UNIVERSITY OF CALIFORNIA,
IRVINE

Edge-Preserving Image Upscaling

THESIS

Submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Anna Díez Manjarrés

Thesis Committee:

Professor Aditi Majumder, Chair

Professor Gopi Meenakshisundaram

Professor Charless Fowlkes

2009

The thesis of Anna Díez Manjarrés is approved:

_____

_____

_____

Committee Chair

University of California, Irvine

2009

to Joaquín

to my family

Table of Contents

List of Figures

Acknowledgments

I want to dedicate some words of acknowledgement to those who made this work possible. First I want to express my gratitude to my advisor Professor Aditi Majumder for giving me such an opportunity and for her guidance and thoughtful advices. Thanks also to my committee members Professor Gopi Meenakshisundaram and Professor Charless Fowlkes for their interest and time.

I would also like to especially thank Mr. Pete Balsells for giving me the opportunity to pursue my graduate studies in the University of California, Irvine, through the financial support of the Balsells Fellowship. My gratitude also goes to Professor Roger Rangel, director of the California-Catalonia Engineering Programs, for his advices, patience and time.

I also want to thank all the members of the Computer Graphics & Visualization Lab, with whom I have shared long working hours, for making me feel like one more in the group.

I am also very grateful for the support received from the catalan community in the University of California, Irvine. Special mention goes to those who were close to me

during my stay; thanks to my second family, Dolo, Marc and Ona, especially to the 'little one' for her unconditional happiness that livened me up in so many moments; thanks also to Vito, Mercè and Alfred for all the experiences and moments we spent together.

Finally, I would like to thank my parents and sister for believing in me and for all their support during these years. And of course, I would like to express the deepest gratitude to my partner, Joaquín.

Abstract of the Thesis

Edge-Preserving Image Upscaling

by

Anna Díez Manjarrés

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2009

Professor Aditi Majumder, Chair

Image upscaling is one of the most elementary image operations, and one of the most widely used in many different areas of application nowadays. It consists on increasing the size of an image and its content, and therefore involves finding unknown color values for pixels inexistent in the original image. Traditionally, image upscaling suffers of blurring and even loss of details and edges, resulting in unfaithful upscaled versions of the original images. The present work explores new approaches to image upscaling intended to preserve edges in the upscaled image, with no other input information than the original image.

Chapter 1

Introduction

Image upscaling is the term generally used to refer to the process of estimating finer resolution images from initial coarse resolution images. Or in other words, image upscaling consists on increasing the size of images.

Image upscaling is a very important topic in the imaging research area. It is an operation basically used for image resizing and zoom-in, which are tools of great importance in many different areas of application nowadays. In surveillance, for example, they are used to inspect video frames with more detail, focusing on specific areas of the frames. Resizing is also very important to enhance traditional television formats NTSC and PAL to better fit new high definition television screens. Along the same line, and generalizing a little bit more, given the huge diversity of display devices and visual content that is available nowadays, image resizing is critical to accommodate them all and guarantee extensive cross compatibilities between devices and contents. Finally, but just to mention a few examples, imaging applications with computational and/or memory constraints

usually tend to store and work with low resolution images, even though higher resolution versions of those images are still highly desirable for display purposes.

Image upscaling is an operation that, formally speaking, involves finding color values for many more pixels than those initially available in the original image, and for which no additional information is generally available, which makes it an especially challenging problem. But additionally, in the process of estimating those color values, structural information of the original image, usually contained in edges and details, needs to be preserved in order to create a faithful upscaled version of it. Traditionally, image upscaling methods pose problems such as blurring and grid-related artifacts, among others, that affect this structural information.

The present work studies new approaches to image upscaling that aim to eliminate the above undesired effects, preserving details and the sharpness of edges in the original image.

Chapter 2

Background

Image upscaling is a specific case of the more generic imaging operation known as image warping. Image warping covers any process of spatially transforming an image by moving its pixels according to a given pixel relocation mapping function $F(m,n)$, which accepts pixel coordinates (m,n) as input, and delivers destination pixel coordinates (u,v) where the input pixel (color) has to be moved.

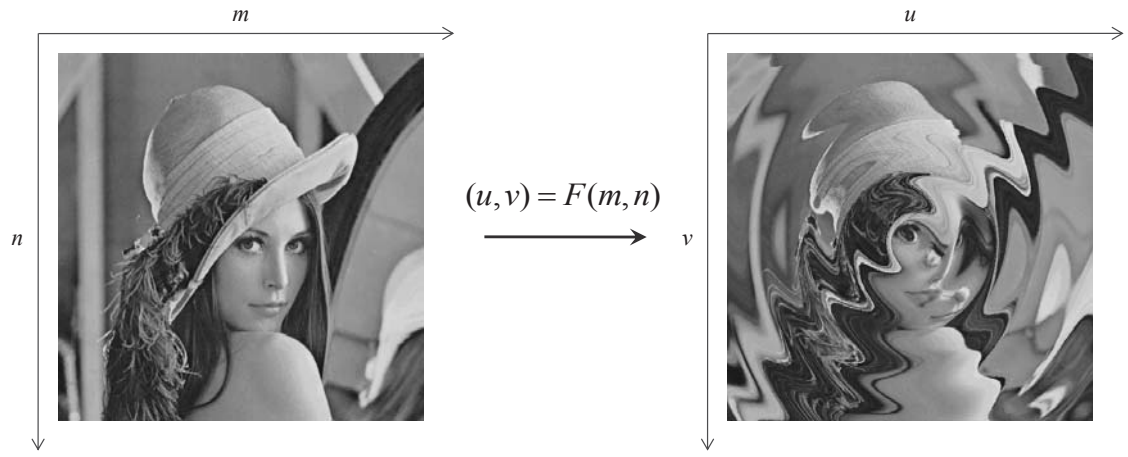


Figure 1: Image Warping

The most common examples of image warping are rotation, scaling and cropping, among many others. Figure 2 shows examples of these typical image warping operations.

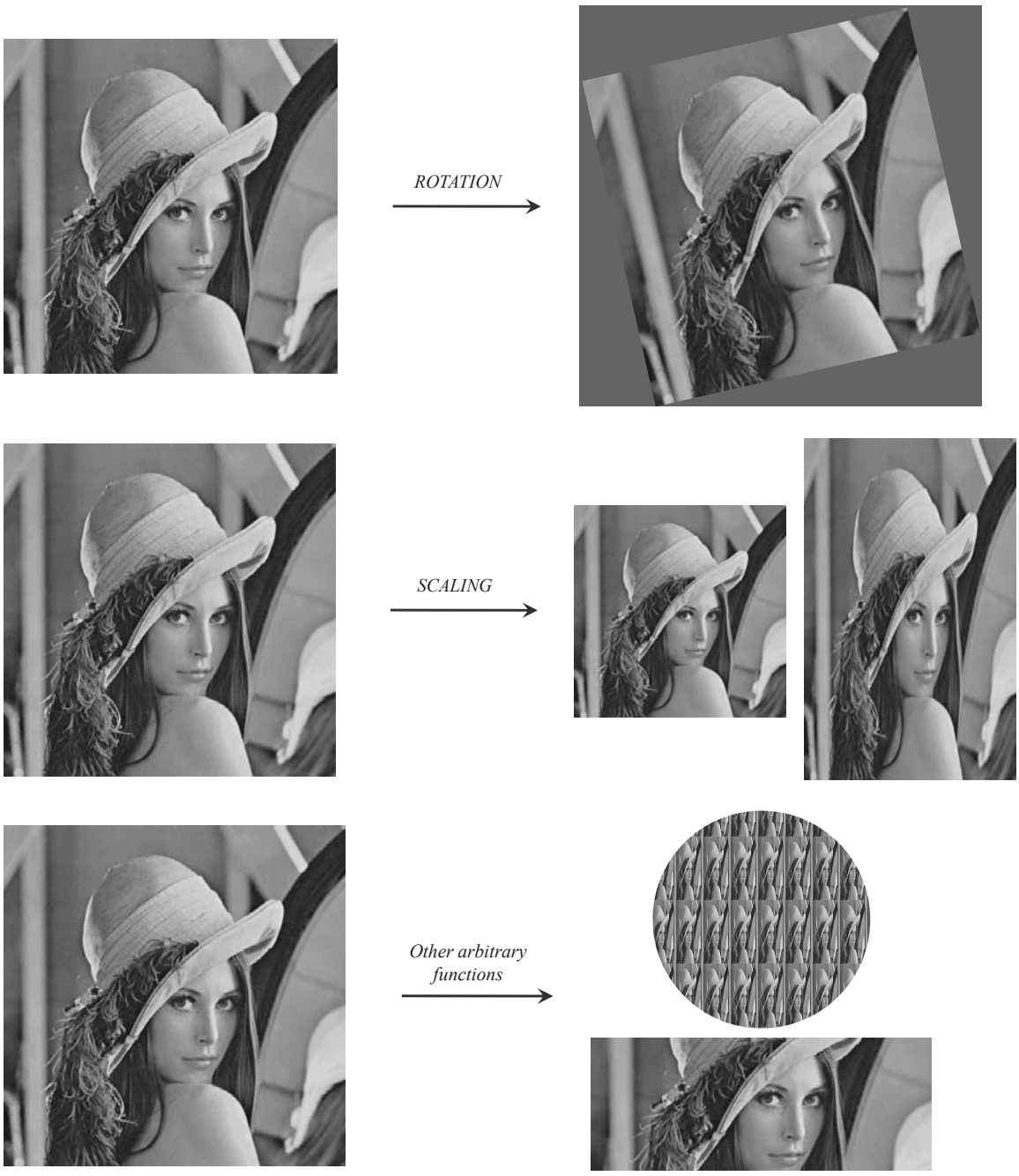


Figure 2: Typical Image Warping Operations

Any type of image warping usually involves two basic and clearly differentiated operations, namely pixel mapping and color assignment, which are explained in detail in the following sections.

2.1 Pixel Mapping

The pixel mapping operation consists on the relocation of the pixels. An association is defined by the pixel relocation function F between the pixel locations in the input image (source) and the pixel locations in the output image (destination) where they must be moved. This is nothing else but a spatial transformation of the pixel coordinates. The movement of pixels in the input image determines the formation of the output image.

In general, the transform function F can be any function $\mathbb{R}$ $\mathbb{R}$, but in practice it can be reduced to the case of positive integer input points (m,n), provided that image pixels are assumed to be always distributed in a regular grid with integer steps starting from $(0,0)$ in the upper left corner of the image. Therefore, in practice F becomes a function $\mathbb{N}$ $\mathbb{R}$.

There are two different approaches to implement the pixel mapping operation, differing in the direction in which the relocation function is applied: forward pixel mapping and backward pixel mapping.

In the case of forward pixel mapping, all the pixels in the input image are scanned, and for each of them its destination location (u,v), where the pixel has to be moved in the output image, is computed by applying the pixel relocation function F to its original

location (m,n), that is $(u,v) = F(m,n)$. At the end, all the pixels from the input image are mapped into locations in the output image.

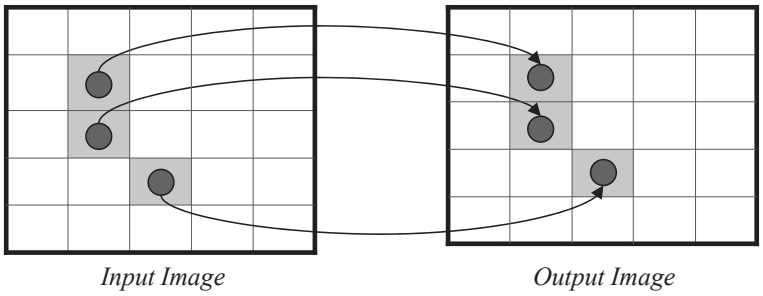


Figure 3: Forward Pixel Mapping

On the other hand, in the case of backward pixel mapping all the pixels in the output image are scanned and for each of them its source location (m,n) from which the pixel has to be moved in the input image is computed by applying the inverse pixel relocation function F^{-1} to its original location (u,v), that is $(m,n) = F^{-1}(u,v)$.

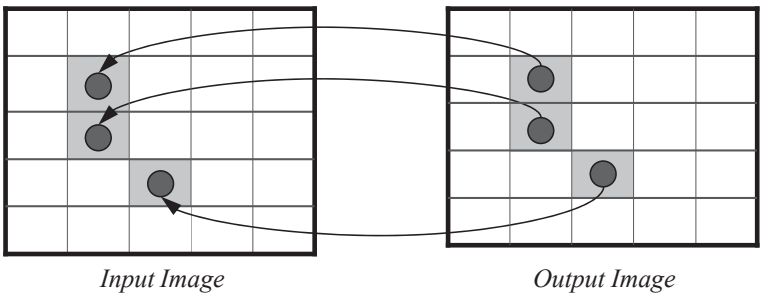


Figure 4: Backward Pixel Mapping

To better understand the pixel mapping operation, the representative case of image rotation is introduced here. As its name indicates, the goal of an image rotation is, given an input image, to rotate all of its pixels β degrees pivoting on the center of the image.

For the purpose of this explanation, let us consider the case of a counter clockwise rotation.

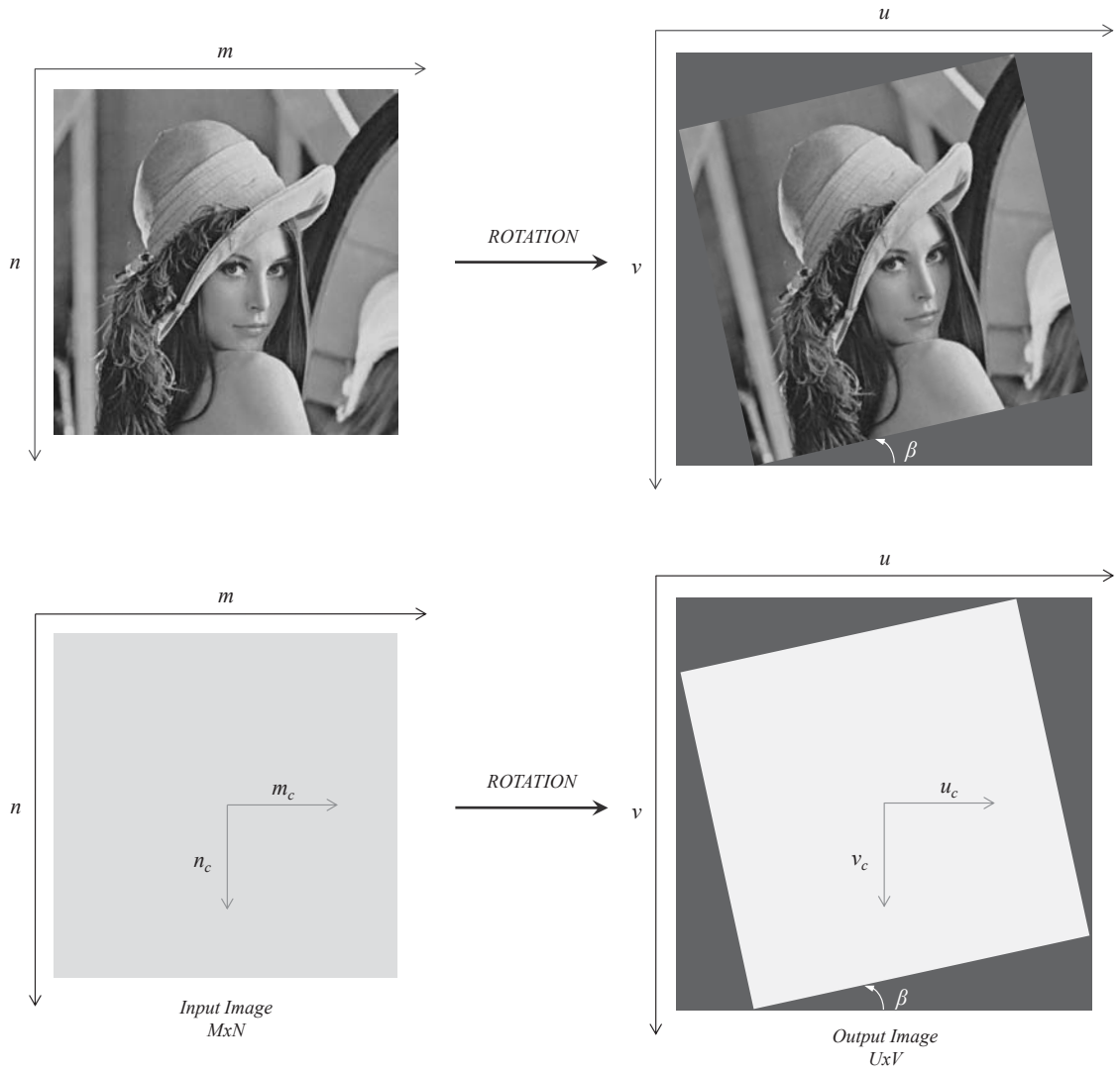


Figure 5: Image Rotation

Given a *MxN* input image, with coordinates (m,n) referred to the upper left corner of the image as shown in Figure 5, we first need to compute the coordinates (m_c,n_c) referred to the center of the image, and transform them into radial coordinates (r,α).

$$m_c = m - \frac{M}{2}$$
$$n_c = n - \frac{N}{2} \tag{2-1}$$

$$r = \sqrt{m_c^2 + n_c^2}$$
$$\alpha = \arcsin\left(\frac{n_c}{r}\right) \in \left[-180°, 180°\right] \tag{2-2}$$

The rotated coordinates (u_c, v_c), referred to the center of the output rotated image, are computed as:

$$u_c = r \cdot \cos(\alpha + \beta)$$
$$v_c = r \cdot \sin(\alpha + \beta) \tag{2-3}$$

And then referred back to the upper left corner of the output rotated image.

$$u = u_c + \frac{U}{2}$$
$$v = v_c + \frac{V}{2} \tag{2-4}$$

Where the size UxV of the output rotated image is computed as:

$$U = M \cdot |\cos\beta| + N \cdot |\sin\beta|$$
$$V = M \cdot |\sin\beta| + N \cdot |\cos\beta| \tag{2-5}$$

Compacting of all the above equations into a single expression we obtain the forward pixel mapping function F, which takes as only inputs the pixel coordinates in the input image (m, n) and the desired angle β.

$$u = \sqrt{\left(m-\frac{M}{2}\right)^2+\left(n-\frac{N}{2}\right)^2}\cdot\cos\left(\arcsin\left(\frac{n-\frac{N}{2}}{\sqrt{\left(m-\frac{M}{2}\right)^2+\left(n-\frac{N}{2}\right)^2}}\right)+\beta\right)+\frac{U}{2}$$
$$v = \sqrt{\left(m-\frac{M}{2}\right)^2+\left(n-\frac{N}{2}\right)^2}\cdot\sin\left(\arcsin\left(\frac{n-\frac{N}{2}}{\sqrt{\left(m-\frac{M}{2}\right)^2+\left(n-\frac{N}{2}\right)^2}}\right)+\beta\right)+\frac{V}{2} \quad (2\text{-}6)$$

Similarly, the equivalent equations for the case of backward pixel mapping can also be found, resulting in a function F^{-1} that takes as only inputs the pixel coordinates in the output rotated image and the desired angle β.

$$m = \sqrt{\left(u-\frac{U}{2}\right)^2+\left(v-\frac{V}{2}\right)^2}\cdot\cos\left(\arcsin\left(\frac{v-\frac{V}{2}}{\sqrt{\left(u-\frac{U}{2}\right)^2+\left(v-\frac{V}{2}\right)^2}}\right)-\beta\right)+\frac{M}{2}$$
$$n = \sqrt{\left(u-\frac{U}{2}\right)^2+\left(v-\frac{V}{2}\right)^2}\cdot\cos\left(\arcsin\left(\frac{v-\frac{V}{2}}{\sqrt{\left(u-\frac{U}{2}\right)^2+\left(v-\frac{V}{2}\right)^2}}\right)-\beta\right)+\frac{N}{2} \quad (2\text{-}7)$$

2.2 Color Assignment

The second operation involved in image warping is color assignment. As pointed out above, for both pixel mapping approaches the result of the relocation function, either direct or inverse, belongs in general to the real numbers, that is $\mathbb{R}$. But digital images only contain pixels in the integer coordinates provided by their underlying regular

sampling grid, so that we are forced to somehow convert the non-integer coordinates given by the pixel mapping process into integer coordinates, valid to move the color values of the pixels from the input image to the output image.

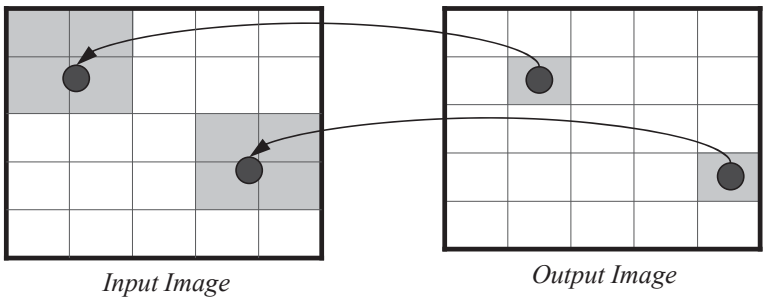


Figure 6: Color Assignment Problem

This problem can be seen as an image resampling problem. For the case of backward pixel mapping, we are interested in computing an estimate of the color value that would have been captured by a color sensor located in a given non-integer point within the original image grid. This type of problem is usually tackled with uniform interpolation techniques that compute the color estimate based on the surrounding color values within the original image grid.

On the other hand, for the case of forward pixel mapping, more complex non-uniform interpolation techniques are necessary to compute the estimates of the color values for all the integer locations within the warped image, based on the available color values in the non-uniform grid formed by the forward mapped non-integer locations. Forward pixel mapping shows then a clear downside compared to backward pixel mapping, due to the higher complexity involved in non-uniform interpolation.

Chapter 3

Image Upscaling

The present work focuses on the specific image warping case of image upscaling, which consists on increasing the size of an image. Given an input image with an initial size $M\,x\,N$, the goal of image upscaling is to generate an output image with a new size $U\,x\,V$, with $U > M$ and $V > N$, resizing the content of the original image accordingly. Figure 7 shows a typical example of image upscaling.

Figure 7: Image Upscaling

Whereas image upscaling can in general operate with different resizing factors for both horizontal and vertical dimensions, the present work only considers, for simplicity, the case of homogeneous upscaling, where both dimensions are resized by the same factor K. Therefore, the size of the upscaled image is computed as:

$$\begin{aligned} U &= \lceil K \cdot M \rceil \\ V &= \lceil K \cdot N \rceil \end{aligned} \tag{3-1}$$

Nevertheless, notice that the methods and results shown in the present study can easily be extended to the general case of non-homogeneous upscaling.

For the purpose of mathematically formulating the image upscaling operation, and similarly as the image rotation is performed by pivoting on the center of the image, let us consider the case where image upscaling is performed by stretching the image outwards using its center as reference. But notice that any other arbitrary point can be used as reference without affecting the final result.

Therefore, the first step is once again to refer the input image coordinates to its center.

$$\begin{aligned} m_c &= m - \frac{M}{2} \\ n_c &= n - \frac{N}{2} \end{aligned} \tag{3-2}$$

Then the upscaling factor is applied to obtain the corresponding upscaled image coordinates referred to its center as well.

$$\begin{aligned} u_c &= K \cdot m_c \\ v_c &= K \cdot n_c \end{aligned} \tag{3-3}$$

And finally the above new coordinates are referred back to the upper left corner of the upscaled image.

$$\begin{aligned} u &= u_c + \frac{U}{2} \\ v &= v_c + \frac{V}{2} \end{aligned} \tag{3-4}$$

Compacting the above equations into a single expression we obtain the forward pixel mapping function F, which takes as only inputs the pixel coordinates in the original image (m,n) and the desired upscaling factor K, and generates the corresponding pixel coordinates in the upscaled image (u,v).

$$\begin{aligned} u &= K \cdot \left(m - \frac{M}{2} \right) + \frac{U}{2} \\ v &= K \cdot \left(n - \frac{N}{2} \right) + \frac{V}{2} \end{aligned} \tag{3-5}$$

Similarly, the equivalent equations for the case of backward pixel mapping can also be found, resulting in a function F^{-1} that takes as only inputs the pixel coordinates in the upscaled image (u,v) and the desired upscaling factor K, and generates the corresponding pixel coordinates in the original image (m,n).

$$\begin{aligned} m &= \frac{1}{K} \cdot \left(u - \frac{U}{2} \right) + \frac{M}{2} \\ n &= \frac{1}{K} \cdot \left(v - \frac{V}{2} \right) + \frac{N}{2} \end{aligned} \tag{3-6}$$

Since forward pixel mapping has been shown to have certain practical drawbacks compared to backward pixel mapping, such as the creation of possible empty gaps in the

output image, from now on a backward approach will always be considered for the pixel mapping operation, unless it is specified otherwise.

Regarding the color assignment operation, as can be observed in the above equations, the image upscaling operation introduces $K-1$ additional pixels in each direction for every pixel in the original image, so that from the original $M\,x\,N$ color values that are available in the original image, at least $M\cdot(K-1)\,x\,N\cdot(K-1)=(U-M)\,x\,(V-N)$ new values have to be estimated.

Furthermore, the downscaled coordinates (m,n) generated by the backward pixel mapping are generally real numbers, i.e. they point to non-integer positions, falling between the integer positions that form the regular pixel grid for which color values are available in the original image. Therefore, it is not possible to directly pull out color values to be used for the corresponding upscaled coordinates (u,v).

This problem can be seen as a resampling problem, where the goal is to compute color value estimates for locations that were not sampled by the sensor that captured the image in first place, based on the available sampled color values. The image resampling problem is traditionally solved using an interpolation-based approach, which estimates each missing color value as some sort of linear combination of the surrounding available color values in the image pixel grid.

3.1 Traditional Approaches

The most extensively used types of interpolation are the nearest interpolation and the bilinear interpolation. In both of them, it is enough to consider the four closest pixels to the desired non-integer position in order to compute the new color value.

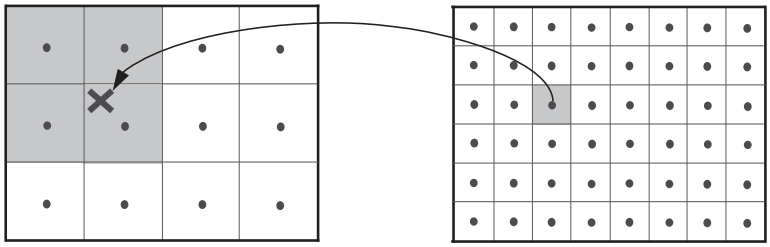


Figure 8: Color Interpolation

3.1.1 Nearest Neighbor Interpolation

In the case of nearest neighbor interpolation, the color value of the non-integer pixel location (m,n) in the original image, which is needed to be assigned to the integer pixel location (u,v) in the upscaled image, is directly estimated as the color value of the nearest integer pixel location within the original image.

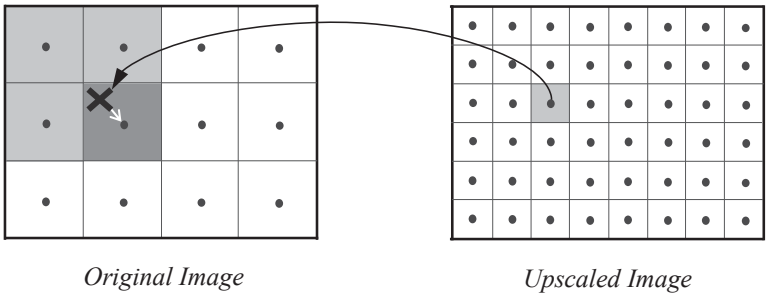
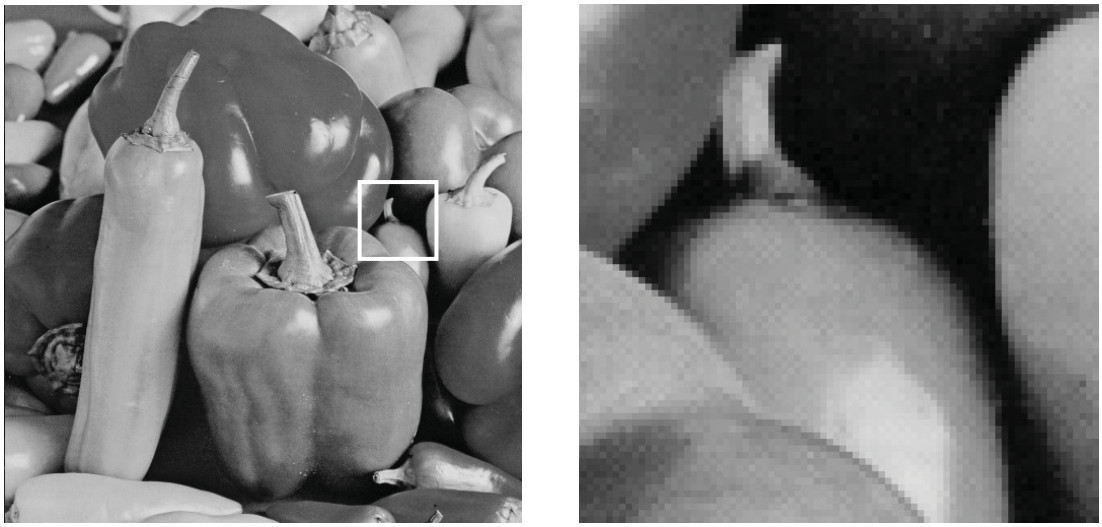


Figure 9: Nearest Neighbor Interpolation

Figure 9 shows an example of this process, where the four neighbors initially considered are in light gray and the chosen nearest neighbor is the in dark gray.

The nearest neighbor interpolation is a very easy and simple interpolation approach, very fast, but it is strongly affected by frequency aliasing and produces undesirable visual effects such as blockiness and jagging artifacts, which significantly distort lines and edges in the image. This can be clearly seen in Figure 10.



(a) **(b)**

Figure 10: Image Upscaling using Nearest Neighbor Interpolation

(a) Original Image; (b) Detail of Upscaled Image

3.1.2 Bilinear Interpolation

In the case of bilinear interpolation, the color value of the non-integer pixel location (m,n) in the original image, which is needed to be assigned to the integer pixel location (u,v) in the upscaled image, is estimated as a linear combination of the four nearest integer pixel location within the original image, with contributions (weights) inversely

proportional to their distances from (m,n). In other words, the interpolated color value is a weighted addition, based on the distance, of the color values of the four nearest neighboring integer pixel locations.

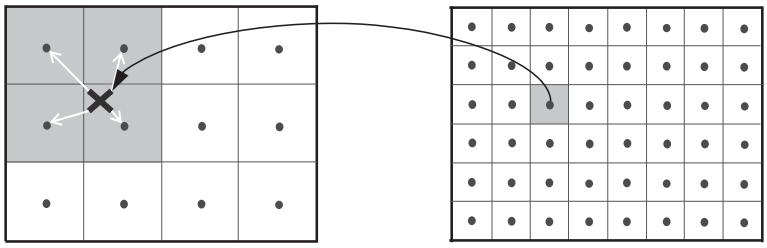


Figure 11: Bilinear Interpolation

Figure 12 shows a generic diagram of the elements involved in bilinear interpolation. The non-integer location is denoted as point G, which falls inside a square area defined by its four nearest neighbors within the image grid, being A the top-left neighbor, B the top-right neighbor, C the bottom-left neighbor, and D the bottom-right neighbor.

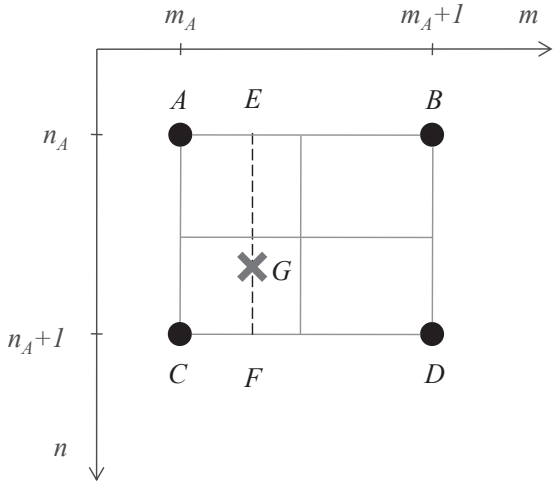


Figure 12: Elements of Bilinear Interpolation

Bilinear interpolation can be seen as two separated linear interpolations, one in each of the two horizontal and vertical directions. First, interpolated values are found for points E and F, as a previous step to the final interpolated value computation. Taking into account that $m_C = m_A$, $m_B = m_D = m_A + 1$, and $m_E = m_F = m_G$, the values for intermediate points E and F are horizontally linearly interpolated as:

$$v_E = \frac{i_B - i_G}{i_B - i_A} \cdot v_A + \frac{m_G - m_A}{m_B - m_A} \cdot v_B = \left(m_A + 1 - m_G\right) \cdot v_A + \left(m_G - m_A\right) \cdot v_B \tag{3-7}$$

$$v_F = \frac{m_D - m_F}{m_D - m_C} \cdot v_C + \frac{m_F - m_C}{m_D - m_C} \cdot v_B = \left(m_A + 1 - m_G\right) \cdot v_C + \left(m_G - m_A\right) \cdot v_D \tag{3-8}$$

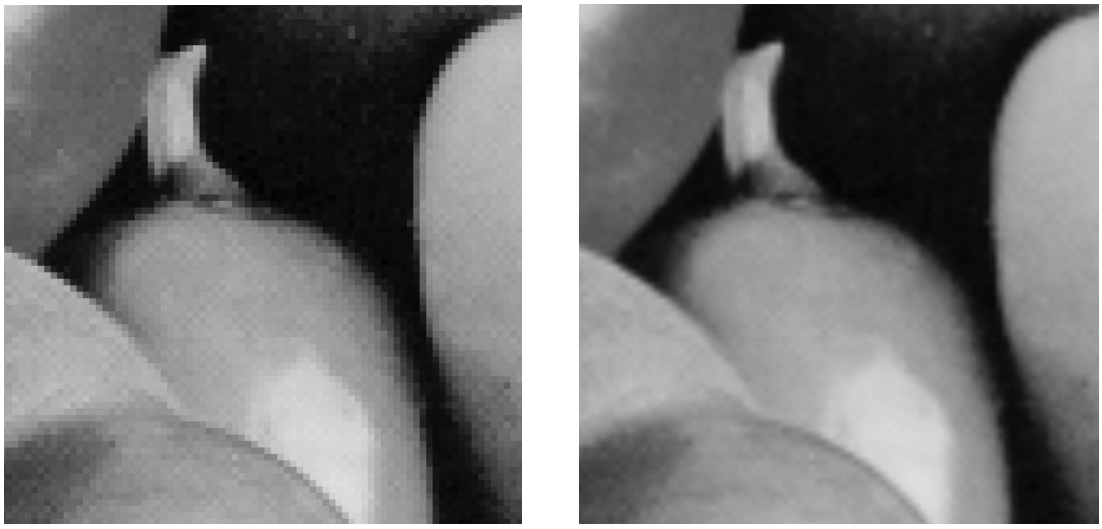
Then the final value for point G is computed as a vertical linear interpolation of points E and F, as shown below, taking into account that $n_E = n_A$ and $n_F = n_A + 1$.

$$v_G = \frac{n_F - n_G}{n_F - n_E} \cdot v_E + \frac{n_G - n_E}{n_F - n_E} \cdot v_F = \left(n_A + 1 - n_G\right) \cdot v_E + \left(n_G - n_E\right) \cdot v_F \tag{3-9}$$

Substituting values v_E and v_F into the above equation, the compact expression for the interpolated value of point G is obtained in terms of the values of the four neighbors A, B, C and D:

$$\begin{aligned} v_G = {} & \left(n_A + 1 - n_G\right) \cdot \left(m_A + 1 - m_G\right) \cdot v_A \\ & + \left(n_A + 1 - n_G\right) \cdot \left(m_G - m_A\right) \cdot v_B \\ & + \left(n_G - n_E\right) \cdot \left(m_A + 1 - m_G\right) \cdot v_C \\ & + \left(n_G - n_E\right) \cdot \left(m_G - m_A\right) \cdot v_D \end{aligned} \tag{3-10}$$

An example of image upscaling using bilinear interpolation is shown in Figure 13, including the nearest interpolation approach as well for comparison purposes.



(a) **(b)**

Figure 13: Upscaled Images Comparison

(a) Nearest Neighbor Interpolation; (b) Bilinear Interpolation

Whereas bilinear interpolation is more complex and computationally demanding than nearest neighbor interpolation, it does eliminate the artifacts produced by the latter approach by avoiding, up to certain point, frequency aliasing. But it does so at cost of blurring the detail and edges present in the original image. The weighted combination of neighbors is nothing else but a low-pass filtering operation, which is intended to eliminate frequency alias, but also fades or even eliminates high frequency components of the image, where detail and edges are contained.

As conclusion, we can say that traditional upscaling techniques based on straight color interpolation, using either nearest neighbor, bilinear or other types of interpolation, tend to introduce artifacts and/or blur the upscaled image, always resulting in a clear distortion and degradation of the edges and details in the image, where most of its important

information is contained. The present work aims to explore new upscaling methods that preserve the edge and detail information contained in the original image.

3.2 Related Work

While traditional approaches introduced in previous sections make use exclusively of the original image as the only available input information, other methods make use of additional information to help find the correct upscaled image. In [Fattal 2007] prior knowledge is used in form of edge dependencies statistics, which are obtained by carrying out measurements and modeling on large sets of training images. Such edge characteristics help them faithfully construct edges in the upscaled image. Although this method provides notable results, it shows certain drawbacks. On one hand, its training is highly time consuming. And on the other hand, its performance is very dependent on the type of images used for training, with very good results being obtained when the training set is well constrained for the type of target input image, but clearly losing quality when generality is increased in the training set or when this one is not adapted to the type of input image.

A similar idea is found in [Sun et al. 2008], where a gradient profile, learned a priori from a large number of natural images, is used to create the upscaled image. This gradient information is used as constraint for the problem of image upscaling, and helps sharpen the edges and details, while also suppressing undesirable artifacts. This method poses

similar problems as [Fattal 2007], namely a computationally demanding training, and results highly dependent on the set of training images.

In [Freeman et al. 2002], images are divided into small square partitions, typically from 5x5 to 7x7 pixels. For a given upscaling factor, a dataset of representative low resolution partitions and their typically associated high resolution partitions is found a priori. In general, for each low resolution partition in the dataset, more than one high resolution partition dependent on the characteristics of neighboring partitions may be considered, and a probabilistic model is trained to define the relationships between the low and high resolution partitions, and between neighboring high resolution partitions. Then, upscaling is performed by finding, for each partition in the original image, the dataset entry that best fits the probabilistic model, and using the corresponding high resolution partition to build the upscaled image. In addition to the facts that, once again, training is needed and results are dependent on the training dataset used, this approach shows little flexibility to apply different upscaling factors, and variable results depending on the partition size.

In [Kopf et al. 2007] a completely different approach is used. It relies on having access to an already existing high resolution image which has certain relationship with the low-resolution image that needs to be upscaled. This is the case in applications where an original high resolution image needs to be downscaled prior to certain required processing, due to computational limitations, and once the processing is performed, the resulting low resolution image has to be upscaled back to the original size. In such cases, the original high resolution image can be used as side information in an upscaling scheme

based on joint bilateral filtering, which will try to preserve high frequency transitions in regions indicated by the original high resolution image. The main problem of this method is that it is restricted to very specific applications, where the above assumptions hold.

In [Avidam and Shamir 2007] a content-aware image resizing method is introduced. For the case of image upscaling, the idea is to introduce the additional pixels that are needed to form the upscaled image in places where their insertion is less noticeable. This is quantified as an energy function: the lower the energy is, the less noticeable the insertion will be. Then, horizontal and vertical paths with minimum accumulated energy transition from side to side of the image are found for all rows and columns, which are referred to as seams. Image upscaling can be achieved by inserting the necessary amount of pixels in all horizontal and vertical seams, using simple interpolation techniques along the seams to compute the new color values. Theoretically, edge and detail areas of the image original image are avoided by the seams, provided that they would significantly increase the accumulated energy transition, and thus they are preserved. In practice, this approach poses several limitations. On one hand, its success is not guaranteed for all images, especially for highly condensed images. But the most important limitation is that, even though the dimensions of the image are uniformly scaled, the content of the image is not.

With the original image as only input information, in [Shan et al. 2008] a feedback-control scheme is used to iteratively adjust the upscaled image. The method is based on finding a model for the formation of the low resolution image from the ideal corresponding high resolution image that is to be found, and invert it. The algorithm is

based on FFT-based operations, which on one hand may be responsible for certain frequency-related artifacts introduced in the upscaled image, and on the other hand makes it computationally demanding.

Finally, in [Thurnhofer and Mitra] an adaptive interpolation algorithm is proposed. The neighborhood of each pixel in the original image is classified into one of three possible categories, namely constant, oriented (well-defined edge or detail structure) or irregular (unclear non-constant structure). Then, such classification along with certain directional information of the original image gradient control the specific type of interpolation used to find the missing color values of the neighboring unknown pixels in the upscaled image, aiming to preserve sharpness of edges and details.

The present work explores new image upscaling methods that aim to preserve edges and details in any type of input image, avoiding traditional artifact and blurring effects, with no prior training or knowledge, and no other information than the original image.

Chapter 4

Proposed Methods

As shown in the previous chapter, traditional techniques used for image upscaling, based on different approaches to straight color interpolation such as nearest neighbor, bilinear and others, tend to introduce degradation of the edges contained in the original image, either in form of jagging artifacts or in form of fading and blurring effects. In all cases, these effects are translated into a clear loss of faithfulness of the upscaled image compared to the original. Edges contain very important structural and semantic information in images, provided that edges determine the boundaries of objects forming the image, and it is critical to preserve them in order to maintain semantics of the original image unaltered

Methods explored in the present work are based on the idea that whereas the traditional interpolation approaches have been proved to degrade edges and thus have to be avoided in edge areas of the image, they do not pose significant problems and behave quite acceptably in smooth and constant areas of the image. Therefore, it would be desirable to

separate edge and non-edge (smooth) areas in the image in order to give to edge areas a special treatment which preserves their content and structure, while still applying conventional interpolation to smooth areas.

The basic goal of the methods presented here is to separate edges from non-edge areas, place them in the upscaled image respecting the color smoothness along the edges, and then fill the remaining non-edge pixels using a smart adaptation of traditional interpolation able to avoid across-edge operations, which are in last instance responsible of the undesired edge blurring effects.

4.1 Edge Detection

Edges are basically associated with significant spatial changes in the intensity of the image, and therefore are traditionally found and located making use of the derivatives of the image as a measure of edge strength. More specifically, the most widely used techniques for edge detection perform peaks detection in the first derivative of the image. While certain more complex methods use derivatives of higher degree, detecting peaks and 0-crosses in the odd and even degree derivatives respectively, the first derivative keeps complexity low while providing reasonable detection performance.

The first derivative of a 2D signal such as an image is the gradient. Given an intensity image I, its gradient ∇I is defined for each pixel as a vector containing the partial derivatives in the horizontal (denoted by m and x) and vertical (denoted by n and y) directions.

$$\nabla I(m,n) \equiv \begin{bmatrix} \dfrac{\partial I(m,n)}{\partial x} \\ \dfrac{\partial I(m,n)}{\partial y} \end{bmatrix} \equiv \begin{bmatrix} G_x(m,n) \\ G_y(m,n) \end{bmatrix} \tag{4-1}$$

The edge strength (S_E) measure of each pixel is usually computed as the magnitude of the gradient vector in that pixel

$$S_E(m,n) = \sqrt{G_x^2(m,n) + G_y^2(m,n)} \tag{4-2}$$

The above expression can be interpreted as the rate of change of the image intensity in the direction of the gradient vector, which in turn is orthogonal to the edge direction (D_E) defined by:

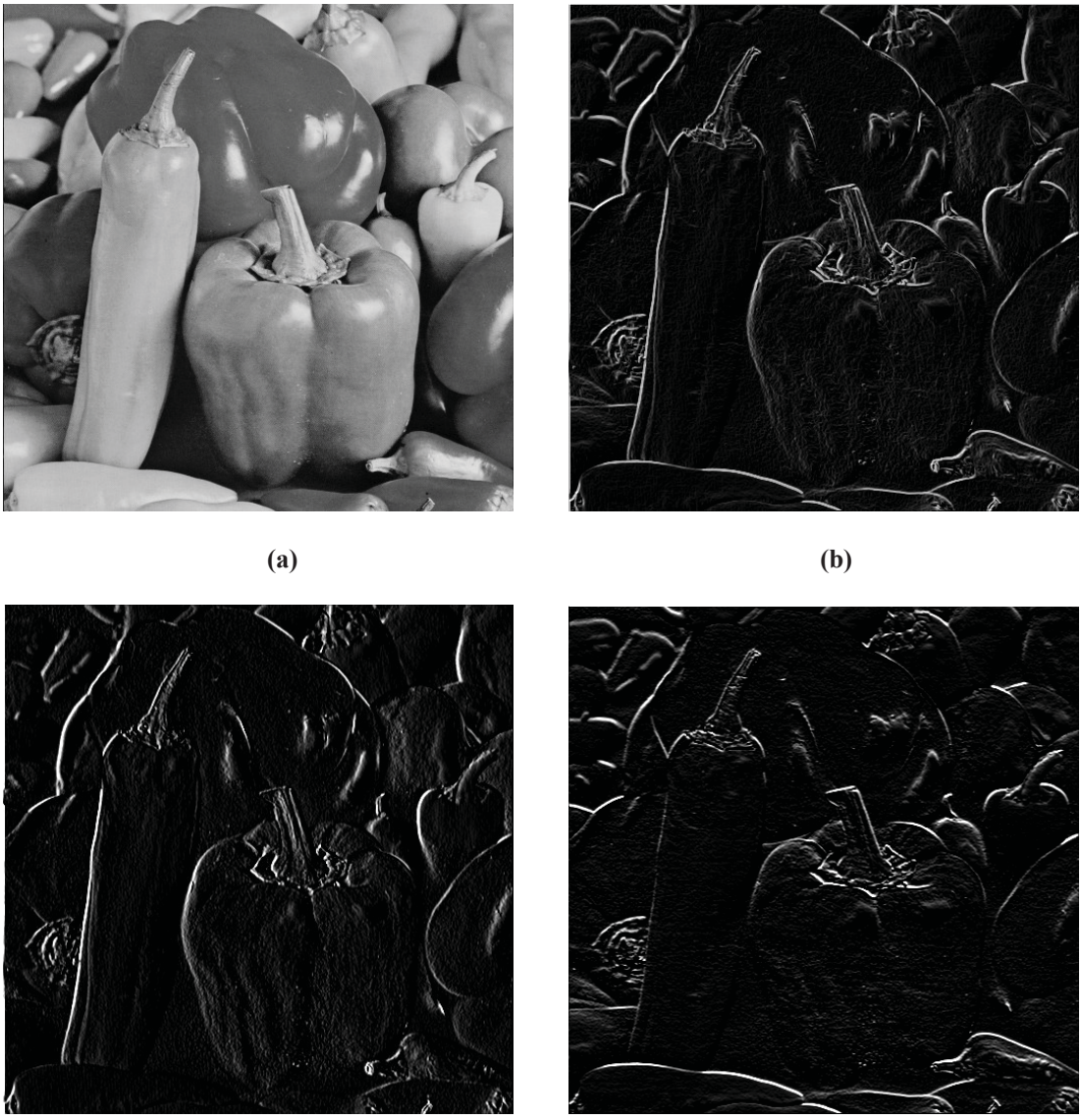
$$D_E(m,n) = \tan^{-1}\left(\frac{G_y(m,n)}{G_x(m,n)}\right) \tag{4-3}$$

In practice, the partial derivatives in each pixel are approximated over certain neighborhood of the pixel using a finite 2D differentiating filter. The most commonly used filter masks for this purpose are the Prewitt and Sobel masks.

-1	0	1	-1	-1	-1	-1	0	1	-1	-2	-1
-1	0	1	0	0	0	-2	0	2	0	0	0
-1	0	1	1	1	1	-1	0	1	1	2	1

(4-4)

Examples of the partial derivative images G_x and G_y, as well as the edge strength image S_E, computed with the Sobel filter, are shown in Figure 14.



(a) **(b)**

(c) **(d)**

Figure 14: Image Derivatives

(a) Input Image; (b) Edge Strength; (c) Horizontal Derivative; (d) Vertical Derivative

The final step in basic edge detection is thresholding of the edge strength image: pixels with an edge strength measure higher than certain minimum T (threshold) are marked in the final edges image as edge pixels, whereas pixels not exceeding such minimum are marked as non-edge pixels.

$$E(i,j) = \begin{cases} 1 & S_E(m,n) > T \\ 0 & S_E(m,n) \le T \end{cases} \tag{4-5}$$

Following the same example, the final result of edge detection using the Sobel filter can be seen in Figure 15.

Figure 15: Edges Detected by Thresholding of Edge Strength

As can be observed in the figure above, for a given reasonable threshold, certain level of image texture and noise may be detected as edges, provided that they show significant gradient magnitude comparable to regular edges. This is generally an undesired effect in edge detection algorithms.

A technique that is widely used to reduce this effect is pre-smoothing: filtering the input intensity image using a low-pass filter to eliminate high frequency texture details and noise while still preserving strong edge intensity transitions. New results using this modified scheme can be shown in Figure 16.

Figure 16: Edges Detected by Thresholding of Edge Strength with Pre-Smoothing

4.1.1 Canny Edge Detector

The Canny edge detector is the most extensively used edge detection tool in the world of image processing. While it is more complex than basic edge detectors based on the simple scheme introduced above, the Canny edge detector clearly outperforms them in detection error rate and localization of the edges.

The Canny edge detector also begins with the filtering of the input intensity image I, using in particular a gaussian filter, to obtain a smoothed version of the image I_S. Then the edge strength and direction images, S_E and D_E, are also computed using I_S.

When thresholding is directly applied to S_E, wide areas around the true edge line (maximum strength) are marked as belonging to the edge. In order to provide thin edge detection, next step performed by the Canny edge detector is nonmaxima suppression,

which consists on somehow discarding from the edge those points that do not show local maximum edge strength.

A basic algorithm to generate the nonmaxima-suppressed edge strength image N_E is to maintain the edge strength measure $S_E(m,n)$ of where such measure is superior to all its immediate neighbors along its edge direction $D_E(m,n)$, while setting to 0 those other pixels that do not meet these requirement.

The final step in the Canny edge detector is to perform hysteresis thresholding on the nonmaxima-suppressed image N_E. Hysteresis thresholding aims to overcome the problems that conventional thresholding show to avoid detecting certain false edge points while discarding some other valid edge points, and in general to detect continuous edges without gaps.

The basic process of hysteresis thresholding begins with the creation of two different thresholded images, E_H and E_L, using high and low thresholds, T_H and T_L. Notice that edge points contained in E_H are a subset of those in E_L. So that next step is to remove from E_L the edge points contained in E_H, to end up with disjuntive edges images E_H and E_L containing strong and weak edge points respectively.

$$E_L = E_L - E_H \tag{4-6}$$

Then, the final step of hysteresis thresholding consists on iteratively transferring from E_L to E_H those weak edge points that are connected to any strong edge point using N-connectivity.

Figure 17: Canny Edges Image

From this point on, N will be assumed to be 8. At the end of this process, E_H is the edges image E we are looking for, containing continuous thin edges without gaps.

4.2 Low-Resolution Edge Separation Upscaling

The Low-Resolution Edge Separation method (LES) makes use of an edge and non-edge separation scheme performed in the original size domain. In Figure 18 the block diagram of LES method is shown.

As a first step, edges are detected in the original image I, generating the edges image ε of the same size $M\,x\,N$ as the original. Edges are then linked and forward mapped using rounding to obtain an upscaled edges image that contains edges with gaps introduced by the upscaling. After performing linear scan conversion on these non-continuous edges, the final upscaled edges image E is found, which contains continuous and complete edges.

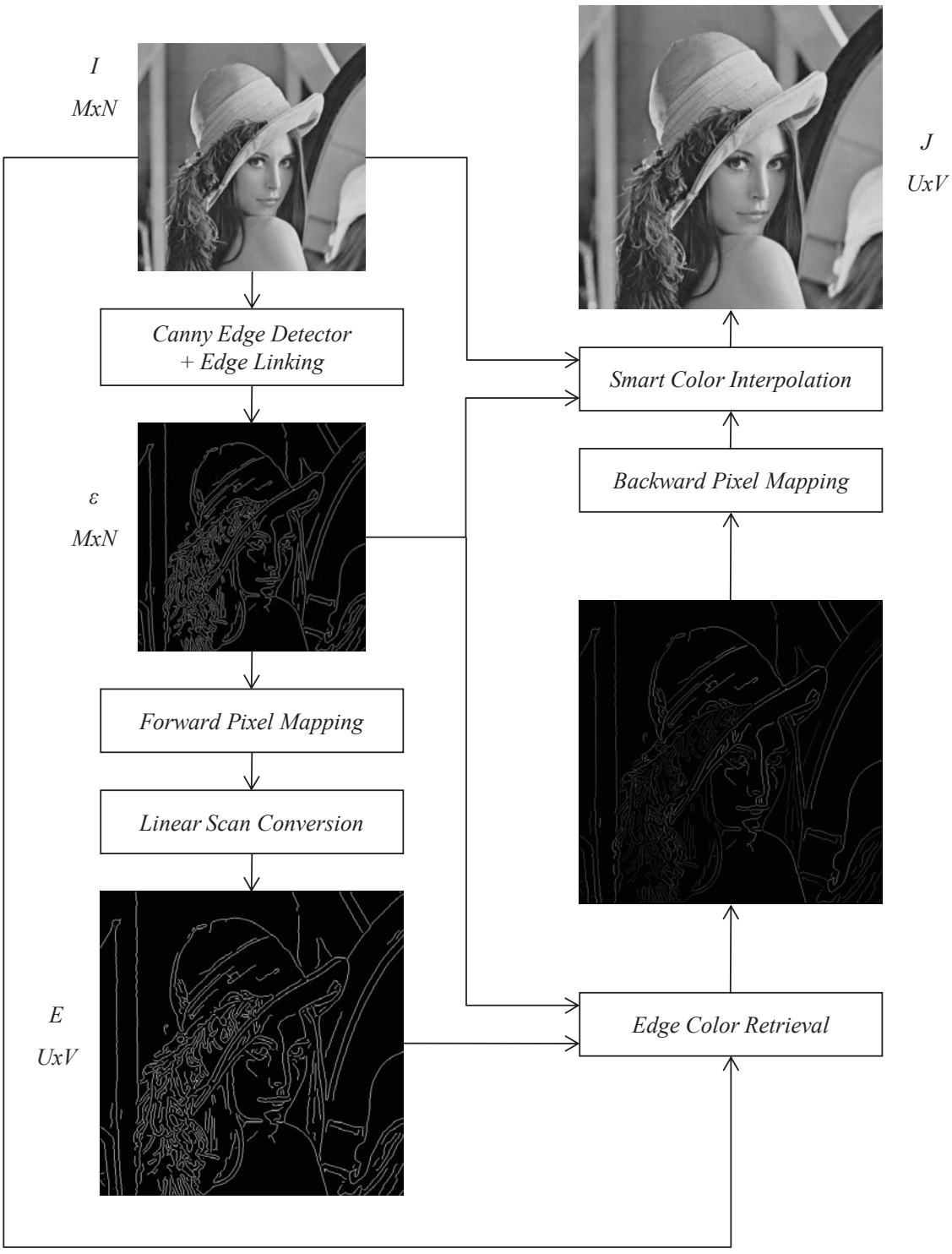


Figure 18: LES Upscaling Block Diagram

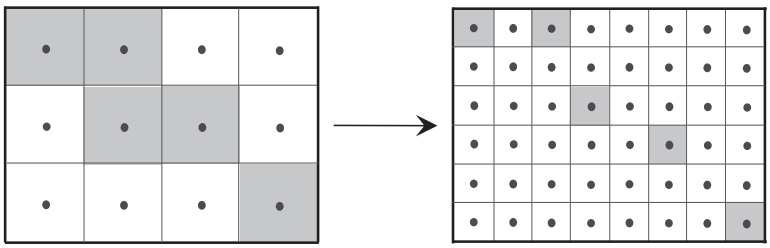
For all the edge pixels in E, color values are retrieved using color information only from the corresponding edge pixels in ε, to obtain an upscaled image J that is partially filled only with colored edge pixels. Then, in the final step the remaining empty pixels in J, which correspond to the non-edge areas, are finally backward mapped to the original image and their color values are found using a smart interpolation technique that avoids across-edge operations.

4.2.1 Detailed Description

4.2.1.1 Edge Separation

The edges image ε is found by applying the Canny edge detector explained in Section 4.1.1 to the input image I. The found edges are then processed by a linking algorithm that converts them into separate logical representations in form of linked lists of points, suitable for separated processing of edges. At the end, every edge is represented by a linked list of ordered and consecutive pixels forming the edge.

The resulting edges, pixel by pixel, are then forward mapped to their final destination in the upscaled image. Since this operation, as explained already in previous chapters, generally leads to non-integer coordinates, rounding is used to finally locate the edge points. The result is an upscaled edges image in which the pixels are disconnected from each other due to the separation introduced by the upcaling. This effect is shown in Figure 19.

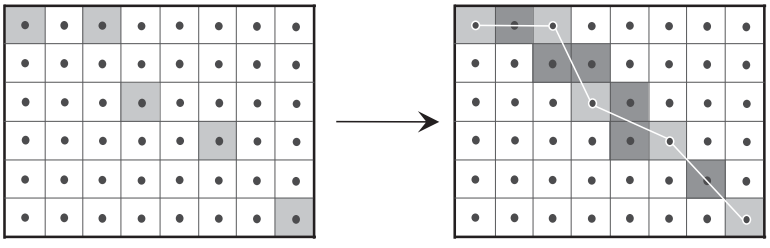


Edges Image *Upscaled Edges Image*

Figure 19: Edge Forward Pixel Mapping

These initial edge pixels in the upscaled image will be referred to as anchor edge pixels, provided that they came directly from the original detected edge.

Then linear scan conversion takes place, linking pairs of consecutive anchor edge pixels. Moving along each of the upscaled edges, from beginning to end, linear scan conversion draws the real line that connects each pair of consecutive anchor edge pixels, deciding for a given type of connectivity which integer points will be forming the final complete and continuous upscaled edge. Linked edge representations are especially suitable for this type of algorithms that require moving along the edges, and notice that the linked representation of the new generated edges can easily be obtained by orderly inserting the new pixels that have been chosen to fill the gaps.



Upscaled Edges Image *Upscaled Edges Image*

Figure 20: Linear Scan Conversion

An example of linear scan conversion can be seen in Figure 20, where the links are drawn as white lines and the new added pixels are painted in dark gray. After processing all edges in ε, the upscaled edges image E is formed, containing fully connected edges for which a linked representation is also available.

4.2.1.2 Edge Coloring

After scan conversion, color has to be retrieved for each upscaled edge based on the color of the corresponding edge detected in the original image. As first step, the color of the anchor edge pixels is directly copied from their corresponding edge pixels in the original image. The result is an edge partially colored as shown in Figure 21, where the non-anchor edge pixels are denoted by dotes.

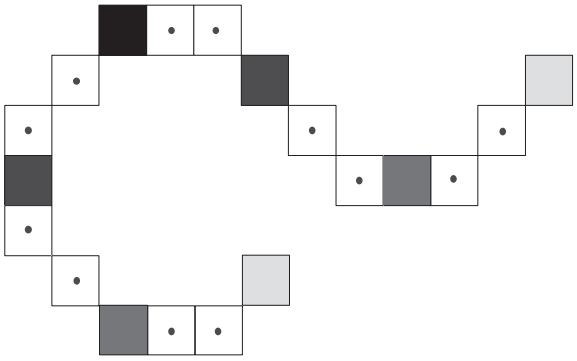


Figure 21: Partially Colored Upscaled Edge

In order to fill with color the remaining non-anchor edge pixels, a parametric representation of the edges is found first. For a given edge of length L, let $P(i)$ be its $i-\text{th}$ pixel, with $i = 0..L$, at coordinates $\big(u(i), v(i)\big)$. The parametric coordinate $t(i)$ is then computed as the accumulated chordal distance along the edge.

$$t(0) = 0$$
$$t(i) = t(i-1) + \sqrt{\left(u(i) - u(i-1)\right)^2 + \left(v(i) - v(i-1)\right)^2} \qquad i = 1..L\text{-}1 \tag{4-7}$$

Let also $C(i)$ be the color value corresponding to pixel $P(i)$. Provided that only the anchor edge pixels have been assigned a color thus far, $C(i)$ is only different from 0 if i corresponds to an anchor edge pixel. The problem of finding color values for the remaining non-anchor pixels is now reduced to the interpolation of C as a function of t, as illustrated in Figure 22.

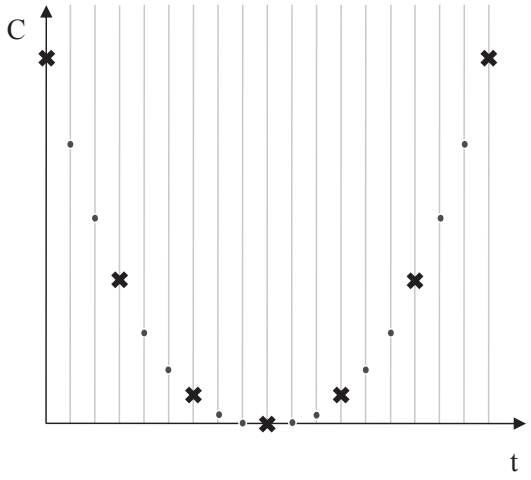


Figure 22: Edge Color as a Function of Parametric Variable *t*

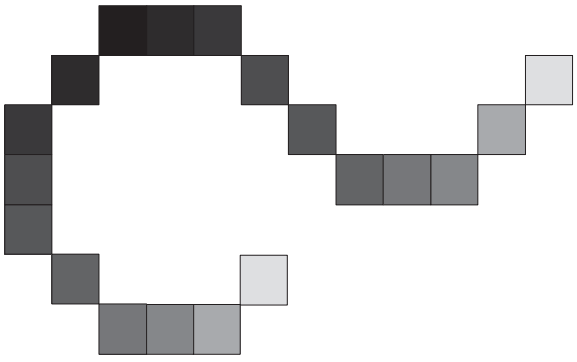


Figure 23: Fully Colored Upscaled Edge

Using a spline interpolation, the resulting colored edge of the example is shown in Figure 23. At the output of the edge color retrieval block, the upscaled image J contains color only in the edges, with all the non-edge areas still empty.

4.2.1.3 Non-Edge Coloring

The remaining two blocks in the LES scheme, namely backward pixel mapping and smart color interpolation, are responsible of filling the non-edge areas in the upscaled image J using a modified bilinear interpolation that avoids the type of across-edge operations that are responsible of the edge blurring.

The upscaled image is scanned, and every non-edge pixel is backward mapped to the original image. The found coordinates, generally non-integer, lead us to the four nearest neighbors, from which the color can be estimated. But this is not trivial task provided that the main objective of the current work is to preserve edges, which is the reason why edges have been treated separately thus far, so now it would not be acceptable to use a conventional bilinear interpolation which would treat all four neighboring pixels equally, and would therefore perform across-edge weighting when approaching edge areas. At this stage, it is critical to guarantee that non-edge pixels are filled with colors semantically coherent with the already filled edge pixels by avoiding across-edge computations during this final color interpolation.

In order to achieve the above goal, it is necessary to make use of additional information during this color interpolation besides the color values of the four closest neighbors: it is necessary to identify the possible edge configurations within the four neighbors in order

to keep for the interpolation only those neighbors that will preserve the edge structure, and discard those others that would lead to averaging across-edge. We can find this additional information in the original edges image ε.

The first step of the smart color interpolation introduced here is to find which of the four neighbors are edge pixels. This is done by looking into ε. Then the best interpolation scheme is determined accordingly, generally using a sub-set of the four neighboring colors. Two basic cases can be distinguished:

- None or all of the four neighboring pixels are edge pixels.
- One, two or three neighboring pixels are edge pixels.

It is straight forward to see that in the first case, conventional bilinear interpolation with the original four neighbors is sufficient for the purpose of preserving edges. It is the second case the one that requires additional logic to find an alternative smart interpolation scheme that avoids across-edge weighting. This new scheme will depend on the number of edge pixels found among the four neighbors and their specific configuration.

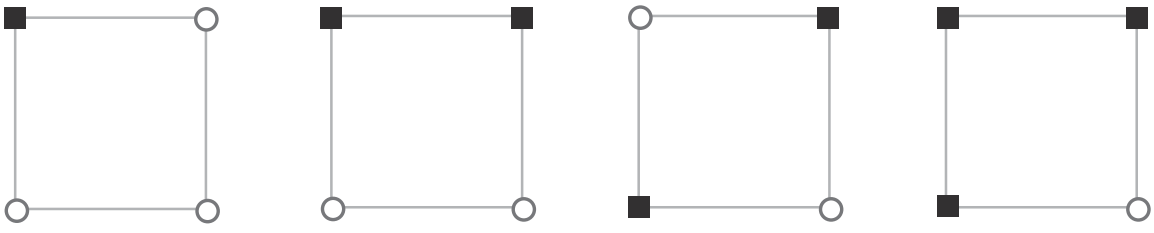


Figure 24: Possible Neighbors Configurations

Total, the four sub-cases shown in Figure 24 have to be considered, where edge pixels are denoted by solid squares: one edge neighbor, two collateral (vertical or horizontal) edge neighbors, two diagonal edge neighbors, and finally three edge neighbors.

The goal is to reduce all the alternative smart interpolation schemes to the case of conventional bilinear interpolation, but with the four neighboring colors rearranged in a way that enables us to effectively discard and keep certain sub-sets of them. The following sub-sections focus on explaining in detail how the neighboring pixel colors are rearranged in each of the above sub-cases, and it is assumed that in all of them bilinear interpolation is applied after the colors rearrangement. Edge pixels are denoted by solid squares, non-edge pixels by circles and the non-integer points from the backward pixel mapping by crosses.

One Edge Neighbor

In this case, the backward mapped point is always considered to be falling in non-edge area. Therefore, whereas the three non-edge neighbors are kept with their original color values, the color value of the edge neighbor is replaced with the average of the two non-edge neighbors falling closest to it, or in other words, that are collateral to it.

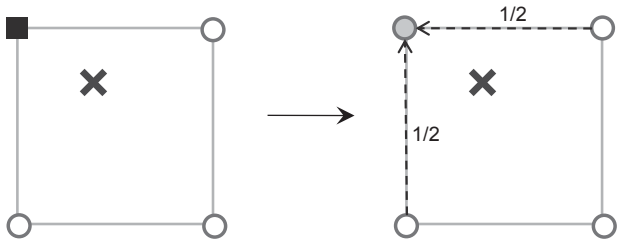


Figure 25: One Edge Neighbor

Two Collateral Edge Neighbors

In this case, the backward mapped point is also always considered to be falling in non-edge area. The color value of each of the two edge neighbors is replaced with the color value of the non-edge neighbor falling closest to it.

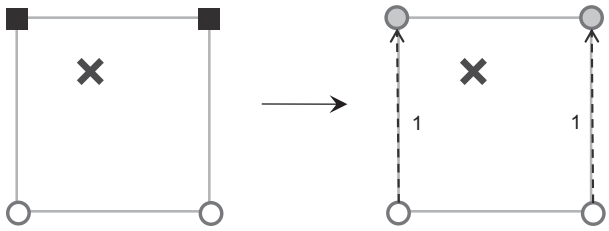


Figure 26: Two Collateral Edge Neighbors

Two Diagonal Edge Neighbors

In this case, the backward mapped point is also always considered to be falling in non-edge area. But since the edge is dividing the four neighbors grid, there are two different non-edge areas now, one on each side of the edge.

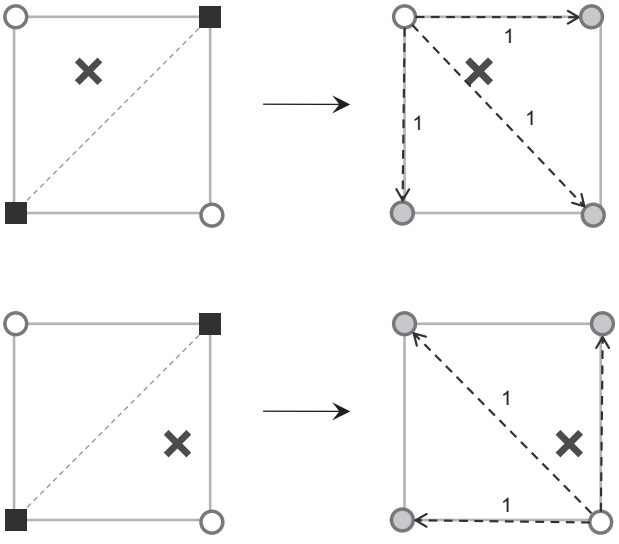


Figure 27: Two Diagonal Edge Neighbors

Therefore, it is necessary to determine on which side of the edge the backward mapped point is falling. This is done by simply finding the non-edge neighbor that falls closest to the backward mapped point, and then its color value is used to replace the color values of all the other edge and non-edge neighbors.

Three Edge Neighbors

In this case, it is necessary to determine whether the backward mapped point falls in edge or non-edge area. Considering the only non-edge neighbor and the edge neighbor opposite to it, the point is considered to fall in non-edge area when it is closer to the former or in edge area when otherwise.

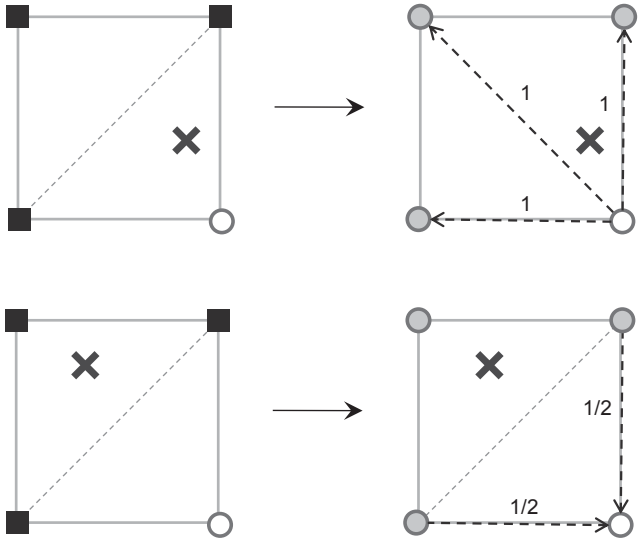


Figure 28: Three Edge Neighbors

If the backward mapped point is found to be falling in non-edge area, the color values of all the edge neighbors are replaced with the color value of the only non-edge neighbor. If on the other hand the backward mapped point is found to be falling in edge area, the color

value of the only non-edge neighbor is replaced with the average of the two edge neighbors falling closest to it, or in other words, that are collateral to it.

4.2.2 Results and Conclusions

The Low-Resolution Edge Separation (LES) scheme introduced in this section has been used to upscale several traditional test images with many different upscaling factors. Starting next page, a set of representative results is shown, using an upscaling factor $K = 4$, and the test images 'Flowers', 'Peppers', 'Monarch' and 'Yacht'. For each one of them, the original image is shown, along with detail zoom of the upscaled image generated using nearest neighbor interpolation, bilinear interpolation, and finally the LES method.

Original Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

Figure 29: LES Results for 'Flowers' Test Image

Original Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

Figure 30: LES Results for ‘Peppers’ Test Image

Original Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

Figure 31: LES Results for 'Monarch' Test Image

Original Image

Nearest Neighbor Interpolation

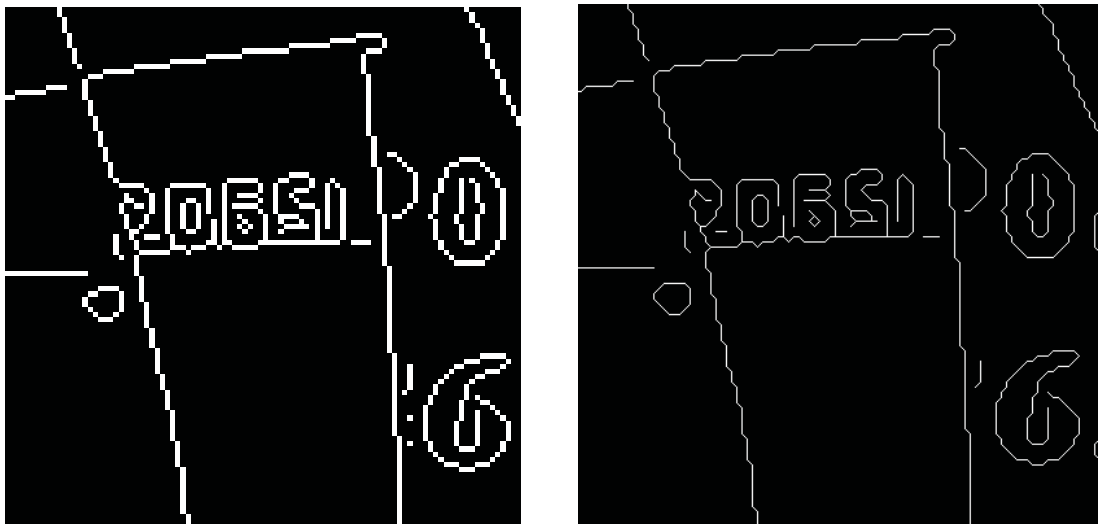
Bilinear Interpolation

LES Method

Figure 32: LES Results for 'Yacht' Test Image

Results clearly show that LES method serves to the objective of preserving edges, these being sharply defined in the upscaled images compared to the nearest neighbor and bilinear interpolation approaches.

On the other hand, edges in the LES upscaled images do not appear to be naturally shaped, with curves becoming successions of straight segments and abrupt steps. This effect is very evident in all the test images shown in the previous pages, but for the purpose of further analyzing the effect, let us focus on the 'Yacht' image.



(a) **(b)**

Figure 33: LES Edge Images ('Yacht')

(a) Original Edges Image; (b) Upscaled Edges Image

As shown in Figure 33, when we look at the detected edges in the original image, and compare side by side with the generated upscaled edges image, it is evident that the upscaled edges follow exactly the shape of the original edges, with all the limitations these have, provided their low resolution definition. Edges in the original image are being

detected with a resolution of integer pixel, which becomes a resolution of K integer pixels in the upscaled image, so that when a straight edge in the original image shifts a single pixel to any side, that becomes an abrupt step of K pixels in the upscaled edge.

4.3 High-Resolution Edge Separation

The High-Resolution Edge Separation method (HES) is intended to solve the problems observed with the Low-Resolution Edge Separation method. HES method also makes use of an edge and non-edge separation scheme, but as opposed to LES method, the edge detection is now performed in the upscaled size domain, which provides finer resolution in the localization of the edges in the upscaled image. The block diagram of HES method is shown in Figure 34.

As first step, an upscaled version of the original image I is initially computed using the conventional bilinear interpolation approach and an upscaling factor K. Edges are then detected in the resulting image $\hat{J}$, of size $U x V$, generating the edges image E of that same size. Next, found edges are linked and backward mapped using rounding to obtain the edges image ε_T of the same size $M x N$ as the original image. Due to the reduction of the edges resolution performed by the backward mapping, ε_T usually contains overdefined thick edges, which need to be thinned to generate the final original size edges image ε. Then, for all the edge pixels in E, color values are retrieved using color information only from the corresponding edge pixels in ε, to obtain an upscaled image J that is partially filled only with colored edge pixels.

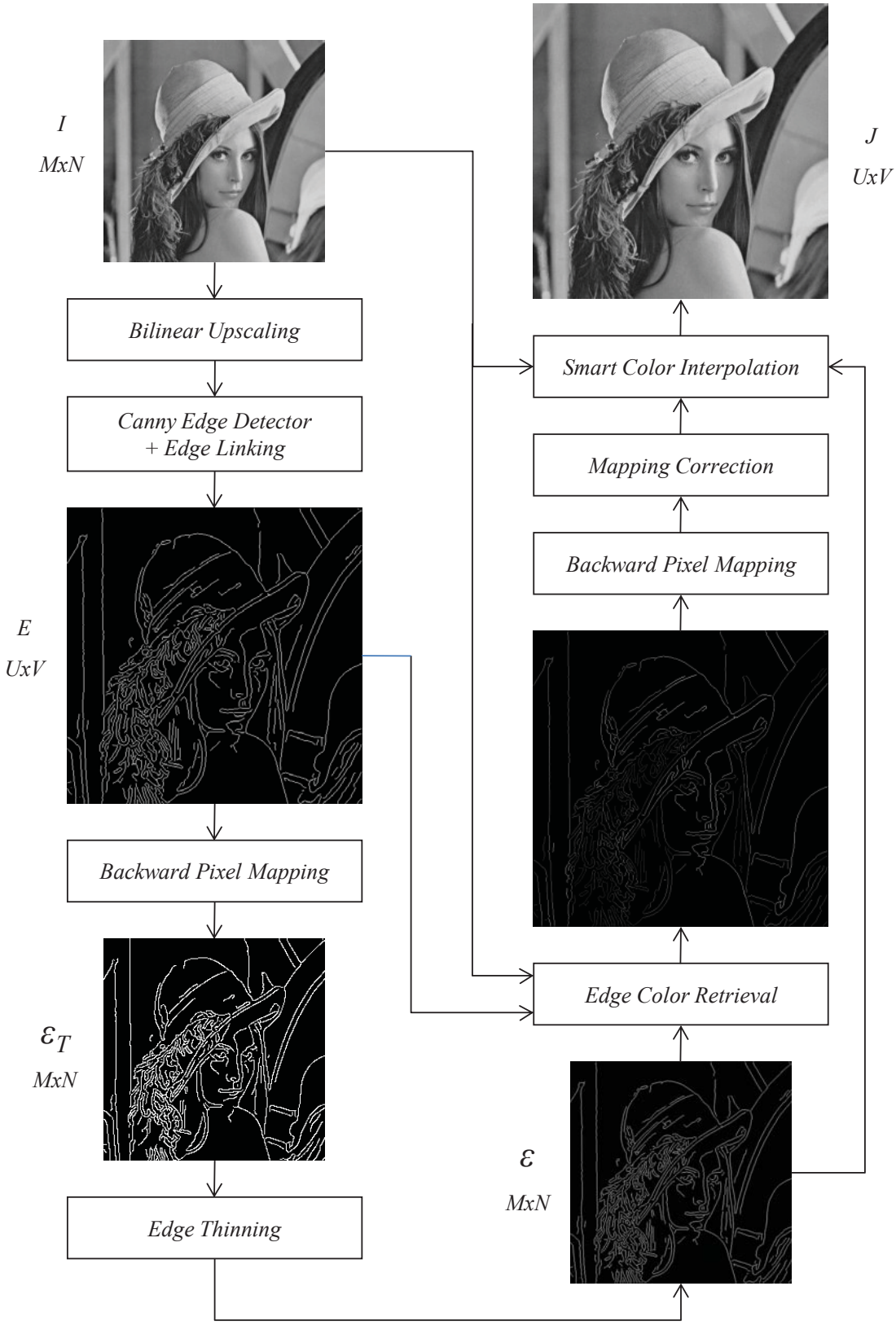


Figure 34: HES Upscaling Block Diagram

In the final step, the remaining empty pixels in J, which correspond to the non-edge areas, are finally backward mapped to the original image, with an additional correction stage, and their color values are found using a smart interpolation technique that uses the original size edge image ε to avoid across-edge operations.

4.3.1 Detailed Description

4.3.1.1 Edge Separation

As pointed out above, the first step in HES method is to perform an initial upscaling by K of the original image I using the traditional bilinear interpolation approach. The generated image $\hat{J}$ has the same size $U x V$ as the desired final upscaled image J, and even though it suffers of the blurring effect explained in previous sections, it is perfectly suitable for edge detection because, as explained in Section 4.1, low-pass filtering is usually the first step of most edge detection algorithms provided that it helps reduce false detections while maintaining the localization of the edges unaltered. Furthermore, due to these characteristics, upscaling is extensively used to detect edges with sub-pixel resolution in the original images.

So next step is to perform edge detection on $\hat{J}$ using the Canny edge detector, obtaining the upscaled edges image E of size $U x V$, and process the found edges with the linking algorithm.

Then, as opposed to the LES method, the resulting edges are backward mapped to the original size image. Since this operation generally leads to non-integer coordinates,

rounding is used to finally locate the edge pixels. The result is an original edges image ε_T in which edges are overdefined and thick due to the resolution reduction performed by the backward pixel mapping. Furthermore, due to the same reason, multiple pixels in a given upscaled edge may point to the same pixel in the corresponding original edge. An example of this effect is shown in Figure 35.

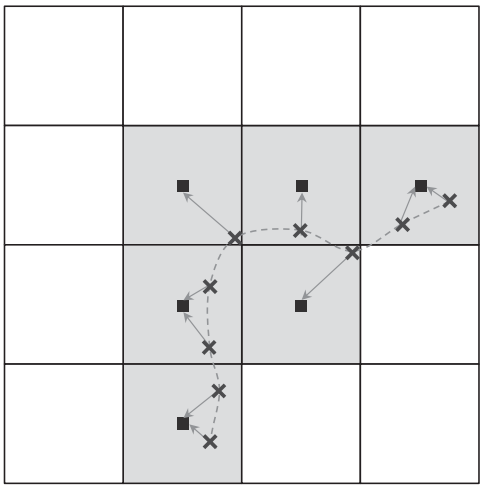


Figure 35: Thick Edge in Original Image

This effect is the opposite of the disconnected pixels effect found in the LES method, which in that case was produced by the resolution increase performed by the forward pixel mapping.

So next step is to reduce the thickness of the edges in ε_T to obtain an original edges image ε equivalent to the one that would have been found if the edge detection had been directly performed in the original image I. For each edge in ε_T, the goal is to find a minimum set of connected pixels among all the pixels in the edge, that properly define the edge.

Let us denote a given edge in the upscaled edges image E as e_H, the corresponding edge in the original thick edges image ε_T as e_{LT}, and the corresponding edge we want to find in the original edges image ε as e_L. The goal can now be restated as to insert into e_L those pixels from e_{LT} that keep 8-connectivity and are sufficient to define the edge. Pixels inserted into e_L are referred to as core pixels.

Initially, the first pixel in e_{LT} is inserted in e_L as the first core pixel, and the first pixel in e_H is picked as the corresponding anchor pixel. Then, the three steps shown below are successively carried out until the last pixel in e_H is picked as anchor pixel.

1. Consider a segment of edge e_H starting right after the last anchor pixel and all the way to its end.

2. Pixels are orderly checked moving along the current segment, looking for the smallest rounding error of the backward pixel mapping, until the corresponding pixel in e_{LT} falls at a distance longer than $\sqrt{2}$ from the last core pixel. Pixels leading to the last core pixel are skipped unconsidered.

3. When step 2 is over, the pixels from e_H and e_{LT} corresponding to the smallest rounding error found are picked as new anchor and core pixels, respectively, with the core pixel being inserted into e_L.

At the end of the process, e_L is a complete and connected thin edge, as shown in Figure 36, where the selected core pixels are denoted in dark gray as opposed to the original

thick edge pixels in light gray, anchor pixels are denoted by white crosses, and arrows show related pairs of anchor and core pixels.

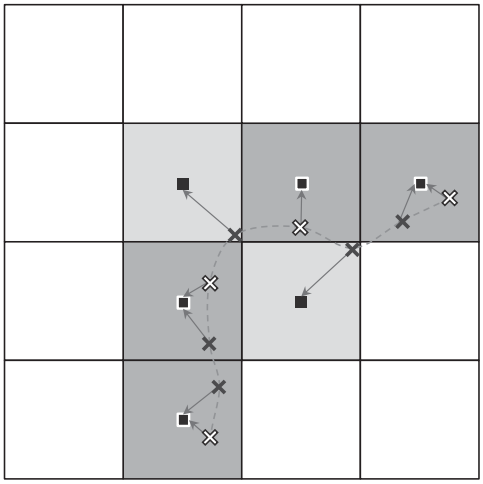


Figure 36: Thin Edge in the Original Image

The original and upscaled edges images generated by the LES method for the 'Yacht' test image are shown in Figure 37, along with the new original and upscaled edges images generated by the HES method.

(a) **(b)**

(c) (d)

Figure 37: LES and HES Edge Images Comparison ('Yacht')

(a) LES Original Edges Image; (b) LES Upscaled Edges Image;

(c) HES Original Edges Image; (d) HES Upscaled Edges Image;

While the edges images generated by both LES and HES methods are almost identical, the new HES method does clearly overcome the limitations shown by the LES method regarding the resolution of the upscaled edges, resulting in much more naturally defined shapes.

4.3.1.2 Edge Coloring

The exact same algorithm explained for the LES method is used here as well, using the anchor edge pixels in the upscaled image found by the above edge separation scheme, along with the core edge pixels as their corresponding edge pixels in the original image.

At the output of the edge color retrieval block, the upscaled image J contains color only in the edges, with all the non-edge areas still empty.

4.3.1.3 Non-Edge Coloring

Similarly as in the LES method, the remaining blocks in the HES scheme are basically responsible of filling the non-edge areas in the upscaled image avoiding the type of across-edge operations that are responsible of the edges blurring.

Backward pixel mapping and smart color interpolation are exactly the same as in the LES method. But in the case of HES method, there is one additional block in between, called mapping correction, that serves to correct an intrinsic effect of detecting edges with higher resolution: the edge ambiguity zone.

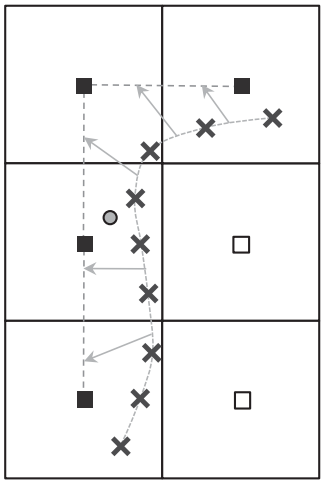


Figure 38: Original and Non-Integer Backward Mapped Upscaled Edges

Let us consider the generic situation shown in Figure 38. The regular square grid represents the original image pixels; black dots represent edges in the original edges image ε, namely original edges; crosses represent the non-integer locations where edges in the upscaled edges image E fall after backward pixel mapping, and will be referred to as mapped edges; and finally the gray dot represents a given non-integer location

backward mapped from a non-edge pixel in the upscaled image, and will be referred to as mapped non-edge pixel.

Due to the rounding used to localize original edges from the mapped edges, which is visualized in Figure 38 with arrows, mapped non-edge pixels falling between both original and mapped edges do not sit on the same side of the edge in both original and upscaled images. For example, considering the situation shown in Figure 38, the gray dot corresponds to a pixel that falls on the left side of the edge in the upscaled image, whereas it is falling on the right side of the edge in the original image. Applying the smart color interpolation without taking care of this new problem first, would lead to semantically incorrect color assignments on both sides of edges.

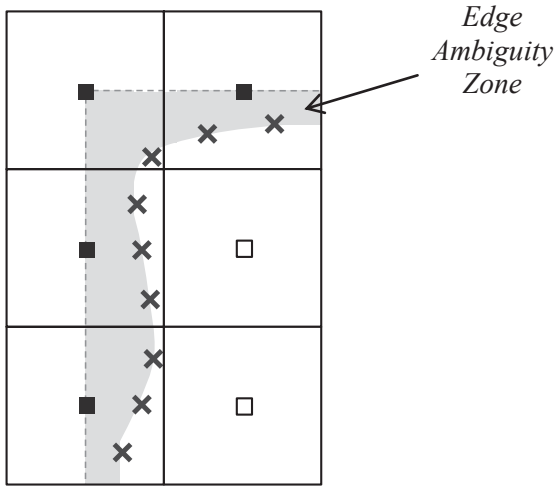


Figure 39: Ambiguity Zone

The zone between the original and mapped edges, where the above problem happens, is referred to as the edge ambiguity zone. The goal of the mapping correction block is to move all the mapped non-edge pixels that fall in an edge ambiguity zone, towards the

corresponding original edge and out of the zone, in order to maintain them on the same side of the edge in both original and upscaled images.

The first important task within the mapping correction module is to determine whether a given mapped non-edge pixel is likely to be falling in an ambiguity zone or not. This is an initial selection process that retains those pixels that will be subjected to further analysis, while clearing the way to the smart color interpolation block for all the unselected pixels. A good approach, which eliminates any possibility of missing affected pixels, is to retain all the mapped non-edge pixels that fall in a thick original edge.

Then, for a given mapped non-edge pixel D that have been selected for further scrutiny, let us consider the four edge pixels falling nearest to it, two from the mapped edge, namely P_1 and P_2, and two from the original edge, namely Q_1 and Q_2.

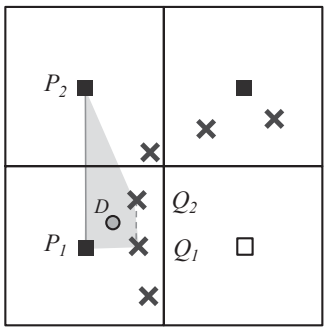


Figure 40: Edge Neighbors of a Backward Mapped Non-Edge Pixel

In order to definitely determine if pixel D is inside the ambiguity zone, lines l_P and l_Q, defined by the pairs of points $\{P_1, P_2\}$ and $\{Q_1, Q_2\}$ respectively, are found in the form $n = a \cdot m + b$, with m and n being as usual the horizontal and vertical coordinates.

The resulting parameters a (slope) and b (intercept ordinate) are shown below.

$$a_P = \frac{n_{P2} - n_{P1}}{m_{P2} - m_{P1}} \qquad a_Q = \frac{n_{Q2} - n_{Q1}}{m_{Q2} - m_{Q1}}$$
$$b_P = \frac{m_{P2} \cdot n_{P1} - m_{P1} \cdot n_{P2}}{m_{P2} - m_{P1}} \qquad b_Q = \frac{m_{Q2} \cdot n_{Q1} - m_{Q1} \cdot n_{Q2}}{m_{Q2} - m_{Q1}} \tag{4-8}$$

Based on these two lines, two different cases have to be considered, namely the case where l_P and l_Q lines are not parallel, and the case where they are parallel. Then finally, if at the end of these two cases, pixel D is found to be in the ambiguity zone, it has to be relocated to make it consistently fall on the same side of the edge in both original and upscaled images.

Non-Parallel Lines

When l_P and l_Q lines are not parallel, that is $a_P \neq a_Q$, point D is determined to be in the ambiguity zone if it falls within the acute angle defined by both lines. In order to mathematically implement such decision, the line passing through the intersection point C and point D, namely l_C, has to be found first, as shown in Figure 41.

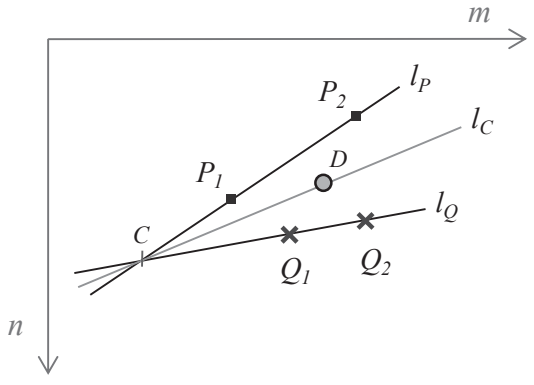


Figure 41: Non-Parallel Edge Lines

The coordinates of point C are computed as:

$$\begin{aligned} m_C &= \frac{b_Q - b_P}{a_P - a_Q} \\ n_C &= \frac{a_P \cdot b_Q - a_Q \cdot b_P}{a_P - a_Q} \end{aligned} \tag{4-9}$$

And the slope and intercept ordinate of line l_C are found as:

$$\begin{aligned} a_C &= \frac{n_D - n_C}{m_D - m_C} \\ b_C &= \frac{m_D \cdot n_C - m_C \cdot n_D}{m_D - m_C} \end{aligned} \tag{4-10}$$

Then, the angles β corresponding to all three slopes computed above are found, along with the corresponding angles $\beta + 180°$, which also define the same lines.

$$\begin{aligned} \beta_P^1 &= \tan^{-1}\left(a_P\right) & \beta_P^2 &= \beta_P^1 + 180° \\ \beta_Q^1 &= \tan^{-1}\left(a_Q\right) & \beta_Q^2 &= \beta_Q^1 + 180° \\ \beta_C^1 &= \tan^{-1}\left(a_C\right) & \beta_C^2 &= \beta_C^1 + 180° \end{aligned} \tag{4-11}$$

And the combination $\{\beta_P^i, \beta_Q^j\}$ leading to an absolute difference angle inferior or equal to 90° is selected, resulting in $\hat{\beta}_P$ and $\hat{\beta}_Q$.

Then, finally, point D is determined to be falling in the ambiguity zone when the condition below is fullfilled.

$$\begin{gathered} \hat{\beta}_P \le \beta_C^k \le \hat{\beta}_Q \\ or \\ \hat{\beta}_P \ge \beta_C^k \ge \hat{\beta}_Q \end{gathered} \tag{4-12}$$

Parallel Lines

When l_P and l_Q lines are parallel, that is $a_P = a_Q$, point D is determined to be in the ambiguity zone if its distance from both lines is inferior or equal to the distance between the lines.

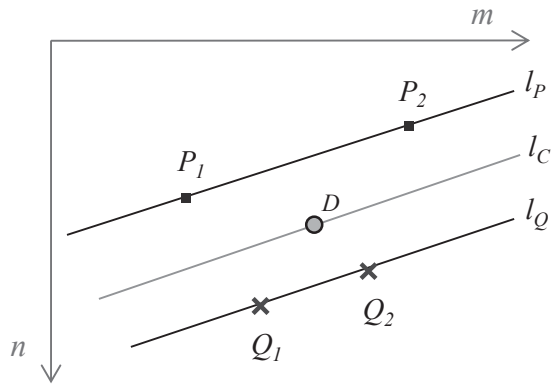


Figure 42: Parallel Edge Lines

The distance between lines l_P and l_Q is computed as:

$$d = \frac{|b_P - b_Q|}{\sqrt{1 + a_P^2}} \tag{4-13}$$

And the distances between point D and both lines are computed as:

$$d_P = \frac{|n_D - a_P \cdot m_D - b_P|}{\sqrt{1 + a_P^2}} \qquad d_Q = \frac{|n_D - a_Q \cdot m_D - b_Q|}{\sqrt{1 + a_Q^2}} \tag{4-14}$$

Therefore, pixel D is determined to be falling in the ambiguity zone when:

$$\begin{gathered} d_P \le d \\ and \\ d_Q \le d \end{gathered} \tag{4-15}$$

Correction

As already explained, this final stage of the mapping correction block is only reached if the mapped non-edge pixel D has been found to fall in the ambiguity zone. The goal here is to move the pixel towards the original edge and out of the zone. This is done by moving the pixel towards the line l_P, in its orthogonal direction, and a distance slightly over d_P, as shown in Figure 43.

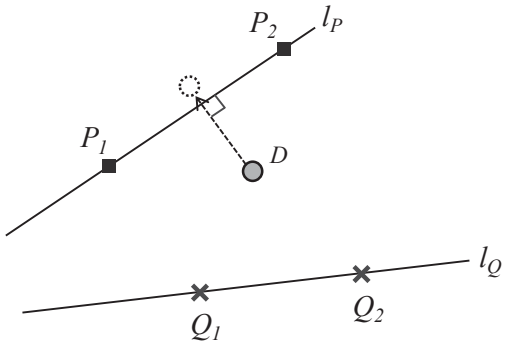


Figure 43: Mapping Correction

First, the slope and the intercept ordinate of the line orthogonal to l_P that passes through D are computed as:

$$a_P^{\perp} = \frac{-1}{a_P}$$
$$b_P^{\perp} = n_D + \frac{1}{a_P} \cdot m_D \tag{4-16}$$

Next, the vector orthogonal to line l_P that moves D onto it is computed as:

$$V_P^{\perp} = \left(\frac{b_P^{\perp} - b_P}{a_P - a_P^{\perp}} - m_D, \quad \frac{a_P \cdot b_P^{\perp} - a_P^{\perp} \cdot b_P}{a_P - a_P^{\perp}} - n_D \right) \tag{4-17}$$

Then, the final relocation of D is performed as shown below, with $\mu = 1.01$.

$$D = D + \mu \cdot V_P^{\perp} = \left((1-\mu) \cdot m_D + \mu \cdot \frac{b_P^{\perp} - b_P}{a_P - a_P^{\perp}}, \quad (1-\mu) \cdot n_D + \mu \cdot \frac{a_P \cdot b_P^{\perp} - a_P^{\perp} \cdot b_P}{a_P - a_P^{\perp}} \right) \tag{4-18}$$

Notice that the above mathematical approach is not valid when pixel D initially falls exactly on line l_P. When this happens, pixel D must be slightly moved orthogonally towards line l_Q before the above movement can be performed, as shown in Figure 44.

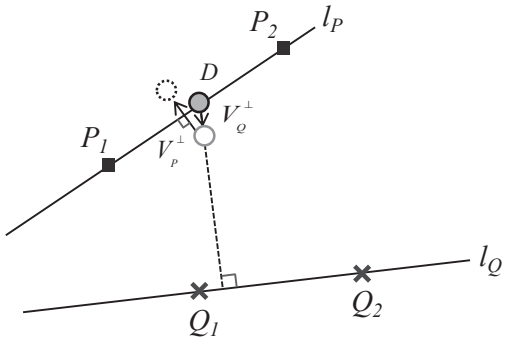


Figure 44: Mapping Correction with Pre-Relocation

This is mathematically done in a completely equivalent way, first computing the vector orthogonal to line l_Q that moves D onto it.

$$V_Q^{\perp} = \left(\frac{b_Q^{\perp} - b_Q}{a_Q - a_Q^{\perp}} - m_D, \quad \frac{a_Q \cdot b_Q^{\perp} - a_Q^{\perp} \cdot b_Q}{a_Q - a_Q^{\perp}} - n_D \right) \tag{4-19}$$

And then performing the pre-relocation of D as shown below, with $\mu = 0.01$.

$$D = D + \mu \cdot V_Q^{\perp} = \left((1-\mu) \cdot m_D + \mu \cdot \frac{b_Q^{\perp} - b_Q}{a_Q - a_Q^{\perp}}, \quad (1-\mu) \cdot n_D + \mu \cdot \frac{a_Q \cdot b_Q^{\perp} - a_Q^{\perp} \cdot b_Q}{a_Q - a_Q^{\perp}} \right) \tag{4-20}$$

4.3.2 Results and Conclusions

The High-Resolution Edge Separation (HES) scheme introduced in this section has been used to upscale several traditional test images with many different upscaling factors. Starting next page, a set of representative results is shown, using the same upscaling factor $K = 4$, and the same test images 'Flowers', 'Peppers', 'Monarch' and 'Yacht', as for the LES method. For each one of them, detail zoom of the upscaled image is shown using nearest neighbor interpolation, bilinear interpolation, the LES method and finally the new HES method.

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

HES Method

Figure 45: HES Results for ‘Flowers’ Test Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

HES Method

Figure 46: HES Results for 'Peppers' Test Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

HES Method

Figure 47: HES Results for 'Monarch' Test Image

Nearest Neighbor Interpolation

Bilinear Interpolation

LES Method

HES Method

Figure 48: HES Results for 'Yacht' Test Image

Results clearly show that the HES method effectively overcomes the problems posed by the LES problem regarding the shape of the edges in the upscaled image. Edges now appear to be naturally and faithfully shaped, yet keeping a high level of sharpness compared to the nearest neighbor and bilinear interpolation approaches.

The HES method has been proved to upscale all types of images, preserving the sharpness and shape of edges, with no other information than the original image, which makes it a very reliable and flexible upscaling method.

Nevertheless, the HES method, as well as the LES method, shows certain painting-like effect that is probably due to an excessive sharpening of the edges produced by the consideration of single line edges. The generated sharp-edged images tend to create a sensation of lack of texture in the non-edge areas, similarly as what can be observed in [Fattal 2007].

Chapter 5

Future Work

Even though the High-Resolution Edge Separation (HES) upscaling scheme proposed in the present work has shown a very promising behavior preserving edges in the upscaled image, it has been shown that it suffers of a painting-like effect. In order to tackle this problem, certain spatial decompositions of edges can be studied to successively apply the HES method on different sections of the edge transitions, which would help to more faithfully map the edge transition profiles from the original image onto the upscaled image.

Bibliography

AVIDAN, S., AND SHAMIR, A. 2007. Seam carving for contentaware image resizing. ACM *Transactions on Graphics (Proceedings of SIGGRAPH 2007)* 26, 10.

CANNY, J. 1986. "A computational approach to edge detection," *IEEETrans. Pattern Anal. Machine Intell*., vol. PAMI-8, pp. 679-698,

DIAMOND A., 2007. Implementation of Bresenham line algorithm. http://www.mathworks.com/matlabcentral/fileexchange/1929

FATTAL, R. 2007. Image upsampling via imposed edges statistics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)* 26, 95.

FREEMAN,W. T., JONES, T.R., PASZTOR, E. C., 2002. Example-based Super-Resolution. *IEEE Computer Graphics and Applications*

GONZALEZ, R. C., WOODS, R. E., 2008. Digital Image Processing. Pearson Prentice Hall Publishers, New Jersey, USA.

KOPF, J., COHEN, M. F., LISCHINSKI, D., AND UYTTENDAELE, M. 2007. Joint bilateral upsampling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)* 26, 96.

KOVESI, P. 2007. edge linking function that forms lists of connected edge points. http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/LineSegments/edgelink.m

SHAN q., ZHAORONG L., JIAYA J., CHI-KEUNG T. Fast Image/Video Upsampling. ACM *Transactions on Graphics (Proceedings of SIGGRAPH 2008)* 27, 5.

SUN, J., SUN, J., XU, Z. B., AND SHUM, H. Y. 2008. Image super-rosolution using gradient profile prior. In *CVPR*.

THUMHOFER, S., MITRA, S., 1996. Edge-Enhanced Image Zooming. *Optical Engineering*, vol. 35, no. 7, pp. 1862-1870.