mathbf{p}_{i,j}$ and neighborhood $\mathcal{N}_{i,j}$ for all the pixels $(i,j) \in \mathcal{O}$. These elements are connected as follows. The neighborhood of each even (k,l) is composed by all the odd pixels whose neighborhoods included (k,l). Mathematically: $\mathcal{N}_{k,l} = \{(i,j) | (k,l) \in \mathcal{N}_{i,j}\}$. As for $\mathbf{u}_{k,l}$ it is computed using the prediction filters of all the odds in $\mathcal{N}_{k,l}$ following the orthogonalizing design presented in [SO09]. Since this sophisticated method has

not been part of the research activity presented in this work, a short summary is presented here (refer to [SO09] for a complete discussion).

First, the image (whose dimensions are $M \times N$) is re-shaped as a $MN \times 1$ vector by creating the index $m_{v,w} = (v-1)M + w$, where (v, w) represents the position of a pixel in the image. Following this notation, the neighborhood of an even pixel $m_{k,l}$ can be written as $\mathcal{N}_{m_{k,l}} = \{z_1, z_2 \dots z_n\}$, where each z_s represents the index of an odd pixel. Let $\mathbf{P}_{k,l}$ be the $MN \times n$ matrix having the prediction vectors of the neighbors of (k, l) as its columns, i.e., $\text{column}_s(\mathbf{P}_{k,l}) = \mathbf{p}_{z_s}$ for $s = 1, 2, \dots, n$. Then we can define the vectors $\mathbf{u}^*_{k,l}$ and $\mathbf{p}^*_{k,l}$ as:

$$\begin{aligned} \mathbf{u}^*_{k,l} &= (\mathbf{u}_{m_{k,l}}(z_1), \mathbf{u}_{m_{k,l}}(z_1), \dots \mathbf{u}_{m_{k,l}}(z_n)), \\ \mathbf{p}^*_{k,l} &= (\mathbf{p}_{z_1}(m_{k,l}), \mathbf{p}_{z_2}(m_{k,l}), \dots \mathbf{p}_{z_n}(m_{k,l})). \end{aligned} \tag{2.13}$$

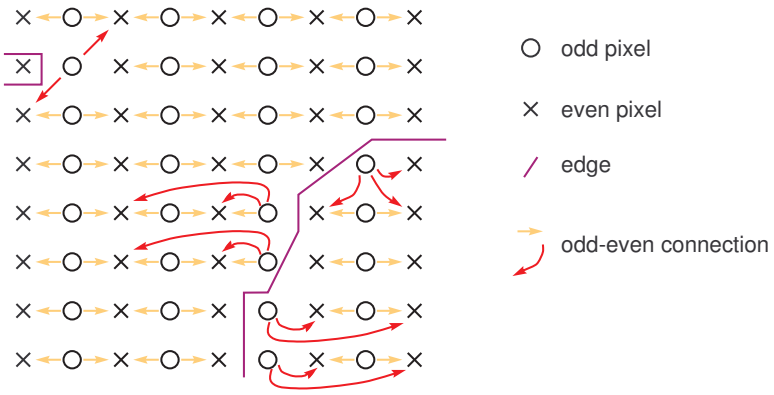
Using this notation, the updating filters are defined by:

$$\mathbf{u}^*_{k,l} = -(\mathbf{P}^t_{k,l}\mathbf{P}_{k,l})^{-1}\mathbf{p}^*_{k,l}. \tag{2.14}$$

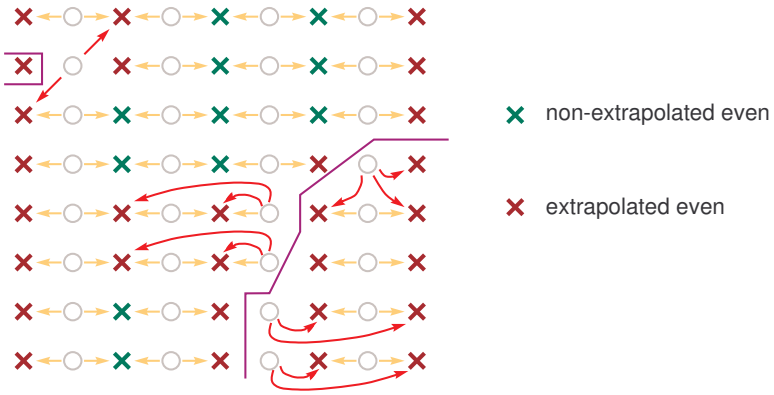
As in the prediction step, the computation of these elements needs to be done for each pixel independently of the rest. Even though the choice of each neighborhood is now a much more straight forward operation than during the prediction step, the whole process may still be computationally demanding. Therefore, ways to optimize the procedure are used. As has been shown in previous sections, the transform behaves as a 5/3 DWT for most of the pixels in the image. Just as all the *non-extrapolated* odds used identical neighborhood structures and filter coefficients, the $\mathbf{u}_{k,l}$ and $\mathcal{N}_{k,l}$ of the evens that are used only by *non-extrapolated* odds need not be computed for each pixel separately.

During the prediction step, the sets of *non-extrapolated* and *extrapolated* evens are formed as the odd pixels create their neighborhoods. The set of *non-extrapolated* set is formed by evens that are only used in the prediction of both their *cousins*, which must also be *non-extrapolated* odds. This means that an even that is used by only one odd pixel, even when this is a *non-extrapolated* odd pixel, will belong to the *extrapolated* evens set. Fig. 2.18 illustrates the process.

When the updating step takes place the two sets of pixels are separated. The updating of each *extrapolated* even is computed individually using the neighborhoods $\mathcal{N}_{k,l}$ formed as explained above and in (2.14) and (2.7). As for the *non-extrapolated* evens,



(a) Example of the neighborhood structure.



(b) *Extrapolated* and *non-extrapolated* evens.

Figure 2.18: Separation into *extrapolated* and *non-extrapolated* evens.

a vector operation similar to the one explained in Section 2.3.2 is applied. Following the equations above, for each even (k, l) in this set, $\mathcal{N}_{k,l}$ is formed by its two *cousins*, while $\mathbf{u}_{k,l} = \{\frac{1}{4}, \frac{1}{4}\}$ for all the pixels in this set.

Once the pixels of both sets have been updated, they are merged again, forming the smooth signal.

2.5 Filter Normalization

Although the expressions introduced in Section 2.1.2 for the predicting and updating operation are completely valid for building a wavelet transform, they usually introduce

one problem, namely, the gain for the high pass (HP) filter, i.e., prediction filter, can be higher than the one for the low pass (LP) filter, i.e., updating filter; this causes the HP coefficients to be scaled higher than the LP coefficients. Thus, when quantifying the resulting transform coefficients for their encoding and transmission, less quantization error is introduced in the HP band and more to LP band, which is suboptimal. This can have negative effects when reconstructing the image.

This problem has been discussed in depth in previous studies, such as [SO09], [DS97] and [Swe96]. The most popular solution consists in **normalizing** the filters so that the total gain for the HP and the LP filters remains equal. Then, the equations computed for the prediction and updating operations are:

$$\begin{aligned} d(i,j) &= X(i,j) - \tfrac{1}{\rho_{i,j}} \textstyle\sum_{(k',l')\in\mathcal{N}_{i,j}} \mathbf{p}_{i,j}(k',l') \cdot X(k',l'), \\ s(k,l) &= X(k,l) + \tfrac{1}{\lambda_{k,l}} \textstyle\sum_{(i',j')\in\mathcal{N}_{k,l}} \mathbf{u}_{k,l}(i',j') \cdot d(i',j'). \end{aligned} \tag{2.15}$$

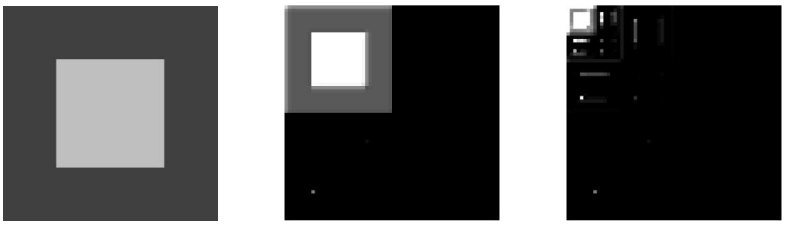
Where $\rho_{i,j}$ and $\lambda_{k,l}$ stand for the normalizing values. In [SO09], for example, these parameters are obtained as follows (using the notation of Section 2.4):

$$\begin{aligned} \rho_{i,j} &= \|\mathbf{p}^*_{i,j}\|, \\ \lambda_{k,l} &= \|\mathbf{P}^t_{k,l} \cdot \mathbf{u}^*_{k,l}\|. \end{aligned} \tag{2.16}$$

However in our work we have found that this method has one serious drawback: since each coefficient is normalized using a different factor, the smooth coefficients (approximation) do not form a PWP signal after the first decomposition. This phenomenon appears when using normalizing strategies where $\lambda_{k,j}$ is not constant $\forall(k,l) \in \mathcal{E}$. This is a major problem in our work, since the algorithm, which assumes a PWP input, is applied iteratively over that approximation. When the signal does not follow a planar shape, the filters designed in Section 2.3.3 cannot approximate the signal correctly, and thus HP coefficients appear.

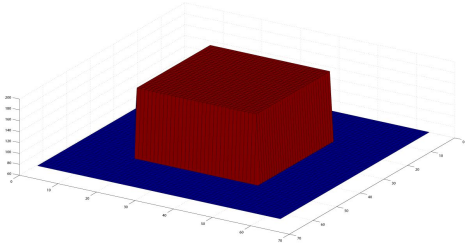
Fig. 2.19 shows an example where a piece-wise constant image (which is a particular case of PWP) is transformed using this kind of normalization. Since the edges are very well defined and the image is completely PWP, our transform should be able to perfectly approximate the signal, so that HP coefficients tend to be all zero. Also, the resulting smooth signal should consist of a reduced version of the original image, i.e., should be PWP for all levels of decomposition. However the evolution of

the transform coefficients as the decomposition increases shows that this is not happening, but, on the contrary, the smooth signal is further from PWP as the level of decomposition is increased. This phenomenon can be observed more clear in Fig. 2.20, where the coefficients in Fig. 2.19 are projected in a 3D space. Consequently, different normalization methods must be found.

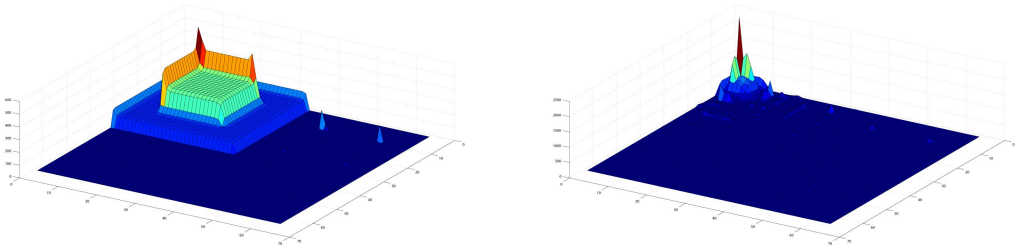


(a) Original Image (b) 1 level of decomposition (c) 3 levels of decomposition

Figure 2.19: Transform coefficients when using standard normalization.



(a) Original Image

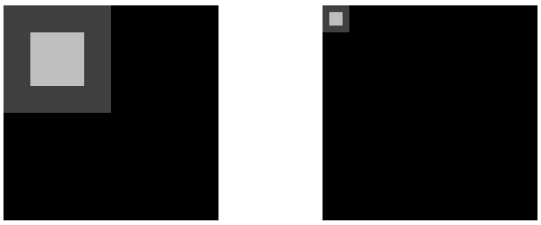


(b) 1 level of decomposition (c) 3 levels of decomposition

Figure 2.20: 3D representation of the transform coefficients when using standard normalization.

In our work $\rho_{i,j}$ is computed as shown in (2.16). However, the even pixels are updated using a constant value $\lambda_{k,l} = \lambda \ \forall (k,l) \in \mathcal{E}$. This value is set as the one used for the *non-extrapolated* evens, $\lambda = 0.8452$, which is the value obtained when using the 5/3 DWT filters and (2.16). This way, as shown in Fig. 2.21 and Fig. 2.22, the coefficients

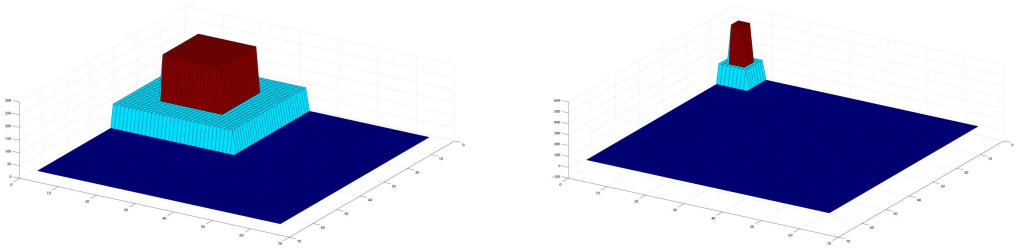
of the lowest pass bands of a PWP image (or a piece-wise constant, as in the example) always form a PWP signal.



(a) 1 level of decomposition

(b) 3 levels of decomposition

Figure 2.21: Transform coefficients when using our normalization.

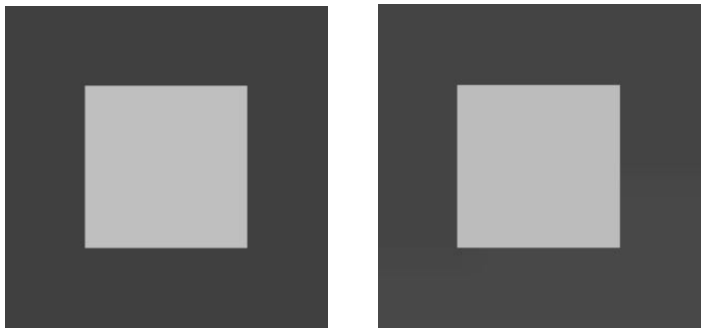


(a) 1 level of decomposition

(b) 3 levels of decomposition

Figure 2.22: 3D representation of the transform coefficients using our normalization.

Thanks to this technique (see Fig. 2.23), the scaling effect is avoided, while no additional errors are introduced. Fig. 2.23(b) shows the result obtained when no normalization is used. Even though, in this case, the edges are perfectly preserved, the colors of the image look slightly brighter than in the original image due to the scaling effect. When using a standard normalization (see Fig. 2.23(c)), this problem is avoided; however, severe artifacts appear because of the mismatch between the shape of the signal and the model that is assumed (PWP). When normalizing all the updating filters using a constant factor (see Fig. 2.23(d)), these problems are avoided, and so the transform preserves both the edges and the colors of the original image. This phenomenon has a strong impact in the final PSNR value of the reconstructed image, as shown in Table 2.1.



(a) Original Image (b) Using no Normalization

(c) Using standard Normalization (d) Using our Normalization

Figure 2.23: Reconstructed images for 0.25bpp and 3 levels of decomposition.

Normalization approach / bpp	1 bpp	2bpp	3bpp
No normalization	21.72	31.46	41.97
Standard normalization	23.07	33.41	43.57
Our normalization	31.99	42.77	44.37

Table 2.1: PSNR (measured in dB) obtained when coding Fig. 2.20(a) at different bit rates, using the different normalization approaches.

Chapter 3

Neighborhood Computation

Since the prediction filters presented in Chapter 2 are specifically designed for planar signals, it must be ensured that all the pixels involved in the prediction operation belong to the same plane in the image. If this was done, the approximation built using those filters could be very accurate and thus the intensity of the detail coefficients would be very low. The selection of each neighborhood $\mathcal{N}_{i,j}$ aims to achieve this goal.

As discussed in previous sections, for each odd pixel (i, j), the structure of the subsets of even nodes $\mathcal{N}_{i,j}$ is not fixed, but, on the contrary, *any* even pixel in the image potentially belongs to $\mathcal{N}_{i,j}$. However, the filter expressions used (Section 2.3.3) imply that these sets are formed by combinations of either two or three pixels. This can lead to many possible neighborhood structures for each odd pixel; hence, among all the possibilities one must be selected. The process for the selection of each $\mathcal{N}_{i,j}$ is done for each odd node (i, j) independently of the others and consists of two steps: the first one finds which even pixels are located in the same plane as (i, j) (we name this *the pixel discarding step*); among all these pixels, the second step finds the group (pair or trio) of evens that is assumed to provide a more accurate approximation (we name this *the neighborhood choice step*).

The first step requires information about the shape and position of the planes in the image. This is obtained by the use of an edge detector. The second one works based on a certain criterion. Both elements (edge information and selection criterion) are shared by the encoder and the decoder. While the criterion is part of the algorithm and so it needs not be specified to the decoder by the encoder, the edge information

must be sent as side information. This way the same neighborhood structures are used in both the forward and inverse transform and the process can be properly reversed. Section 3.1 explains the pixel discard step, while Section 3.2 shows how the most appropriate combination of evens is selected among all possible. In case that there is not a single valid combination of even pixels for the approximation of an odd, we say that the pixel is *isolated* in its plane. Section 3.3 shows the procedure that is followed in such cases.

Note that in this thesis we describe the algorithm to identify the best neighborhood set in terms of a metric computation and a minimization, but the number of likely edge positions is finite so it will be possible to select $\mathcal{N}_{i,j}$ based on simple look-up tables.

3.1 Pixel Discard

In order to determine which pixels are located in each plane, the position of the edges in the image is used. Instead of using existing edge detection methods, in this work, a novel edge detector that suits the needs of our algorithm has been designed; Appendix A describes it in detail. The most important feature of this edge detector is that it provides information about the presence of edges between each pixel and its six immediate neighbors (i.e., the ones located next to it in the horizontal, vertical, and both diagonals directions) by extending the pixel grid as shown in Fig. 3.1. We will refer to this new grid as the *pixel and edge grid*.

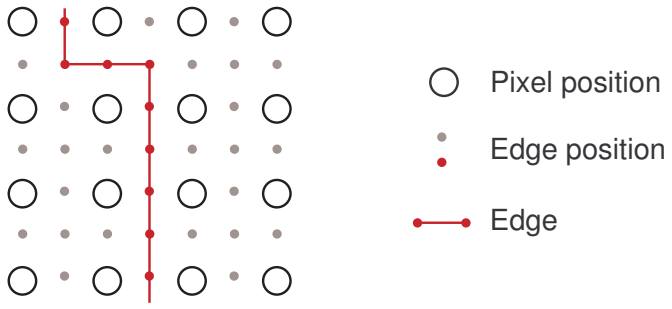


Figure 3.1: Example of the extended pixel and edge grid

This edge grid is very similar to the one presented in [MD08], in which edges where located in a grid dual to the pixels grid. The resemblances and differences between

this method and ours are discussed in Appendix A.

Thanks to this technique it is trivial to discern whether a pixel is separated from its immediate neighbors by an edge or not. However, in order to select all the evens that are located in the same plane as the odd to be predicted, the algorithm needs to extend this information to pixels located at greater distances. Section 3.1.1 explains the ideas on which the extension of the edge information is based. However, for computational reasons, the algorithm does not work exactly as presented there, but computes the information in a faster and simpler way while being consistent with its main principles. The actual operation of the algorithm is fully described in Section 3.1.2.

3.1.1 Edge Grid Expansion

Given information about the presence of edges between each pixel and its immediate neighbors, we seek to know whether any pair of pixels belong to the same plane in the image. First of all, the following assumption is made:

Fact 1 *If there is not an edge between two pixels then both of them belong to the same plane.*

Note that this is not an equivalence. Since edges can have totally arbitrary shapes, it might happen that an edge is found between two pixels of the same plane. The keyword is **between**. As can be observed in Fig. 3.2, depending on the path that we draw from one pixel to another we may or may not find edges even though both pixels belong to the same plane. However, we will always find an edge between a pair of pixels that belong to different planes regardless of the path that is followed. Given two pixels, in order to ensure that they are part of the same plane one would need to look at all the possible paths that could connect them until a path which crossed no edges was found. Nevertheless, this is computationally impractical, and, therefore, a fixed method that easily designs paths between pixels is needed.

Let (i, j) and (k, l) be two pixels in the image. We define $e_{(i,j)(k,l)}$, a variable that is equal to 1 in case there is an edge between both pixels and 0 otherwise. Also, let $\mathcal{S}^n_{i,j}$ denote the set of pixels surrounding of (i, j) of level n. Mathematically $\mathcal{S}^n_{i,j} =$

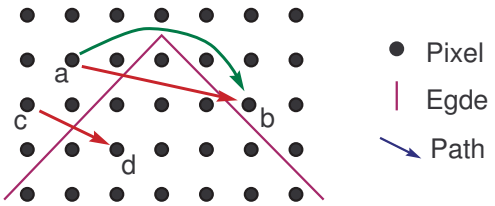
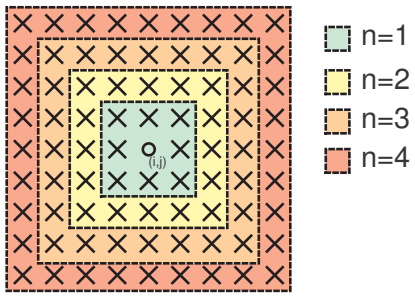


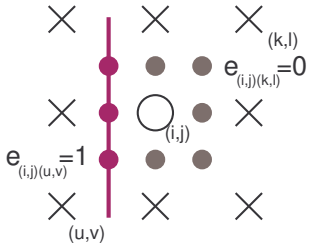
Figure 3.2: Pixels a and b are located in the same plane; depending on the path an edge is found between them or not. Pixels c and d are located in different planes; all the paths that connect them cross edges.

$\{(k,l) \mid dist[(i,j),(k,l) = n\}$ where *dist* measures the distance between pixels as shown in Fig. 3.3(a).

Finally, define $\mathcal{A}^n_{i,j} = \{e_{(i,j)(k,l)}\ \forall (k,l) \in \mathcal{S}^n_{i,j}\}$. Note that $\mathcal{A}^1_{i,j}\ \forall (i,j)$ is defined by the pixel and edge grid previously described (refer to Fig. 3.3(b) for an intuitive demonstration).



(a) Distribution of the pixels of each $\mathcal{S}^n_{i,j}$



(b) $\mathcal{A}^1_{i,j}$

Figure 3.3: Example of the definition of $\mathcal{S}^n_{i,j}$ and $\mathcal{A}^1_{i,j}$.

Extending Fact 1, if there is not an edge between (i,j) and (k_1,l_1) and there is not an edge between (k_1,l_1) and (k_2,l_2), this means that the three pixels belong to the same plane, thus it is only natural to assume that there is no edge between (i,j) and (k_2,l_2). Mathematically:

$$e_{(i,j)(k_2,l_2)} = e_{(i,j)(k_1,l_1)} \vee e_{(k_1,l_1)(k_2,l_2)}. \tag{3.1}$$

Where $\vee$ stands for the mathematical operation *"or"*. We can say that a path from (i,j) to (k_2,l_2) that passes through (k_1,l_1) has been drawn. In our algorithm paths are drawn by connecting adjacent pixels. Fig. 3.4 shows the shape of the paths that the algorithm would follow in order to detect the presence of edges between a central

pixel (i, j) and any other pixel on the image. Then, if there is no edge between any pair of adjacent pixels along the path that connects (i, j) and (u, v) it is assumed that there is no edge between them, and, consequently, that they belong to the same plane.

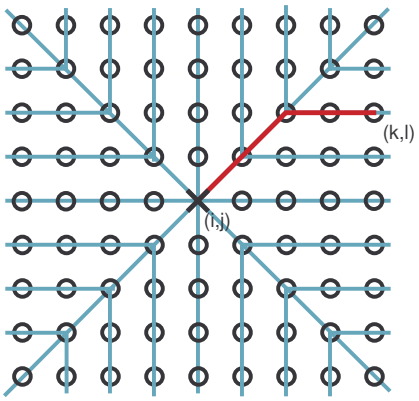


Figure 3.4: Example of the creation of paths for a certain (i, j). In red, path from (i, j) to (k, l)

Mathematically speaking, in order to know if there is an edge between two pixels, e.g., (i, j) and $(u, v) \in \mathcal{S}_{i,j}^{n}$, the following operation is computed:

$$\begin{aligned} e_{(i,j)(u,v)} &= e_{(i,j)(w,z)} \vee e_{(w,z)(u,v)}, \\ (w, z) &= \underset{(k,l)\in\mathcal{S}_{i,j}^{n-1}}{\operatorname{argmin}} \left[\|(u, v) - (k, l)\| \right]. \end{aligned} \tag{3.2}$$

This way, $\mathcal{A}_{i,j}^{n}$ $\forall n$ can be obtained using only $\mathcal{A}_{i,j}^{n-1}$ and $\mathcal{A}_{k,l}^{1}$ $\forall (k, l) \in \mathcal{S}_{i,j}^{n-1}$ (which is known thanks to the edge detector introduced before). Consequently, the information provided by the edge detector, which only indicated the presence of edges between adjacent pixels, can be extended to any pair of pixels in the image.

Nonetheless, even though this procedure is mathematically valid, running it for every pair of pixels is not computationally practical since, for most images, the number of operations needed would be too large. Moreover, in most cases these operations will not be necessary for various reasons. Section 3.1.2 describes how, using the principles presented above, the algorithm easily discards all the even pixels that are separated by an edge from the odd pixel which is being predicted.

3.1.2 Pixel Discard Algorithm

First of all, given a certain odd pixel (i, j) not all the evens in the image are considered candidates. Among all the pixels in $\mathcal{E}$, the subset $\mathcal{Q}_{i,j} \subset \mathcal{E}$ is formed by the evens located within a certain distance of (i, j). The search area is limited to a square region around (i, j) and only the pixels inside that region can be considered for the latter formation of $\mathcal{N}_{i,j}$. This area is formed as shown in Fig. 3.5, which comprises a fixed number W of even pixels. Clearly, since images are finite signals, the pixels located close to the boundaries will have less candidate nodes than the others (card($\mathcal{Q}_{i,j} \leq W$), where card stands for the cardinality of the subset, i.e., the number of elements in the subset). The reason for limiting the number of candidates is that it is assumed that if a valid combination of pixels cannot be found using the W evens comprised inside the region defined, it means that the pixel is isolated and therefore enlarging the area of search would not provide any new valid evens but only will slow down all the algorithm.

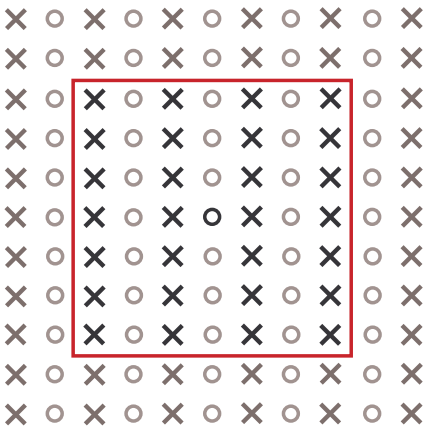


Figure 3.5: Only the evens inside the red square are considered for the prediction of the central odd

Once we know which pixels conform the set of candidate evens, $\mathcal{Q}_{i,j}$, the pixel discarding step can start. As explained, all the pixels that are separated from the odd in question by an edge must not be considered further. Therefore, the resulting subset $\mathcal{R}_{i,j} \subset \mathcal{Q}_{i,j}$ only includes even pixels that belong to the same plane as (i, j).

Contrary to what may have been inferred from the explanation of how the edge information is used, pixel discarding is not *pixel-oriented* but *edge-oriented*. This means that instead of checking the path from (i, j) to each even, the algorithm seeks

for the edges in the area of interest and then discards the evens that are separated from the odd by each one of these edges. Since the structure of the paths is fixed one can trivially identify which evens will be assumed to be located in a different plane given a certain edge.

To this purpose it must be remembered how the paths are drawn (Fig. 3.4). Bearing this in mind, clearly, different edge positions imply the discarding of different number of evens. For example, an edge located at $(i+0.5, j)$ (e.g., edge **a** in Fig. 3.6) leads to discarding all the pixels with position (h, j) $\forall h > i$ (pixels inside the pink area next to **a**), because all the paths that would be drawn from the odd to any of these pixels would cross at least that edge. In a similar way, an edge located in a position such as $(i-1.5, j-1.5)$ (e.g., edge **b** in Fig. 3.6) will not only lead to discarding all the pixels located in the diagonal line that the vector from the odd to the edge defines (pixels with position $(i-\alpha, j-\alpha)$ $\forall \alpha \geq 2$) but also all the horizontal and vertical paths that are derived from it ($(i+\delta, j+\rho$ $\forall \delta, \rho \geq 2$, pixels inside the pink area next to **b**). Other similar cases can be found depending on the position of the edges. If all these discarded evens are stored in the subset $\mathcal{T}_{i,j} \subset \mathcal{Q}_{i,j}$, then the output of this block is $\mathcal{R}_{i,j} = \mathcal{Q}_{i,j} \setminus \mathcal{T}_{i,j}$; where $\setminus$ represents the difference of sets.

It is worth noting that some edge positions will never have an effect on the pixel discarding. For example, edges located between two odd pixels (e.g., edge **c** in Fig. 3.6); can be ignored since, given the shape of the paths used, no evens would be discarded by that edge. Chapter 4 explains in detail which consequences can these, and other similar cases, have on the performance of the algorithm and how they are dealt with.

Summarizing, the pixel discarding step operates as follows: Firstly, the set of candidates is restricted to the evens located inside a certain region around the odd pixel (generation of set $\mathcal{Q}_{i,j}$). Then, all the edges located inside the region are considered. Due to the structure of the paths each edge implies the discard of certain evens. All these evens are grouped into a new set of eliminated evens ($\mathcal{T}_{i,j} \subset \mathcal{Q}_{i,j}$). Once these are eliminated from $\mathcal{Q}_{i,j}$, the remaining even pixels (which form $\mathcal{R}_{i,j}$) are assumed to belong to the same plane as the (i, j) and are used as the input of the following step, namely the *neighborhood choice*, which, from among all of them, chooses the pixels that will form the neighborhood $\mathcal{N}_{i,j}$, used in the predicting operation.

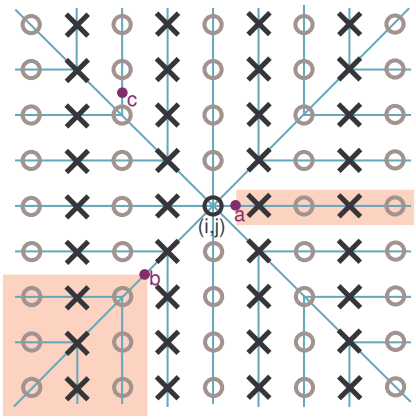


Figure 3.6: Evens inside the pink areas are discarded by the edges **a** and **b** (positions $(i+0.5, j)$ and $(i-1.5, j-1.5)$). The edge **c** has no influence in the discard.

3.2 Neighborhood Choice

The filter expressions described in Section 2.3.3, which will be applied to the pixels in $\mathcal{N}_{i,j}$, require that the neighborhoods be formed by either two or three evens. Two-pixel neighbor sets are such that the even pixels in the set are collinear with the odd pixel to be predicted. Three-pixel neighbor sets are such that the three even pixels are not collinear. Generally, there will be many combinations of pixels that meet these conditions. Using $\mathcal{R}_{i,j}$, the set of all the evens that are not discarded by the previous block, the set $\mathcal{M}_{i,j}$ is formed. This set contains all the possible neighborhoods $\mathcal{N}_{i,j}$ that can be formed by combining pairs and trios of pixels in $\mathcal{R}_{i,j}$ as described above.

Until this point, we have used the assumption of piece-wise planarity. If the image was exactly PWP and assuming that there were no errors in the pixel discarding step, any $\mathcal{N}_{i,j} \in \mathcal{M}_{i,j}$ would provide a perfect prediction of the odd's value. However, if we bear in mind that the PWP assumption is just an approximation and that the edge detector may be inaccurate sometimes, it is natural to conclude that there are neighborhoods that are likely to provide a more accurate prediction than others. Therefore, our goal is to find the optimal neighborhood ($\mathcal{N}^*_{i,j}$). The concepts on which the selection of a certain neighborhood over another is based on are the following: first, the information corresponding to pixels close to each other is assumed to be more correlated. Since the intensity of the DM may not exactly follow a planar function, we favor $\mathcal{N}_{i,j}$ choices that cover a small area, which will tend to increase the

probability that depth information can be well approximated by a plane. Following the same intuition we will favor those $\mathcal{N}_{i,j}$ that are closer in distance to (i, j). Second, pixels close to edges are considered less *reliable*. This is so because there is always the possibility that the edge detector is not completely accurate in those areas, and so, in reality, it may happen that those pixels belong to a different plane. Pixels located far away from edges, on the contrary, are not likely to belong to different planes.

The adequacy of a certain neighborhood $\mathcal{N}_{i,j} = \{(k_1, l_1) \dots (k_L, l_L)\}$ (with $L = 2, 3$) is measured according to these ideas by computing the parameters $C(\mathcal{N}_{i,j})$, $P(\mathcal{N}_{i,j})$ and $G(\mathcal{N}_{i,j})$. In order to explain the meaning of each parameter we will use the concept that the positions of the pixels in each neighborhood define a *geometrical figure* (GF) in 2D space. This way, neighborhoods composed of three pixels form *triangles*, while the ones composed of two pixels form *segments* of straight lines. Fig. 3.7 illustrates this idea.

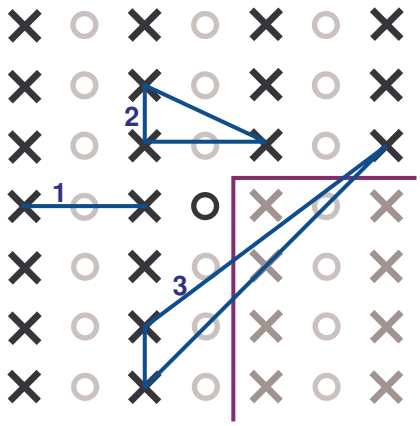
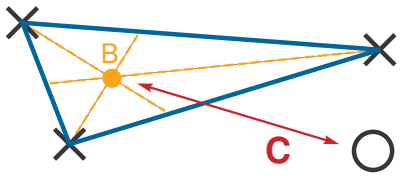


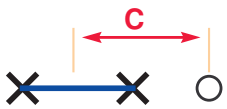
Figure 3.7: Example of three GFs (1,2 and 3 in the image) that can be build using the available evens.

The function $C(\mathcal{N}_{i,j})$ reflects how close this GF is to the odd pixel in question. $C(\mathcal{N}_{i,j})$ is computed as the Euclidean distance from the odd pixel (i, j) to the centroid $(B(\mathcal{N}_{i,j}))$ of the GF. This centroid represents the *center of gravity* of the GF, and in the case of segments it is located in the middle point of it, equidistant to both extremes. Fig. 3.8 graphically exemplifies the computation of $C(\mathcal{N}_{i,j})$. Mathematically:

$$\begin{aligned} B(\mathcal{N}_{i,j}) &= \tfrac{1}{L} \cdot \textstyle\sum_{m=1}^{L} (k_m, l_m), \\ C(\mathcal{N}_{i,j}) &= \|(i, j) - B(\mathcal{N}_{i,j})\|. \end{aligned} \tag{3.3}$$



(a) Triangular GF (3 evens)



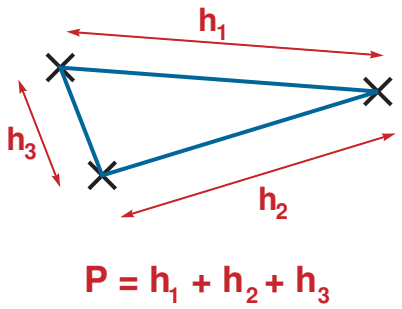
(b) Linear GF (2 evens)

Figure 3.8: Example of the computation of $C(\cdot)$ for different GFs.

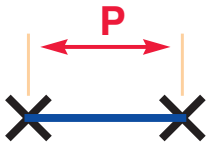
Clearly, for GFs located around (i, j) the parameter $C(\mathcal{N}_{i,j})$ would have small values, which is consistent with the decision of enforcing the use of neighborhoods formed by pixels that are located close to the odd pixel.

Parameter $P(\mathcal{N}_{i,j})$ is computed as the perimeter of the GF formed by $\mathcal{N}_{i,j}$. For neighborhoods of two pixels $P(\mathcal{N}_{i,j})$ is equivalent to the length of the segment. Fig. 3.9 illustrates the computation of $P(\mathcal{N}_{i,j})$. Mathematically:

$$P(\mathcal{N}_{i,j}) = \sum_{m=1}^{L-1} \sum_{m'=m+1}^{L} \|(k_m, l_m) - (k_{m'}, l_{m'})\|. \tag{3.4}$$



(a) Triangular GF (3 evens)



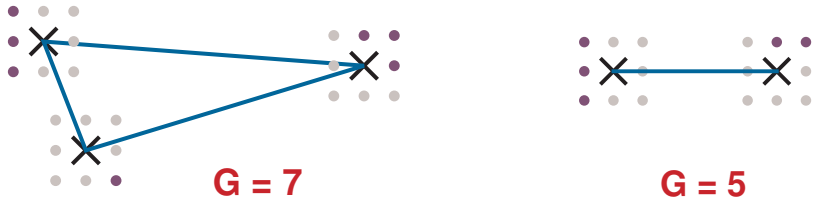
(b) Linear GF (2 evens)

Figure 3.9: Example of the computation of $P(\cdot)$ for different GFs.

Thus, neighborhoods that are formed by pixels located in a small area will have a lower cost $P(\cdot)$. This meets one of the requirements presented before, since the closer the pixels the more likely that their values follow a planar shape.

Lastly, parameter $G(\mathcal{N}_{i,j})$ is computed, using the extended pixel and edge grid, as the number of edges located between each of the pixels that conform $\mathcal{N}_{i,j}$ and their respective immediate neighbors. Fig. 3.10 shows an example. Mathematically:

$$G_{\mathcal{N}_{i,j}} = \sum_{\forall (k,l) \in \mathcal{N}_{i,j}} \sum_{\forall (u,v) \in S^1_{k,l}} e_{(k,l)(u,v)}. \tag{3.5}$$



(a) Triangular GF (3 evens) (b) Linear GF (2 evens)

Figure 3.10: Example of the computation of $G(\cdot)$ for different GFs.

It must be remarked here that in the computation of $G(\cdot)$ it is used edge information prior to any downsampling, regardless of the level of decomposition. Therefore, if a certain pixel is separated by an edge from its closest neighbors, this edge will only be counted if it was located immediately next to the even in question before any downsampling was done (see edge **a** in Fig. 3.11), and not if it was located in any other position between the pixel and its neighbor (see edge **b** in Fig. 3.11). This technique is consistent with the nature of the parameter $G(\cdot)$, since the reliability of a pixel does not depend on the level of decomposition but only on the distance from that position to the edges in the original image. If the edge information was computed based on the downsampled grid, some pixels could become less reliable as the level of decomposition increased, which physically does not make sense.

Figure 3.11: After a level of decomposition has been reached, the pixel positions in gray are no longer considered. Therefore, the central pixel is separated from its two new neighbors by edges. However, on the computation of $G(\cdot)$ only the edge **a** will be counted.

Using these parameters, the best neighborhood is found as the one with the lowest cost, with the cost of a each neighborhood $\mathcal{N}_{i,j}$ being computed using the following expression:

$$J(\mathcal{N}_{i,j}) = \alpha \cdot C(\mathcal{N}_{i,j}) + \beta \cdot P(\mathcal{N}_{i,j}) + \gamma \cdot G(\mathcal{N}_{i,j}). \tag{3.6}$$

This way, not only the best neighborhood can be easily found, but the same operation can be identically reproduced at the decoder. This is essential, for a different selection

of the neighborhood would lead to an incorrect inverse transform, and, consequently, to errors in the reconstructed image.

As for the α, β and γ factors, they are used to weight the relative importance that each parameter (C, P and G) has in the cost function. In our experiments a simple configuration which we refer to as the *standard weighting* is used, this is $\alpha = \beta = \gamma = 1$. However, if for some reason any parameter(s) had to be given more or less importance in the approximation, these factors could be changed. Obviously, these values have to be common to both encoder and the decoder. In practice, they can be chosen independently of a specific image and thus do not require overhead to be sent.

Fig. 3.12 shows different possible neighborhood structures (GF) for an odd pixel. Bearing the meaning of each parameter in mind, it is intuitively possible to see which GF would have greater cost and which ones a lower one.

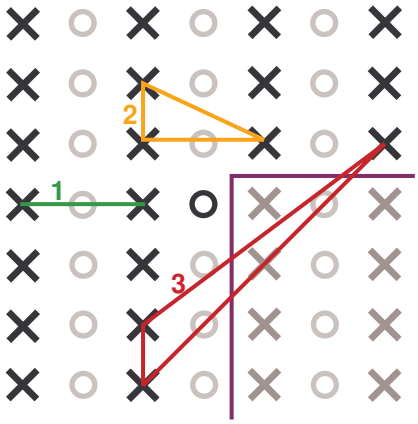


Figure 3.12: $J(GF1) < J(GF2) < J(GF3)$

Note that if the *non-extrapolated* pixels were not computed in a separate procedure, the number of candidate pixels and, consequently, of possible neighborhoods ($card(\mathcal{M}_{i,j})$) for each one of them would be very large. Moreover, in those cases both *cousins* would always be available (not separated from the odd by an edge). The neighborhood composed by these two *cousins* would always show a zero value for C; and P would have the minimum possible value. As for G, since these nodes are located far from edges, it would have zero or very small value ($G = 0$) in most of the cases. Hence the function $J(\mathcal{N}_{i,j})$ would give the minimum cost for the neighborhood formed by the odds' *cousins*, and that would be the $\mathcal{N}^*_{i,j}$ used in the prediction. This process,

however, would be computationally very demanding. Consequently, the process that computes these pixels separately (presented in Section 2.3.2) is not only consistent with the criteria of the transform, but also makes the whole algorithm much more efficient than what it would be if all the pixels were computed individually.

So far, the main ideas behind the selection of the neighborhoods have been given. In Section 3.2.1, the steps that are taken in its implementation are shown in detail.

3.2.1 Neighborhood Selection Algorithm

Since for every odd pixel (i, j) there may be a large number of possible neighborhoods $\mathcal{N}_{i,j}$ inside $\mathcal{M}_{i,j}$, it may seem that the calculation of all the costs $J(\mathcal{N}_{i,j})$ can be a very demanding process in terms of computational resources and hence it can make the algorithm very slow. However, by defining a fixed and optimized procedure, the process can be accelerated considerably. This procedure is summarized in Fig. 3.15.

First of all, in our algorithm, the size of the square region used for the discard is set to 5×5, which implies that $W = 28$ (see Fig. 3.13). This measure has been decided based on experiments in which it was concluded that bigger regions did not improve the performance of the algorithm, but only made it considerably slower.

In order to compute the cost of all the valid combinations of evens we treat differently two kinds of neighborhoods, the ones formed by two pixels and the ones formed by three pixels. The algorithm firstly seeks for the pairs of even pixels collinear with the odd (i, j) ($card(\mathcal{N}_{i,j}) == 2$). As shown in Fig. 3.13, since the set of candidate evens has been restricted to a certain area, there are only a few possible two-pixels neighborhoods. These include the combinations of the evens located in either the same row/column as the odd node (depending on the TD all the pixels in a row/column will belong to the odd set $\mathcal{O}$), and the ones that use pixels in the diagonals. The maximum number of possible two-pixel neighborhoods is 18, since there are 3 directions with 4 even pixels in each, which are grouped in pairs ($3 \cdot \binom{4}{2} = 18$). However, since some of the evens will be discarded because of the edges (otherwise the odd will be a *non-extrapolated* pixel and would not be processed in this block) the real number of combinations to compute will always be lower.

These combinations are found one at a time, and, for each one of them its cost is

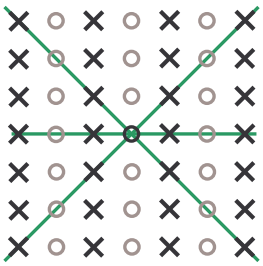


Figure 3.13: Pairs of even pixels collinear with the odd can only be found along the green directions.

computed and compared against the minimum cost found until that moment. For the computation of the parameters C and P the positions of the evens in the 2D space are used as explained in the Section 3.2; as for G, a function called $g(u, v)$ is used. This function is created right after the edge detector is applied and contains the number of edges located around each pixel. Using the notation introduced before $g(k, l) = \sum_{\forall(u,v)\in\mathcal{S}^1_{k,l}} e_{(k,l)(u,v)}$. This way $G(\mathcal{N}_{i,j}) = \sum_{\forall(k,l)\in\mathcal{N}_{i,j}} g(k, l)$. Once all the two-pixel combinations have been considered the best neighborhood among them, which we denote $\mathcal{N}^2_{i,j}$, is selected. The cost of this neighborhood will be later compared against the minimum cost found among all the three-pixel neighborhoods. Note that it could also happen that due to the presence of edges it is not possible to find two evens collinear with the odd one.

The next step consists in finding all the combinations of three non-collinear even pixels. Since the number of possibilities is now higher, checking one possibility at a time is not computationally efficient. This time, all the combinations of three non-collinear evens are found at once, and the positions of the pixels that conform the neighborhoods are ordered in a matrix. Then, the cost associated with each parameter (P, C and G) is computed using the adequate vector functions over that matrix. The best neighborhood is trivially found by picking the one with minimum total cost (J), we name this neighborhood $\mathcal{N}^3_{i,j}$.

Finally, as shown in Fig. 3.15, the best two-pixel neighborhood is compared against the best three-pixel one. The one with lower cost is then chosen as $\mathcal{N}^*_{i,j}$, which will be used in the prediction of (i, j). Mathematically: $\mathcal{N}^*_{i,j} = \underset{\mathcal{N}_{i,j}=\mathcal{N}^2_{i,j},\mathcal{N}^3_{i,j}}{\operatorname{argmin}} [J(\mathcal{N}_{i,j})]$.

Note that, since the edge information is shared by the encoder and the decoder, the neighborhood selection process can be replicated at the decoder; which implies that

by sending only the edge map as overhead, the transform can be inverted.

At this point, it can also happen that, for certain odd pixels, it is not possible to find any valid combination of evens due to the presence of edges. We then say that the pixel is *isolated*; Section 3.3 explains how the algorithm proceeds in this case.

3.3 Isolated Pixels

So far, it has been assumed that for each odd there will always be enough candidate even pixels to form at least one valid neighborhood. Nonetheless, it may happen that for certain odd pixels there are no combinations of evens that fit the collinearity conditions. This phenomenon is more common as the level of decomposition increases, since the number of pixels in each plane is reduced and the pixels that remain are located further apart, which increases the probability that there exist edges between them.

The algorithm is modified in these cases. There are two possibilities: either search alternative filtering operations using the pixels that are known to belong to the same plane (if there are any) or accept filtering operations that cross edges. Since the main goal of the transform is the avoidance of the high frequency values generated when filtering across edges, the algorithm tries to avoid the second option as much as possible. The algorithm, which is illustrated in Fig. 3.15, works as follows.

When, given an odd pixel (i, j), $\mathcal{M}_{i,j}$ is found to be empty, the algorithm checks if there is an edge between the odd and one of its *cousins*. Note that at least one of them is separated by an edge, for, if it was not, the odd would have been treated as a *non-extrapolated* pixel. In case one of the *cousins* (say (k, l)) is located in the same plane, the approximation is built using only that one pixel, $\mathcal{N}_{i,j} = (k, l)$ and $\mathbf{p}_{i,j} = 1$. This way, the detail coefficient is computed as the difference of intensity between the two pixels. Mathematically:

$$d(i, j) = X(i, j) - X(k, l). \tag{3.7}$$

Contrary to *non-isolated* pixels, even if the signal is exactly planar, the detail coefficient is not expected to have intensity zero. However, since the two pixels are located close together and in the same plane, the difference between them is expected to be small. Note that this method is equivalent to using a mirroring technique that

extrapolates the value of one pixel at the other side of the edge then filtering with coefficients $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}$ (as in Fig. 3.14). This is equivalent to the method described in [MD08], using only one even.

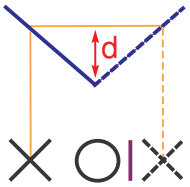


Figure 3.14: Prediction of an odd pixel using only one of its *cousins*. The purple line represents an edge.

In the case that both *cousins* are separated by edges, the node is assumed to be completely isolated in its plane. No other candidates are searched for. When this happens, the odd is predicted using its two *cousins* (with $\mathbf{p}_{i,j} = \{\frac{1}{2}, \frac{1}{2}\}$) regardless of the edges between them. Due to the fact that the pixels belong to different planes the detail coefficient is not minimized; nevertheless, this selection makes the whole transform closer to orthogonality. As shown in Chapter 5, this can have a positive influence in the overall quality of the reconstructed image.

Note that the fact that this point is reached does not mean that all the even pixels were discarded during the *Pixel Discard* block. On the contrary, there may still remain some even pixels that are not separated by edges, even though a valid neighborhood cannot be formed. One alternative to our method could consist on finding any of those pixels and applying (3.7). However, in that case the detail coefficient is not ensured to be minimized either, while the transforms loses orthogonality and the computation becomes much more complex and slow. In our work we have decided to directly use the two *cousins* without checking other possibilities.

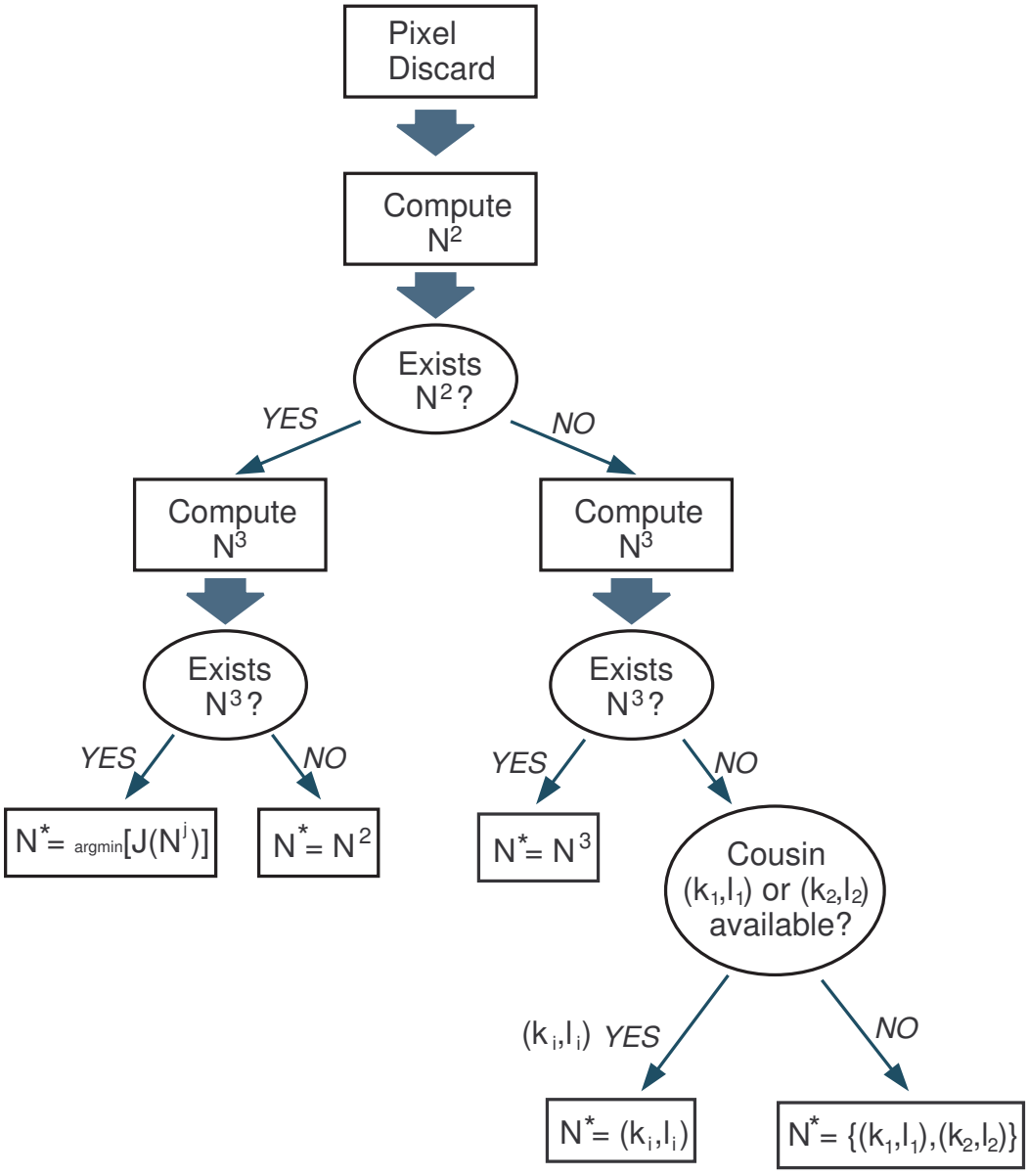


Figure 3.15: Neighborhood choice process.

Chapter 4

Extension to Multiple Levels of Decomposition

During the iteration of a lifting transform, the number of inputs on which the transform needs to be computed changes as the level of decomposition increases. This is due to the downsampling that is implied when the pixels are split into the $\mathcal{E}$ and $\mathcal{O}$ sets. Moreover, since the transform uses a separable structure, the signal is always downsampled along one direction before computing the transform along the orthogonal one. Consequently, the input (image) does not only change the number of coefficients, but also their spatial distribution, i.e., at some point the coefficients will form a square and at others they will form a rectangle. In this chapter, we provide detail on how the operations explained in previous chapters are adapted to situations when multiple levels of decomposition are employed.

First of all, it is important to understand how the spatial distribution of the coefficients evolves during the transform process. As shown in Fig. 4.1, given an image $X(i,j)$, of size $N \times M$, the transform is first computed along rows, which means that during the coefficient split, the TD is the horizontal (rows) while AD is the vertical (columns). Using this structure, the pixels are predicted and updated. Once this is done, the pixels corresponding to the different sets are separated. This results in two 2D signals of size $\frac{N}{2} \times M$ (rectangular shape). In order to complete the transform, the operation is repeated over these two signals separately. This time, the TD corresponds to the vertical one, which means that the parity is alternated along

columns. Once the two transforms have been computed, the result consists on 4 2D signals of size $\frac{N}{2} \times \frac{M}{2}$ (note that the total number of coefficients has not changed, since $4 \cdot \frac{N}{2} \cdot \frac{M}{2} = N \cdot M$). When this point has been reached, the transform has competed a *level of decomposition*; if further decomposition is needed the process is repeated iteratively on the coefficients resulting from both updating operations (smooth horizontal-smooth vertical).

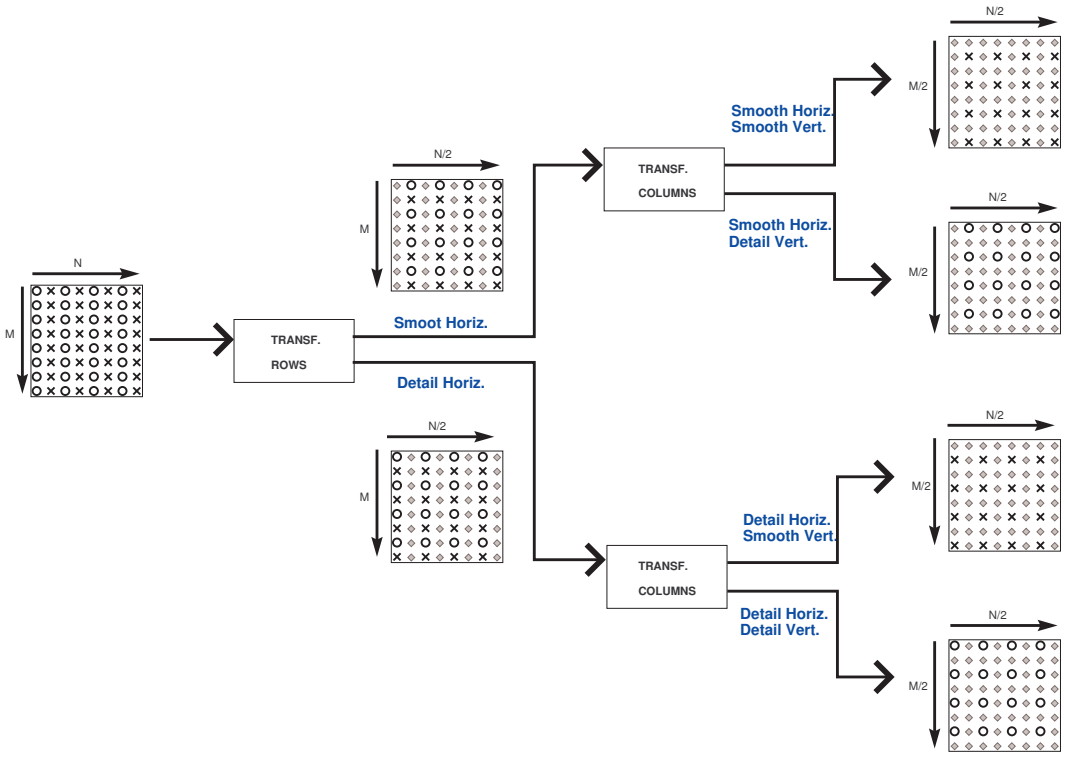
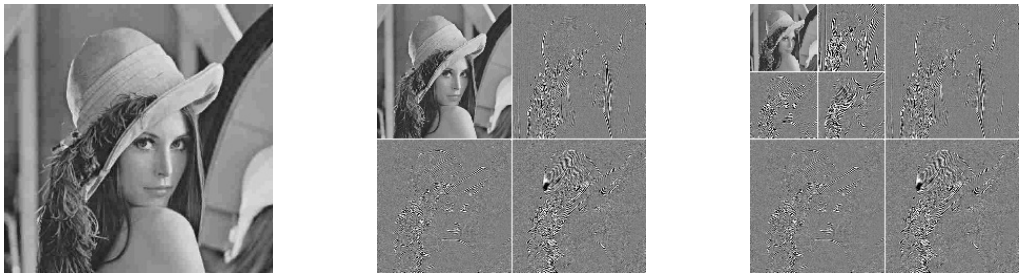


Figure 4.1: Block diagram of a separable lifting transform. The gray squares represent pixels that are not further used because of downsampling.

Given the structure in which the DWT coefficients are normally represented (Fig. 4.2), it is possible to think the transform coefficients are relocated (separating in space all the frequency bands) after each level of decomposition. However, this could have very damaging consequences when computing our transform over those coefficients, since, for example, 3 positions that are collinear at a certain level of decomposition may not be so if the coefficients are relocated as in Fig. 4.2(b) and 4.2(c), this would affect the definition of the neighborhoods and the filters; also, the way that the edges are dealt with should change in order to be adapted to a new structure. Therefore, as shown

in Fig. 4.1, each detail/smooth coefficient is located exactly in the same position as the corresponding original pixel, during the whole transform process.



(a) Original image

(b) DWT coefficients after 1 level of decomposition

(c) DWT coefficients after 2 levels of decomposition

Figure 4.2: Example of the application of DWT to an image. In (b) and (c), the transform coefficients are represented separately in space and frequency.

In our algorithm, the downsampling works as follows. Let q be the level of decomposition at a certain stage of the transform. The splitting is first computed along rows, which means that (i, j) will belong to $\mathcal{O}$ if $j = 2^q n - 1$ $(n \in \mathbb{Z})$ and to $\mathcal{E}$ otherwise $(j = 2^q n)$. After the sets are separated, the pixels are split again in order to continue with the transform, in this case a pixel belongs to $\mathcal{O}$ if $i = 2^q n - 1$ and to $\mathcal{E}$ if $i = 2^q n$. Thanks to this method, the position of a coefficient in relation to another does not change as the level of decomposition increases. However, this also implies that some parts of the algorithm must be adapted to each stage of the transform, these are: the square region used during the even discarding and the edge map (EM). Fig. 4.6 illustrates how these elements, which are external to the lifting transform but have a deep repercussion on its result, change as the level of decomposition grows. These adaptations are detailed next.

First, as shown in Fig. 4.6 the square region used in order to discard even pixels, which is introduced in Section 3.1.2, is enlarged after each step so that the number of evens inside the region remains constant, and hence, the same possible neighborhoods can be formed. Let dx_r^q (dy_r^q) be the maximum distance from the odd (i, j) being predicted, to any pixel (k, l) inside the restricted area when $k = i$ $(l = j)$ (see Fig. 4.3), at the level of decomposition q, when the transform is computed along rows. When the transform is computed along columns, we denote this distance dx_c^q (dy_c^q). For the first level of decomposition $dx_r^1 = dy_r^1 = 3$, which forms the square region of size 5×5 introduced in Section 3.2.1. After the image is downsampled along rows, the

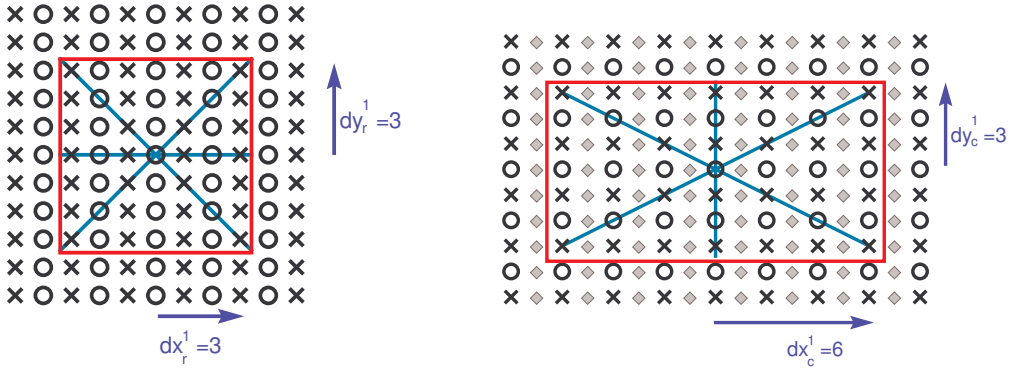
horizontal component is doubled, making the area rectangular instead of square, i.e.,

$$\begin{aligned} dx_c^q &= 2 \cdot dx_r^q \\ dy_c^q &= dy_r^q. \end{aligned} \tag{4.1}$$

Once the column downsampling has been performed, the vertical component is doubled too, which returns the region to a square shape:

$$\begin{aligned} dx_r^{q+1} &= dx_c^q, \\ dy_r^{q+1} &= 2 \cdot dy_c^q. \end{aligned} \tag{4.2}$$

Fig. 4.3 shows how the region changes as the number of levels of decomposition is increased. Note that thanks to this method, all the neighborhoods structures described in Section 3.2.1 can be formed regardless of the level of decomposition, since neither the number of evens nor the possible combinations of collinear/non-collinear pixels are changed.



(a) Square region when transform along rows

(b) Rectangular region when transform along columns

Figure 4.3: The region of non-discarded evens changes its shape and size in order to ensure that there is always the same number of possible neighborhoods.

Second, the EM can be updated before each step of the transform in order to accelerate its computation, without changing the resulting output. For this purpose, we take a closer look at the technique designed for drawing paths from pixel to pixel, which is used for discarding the evens that belong to different planes (see Section 3.1). When using this method some edges will never be crossed by any path, and thus, the result of the discard will not be affected if they are not considered. However, this edges slow the algorithm significantly. Therefore, by eliminating them from the EM before computing the transform, the algorithm can be computed much faster. The location of these *useless* edges is the following:

For start, a path that connects two odd coefficients of the same row or column will never lead to an even one. Hence, no even will be discarded due to the presence of edges that cross such path (see edge **c** in Fig. 4.4), i.e., the choice of $\mathcal{N}_{i,j}$ will not be changed by those edges. Using a similar argument, the edges which are located in rows/columns that have been eliminated by downsampling (see edge **a** in Fig. 4.4) will be of no use for the pixel discard operation. On the contrary, edge positions that represent edges between diagonal neighbors (see edge **b** in Fig. 4.4) cannot be eliminated so easily, even when the closer pixels are odds, for there is always the possibility that some paths pass through those positions no matter the level of decomposition.

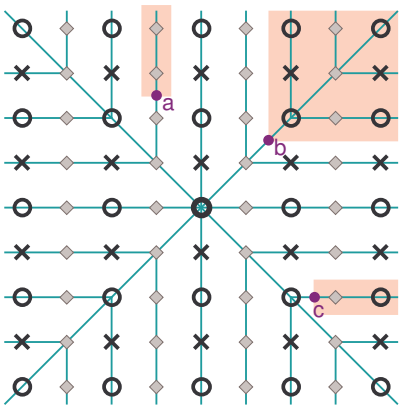
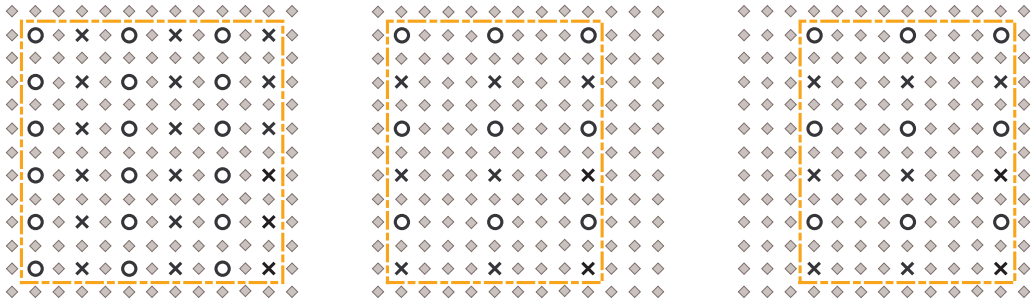


Figure 4.4: The edges a and c do not imply the discard of any evens, while b does.

Also, as shown in Fig. 4.5, as the image is downsampled, the last rows/columns are not transformed anymore. Therefore, the edges located in those areas are no longer needed, for they only lead to positions where the pixels have been eliminated by downsampling. For a certain level of decomposition q, these useless positions are the following: when the transform is computed along rows, all edges with position (i, j) so that $i > 2^{m_r}$ and $j > 2^{m_c}$, where $m_r = \log_2(N) - q$ and $m_c = \log_2(M) - q$, can be eliminated without altering the result of the transform. On the next stage, when the transform is computed along columns, those edges are still not used, but also some additional positions can be eliminated: for the transform of the detail coefficients resulting from the previous stage (detail horiz. in Fig. 4.1), these correspond to the columns with $i > 2^{m_r-1}$; while for the smooth coefficients (smooth horiz. in Fig. 4.1), the first columns ($i < 2^q$) are the useless ones.



(a) Transform computed along rows

(b) Transform along columns on the detail coeffs.

(c) Transform along columns on the smooth coeffs.

Figure 4.5: Structure of the pixels for a certain level of decomposition. The edges outside the region marked by the orange line will not influence the result.

In summary, given a certain EM $BW^0(u,v)$, output of the edge detector, whose size is $(2N-1)\times(2M-1)$ because it uses the pixel and edge grid, the EMs $BW_r^q(u,v)$, $BW_{ce}^q(u,v)$ and $BW_{co}^q(u,v)$ that are used in the the lifting operations as shown in Fig. 4.6, are computed as:

$$BW_r^q(u,v) = \begin{cases} 0 \text{ if } u = (n+1)\cdot 2^q + n + 1 \text{ or } v = (n+1)\cdot 2^q + n + 1;\ n \in \mathbb{Z} \\ BW_{ce}^{q-1}(u,v) \text{ otherwise} \end{cases} \tag{4.3}$$

$$BW_{ce}^q(u,v) = \begin{cases} 0 \text{ if } u = n\cdot 2^{q+1} + 2^q - 1 \text{ or } v = 2^q + 2\alpha;\ n,\alpha \in \mathbb{Z}; \alpha \leq 2^{q-1} - 1 \\ BW_r^q(u,v) \text{ otherwise} \end{cases} \tag{4.4}$$

$$BW_{co}^q(u,v) = \begin{cases} 0 \text{ if } u = (n+1)\cdot 2^{q+1} - 1 \text{ or } v = (2n+1)\cdot 2^q - 1;\ n \in \mathbb{Z} \\ BW_r^q(u,v) \text{ otherwise} \end{cases} \tag{4.5}$$

Once all these adaptations have been done, the transform can be computed at any level of decomposition as explained in Chapters 2 and 3.

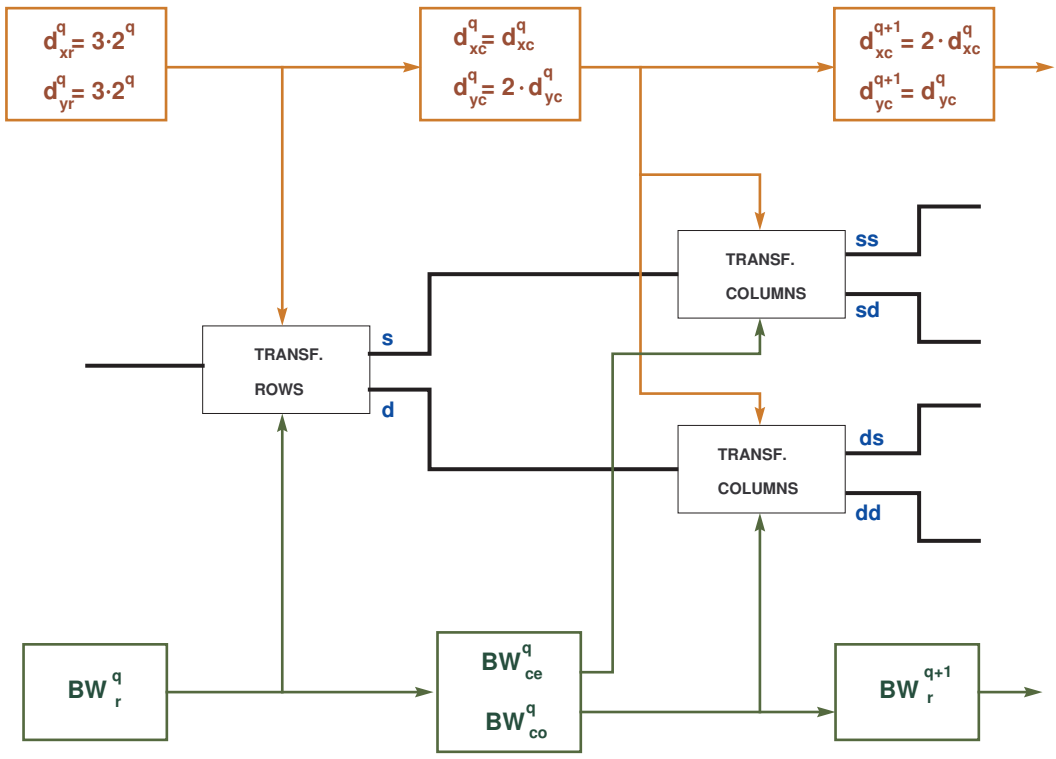


Figure 4.6: Block diagrams of the operation computed parallel to the lifting transform: Discard region resizing (on top, in orange) and updating of the EM (on bottom, in green).

Chapter 5

Experimental Results

The initial goal of this work was the design of a wavelet transform that efficiently encodes depth maps while preserving the edges on those DM images. This way, it was assumed, the resulting synthesized images obtained by using these DMs would have a better perceptual quality than the ones obtained when the DMs were encoded using other existing methods.

In Section 5.1 we experimentally prove that when DMs are coded using our method, the PSNR values achieved are higher than the ones obtained with standard transforms. Also, we prove that the edges are well preserved when using our transform while blurred when using non-adaptive filtering techniques. Then, we also compare the performance of our algorithm against the method presented in [MD08] and reach important conclusions about the different edge-avoiding approaches.

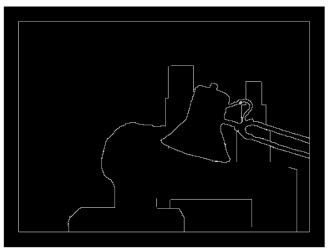
Moreover, since the ultimate use of DMs is not their direct viewing but their use in view synthesis operations, we also compare the resulting synthesized images obtained with DMs that have been coded with our transform against others obtained using different methods. This is done in Section 5.2.

Note that all the results presented in this chapter have been obtained using three levels of decomposition. Also, the lifting filters of our transform are normalized using the technique introduced in Section 2.5 and the edge detector used is the one presented in Appendix A.

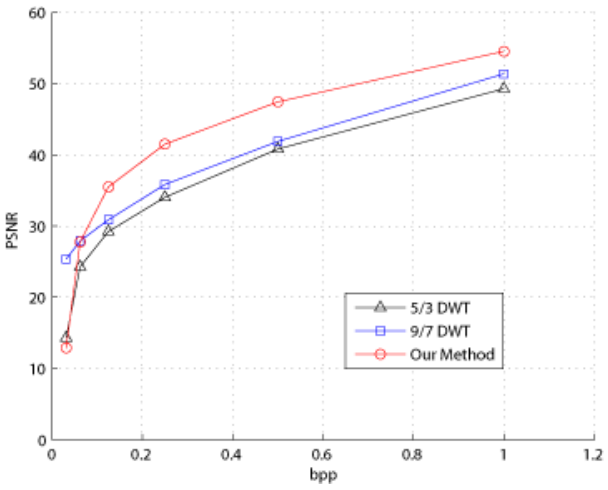
5.1 Compressed Depth Maps

First, we compare the performance of our proposed transform against the standard 9/7 and 5/3 DWT. For this purpose we use some of the DMs provided in the Middlebury data set [Mid] and code them for different bit rate values. The obtained RD curves are presented in Fig. 5.1 to 5.5. As the results show, our algorithm performed significantly better than the other methods for high bit rates, gaining up to 15 dB in some cases. However, for very low bit rates (0.125 bpp or below), our method achieved similar PSNR values as the standard ones, or in some cases had a worse performance. This is due to the inclusion of EMs as side information; since the size of the EM is fixed, the percentage of bits dedicated to this information grows as the bit rate decreases. This phenomenon has a deeper repercussion for images with a larger number of edges (see Fig. 5.2), whereas for DMs with few edges (see Fig. 5.5) fewer bits are dedicated to the overhead, and thus, our results are not affected as dramatically. To illustrate this fact, the total bits per pixel dedicated to each EM is specified together with the RD curves. However, note that for most applications, DMs with PSNR lower than 25 dB introduce severe distortion. Therefore, a comparison of the performance of the algorithms below that point does not have a practical use, since in all cases the interpolated images would ave very deteriorated quality.

(a) Original Image

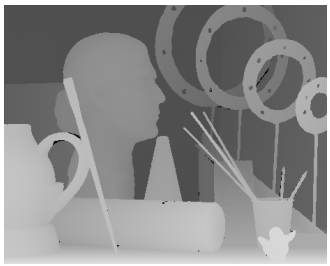


(b) Edge Map



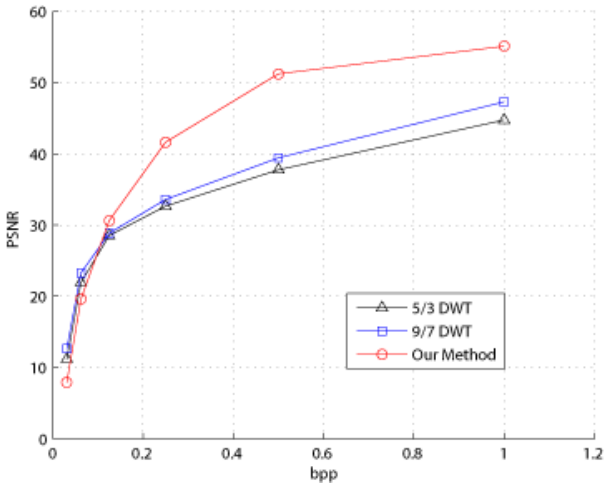
(c) RD curves

Figure 5.1: Tsukuba DM. 0.0059 bpp dedicated to EM



(a) Original Image

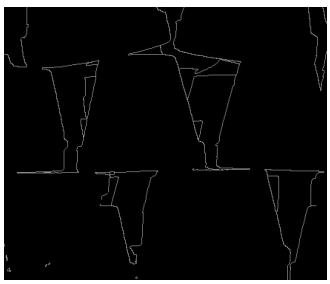
(b) Edge Map



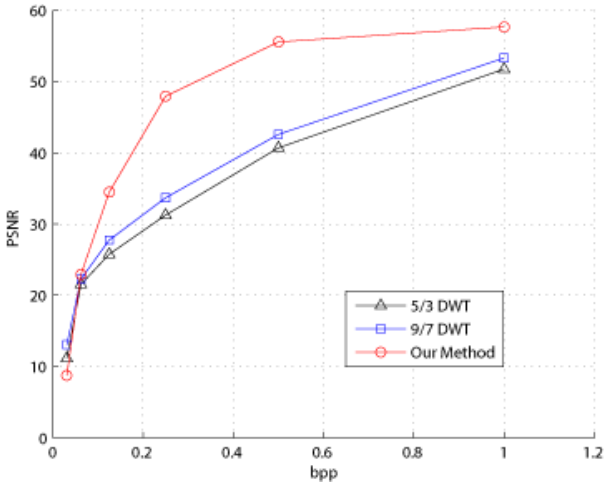
(c) RD curves

Figure 5.2: Art DM. 0.0099 bpp dedicated to EM

(a) Original Image

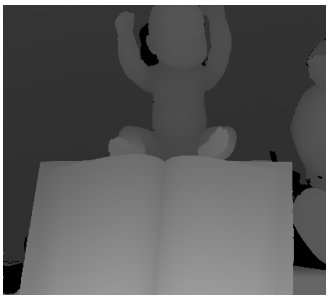


(b) Edge Map

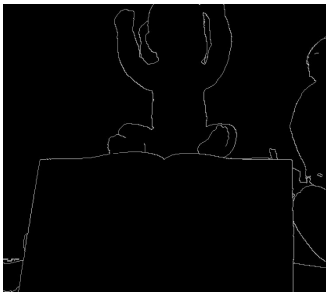


(c) RD curves

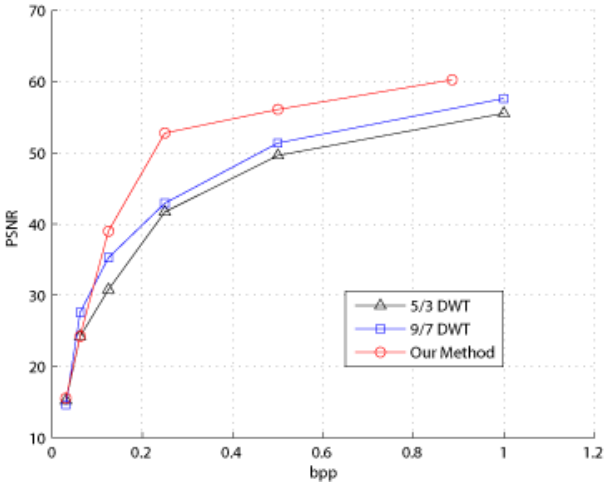
Figure 5.3: Flowerpots DM. 0.0059 bpp dedicated to EM



(a) Original Image

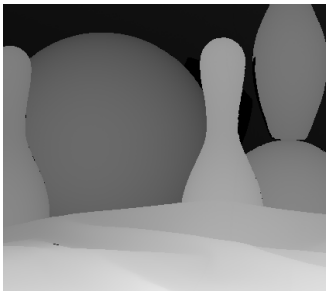


(b) Edge Map

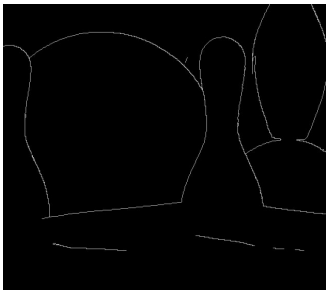


(c) RD curves

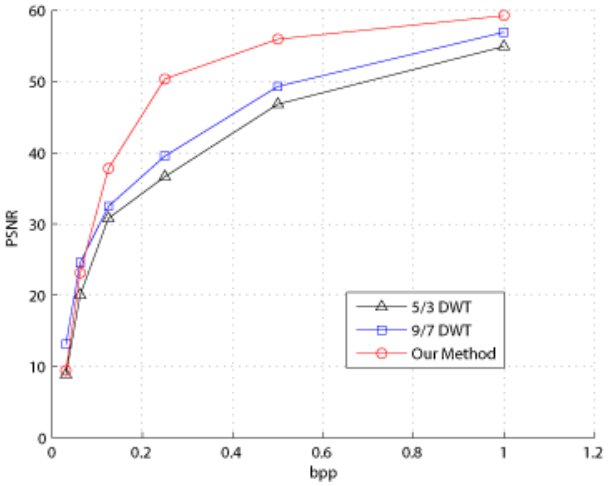
Figure 5.4: Baby DM. 0.0060 bpp dedicated to EM



(a) Original Image



(b) Edge Map

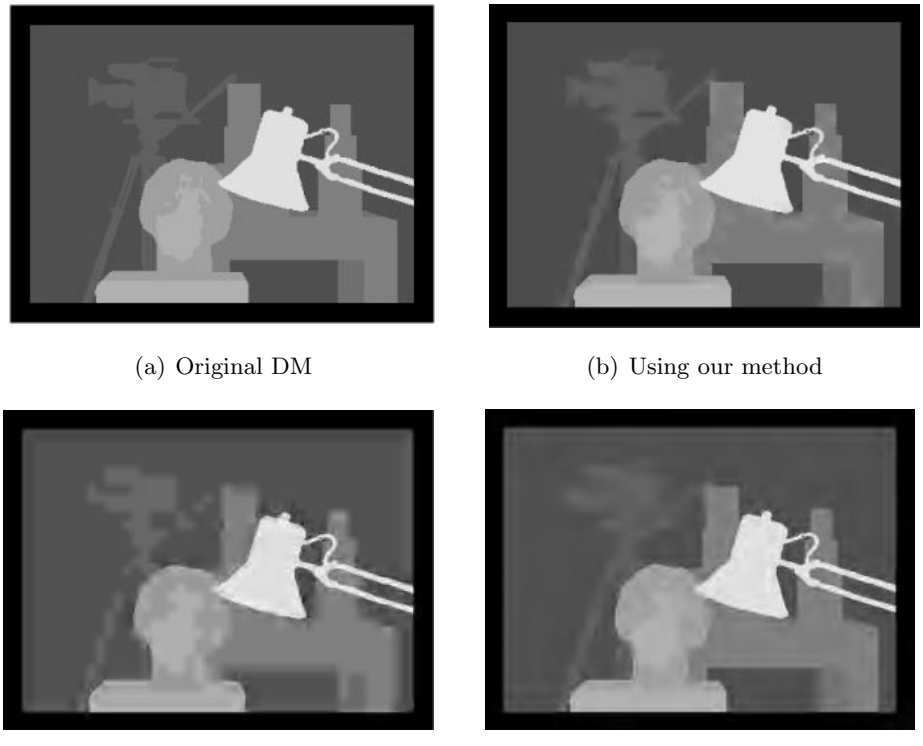


(c) RD curves

Figure 5.5: Bowling DM. 0.0049 bpp dedicated to EM

Another aspect that is worth considering is the fact that the gain obtained with our transform (as compared to standard methods) is greater for images with stronger edges, i.e., greater difference of intensity between the pixels at each side of the edge, (e.g., Fig. 5.3) than for ones with weaker edges (e.g., Fig. 5.4). This is so because the error incurred when standard methods mix pixels from different planes is lower in the second group of images, and hence, the energy of the coefficients of the HP bands is inferior. Thus, the gain achieved when avoiding filtering across edges is higher when edges are stronger.

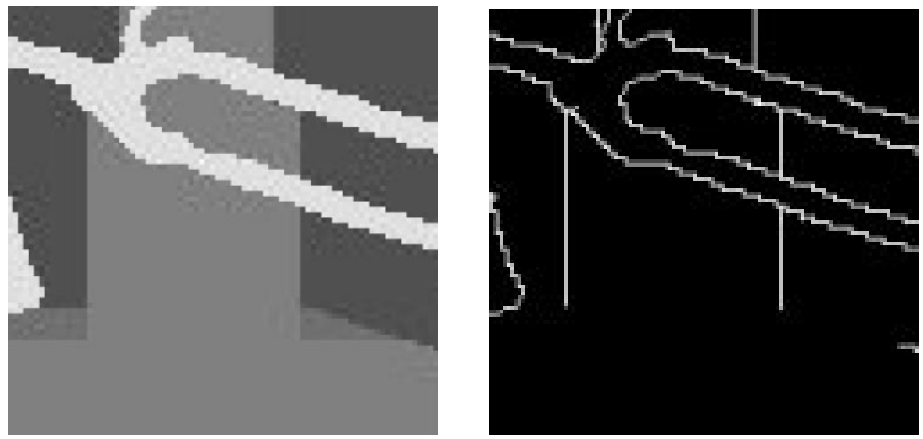
One of the main objectives of our transform was the preservation of the sharpness of the edges. As shown in Fig. 5.6, thanks adaptive filtering, the blurring effect introduced by the standard methods does not appear in our case (even when all images have similar PSNR values). This is important when the coded DMs are later used in View Interpolation operations. Also, note that in areas where the edge detector fails to find edges, our transform also introduces severe blurring (see Fig. 5.7). This reflects the importance of a reliable approach for finding the edges.



(a) Original DM (b) Using our method

(c) Using 5/3 DWT (d) Using 9/7 DWT

Figure 5.6: Reconstructed Tsukuba image after coding at 0.125bpp.



(a) Section of original DM

(b) Edge Map

(c) Using our method

(d) Using 9/7 DWT

Figure 5.7: Zoomed area of the Tsukuba DMs. Using our method, the edges indicated in the EM are preserved, while the rest are blurred. The standard method blurs all edges.

Lastly, another factor that damages the performance of our algorithm is that SPIHT (which is used for encoding the transform coefficients) is optimized for orthogonal transforms, i.e., it tends to prioritize the transmission of larger transform coefficients. However, due to the filtering techniques used, our method is far form being orthogonal. More specifically, if $Y(i,j)$ denotes the resulting coefficients from transforming $X(i,j)$, then if the transform was nearly orthogonal, we should expect that $\delta = \sum_{i,j}|Y(i,j)|^2 / \sum_{i,j}|X(i,j)|^2 \approx 1$. However, our transform is, in general, not even close to this condition. Table 5.1 shows the values of δ for the images presented in Figs. 5.1 to 5.5. Clearly, our transform is far from orthogonality, while the standard methods are not. Thus, using SPIHT to encode our transform coefficients is inher-

ently suboptimal, and it is one of the reasons (together with the side information bit consumption) why our transform does not achieve much gain at low bit rates. Future studies should be focused in both the causes and the consequences of this phenomenon. In an improved version of our algorithm it would be necessary to consider filter techniques that make the overall transform more orthogonal. A possible solution could consist of improving the filter normalization approach so that it does not only ensure that the transform coefficient follow a PWP model (as it does now) but also that the overall transform is closer to orthogonality.

Transform / Image	tsukuba	art	flowerpots	baby	bowling
Proposed	0.1164	0.1172	0.1164	0.1179	0.1188
5/3 DWT	1.0353	1.0352	1.0417	1.0277	1.0424
9/7 DWT	0.9872	1.0219	1.0149	1.0173	1.0318

Table 5.1: $\frac{\sum_{i,j}|Y(i,j)|^2}{\sum_{i,j}|X(i,j)|^2}$ for various transforms and images.

We also compare the performance of our algorithm against the one presented in [MD08] (which we refer to as *Minh Do* method). Since the normalization approach introduced in Section 2.5 is a part of our work we first compare the PSNR values obtained with our transform (using our normalization method, which we refer to as uniform normalization) against the ones obtained with [MD08] (using the standard normalization approach). As the RD curves of Fig. 5.8 show, our transform performed significantly better than their algorithm for all images and bit rates (achieving up to 8dB of gain when using our transform).

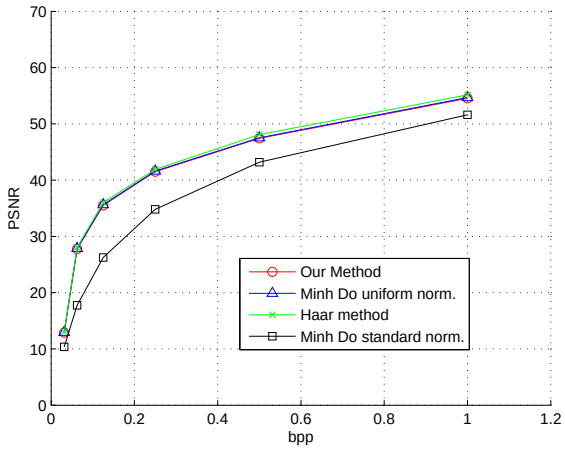
Moreover, in order to compare the neighborhood selection approach of our transform against the one in [MD08], we combine the transform in [MD08] with our normalization method. This way, the transform that obtains higher PSNR values will be the one with the most efficient neighborhood selection algorithm. Since the number of possible neighborhood structures is greater in our algorithm, intuition says that DMs coded using our method should have better quality. However, as the RD curves show (see Fig. 5.8), the results obtained with this method are very close to ours (in some cases even better). This implies that the gain obtained when choosing our method over the one in [MD08] is due to the normalization method introduced, rather than to the larger number of possible neighborhood structures.

This can be attributed to the fact that DMs are, locally, almost constant. In [MD08],

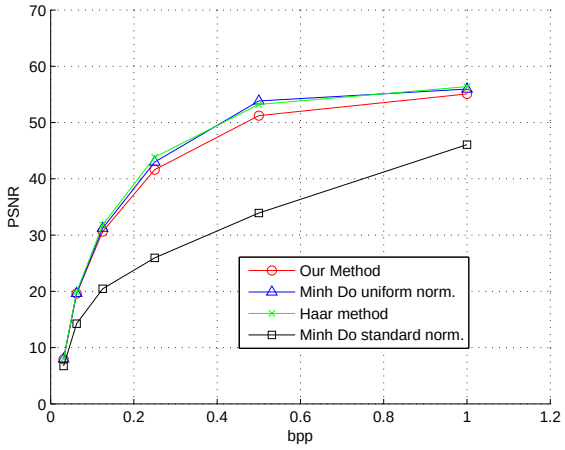
when an odd's neighborhood cannot be formed by the odd's cousins or by a combination of two other evens located in the same row/column as the odd, the pixel is assumed to be isolated. In this case (see Section 3.3) the detail coefficient consists of the difference between the intensity of the odd pixel and the one from a certain even (which belongs to the same plane). If the image is perfectly PWP, the intensity of this detail coefficient will (in general) not be zero, and thus high frequency components appear on those areas. Our method, on the contrary, is capable of searching for alternative filtering strategies that generate a zero intensity detail coefficient. This way, with our transform, the coding of the HP coefficients requires less bits. This should lead to higher quality DMs when using our method, than when using the transform in [MD08]. However, since the DMs used are very close to being locally constant, the detail coefficients obtained by the algorithm in [MD08] have very low value. Then, in order to code these HP coefficients very few bits are required, as well; and so the appearance of these HP components does not have a strong impact on the overall quality of the reconstructed images. Moreover, since most DMs are not exactly PWP, the errors that our transform generates when approximating the signal for a plane are comparable, in terms of HP components intensity, to the ones obtained when using [MD08]. Also, the search of alternative neighborhood structures often leads to crossing edges when filtering, due to errors during the generation of the EM, which introduces high intensity HP coefficients. All this implies that the use of a larger kernel of possible neighborhoods is does not improve the quality of the reconstructed DMs as much as expected.

In order to prove this theory, we have developed a Haar-like transform where each odd is predicted using only its available cousins, i.e., both of them (as in 5/3 DWT) if they are not separated by edges, and, in case one of them belongs to a different plane, the prediction is computed using only the remaining one. The normalization method used in this case corresponds to the one presented in Section 2.5. Contrary to [MD08], whose algorithm could use pixels located further away from the odd in order to extrapolate the signals values, in this Haar-like transform, only the immediate neighbors of each odd, along TD, can be used for its prediction. This would introduce strong errors in case that the image was not close to being locally constant. The results (see Fig. 5.8) show that this transform achieves similar results (and even better in some cases) as both the one proposed in this work and the one in [MD08]. Consequently, it is deduced that DMs can be considered locally constant.

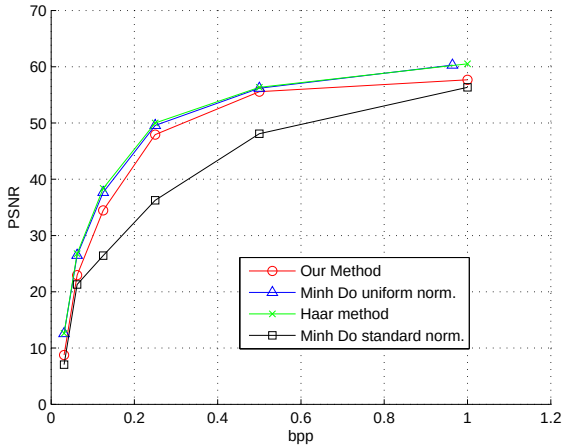
Thus, assuming that DMs are piece-wise constant (PWC) images can be sufficient for developing coding algorithms to be applied to DMs. Nevertheless, since PWC is a particular case of PWP, all the assumptions and statements presented in this work are still valid. Moreover, our transform, and all the tools presented in this work, can be extended to other PWP signals in different scenarios.



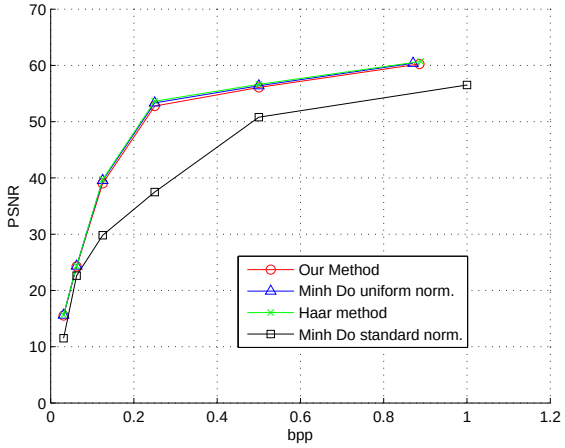
(a) Tsukuba



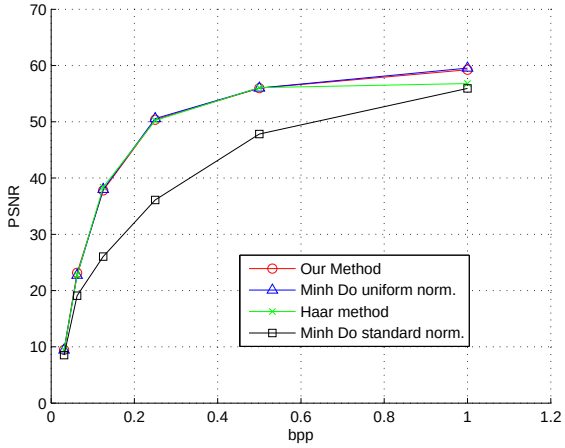
(b) Art



(c) Flowerpots



(d) Baby



(e) Bowling

Figure 5.8: RD curves obtained using different edge-avoiding filtering techniques.

5.2 Interpolated Views

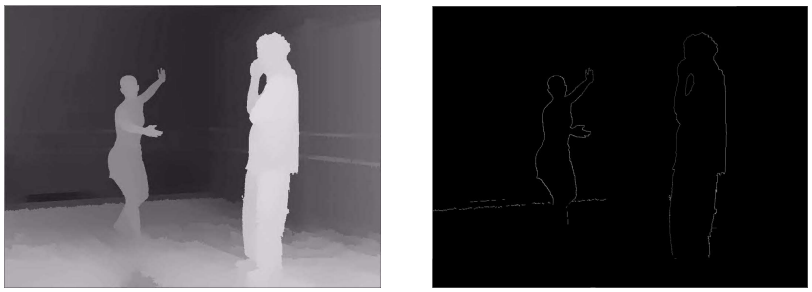
We now compare the perceptual quality of the synthesized images that are obtained when using our method against the one obtained when using standard methods. For this, we consider the interpolation of individual frames in the standard ballet and breakdancers sequences using two adjacent views. The frames whose DMs are compressed are shown in Fig. 5.9 and 5.10. For simplicity, we only compress the depth maps but not the individual frames. Joint encoding of video and depth is beyond the scope of this work. The DMs to be compressed, together with their corresponding EMs, are show in Fig. 5.11 and 5.12. All the results have been obtained using the interpolation software developed by Nagoya University [Uni].

(a) View 3 (b) View 5

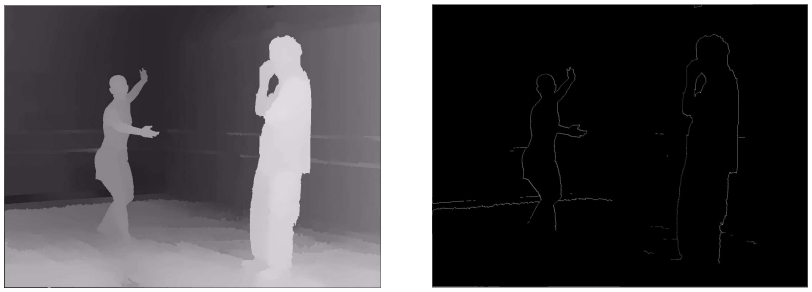
Figure 5.9: Frames from the Ballet sequence

(a) View 3 (b) View 5

Figure 5.10: Frames from the Breakdancers sequence



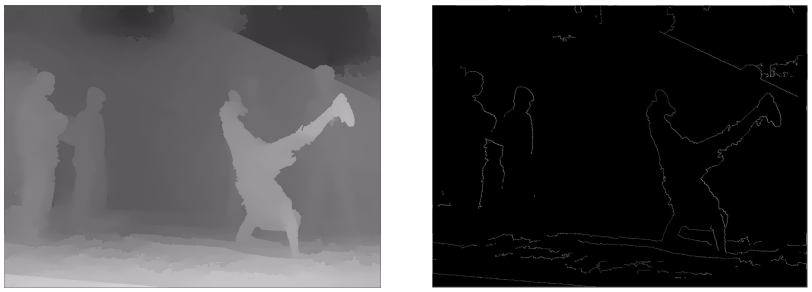
(a) View 3 DM (b) View 3 EM



(c) View 5 DM (d) View 5 EM

Figure 5.11: Ballet frames DMs and EMs

(a) View 3 DM (b) View 3 EM



(c) View 5 DM (d) View 5 EM

Figure 5.12: Breakdancers frames DMs and EMs

Note that even when the DMs are not compressed the interpolated images show severe artifacts (see Fig. 5.13).

Figs. 5.16 and 5.15 show the images obtained when the DMs are compressed at 0.25 bpp for both our method and the 9/7 DWT. Clearly, the standard method deforms the shape of the objects, which degrades the perceptual quality of the image. Thanks to our adaptive filtering technique, the edges are preserved, and, consequently, the shape of the objects maintains its naturalness. This is even more clear in Fig. 5.16 and 5.17, which shows in detail a zoomed area of the resulting images. Note that the artifacts introduced by our transform are very similar to the ones that appear when the DMs are not compressed. Thus, DMs coded with our transform require less bits for storage and transmission, while the perceptual quality of the image is not severely affected.

Also, it is worth noting that even though the PSNR values obtained for the DMs are very similar for both transforms (see Table 5.2), the perceptual quality of the resulting synthesized images is very different. This is due to the fact that the pixels located around edges represent a small percentage of the total set of pixels, thus, the errors generated on those areas have little impact in overall PSNR of the DM, while they are determinant in the result of the view interpolation.

	Ballet v. 3	Ballet v. 5	Breakdancers v. 3	Breakdancers v. 5
Proposed	46.07	48.06	52.23	51.93
9/7 DWT	52.09	47.77	51.53	51.13

Table 5.2: PSNR (in dB) of the reconstructed DMs.

Note that, again, the standard method introduces more error when the edges are strong (e.g., in the Ballet sequence, Fig. 5.14(b)), while achieving considerably good results for images with weak edges (e.g., the Breakdancers sequences, Fig. 5.15(b)). For both types of images our transform achieves very good visual results (see Figs. 5.14(a) and 5.15(a)), since the edges are preserved in both cases.

(a) Ballet Frame

(b) Breakdancers

Figure 5.13: Interpolated Frames obtained with uncompressed DMs

(a) Using our method

(b) Using 9/7 DWT

Figure 5.14: Ballet frames obtained with compressed DMs

(a) Using our method

(b) Using 9/7 DWT

Figure 5.15: Ballet frames obtained with compressed DMs

(a) Using uncompressed DMs (b) Using our method (c) Using 9/7 DWT

Figure 5.16: Zoomed area from the interpolated Ballet images

(a) Using uncompressed DMs (b) Using our method (c) Using 9/7 DWT

Figure 5.17: Zoomed area from the interpolated Breakdancers images

Chapter 6

Conclusions and Future Work

In this work we have presented an innovative wavelet transform that encodes DMs taking into consideration the final result of their use in view synthesis. The transform has been defined using the lifting procedure and based on the assumption that a DM is a PWP signal and that the sharpness of its edges need to be preserved. Starting from that point, the different elements of the lifting procedure have been adapted to the scenario, creating an innovative edge-avoiding filtering technique. The coefficient split and the prediction filters have been defined for planar signals, while the neighborhood of each odd is chosen so that filtering across edges is avoided. The algorithm also includes a new filter normalization approach which ensures that the transform coefficients follow a PWP model after transforming a PWP signal.

In order to suit the needs of our algorithm, a new edge detector technique, which renders the edge information in a new grid structure, has been designed as well.

Thanks to these adaptations, we not only improve the quality of the reconstructed DMs (as compared to existing methods), but also eliminate the artifacts on the synthesized images, which are induced by filtering operations that cross edges. When DMs are encoded using our method, the synthesized images obtained by view interpolation present a significantly better perceptual quality.

In our experiments, we have also seen that assuming that DMs are piece-wise constant images is sufficient, in the vast majority of the cases, for designing adaptive wavelet transforms. Nonetheless, since this is just a particular case of PWP images, our

transform is completely valid for DMs, while also extensible to different scenarios based on PWP signals.

As for future studies, many elements of this work can be revisited and improved in order to be adapted to different scenarios. First, future lines of work could focus in the joint coding of depth and video sequences. Also, the PWP model should be extended to a piece-wise polynomial one so that it can be adapted to other signals apart from DMs. Ways for making the transform closer to orthogonality should be developed too. In addition, the basis set by our edge detector technique should also be exploited in order to achieve greater precision in the detection of edges; which has a deep repercussion in the performance of our transform and many other applications. Other studies could be focused in adapting the normalization technique presented in this work to other applications that require that the model of the signal remains constant through the whole transform process. Moreover, in an improved version of our transform, it would be necessary to consider using simplified EM (requiring fewer bits) for lower bit rates. Other elements of our transform, such as the shape of the paths and the criteria presented in Chapter 3, should be revisited and adapted when developing later versions of our transform in different scenarios.

Appendix A

Edge Detector

As has been shown in previous chapters, the location of the edges is a very important element in our algorithm. In Chapter 3 the algorithm use of the edge information was explained in detail. There, a new grid in which some positions corresponded to the pixels in the image while some others were only used for locating the edges was introduced. Note that, most existing edge detection techniques do not render their outputs in such grid but, on the contrary, they use what we have named *edge pixels*. This means that when an edge is found between a pair of adjacent pixels, the edge detector assigns to this edge the position of one of these two pixels. The decision of which pixel is chosen to be an *edge pixel* does not take into consideration which side of the edge each pixel is located in. In piece-wise planar images, this implies that one cannot tell which plane *edge pixels* belong to. Fig. A.1 shows an EM obtained by applying one of these techniques, where, clearly, the location of the *edge pixels* is not consistent with the position of the pixels in the planes of the image. Since our algorithm requires precise information about the location of edges in the image so that pixels from different planes do not get filtered together, these methods are not valid for our purposes. Using a *pixel and edge grid* such as the one presented in Chapter 3, on the contrary, one can always tell on which side of the edge each pixel is located, and, thus, mixture between pixels of different planes can be avoided.

Our work includes the creation of a novel edge detection technique that locates the edges in the *pixel and edge grid*. Since the edge detection is not the main objective of our work, the improvement of such technique is left for future studies. Section A.1

Figure A.1: Edge Map obtained using Canny ([Can86]) edge detector. The vertical edge on the left should be completely straight.

explains in detail how the edges are found and presented in the output, while Section A.2 shows how this information has been manipulated and coded so that the bit rate consumed by its transmission to the decoder is minimized.

A.1 Edge Detection Algorithm

As the vast majority of edge detectors, ours works based on the directional gradient of the image. This means that the gradient is computed all around the image, and it is assumed that there is an edge in those positions where the intensity of the gradient is higher than a certain threshold. The edge detection algorithm is shown in Algorithm 1.

Given a DM $X(i,j)$ of size $N \times M$ the first step that our edge detector takes is the creation of a new variable, which we will name $Y(u,v)$ of the size of the mentioned *pixel and edge grid*; i.e., it has $(2N-1) \times (2M-1)$ positions. For each pixel (i,j) in X the corresponding position in the new grid is $(2i-1, 2j-1)$. Therefore, all the positions of the form $(2n-1, 2m-1)$ $\forall n, m \geq 1$ represent the positions of the pixels in the images, whereas the rest of the positions are potentially edges. These potential edges are located between each pixel and its six immediate neighbors in the vertical, horizontal and both diagonal directions. As can be observed in Fig. A.2, the positions of the form $(2n-1, 2m)$ and $(2n, 2m-1)$ correspond to edges between neighbors along the vertical and horizontal direction respectively, whereas positions such as $(2n, 2m)$ correspond to edges in the diagonal directions. Note that while

each horizontal/vertical edge separates a pair of pixels, each diagonal edge implies the separation of two pairs of pixels.

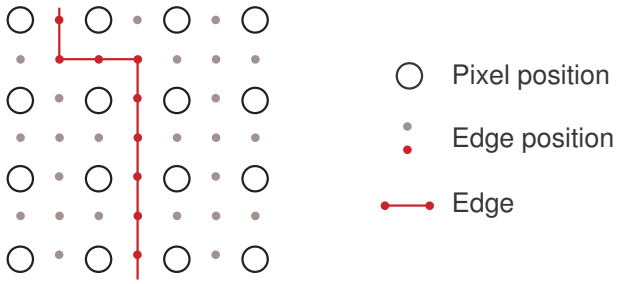


Figure A.2: Representation of an edge in the *pixel and edge grid*.

This method is very similar to the one used in [MD08]. There, the edge grid is formed only by the positions corresponding to the diagonal edges (see Fig. A.3). This way, the edge grid is dual to the one used by the pixels of the image. The horizontal and vertical edges are obtained by combining nearby diagonal pixels. As seen in Section A.2 after our edge map is coded, the resulting edge map is equivalent to the one in [MD08].

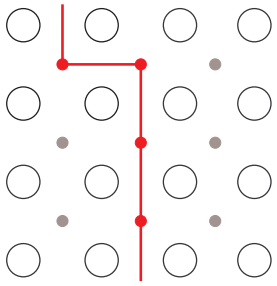


Figure A.3: Edge grid used in [MD08].

Once $Y(u, v)$ is created, each edge position is assigned the value of the gradient of the signal in that location. For the horizontal and vertical edges, this value consists of the difference of intensity between the two adjacent pixels on the image. As for the diagonal edges, since there are two pairs of pixels involved, the value assigned is computed as the maximum of the two differences of intensity. These differences are given as absolute values. Obviously, the coefficients corresponding to the pixels in the original image have zero value. Mathematically:

$$
\begin{aligned}
&Y(2n-1,2m-1) = 0,\\
&Y(2n-1,2m) = |X(n,m) - X(n,m+1)|,\\
&Y(2n,2m-1) = |X(n,m) - X(n+1,m)|,\\
&Y(2n,2m) = max\{|X(n,m) - X(n+1,m+1)|, |X(n,m+1) - X(n+1,m)|\}.
\end{aligned} \tag{A.1}
$$

Once the gradient has been computed in all positions, the threshold is defined. Let $q = \{Y(2n-1,2m), Y(2n,2m-1), Y(2n,2m)\ \forall n,m \geq 1\}$ be the set of all the gradient values obtained in the previous step, then the threshold τ is computed as $\tau = m_q + 0.6 \cdot \sigma_q$, where m_q stands for the mean of the values in q and σ_q represents its standard deviation.

Finally the *edge map* is created as a binary image of size and structure equal to $Y(u,v)$. This image, which we name $Z(u,v)$, contains value 1 for the positions where the gradient is higher than the threshold τ and 0 elsewhere. Mathematically:

$$
Z(u,v) = \begin{cases} 1 \text{ if } Y(u,v) > \tau \\ 0 \text{ if } Y(u,v) < \tau \end{cases}
$$

When this is done, $Z(u,v)$ represents the edge map of the DM $X(i,j)$. Fig. A.4 represents the whole edge detection process.

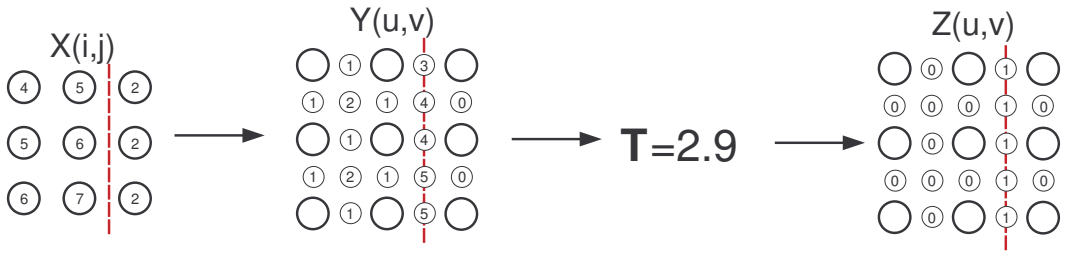


Figure A.4: Example of the computation of the *edge map*

The use of this technique has one major drawback (as compared with other methods): the output of this system has almost twice as many positions as the output of normal edge detectors (whose *edge maps* have the same size as the original image) which

means that it requires more resources for its storage and transmission. Section A.2 explains how this *edge map* is encoded so that this problem is overcome.

A.2 Edge Map Coding

Since edge maps are sent to the decoder as side information, their size and bit rate consumption can have a great repercussion on the performance of the whole algorithm. Moreover, using the *pixel and edge grid* explained before there is a great quantity of unnecessary information. The EM coding algorithm is shown in Algorithm 2.

First, all the positions corresponding to pixels in the image do not provide information about the presence of edges, therefore they should not be sent to the decoder. These positions represent 36% of the total positions on the image, so by simply eliminating these pixels the size of the whole edge map is significantly reduced. Nonetheless, the size of the resulting edge map is still considerably greater than the one provided by methods that do not use the *pixel and edge grid*.

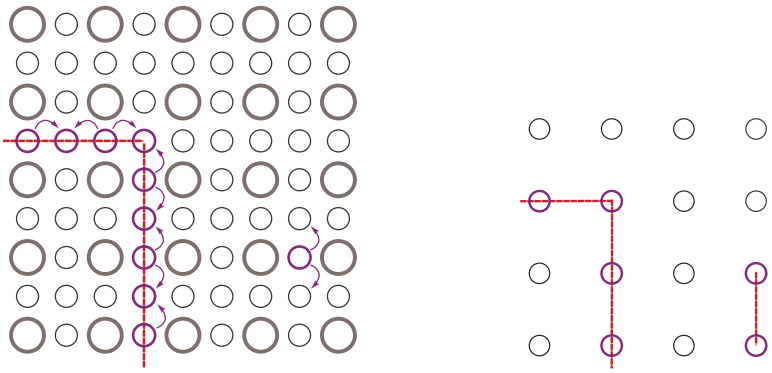
In order to achieve greater compression, we will only retain the edge positions corresponding to the diagonal edges, which generates an EM of size $(N-1) \times (M-1)$. The information of vertical an horizontal edges is retained as follows:

Let $(w, z) = (2n, 2m-1)$ ($(w, z) = (2n-1, 2m)$) be a position on $Z(u, v)$ that represents an horizontal (vertical) edge, i.e., $Z(w, z) = 1$. Then, the closest positions that represent diagonal edges are forced to have value 1, $Z(2n, 2m-2) = Z(2n, 2m) = 1$ ($Z(2n-2, 2m) = Z(2n, 2m) = 1$) regardless of their previous values. Note that the edges positions located on the boundaries of the image will only change the value of one diagonal edge pixel. Also, all the diagonal edges that had already value 1 will retain their value while the ones that had value 0 might change it. Fig. A.5 illustrates the operation.

Mathematically, if $Z_2(i, j)$ represents the re-sized $(N-1) \times (M-1)$ Edge Map, and $\vee$ stands for the logical operator "*or*", then:

$$Z_2(i, j) = Z(2i, 2j) \vee Z(2i-1, 2j) \vee Z(2i, 2j-1) \vee Z(2i+1, 2j) \vee Z(2i, 2j+1). \quad \text{(A.2)}$$

In fact, Z_2 is only used for the storage and transmission of the edge information. Since our algorithm requires that the edges be located in the $(2N-1) \times (2M-1)$ grid,

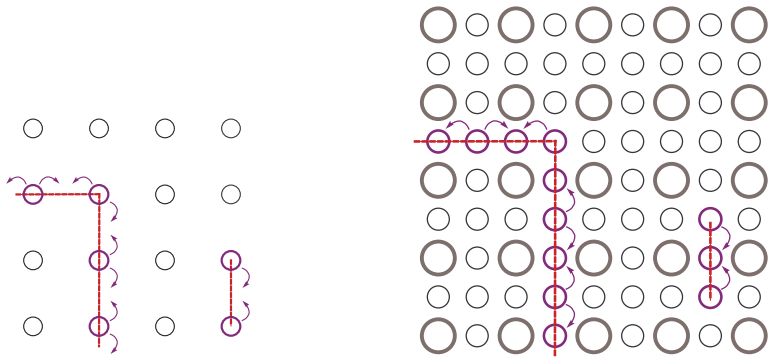


(a) Extension of the information in Z (b) Resulting Z_2

Figure A.5: Generation of Z_2.

the information needs to be resized before applying the transform. For this operation the positions of potential horizontal and vertical edges both of whose closest diagonal edges position have value 1 are considered to be representing an edge, and therefore, they are also given value 1. The edge map resizing algorithm is shown in Algorithm 3. Mathematically, if $Z_3(u, v)$ is the reconstructed edge map, and $\wedge$ stands for the logical operator "*and*", then:

$$\begin{aligned} &Z_3(2i, 2j) = Z_2(i, j), \\ &Z_3(2i+1, 2j) = Z_2(i, j) \wedge Z_2(i+1, j), \\ &Z_3(2i, 2j+1) = Z_2(i, j) \wedge Z_2(i, j+1), \\ &Z_3(2i-1, 2j-1) = 0. \end{aligned} \tag{A.3}$$



(a) Extension of the info. from Z_2 (b) Resulting Z_3

Figure A.6: Generation of Z_3.

Clearly, A.2 is not injective, thus it is not invertible. This means that $Z_3 \neq Z$; however, the information of both EM is very similar. Since both the encoder and the decoder need to use the same edge information, they will both use $Z_3(u, v)$, while the EM sent will be $Z_2(i, j)$.

Note that this method is equivalent to the one presented in [MD08], where only the information of the diagonal edges was used (see Fig. A.3). In their algorithm, only the paths that connect pixels in the same row/column are considered, thus, a grid that only includes diagonal edges, together with an easy way for extending this information to obtain horizontal and vertical edges, is sufficient for their algorithm. Since in our case the shape of the paths is more complex, the pixel and grid used by the algorithm is defined so that the horizontal and vertical edges are also included (as in Fig. A.2), which makes the computation of any $e_{(i,j)(k,l)}$ much simpler.

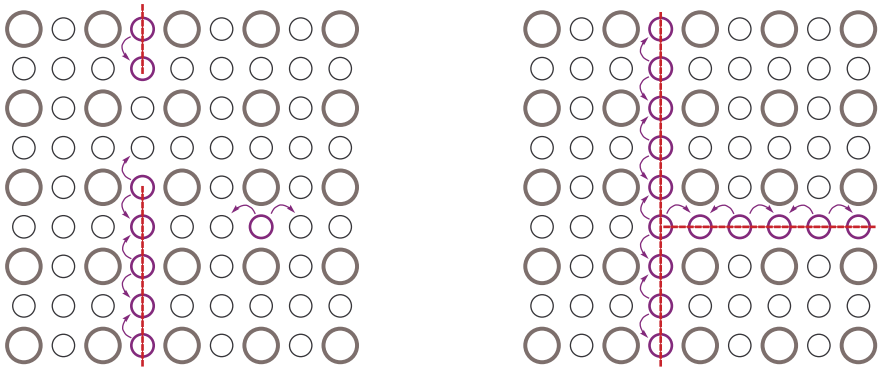
The use of this technique has one major benefit: the edge map is compressed. However, there are other aspects that are implied:

These technique may close edges that were incorrectly left open by the edge detector explained in the previous section. Fig. A.7 shows a simple example of how this phenomenon can take place. This is good for our transform, since now the planes are more correctly separated.

However, other edges are created in positions where there were not supposed to be any. As shown in Fig. A.7 two separate edges located close together can easily generate new edges in positions located between them. This phenomenon can be extremely harmful on noisy EM; since, by using this method, positions that represent false isolated edges would be connected to other edges, as it is also shown in Fig. A.7.

On the whole, the gain achieved thanks to the reduction in the bit consumption compensates the possible errors generated by the re-sizing. Moreover, since the generation and treatment of EMs is not an objective of this work, a more optimal solution is not studied here, but left for future studies.

Table A.1 shows some examples of the bit consumption required when the EMs are coded and when they are not, together with the Hamming distance existent between the uncoded EM (Z) and the one obtained using the coding and decoding techniques explained above (Z_3). Note that since the EMs are coded using JBIG (which is very efficient for binary signals regardless of their size), even though the number of



(a) Generation of Z_2 (b) Generation of Z_3

Figure A.7: The re-sizing of the EM can close incomplete edges, but also generate false ones.

positions of the EM has been greatly reduced thanks to the coding technique used (the number of positions is reduced to around $\frac{1}{4}$ in the coded EM), the gain in terms of bit consumption does not decrease as dramatically (bpp decreases around 20%). Also, note that the size of the DMs is 1024×768, which means that the number of positions of the corresponding EMs is 2047×1535 (3142145 positions), thus a Hamming distance of 5000 can be considered very small.

Image	bpp original EM	bpp coded EM	Hamming distance
Ballet view 3	0.0024	0.0019	5032
Ballet view 5	0.0022	0.0018	5065
Breakdancers view 3	0.0042	0.0035	4664
Breakdancers view 5	0.0047	0.0039	5076

Table A.1: Bits per pixel dedicated to the EM and Hamming distance from the uncoded EM to the coded one for different images.

Algorithm 1 Edge detection algorithm.

$[N_x, N_y] \leftarrow size(X)$
$Y \leftarrow$ matrix of size $(2 \cdot N_x - 1) \times (2 \cdot N_y - 1)$
$Y(n,m) \leftarrow 0 \ \forall (n,m)$
{Compute gradient}
for $n = 1$ to N_x **do**
  for $m = 1$ to N_y **do**
    $Y(2n-1, 2m) \leftarrow |X(n,m) - X(n,m+1)|$
    $Y(2n, 2m-1) \leftarrow |X(n,m) - X(n+1,m)|$
    $Y(2n, 2m) \leftarrow max\{|X(n,m) - X(n+1,m+1)|, |X(n,m+1), -X(n+1,m)|\}$
  end for
end for
{Compute mean value of the gradient.}
{ First compute the number of edge positions.}
$w \leftarrow (2 \cdot N_x - 1) \cdot (2 \cdot N_y - 1) - N_x \cdot N_y$
{ Compute mean.}
$M \leftarrow \frac{\sum_{\forall(n,m)} Y(n,m)}{w}$
{ Compute standard deviation.}
$S \leftarrow \sqrt{\frac{\sum_{\forall(n,m)} Y^2(n,m)}{w} - M^2}$
{ Compute threshold.}
$T \leftarrow M + 0.6 \cdot S$
{ Generate binary edge map.}
$Z \leftarrow$ matrix of size$(2N_x - 1) \times (2N_y - 1)$
for $n = 1$ to N_x **do**
  for $m = 1$ to N_y **do**
    if $Y(n,m) \geq T$ **then**
      $Z(n,m) \leftarrow 1$
    else
      $Z(n,m) \leftarrow 0$
    end if
  end for
end for

Algorithm 2 Edge map coding algorithm

$[2 \cdot N_x - 1, 2 \cdot N_y - 1] \leftarrow size(Z)$
$Z_2 \leftarrow$ matrix of size $(N_x - 1) \times (N_y - 1)$
{Compute reduced EM.}
for $i = 1$ to $N_x - 1$ **do**
  for $j = 1$ to $N_y - 1$ **do**
    $Z_2(i,j) \leftarrow Z(2i, 2j)$ or $Z(2i - 1, 2j)$ or $Z(2i, 2j - 1)$ or $Z(2i + 1, 2j)$ or $Z(2i, 2j + 1)$
  end for
end for

Algorithm 3 Edge map decoding algorithm

$[N_x - 1, 2N_y - 1] \leftarrow size(Z_2)$
$Z_3 \leftarrow$ matrix of size $(2 \cdot N_x - 1) \times (2 \cdot N_y - 1)$
{Compute full-size EM.}
for $i = 1$ to $N_x - 1$ **do**
  for $j = 1$ to $N_y - 1$ **do**
    if $Z_2(i, j) = 1$ **then**
      if $Z_2(i + 1, j) = 1$ **then**
        $Z_3(2 \cdot i + 1, 2 \cdot j) \leftarrow 1$
      end if
      if $Z_2(i, j + 1) = 1$ **then**
        $Z_3(2 \cdot i, 2 \cdot j + 1) \leftarrow 1$
      end if
      {First and last rows/columns of Z_3 must be computed separately.}
      if $i = 1$ **then**
        $Z_3(1, 2 \cdot j) \leftarrow 1$
      end if
      if $i = N_x - 1$ **then**
        $Z_3(2 \cdot N_x - 1, 2 \cdot j) \leftarrow 1$
      end if
      if $j = 1$ **then**
        $Z_3(2 \cdot i, 1) \leftarrow 1$
      end if
      if $j = N_y - 1$ **then**
        $Z_3(2 \cdot i, 2 \cdot N_y - 1) \leftarrow 1$
      end if
    end if
  end for
end for

Appendix B

Mathematical Proofs

In Section 2.3.3 two mathematical equations that can be used in the computation of filters with two vanishing moments have been provided. In this appendix, the proof of why these expressions minimize the intensity of the detail coefficients is given. The collinearity conditions stated in that same section are also proved here. For all the following proofs, all the pixels are assumed to form a perfect 2D plane. Therefore, the value of each pixel in the image is equal to:

$$X(i,j) = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.1}$$

We also recall the mathematical operation applied during the prediction of a certain odd (i,j), since it will be the starting point of all mathematical reasoning:

$$d(i,j) = X(i,j) - \sum_{(k',l') \in \mathcal{N}_{i,j}} p_{i,j}(k',l') X(k',l'). \tag{B.2}$$

Section B.1 explains the mathematical reasoning behind the filters expression used for two-pixel neighborhoods while Section B.2 does the same with three-pixel neighborhoods.

B.1 Two-pixel Neighborhood Approximation

Given a certain odd pixel (i, j) and a neighborhood $\mathcal{N}_{i,j} = \{(k_1, l_1), (k_2, l_2)\}$ of a planar 2D signal, it can be proved that an exact approximation of the odd's value can be build using these two evens if the three pixels (two evens plus the odd) are collinear. When using these three pixels, the expressions (B.1) and (B.2) can be combined in order to express the detail coefficient computation as a vector operation:

$$d(i,j) = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.3}$$

Since the three nodes are collinear, their positions can be expressed as $v = m \cdot u + n$ $\forall (u, v) = (i, j), (k_1, l_1), (k_2, l_2)$. Written as a vector operation:

$$\begin{pmatrix} i & j & 1 \end{pmatrix} = \begin{pmatrix} i & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & m & 0 \\ 0 & n & 1 \end{pmatrix}. \tag{B.4}$$

Note that if $i = k_1 = k_2$ the parameter m will not exist ($m \to \infty$). However, in such case the operation can be computed identically by using the ordinate coordinate instead of the abscissa, and $m = 0$ ($u = n$). Mathematically:

$$\begin{pmatrix} i & j & 1 \end{pmatrix} = \begin{pmatrix} j & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 \\ n & 0 & 1 \end{pmatrix}. \tag{B.5}$$

For the rest of our reasoning we will assume that i, k_1 and k_s are not equal, and, therefore (B.4) will be the expression used. Applying (B.4) in (B.3) leads to:

$$d(i,j) = \begin{pmatrix} i & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & m & 0 \\ 0 & n & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & 1 \\ k_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & m & 0 \\ 0 & n & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.6}$$

Applying the distributive property of the sum:

$$d(i,j) = \left(\begin{pmatrix} i & 1 \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & 1 \\ k_2 & 1 \end{pmatrix} \right) \cdot \begin{pmatrix} 1 & m & 0 \\ 0 & n & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.7}$$

If the prediction was perfect, the intensity of the detail coefficient would be zero.

$$d(i,j) = 0 = \left(\begin{pmatrix} i & 1 \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & 1 \\ k_2 & 1 \end{pmatrix} \right) \cdot \begin{pmatrix} 1 & m & 0 \\ 0 & n & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.8}$$

Which leads to:

$$\begin{pmatrix} i & 1 \end{pmatrix} = \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & 1 \\ k_2 & 1 \end{pmatrix}. \tag{B.9}$$

Finally, the expression of the filter is:

$$\overrightarrow{p}_{i,j} = \begin{pmatrix} i & 1 \end{pmatrix} \cdot \begin{pmatrix} k_1 & 1 \\ k_2 & 1 \end{pmatrix}^{-1}. \tag{B.10}$$

Which is the same as the one given in Section 2.3.3.

Note that if $k_1 = k_2$ the inverse cannot be computed. However, in this case, as mentioned above, the coordinates used would be j, l_1 and l_2, which would provide the expression:

$$\overrightarrow{p}_{i,j} = \begin{pmatrix} j & 1 \end{pmatrix} \cdot \begin{pmatrix} l_1 & 1 \\ l_2 & 1 \end{pmatrix}^{-1}. \tag{B.11}$$

So far, an expression of the prediction filters has been given only for the case where the three pixels are collinear. However, can the approximation be formed in the case that the three nodes are not located in the same 1D line inside the 2D space? Intuition says that this is not possible, since the function $X(i,j)$ has three dimensions (the two position values and the intensity). Therefore, in order to parameterize the function that passes though the three pixels, two evens do not provide enough information.

Applying a similar mathematical reasoning to the one followed for collinear pixels, we can see how the detail coefficient cannot be minimized completely in the case that $(i,j),(k_1,l_1)$ and (k_2,l_2) are not collinear.

First of all, since pixels are not collinear, (B.4), (B.5) and (B.6) cannot be used. Therefore, the new line of reasoning must start at (B.3). The step from that point is trivial:

$$d(i,j) = 0 = \left(\begin{pmatrix} i & j & 1 \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \end{pmatrix} \right) \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.12}$$

Consequently:

$$\begin{pmatrix} i & j & 1 \end{pmatrix} = \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \end{pmatrix}. \tag{B.13}$$

We will use the notation $\mathbf{H}_2 = \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \end{pmatrix}$. Since $\mathbf{H}_2$ is a *full rank* matrix, and has *linearly independent* (LI) rows, the pseudoinverse [RM72] ($\mathbf{H}_2^+$) can be computed so that $\mathbf{H}_2 \cdot \mathbf{H}_2^+ = \mathbf{I}$, where $\mathbf{I}$ stands for the *identity matrix*. Therefore:

$$\overrightarrow{p}_{i,j} = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \mathbf{H}_2^+. \tag{B.14}$$

And (B.3) is then:

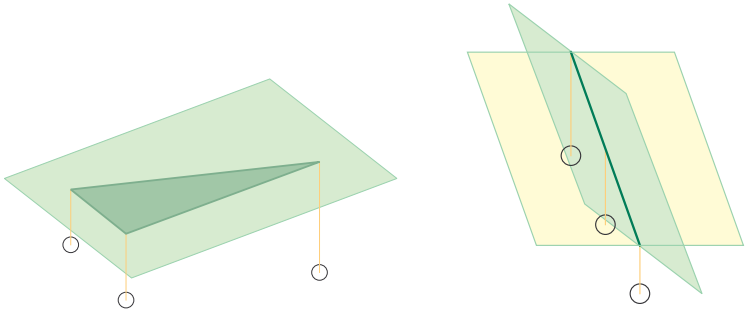
$$d(i,j) = 0 = \begin{pmatrix} i & j & 1 \end{pmatrix} - \left(\mathbf{I} - \mathbf{H}_2^+ \cdot \mathbf{H}_2 \right) \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.15}$$

However, since $\mathbf{H}_2^+ \cdot \mathbf{H}_2 \neq \mathbf{I}$ (this is a property of every *pseudoinverse* matrix of this kind) it cannot be ensured that the detail coefficient is minimized.

Consequently, it has been proved that for a planar signal, a neighborhood formed by two evens can only minimize the detail coefficient's intensity if those evens are collinear with the odd to be predicted.

B.2 Three-pixel Neighborhood Approximation

Given the odd node (i, j) and its neighborhood $\mathcal{N}_{i,j} = \{(k_1, l_1), (k_2, l_2), (k_3, l_3)\}$, an approximation of the value $X(i, j)$ is computed as the intensity of the plane formed by the three even pixels in the odd's position. In order to define this plane completely, the even pixels must not be collinear in the 2D space, since, if they are there is always a degree of freedom and the plane is not correctly defined in one dimension. This can be intuitively understood thanks to Fig. B.1.



(a) Non-collinear pixels define a plane (b) Collinear pixels cannot define a plane

Figure B.1: Definition of planes using 3 pixels.

Also, when finding the mathematical expression of the prediction filters, this condition appears very clear. Similarly to (B.3), the prediction of (i, j) in the case that the image is a perfect plane is given by:

$$d(i,j) = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \\ k_3 & l_3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{B.16}$$

Applying a reasoning equivalent to the one in the previous section: in order to get intensity zero for the detail coefficient $(d(i, j) = 0)$ we have:

$$\begin{pmatrix} i & j & 1 \end{pmatrix} = \overrightarrow{p}_{i,j} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \\ k_3 & l_3 & 1 \end{pmatrix}. \tag{B.17}$$

Consequently:

$$\overrightarrow{p}_{i,j} = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \\ k_3 & l_3 & 1 \end{pmatrix}^{-1} . \tag{B.18}$$

This operation can only be computed when $\mathbf{H}_3 = \begin{pmatrix} k_1 & l_1 & 1 \\ k_2 & l_2 & 1 \\ k_3 & l_3 & 1 \end{pmatrix}$ is invertible. Basic algebra proves that this is only possible if the three vectors $\begin{pmatrix} k_n & l_n & 1 \end{pmatrix}$ are LI. This implies that the three positions in the 2D space (k_n, l_n) are not collinear. If this condition is followed and the four pixels (three evens and one odd) form a perfect plane the detail coefficient will be zero:

$$d(i,j) = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} - \overrightarrow{p}_{i,j} \cdot \mathbf{H}_3 \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} i & j & 1 \end{pmatrix} \cdot \left(\mathbf{I} - \mathbf{H}_3^{-1} \cdot \mathbf{H}_3 \right) \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = 0. \tag{B.19}$$

Bibliography

[Can86] J. F. Canny, *A computational approach to edge detection*, IEEE Trans Pattern Analysis and Machine Intelligence **8** (1986), no. 6.

[Cut52] C. C. Cutler, *Differential quantization of communication signals*, July 1952.

[DS97] I. Daubechies and W. Sweldens, *Factoring wavelet transforms into lifting steps*, Journal of Fourier Analysis and Applications **4** (1997), no. 3, Journal of Fourier Analysis and Applications.

[Hei49] W. Heisenberg, *The physical principles of quantum theory*, 1st ed., General Publishing Company, 1949.

[jbi93] *JBIG, Progressive Bi-level Image Compression*, ISO/IEC ITU Recommendation T.82, 1993.

[KOL+09] W.S. Kim, A. Ortega, P. Lai, D. Tian, and C. Gomila, *Depth map distortion analysis for view rendering and depth coding*, Proc. of ICIP'09, 2009.

[KP97] B.-J. Kim and W. A. Pearlman, *An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT)*, Data Compression Conference **0** (1997).

[LOD+09] P. Lai, A. Ortega, C. Dorea, P. Yin, and C. Gomila, *Improving view rendering quality and coding efficiency by suppressing compression artifacts in depth-image coding*, Proc. of VCIP'09, 2009.

[Mal08] S. G. Mallat, *A wavelet tour of signal processing*, 3rd ed., Academic Press, 2008.

[MD08] M. Maitre and M.N. Do, *Joint encoding of the depth image based representation using shape-adaptive wavelets*, Proc. of ICIP'08, 2008.

[Mid] Middlebury, *Middlebury stereo vision data set*, http://vision.middlebury.edu/stereo/.

[PM05] E. Le Pennec and S. Mallat, *Sparse geometric image representations with bandelets*, IEEE Transactions on Image Processing **14** (2005), no. 4, 423–438.

[RM72] C.R. Rao and S. K. Mitra, *Generalized inverse of matrices and its applications*, Proc. Sixth Berkeley Symp. on Math. Statist. and Prob. **1** (1972), 601–620.

[SHLY08] N. I. Cho S. H. Lee, S. H. Lee and J. H. Yang, *A motion vector prediction method for multi-view video coding*, International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2008.

[SMS+07] A. Smolic, K. Mueller, N. Stefanoski, J. Ostermann, A. Gotchev, G.B. Akar, G. Triantafyllidis, and A. Koz, *Coding algorithms for 3DTV: A survey*, Circuits and Systems for Video Technology, IEEE Transactions on **17** (2007), no. 11, 1606–1621.

[SO09] G. Shen and A. Ortega, *Tree-based wavelets for image coding: Orthogonalization and tree selection*, Proc. of PCS'09, May 2009.

[SP] M. Sakalli and W. A. Pearlman, *Set partitioning in hierarchal trees (spiht) algorithm in matlab program language*, http://www.cipr.rpi.edu/research/SPIHT/.

[Swe96] W. Sweldens, *The lifting scheme: A custom-design construction of biorthogonal wavelets.*, Appl. Comput. Harmon. Anal. **3** (1996), 186–200.

[Tan04] M. Tanimoto, *Free viewpoint television - ftv*, Proceedings of 2004 Picture Coding Symposium (2004).

[Tol76] G. P. Tolstov, *Fourier series*, 1st ed., General Publishing Company, 1976.

[Uni] "Tanimoto Lab Nagoya University", http://www.tanimoto.nuee.nagoya-u.ac.jp/english.

[VBLVD06] V. Velisavljevic, B. Beferull-Lozano, M. Vetterli, and P.L. Dragotti, *Directionlets: Anisotropic multidirectional representation with separable filtering*, IEEE Transactions on Image Processing **15** (2006), no. 7, 1916–1933.

[WN03] R. M. Willet and R. N. Nowak, *Platelets: A multiscale approach for recovering edges and surfaces in photon-limited medical imaging*, IEEE Transactions on Medical Imaging **22** (2003), no. 3.

[XGZ05] W. Gao X. Guo and D. Zhao, *Motion vector prediction in multiview video coding.*, ICIP 2005: Proceedings of the 2005 International Conference on Image Processing, 2005.

[YKK+07] K. Yamamoto, M. Kitahara, H. Kimata, T. Yendo, T. Fujii, M. Tanimoto, S. Shimizu, K. Kamikura, and Y. Yashima, *Multiview video coding using view interpolation and color correction*, Circuits and Systems for Video Technology, IEEE Transactions on **17** (2007), no. 11, 1436–1449.

[YKS07] J. Kim Y. Kim and K. Sohn, *Fast disparity and motion estimation for multi-view video coding*, IEEE Transactions on Consumer Electronics **53** (2007), no. 2, 712719.