Image Space and Time Interpolation for Video Navigation

Master Thesis

Author: Angel Tamariz Sánchez

Director: Dr. Josep Ramon Casas

October 2011

A mi hermano Gabriel y a mi amigo Angel

Contents

List of Figures

Abstract

The aim of image-based video navigation is essentially to achieve a continuous change in the viewpoint without the need of a complete camera coverage of the space of interest. By making use of image interpolation, the need for video hardware can be reduced drastically by replacing them, in the desired viewpoints, with virtual video cameras. In this work, based on previously published approaches, an algorithm for time and space image interpolation is developed with a video application perspective, where human-eye image interpretation is taken as a priority over exact geometry construction of the image sequences.

L'objectiu de la vídeo navegació basada en imatges és, principalment, aconseguir múltiples vistes d'un mateix escenari sense la necessitat de cobrir completament l'espai d'interès amb l'ús de càmeres de vídeo. Fent us de la interpolació d'imatges, la necessitat de hardware de vídeo pot ser reduïda dràsticament al ser reemplaçat, en els punts de vista desitjats, amb càmeres de vídeo virtuals. En aquest treball, basat en aproximacions publicades anteriorment, desenvolupem un algoritme per a la interpolació d'imatges en temps i espai amb una perspectiva d'aplicació de vídeo, on la interpretació des del punt de vista de l'ull humà es pren com a prioritat sobre la construcció exacte de figures geomètriques en les seqüències d'imatges.

El objetivo de la video navegación basada en imágenes es principalmente lograr un cambio continuo de perspectiva sin la necesidad de cubrir completamente el espacio de interés con el uso de cámaras de video. Haciendo uso de interpolación de imágenes, la necesidad de hardware de video puede ser reducido drásticamente al ser remplazado, en los puntos de vista deseados, con cámaras de video virtuales. En este trabajo, basado en aproximaciones publicadas anteriormente, desarrollamos un algoritmo para la interpolación de imágenes en tiempo y espacio con una perspectiva de aplicación de video, donde la interpretación desde el punto de vista del ojo humano se toma como prioridad sobre la construcción exacta de figuras geométricas en las secuencias de imágenes.

Prologue

We are human, and we feel.

Communicating our feelings has always meant a challenge for both men and women throughout the history of Mankind. Art is the space that functions as a means to express what sometimes cannot be expressed through dialogue.

Art is expression of feelings. It is the opportunity of using the body to give an intimate message to possible spectators of the work. Great artists are distinguished by having the ability to achieve the connection between what is in the heart and what is in the exterior through some particular tool; whether it be a pencil, a musical instrument, clay, or perhaps a canvas. Those artworks which succeeded in moving the feelings of those who contemplate them, transmitting and sharing what the artist has felt at the time of creation, have achieved their goal.

We are human, and we reason.

In the matter of reducing the complexity of living, or rather of surviving, Humans have made use of their reasoning abilities to create technology. From a simple sheet of paper to the most complex aircraft, every technological invention is characterized as being a resort to adapt to the environment in which men and women live, according to their needs and concerns. The paradigmatic case, in the context of the Information Age, is that of computers.

Day after day life experiences are more influenced by the incessant improvement of information and computing technologies. The processing of algorithms through computers is, as time moves on, less and less limited to the resolution of theoretical problems, which may be beyond the reach of the five senses of ordinary people, and influence, in a more perceptible way, daily life experiences. The speed of operation resolution that computers currently reach, and the advantage that this gives regarding time and resource savings, has allowed that they affect every, or almost every field of knowledge nowadays.

We are human, and we imagine.

Throughout history, art and technology have found ways to compliment one another. Sometimes their coexistence has proven to be a way of reinforcing each other. I think of architecture as a clear example for this approach; however, it is also usual to hear people refer to a Formula 1 engine as artwork.

However, the dehumanization of technological tools, which could be understood as the misunderstanding of how these tools work, given the increasing

complexity throughout their evolution, makes it difficult to conceive machines as vehicles for the expression of feelings. It is, in itself, already a challenge to express them! Why should we further complicate our lives by making use of new technologies?

But some environments exist in which the convergence of both fields is not inconceivable. On the one hand, an image can be defined depending on the angle from which it is interpreted. It may be a graphical representation on paper, it can be a drawing, a photograph or a painting from an artistic approach. On the other hand, an image may also be defined as a two-dimensional spatial signal, if done from an engineering perspective.

Images represent an area of reconciliation between art and technology, between art and engineering. Advances in the signal processing field have allowed great versatility in the manipulation of images, and as a consequence, the opportunity to venture into this activity with an artistic vision. The use of computational tools for graphical design is, for example, increasingly common.

The whole computer art scene, which represents a community of people interested and active in the creation of computer-based artwork, has found in this reconciliation one more occasion to strengthen bonds between arts and technology; between creativity and ingenuity; between feelings and reasoning. All this through the use of imagination.

Based on algorithms that take into account, as an efficiency factor, plausibility of the results from a visual perception point of view over merely numeric parameters, this Master thesis is simply an additional contribution in the use of information technologies to reinforce the artistic virtues of image processing; making it a possible motivation for filmmakers, video creators or people interested in imagery arts in general.

Our specific topic, free-viewpoint navigation systems, which have the objective of rendering photo realistic views of the real world scenes from arbitrary perspectives, offers the opportunity to create physically impossible effects such as frozen-time scene exploration and extreme slow motion.

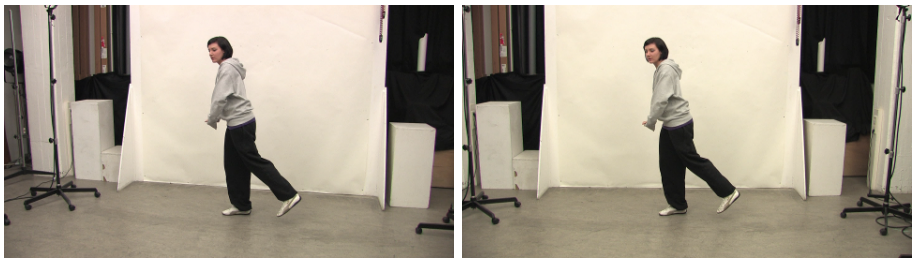
Introduction

On the research paper titled Perception-Motivated Interpolation of Image Sequences by Stich et al. [4] an approach for image-based interpolation is presented based on a series of algorithms focused primarily in plausibility from a human perception perspective. This Master Thesis uses Stich's work as a base-point for achieving once again convincing interpolated images, which work as a starting point for free-viewpoint video navigation, but experimenting with different strategies and using Matlab as programming language and numerical environment.

Objective

If anyone is asked to analyse Figure 1 which corresponds to a stereo-vision scene, where the space difference between viewpoints is no more than 10 degrees, and to tell which parts in the first image correspond to the ones in the second, he or she would not have any difficulty in answering. Anyone can easily match every part of the dancer's body (shoes, hair, hand, etc) between both images, match every figure in the background and tell which parts of the image have been occluded or which figures are new in the distinct view-point. We know that at the edges of the images we will be able to see things from one perspective which are invisible to the other. Our brains are trained to understand about changes in perspective and we have previous knowledge about the objects in the pictures, e.g we know how an arm or a leg moves, and we have a clear idea of how color tones behave in a human face. Depth perception is also an important tool that allows us to know in advance if an object will cover a farther one if we, as observers, move left or right, up or down. As humans, we are able to make these "logical" statements about the two images in Figure 1, but computers cannot. Computers do not *see* a dancer, computers *see* a two-dimensional discrete signal that, in principle, has independent elements. Which criteria will a computer use to know how perspective behaves? Or, if instead of space difference we analyse time difference, which criteria will a computer use to match an arm that has moved? The intention of this work is to process two images from a same scene but from different perspectives and/or different time instants, in a way that computed correspondences between both may be as close as the ones anyone would point out with her or his finger. By achieving this it will then be possible to synthesize a plausible *intermediate* view of the scene (interpolation of images). It can also be seen as an endeavour to decoratively reduce the abysmal distance between human and computer image interpretation.

The goal of this thesis is, therefore, to generate an algorithm that, by taking



(a) (b)

Figure 1: Stereo vision scene

advantage of some previous work done in the field, is able to synthesize these interpolated images and afterwards use them for video navigation applications. Visual effects such as *bullet time* and *slow motion*, are the intended aims to be reached with the procedures shown in this work. It is an effort to improve post-processing techniques in order to reduce the costs that this kind of video experiences represent. The goal is also to obtain results that may be evaluated, by comparing them with *real* intermediate images. This will help to know which aspects of the strategy deserve more attention for students who may be willing to improve the algorithm in the future.

It should be taken into account that even if a real-time solution is, without a doubt, of great interest in this field of knowledge, the developed algorithm does not offer a solution of this kind. The main goal is centered in finding a correct implementation of image processing algorithms that will construct the desired interpolations. Although a great effort has been injected in improving the efficiency of the code, this by itself is not a major objective. Also, it is worth saying that in what refers to Matlab, real-time is not precisely its strongest quality.

Related work

Even though video navigation is to be the goal of the algorithm designed in this work, it is mostly concentrated in the generation of the interpolated images.

The Computer Graphics Lab at TU Braunschweig in Germany has been working in this area for some years now. Besides the interpolation paper, based on perception by Stich et. al. [4], a lot of effort has been dedicated to the study of different mechanisms for the optimization of video navigation. Lipski et. al. put together previous work done on image interpolation based on dense correspondences and post processing methods to achieve some interesting results in their article titled "Virtual Video Camera: Image-Based Viewpoint Navigation Through Space and Time" [5].

Another contribution from Lipski et. al. [6], is dedicated to the optimization of the estimation of the dense correspondences, which is, in fact, one key step for the interpolation of the images. This article is focused on several video post-processing maneuvers such as image morphing, optical flow estimation, stereo rectification, disparity/depth reconstruction and baseline adjustment.

In the movie industry, algorithms such as the ones presented by Beier and

Neely [7] are often used to create visual effects in order to smooth transitions between actor appearances during the film. This is known as image morphing, where determined objects in the images are interpolated based on user-defined correspondences; and it is also a concept closely related to this work.

In what refers to topics related with image perception, Vangorp et. al. [8] worked on the influence of material of objects to shape perception showing interesting dependencies between differentiability of material properties and surfaces. One year later, Ramanarayanan et. al. [9] published an article titled "Perception of complex aggregates" where they worked on user studies focused on visual equivalence of rendered images, and the equivalence of the rendering of groups of objects depending on shape and texture.

Divergences with the original approach

In general, the approach followed in this work is very similar to Stich et al. [4], but some important differences are worth mentioning. In relation to the dense correspondence computation, which basically relates both images to be interpolated by means of the edges of the figures in the scene, Stich's approximation is based on finding the gradient of the image using the Compass operator [10], and afterwards relating the pixels of the edges marked by the gradient (*edglets*) with regions computed by an algorithm proposed by Felzenszwalb and Huttenlocher [11], essentially based on neighboring pixel similarities. In this approach, edglets and homogeneous regions are found using a segmentation process called the Watershed algorithm, which is fully explained in a later chapter. Edglets are therefore defined directly as the edges of the regions computed by the algorithm itself. Another difference is that Stich's approach makes use of additional manual corrections once the edge correspondences are computed, assuring extra accuracy in this stage. While manually corrected interpolations do offer better quality results, this approach aims at no human intervention during the computation. Notorious differences are found in the rendering stage, mainly because in this approach no real-time processing is needed. These, together with other minor processes, alternative to the original approach, will be mentioned during the description of the whole strategy.

Structure of the thesis

This thesis is mainly divided into four chapters:
Chapter 1 is dedicated to describing the algorithms and concepts used to achieve our goal. The most important algorithms used are the Watershed, the Shape Context descriptor, the Auction algorithm, the RANSAC process and Inpainting. All these algorithms are used to play with and handle *translets* which will more deeply be defined later, but that consist of a determined image segment and its corresponding homography. Homographies are then a basic concept that is also described in detail in this chapter.
Chapter 2 is dedicated to explaining how these tools and algorithms are implemented in order to obtain the desired results. Some figures are used to show the procedure and how the algorithms interact with the images to be interpolated.
Chapter 3 shows the results after the whole algorithm is applied to several pairs of images. The issues encountered in the results are explained in detail making

emphasis in the possible reasons for their existence and the different solutions that can be implemented to face them.

Finally, in Chapter 4 I try to give certain guidance to future intentions on the continuation of this line of work, focusing on what needs to be improved from the code and from the strategy itself.

Chapter 1

Concepts and tools

The basic idea used for the computation of the interpolation of the images is to consider the second image as a geometrical deformation of the first one, as a consequence of a different projection of the 3D space. Both images are segmented into regions which respond to determined area homogeneities. Based on each region's position and shape, a correspondence between each one of them is found from one image to the other. The tool that will help us model the deformation of the corresponding regions is called *homography*. The key for a plausible interpolation will therefore rely on the accurate computation of these homographies, which in turn rely on an accurate determination of the correspondences of the edge pixels between the two images. To achieve this task, a series of algorithms are employed. These, and some other basic concepts used throughout the Master Thesis, are described in this chapter.

1.1 Homography, the basic tool

Hartley and Zisserman [12] define a homography as the projective transformation that takes each point $\mathbf{x}_i$ in $\mathbb{P}^2$ to its corresponding $\mathbf{x}'_i$ likewise in $\mathbb{P}^2$. In a practical situation, the points $\mathbf{x}_i$ and $\mathbf{x}'_i$ are points in two images, each image being considered as a projective plane $\mathbb{P}^2$. Put in other words, the homography describes what happens to the perceived positions of observed objects when the point of view of the observer changes.

So the problem relies on estimating this 2D projective transformation. A set of point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ between two images will be considered as a starting point. The problem is to compute a 3×3 matrix $\mathbf{H}$ such that

$$\mathbf{H}\mathbf{x}_i = \mathbf{x}'_i \tag{1.1}$$

for each i.

The Direct Linear Transformation (DLT) is a simple linear algorithm for the determination of matrix $\mathbf{H}$. Expressing Equation 1.1 in terms of the vector cross product will enable a simple linear solution for $\mathbf{H}$ to be derived:

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = 0.$$

Consider that each point is represented as a homogeneous 3-vector, i.e.

$\mathbf{x}_i = (x_i, y_i, w_i)^\mathsf{T}$ and $\mathbf{x}'_i = (x'_i, y'_i, w'_i)^\mathsf{T}$, being w_i and w'_i the scale factor. Then, if the j-th row of the matrix $\mathbf{H}$ is denoted by $\mathbf{h}^{j\mathsf{T}}$:

$$\mathbf{H}\mathbf{x}_i = \begin{pmatrix} \mathbf{h}^{1\mathsf{T}}\mathbf{x}_i \\ \mathbf{h}^{2\mathsf{T}}\mathbf{x}_i \\ \mathbf{h}^{3\mathsf{T}}\mathbf{x}_i \end{pmatrix}, \tag{1.2}$$

the cross product may be given explicitly as

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = \begin{pmatrix} y'_i\mathbf{h}^{3\mathsf{T}}\mathbf{x}_i - w'_i\mathbf{h}^{2\mathsf{T}}\mathbf{x}_i \\ w'_i\mathbf{h}^{1\mathsf{T}}\mathbf{x}_i - x'_i\mathbf{h}^{3\mathsf{T}}\mathbf{x}_i \\ x'_i\mathbf{h}^{2\mathsf{T}}\mathbf{x}_i - y'_i\mathbf{h}^{1\mathsf{T}}\mathbf{x}_i \end{pmatrix}. \tag{1.3}$$

Since $\mathbf{h}^{j\mathsf{T}}\mathbf{x}_i = \mathbf{x}_i^\mathsf{T}\mathbf{h}^j$ for $j = 1, ..., 3$, this gives a set of three equations in the entries of $\mathbf{H}$, which may be written in the form

$$\begin{bmatrix} \mathbf{0}^\mathsf{T} & -w'_i\mathbf{x}_i^\mathsf{T} & y'_i\mathbf{x}_i^\mathsf{T} \\ w'_i\mathbf{x}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} & -x'_i\mathbf{x}_i^\mathsf{T} \\ -y'_i\mathbf{x}_i^\mathsf{T} & x'_i\mathbf{x}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \mathbf{0}. \tag{1.4}$$

Although there are three equations in (1.4), only two of them are linearly independent (since the third row is obtained from the sum of x'_i times the first row and y'_i times the second). Thus each point correspondence gives two equations in the entries of $\mathbf{H}$, and the set of equations becomes

$$\begin{bmatrix} \mathbf{0}^\mathsf{T} & -w'_i\mathbf{x}_i^\mathsf{T} & y'_i\mathbf{x}_i^\mathsf{T} \\ w'_i\mathbf{x}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} & -x'_i\mathbf{x}_i^\mathsf{T} \end{bmatrix} \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix} = \mathbf{0}. \tag{1.5}$$

These equations have the form $\mathbf{A}_i\mathbf{h} = \mathbf{0}$, where $\mathbf{A}_i$ is a 2×9 matrix, and $\mathbf{h}$ is a 9-vector made up of the entries of the matrix $\mathbf{H}$:

$$\mathbf{h} = \begin{pmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{pmatrix}, \qquad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \tag{1.6}$$

with h_i the i-th element of $\mathbf{h}$.

Remember that the equations hold for any coordinate representation $(x'_i, y'_i, w'_i)^\mathsf{T}$ of the point $\mathbf{x}'_i$. If $w'_i = 1$ is chosen, it means that (x'_i, y'_i) are the coordinates measured in the image [12].

At least four point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ have to be specified in order to fully constrain $\mathbf{H}$. If these four correspondences are given exactly, then an exact solution for the matrix $\mathbf{H}$ is possible. However, since in practice points are measured inexactly (*noise*), if more than four correspondences are given, they may not be fully compatible with any projective transformation. Therefore, it is necessary to determine the *best* $\mathbf{H}$ from the obtained data. This will be done by minimizing a determined cost function. Hartley and Zisserman [12] present several cost functions and the methods for minimizing them. For this work the Gold Standard algorithm was used, which is optimal in the sense that the $\mathbf{H}$ that minimizes it gives the best possible estimate of the transformation under certain assumptions.

A *good* computation of the projective transformation will depend on the accuracy of the determination of the point correspondences. In the following sections the algorithms that work in order to achieve this objective are presented.

1.2 The Watershed Algorithm

The first step is to define edges and homogeneous regions. The Watershed algorithm is a morphological approach to image segmentation through a region growing method which uses the gradient value of the image and interprets it as a topographic map (Fig. 1.1(b)). Image segmentations consist of partitioning the image into regions which are homogeneous with respect to some property. In the watershed case gray value is taken as the criterion. Gray level in the gradient image (Fig. 1.1(a)) will define high topographic areas for high gradient values, which correspond to edges in the original image. Catchment basins in the image are therefore generated by low gradient magnitude intensity limited by the high gradient pixels. The concept of watershed is used because region growth behaves as water floods beginning from predefined local minima in the image (Fig. 1.1(c)). Wherever the floods of each basin meet, dams are built and when the water reaches the highest peak, the process is stopped. These dams that separate the basins or regions of the image are called Watersheds.

Once the gradient of the image is computed, local minima must be found to perform the initial floods. The pixel with the gray value closest to the flooded level is assigned to its corresponding catchment basin. If no preprocessing of the image is done in order to simplify the image, over-segmentation can be encountered. In a typical real image, gray level variations exist, there where for the human eye it may seem as a flat area. This generates into a creation of multiple catchment basins in the regions that would be considered homogeneous. Simplification of the image consists of smoothing the image, eliminating noise in homogeneous regions, but without losing the edges of interest.

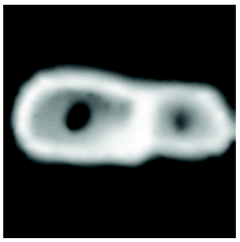
After the location of local minima in the gradient of the simplified image is carried out, flooding takes place. As mentioned before, pixels with the closest value to the flooded level are assigned to the basin, either considering 4-connectivity, or 8-connectivity. In this work 8-connectivity was chosen for the Watershed algorithm.

Different watershed results will be obtained depending mainly, besides pixel connectivity, on the image simplification method, the criterion for local minima determination, and the gradient computation method employed.

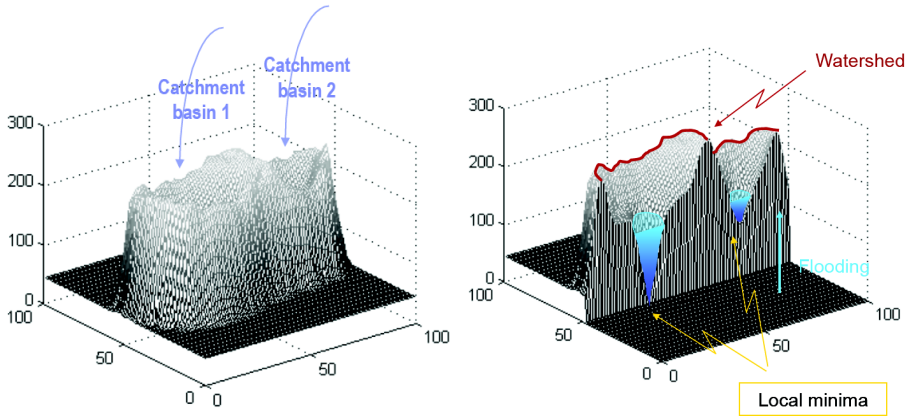
1.3 Shape Context Descriptor

Figure 1.2 is an example of a known character (letter A) but represented with different shapes. For a person who reads Latin characters, both images have the same meaning and could draw the edge correspondences that would help computing the homography between (a) and (b) considering that the latter is a deformation of the prior. For a computer this task is not necessarily trivial.

The shape context descriptor is a strategy proposed by Belongie et al. [2] where precisely each pixel of an image is described by its shape context, i.e.



(a)



(b) (c)

Figure 1.1: Example of a Watershed algorithm procedure. (a) Gradient of an image. (b) Topographic representation of (a). (c) Flooding process and watershed determination. Images have been taken from [1]

.

(a) (b)

Figure 1.2: Example of two different shapes for a same character

by the shapes that surround the pixel. This is achieved by computing distance and direction histograms for each pixel. The information is then used for shape matching.
Let us see how it works. Considering a shape with n number of edge pixels, for a point p_i on the shape, a coarse histogram h_i of the relative coordinates of the remaining $n-1$ points is computed:

$$h_i(k) = \#\{q \neq p_i : (q - p_i) \in bin(k)\} \tag{1.7}$$

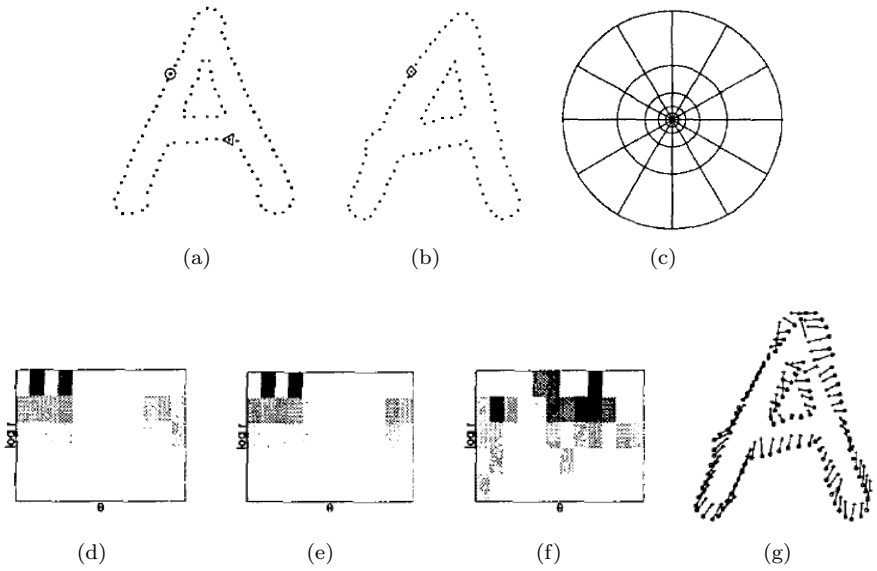
This histogram is defined to be the *shape context* of p_i. The descriptor must be more sensitive to differences in nearby pixels, therefore a log-polar coordinate system is used as in Figure 1.3(c). Once the histograms are computed for every edge pixel, the matching decisions are made using a determined cost function. Consider an edge point p_i from the first shape, and an edge point q_j from the second one. Let $C_{ij} = C(p_i, q_j)$ denote the cost of matching these two points:

$$C_{ij} = \frac{1}{2}\sum_{k=1}^{K}\frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)} \tag{1.8}$$

where $h_i(k)$ and $h_j(k)$ denote the K-bin normalized histogram at p_i and q_j, respectively.
Given the set of costs C_{ij} between all pairs of points i on the first shape and j on the second shape, we want to minimize the total cost of matching subject to the constraint that the matching be one-to-one. This is an instance of the weighted bipartite matching problem which, in this thesis is solved using the algorithm proposed by Bertsekas [13]. The input to the assignment problem is a square cost matrix with entries C_{ij}. The result is a permutation $\pi(i)$ such that the sum $\sum_n C_{i,\pi(i)}$ is minimized.
When the number of edge points on two shapes is not equal, the cost matrix can be squared by adding "dummy" nodes to each point set with a constant matching cost of ϵ_d. This can be also used, even when having the same number of edge points, to allow a better handling of outliers. In this case, a point will be matched to a "dummy" whenever there is no match available at smaller cost than ϵ_d. Thus, ϵ_d is used as a threshold parameter for outlier detection. The cost C_{ij} for matching points can include an additional term based on the local *appearance similarity* at edge points p_i and p_j. This is particularly useful when we are comparing shapes derived from gray-level images instead of line drawings, as in the case of this work, where real-life images are used. For example, one can add a cost based on color or texture similarity.

1.4 The Auction Algorithm

As proposed in [4], the term *edglet* will be used in this work to refer to those pixels belonging to the predefined edges of the image. In this case, pixels that separate the regions after the Watershed algorithm has been applied.
The Shape Context descriptor sets a cost for matching each edglet with every other from the second image. Once these costs have been computed, an optimal matching has to been chosen for each one of the edglets. This kind of problem is known as a *weighted bipartie graph matching problem* and the Auction algorithm proposed by [13] has proven to perform very efficiently in solving them.



(a) (b) (c) (d) (e) (f) (g)

Figure 1.3: Shape context computation and matching. (a,b) Sampled edge points of two shapes. (c) Diagram of log-polar histogram bins used in computing the shape contexts. In this example 5 bins for $log(r)$ and 12 bins for θ. (d-f) Example shape contexts for reference samples marked by $\circ, \diamond, \triangleleft$ in (a,b). Each shape context is a lo-polar histogram of the coordinates of the rest of the point set measured using the reference point as the origin (dark = large value). There is a clear similarity between the $\circ$ and the $\diamond$ histograms, which were computed for two relatively similar points in each shape. On the other hand, the shape context for $\triangleleft$ is notoriously different. (g) Correspondences found using bipartie matching, which will be explained in the next section. Images taken from [2].

So again, the objective is to find optimal one-to-one assignments between edglets from both images. An intuitive way of understanding the functionality of the Auction algorithm is in an economic context. Each edglet i from the first image needs to pay a price p_j to be assigned with another edglet j from the second image (which we will call *object*). A benefit a_{ij} exists when matching i with j. Therefore the value for each matching is $a_{ij} - p_j$. The edglet will be *happy* if it can obtain the maximal possible value:

$$a_{ij_i} - p_{j_i} = \max_{j=1,\ldots,n}\{a_{ij} - p_j\}. \tag{1.9}$$

However, considering that the interest of the algorithm is to find *global satisfaction* and fair *wealth distribution*, the objective will be to obtain the maximum total benefit $\sum_{i=1}^{n} a_{ij}$, and prices will be in *equilibrium* when all edglets are *happy* or *almost happy*; as will be explained later on.

The Auction algorithm consists of performing iterations starting with any set of assignments at any set of prices. If every edglet is *happy*, the process terminates. Otherwise, some edglet i that is not *happy* is selected, and finds an object (edglet from the second image) j_i which offers maximal value:

$$j_i \in \max_{j=1,\ldots,n}\{a_{ij} - p_j\}. \tag{1.10}$$

It then exchanges objects with the edglet assigned to j_i at the beginning of the iteration, and sets the price of the best object j_i to the level at which it is indifferent between j_i and the second best object, that is, the edglet sets p_{j_i} to

$$p_{j_i} + \gamma_i, \tag{1.11}$$

where

$$\gamma_i = v_i + w_i. \tag{1.12}$$

v_i is the best object value,

$$v_i = \max_j\{a_{ij} - p_j\}, \tag{1.13}$$

and w_i is the second best object value,

$$w_i = \max_{j \neq j_i}\{a_{ij} - p_j\}, \tag{1.14}$$

that is, the best object value other than j_i.
γ_i is the largest increment that the edglet i can do to the best object price p_{j_i}, having object j_i still as the best option. This is why it is called *auction* algorithm. Once an edglet (bidder) finds the object with the best value, it raises its price (by a bidding increment γ_i) up to a maximum it is willing to pay, that is, just below the price of the *second best* option. Iteratively, edglets will be increasing prices, losing objects, winning others, until a one-to-one assignment is reached and everybody is *happy*.

However, this iteration process does not always work. The problem arises when the bidding increment γ_i is zero because of the existence of more than one *best object*. As a result, a never ending cycle happens when several edglets are interested in a lower number of objects without increasing their prices. This is why we talked about *almost happy* edglets previously. To break such never

ending cycles, an extra minimum increment ϵ is imposed to the price of the object that has been bid for, as it happens in real auctions. Therefore, it is said that an edglet i is *almost happy* with an assignment if the value of its assigned object j_i is within ϵ of being maximal:

$$a_{ij_i} - p_{j_i} \geq \max_{j=1,\ldots,n}\{a_{ij} - p_j\} - \epsilon. \tag{1.15}$$

So now we will have a set of *almost happy* edglets at the end of the process. This way we avoid having several *best options* for a single edglet. The whole process is the same as described before, with the exception that the *bidding increment* γ_i is now:

$$\gamma_i = v_i - w_i + \epsilon, \tag{1.16}$$

rather than Eq. 1.12.
Once the process has finished for each and every edglet, a one-to-one assignment will be achieved. These matches will then be used in the computation of the homographies. However, some *outliers* may be obtained from the Auction algorithm. The RANSAC algorithm is used to respond to this issue.

1.5 RANSAC

As said before, some of the edglets will be matched to *dummy* pixels, which represent, by definition, an outlier. These matches must and will be immediately discarded from the set used to compute the corresponding homography. But not only these matchings will behave as outliers. Many correspondences calculated by the *auction* algorithm will not be accurate. While discarding the *dummy* correspondences may not present a hard challenge, getting to know which of the rest of them are outliers is not easy. Marco Zuliani [14] defines an outlier as

> a datum that does not fit the "true" model instantiated by the "true" set of parameters within some error threshold that defines the maximum deviation attributable to the effects of noise.

But, which is the "true" model in our case? The homography, of course. Nevertheless, considering that the deformation of each region, which can be seen as a 2D geometrical figure that changes its shape, depends on a change of vision perspective and/or random movement of the objects in the image, getting to know which of the matchings are outliers and which are inliers is not trivial. Fortunately, in 1981, Fischler and Bolles [15] introduced the Random Sample and Consensus (RANSAC) algorithm, which basically consists of a method to estimate the parameters of a determined model, starting from a set of data which is originally strongly contaminated (it can be larger than 50%) with outliers. For a better understanding of how RANSAC works, a simple example is shown in Figure 1.4. In this case, the purpose of the data is to define the model of a straight line. The existence of outliers, which exceed the error threshold, will influence towards a wrong determination of the line's model and therefore must be removed. In this work's case, our models are the homographies, and bad assignments between edglets will translate into their wrong computation.

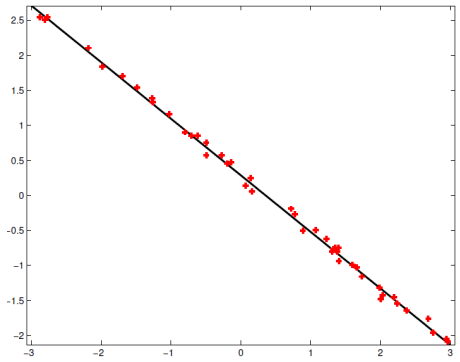


Figure 1.4: Example of a data set for the determination of the model of a straight line, contaminated with outliers

The RANSAC algorithm is essentially composed of two steps which are repeated iteratively [14]:

- **Hypothesis**. First minimal sample sets (MSSs) are randomly selected from the input dataset and the model parameters are computed using only the elements of the MSS. The cardinality of the MSS is the smallest sufficient to determine the model parameters (as opposed to other approaches, such as *least squares*, where the parameters are estimated using all the data available, possibly with appropriate weights).

- **Test**. In the second step RANSAC checks which elements of the entire dataset are consistent with the model instantiated with the parameters estimated in the first step. The set of such elements is called *consensus set* (CS).

RANSAC terminates when the probability of finding a better ranked CS drops below a certain threshold. It is important to take into consideration that the first MSS is selected randomly. This means that for different RANSAC processes for the same dataset, we may obtain different results. This is why, as it will be shown in the next chapter, it is important to priorly *filter* some of the correspondences computed by the Auction algorithm, which are clearly outliers, in order to reduce the divergence in the different possible results obtained from RANSAC.

The last CS obtained will represent the set of assignments that will be considered to compute the homography for each region.

1.6 Inpainting

One of the main issues to deal with in image interpolation, especially when the interest is focused on generating a new viewpoint between two different perspectives of a determined scene, are occlusions. When changing view perspective, closer objects in the scene move *faster* than farther objects. This means that between two different views, we may *cover* or *uncover* part of the background with the foreground. These particularities are called occlusions. It is a two-sided problem then; one is to solve fold-overs, i.e. *covered* regions or simply any

case where more than one pixel end up in one same location, and the second is deciding what information must be used to *fill in* the missing regions, i.e. the *uncovered* or *disoccluded* areas. For the latter a possible solution is using Inpainting.

The concept *inpainting* comes from the practice of restoration of damaged paintings, murals and other kind of artistic creations over a surface, e.g. mosaics. However, this principle has been applied also to photography and film. The aim is to retouch the image in a naturally extended way that it is non-detectable by an observer who does not know the original version [3].

With the interest in photographic and film restoration, digital techniques find their niche. Removing scratches, publicity, subtitles, or entire undesired objects from the images have replaced the artistic objective of reverting deterioration when talking about digital methods (see Figure 1.5).

Since the issue we are interested in solving is occlusions, we will be dealing with texture and structures. This means that the inpainting algorithm will not only be limited to filling desired regions by means of pattern repetition, as could happen with textures.

Inpainting is, of course, a subjective procedure. There is no "correct" way of doing it. But as it is stated in the objective of this Master thesis, the whole procedure is focused on human perception, and therefore, if all these rectifications are not *too* notorious, we will be satisfied.

It is not the purpose of this work to deeply explain, step by step, how the algorithm works, but to give a general idea of the principles on which it is based. For a detailed explanation refer to Bertalmio et al. [3].

The basic idea is the following. Let Ω represent the region intended to be inpainted, and $\partial\Omega$ its boundary. The algorithm will prolong the lines of equal gray values arriving at $\partial\Omega$, while maintaining the angle of "arrival". The drawing from $\partial\Omega$ proceeds inwards in this way, while curving the prolongation lines progressively to prevent them from crossing each other.

This algorithm is also based on performing iterations of a determined process. This process consists mainly of the following:

1. The structure of the surrounding area Ω is continued into the gap. Contour lines are drawn via the prolongation of those arriving at $\partial\Omega$

2. The different regions inside Ω, as defined by the contour lines, are filled with color, matching those of $\partial\Omega$

Simultaneous iterations are performed, which will progressively *shrink* the gap Ω by inwardly prolonging, in a smooth manner, the lines arriving at the boundary $\partial\Omega$.

Figure 1.5: Example of photography inpainting. Image taken from [3].

Chapter 2

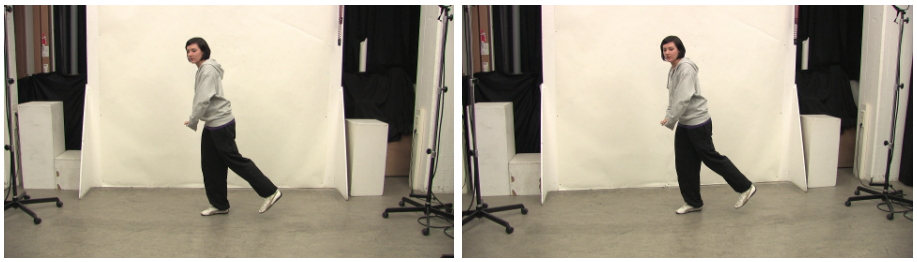
Implementation

Now that the basic theory of some algorithms has been explained in Chapter 1, we are ready to give a detailed explanation of the implementation of the needed tools in order to achieve the objective of this thesis.

The starting point consists of two images of a determined scene, seen from different but close (not more than 10 degrees angle between cameras) viewpoints (Figure 2.1), or different (consecutive) time instants (Figure 2.2). We aim to generate a deformation field between these two images that, by scaling it, may help to synthesize an intermediate (in time or space) view of the scene, i.e. interpolating the two images. The result of the interpolation does not necessarily need to be a middle position; the scalability property of the deformation field will let us *move* arbitrarily in this aspect. The computation of the deformation vector field, together with the rest of the procedure, will be illustrated in the following sections.

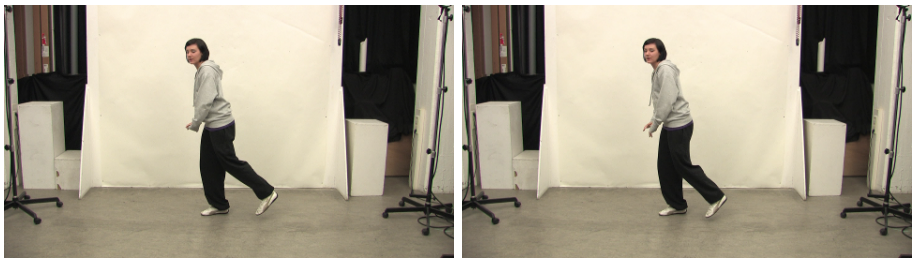
2.1 Edges and homogeneous regions

The first step is to extract from both images the components that will be used to define a correspondence between them: *edges* and *homogeneous regions*. The hypothesis in which we rely on is that the movement behaviour of these components is similar between the two images; similar enough to be modelled. If



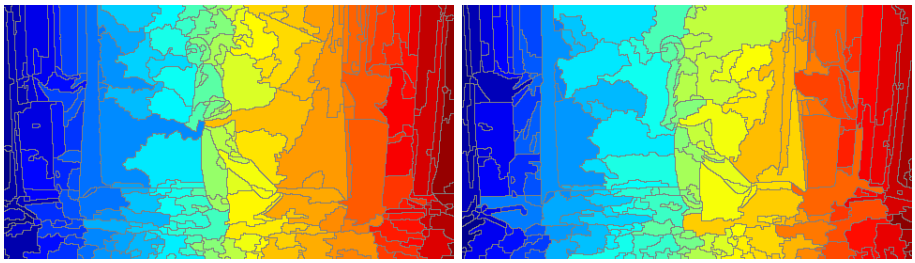
(a) (b)

Figure 2.1: Example of two images of the same same scene, same time instant, but different views.



(a) (b)

Figure 2.2: Example of two images of the same same scene, same view, but different time instants.



(a) (b)

Figure 2.3: (a) Figure 2.1(a) oversegmented. (b) Figure 2.1(b) oversegmented. This is a case of a wrong simplification of the images. Finding a match between regions would be quite difficult.

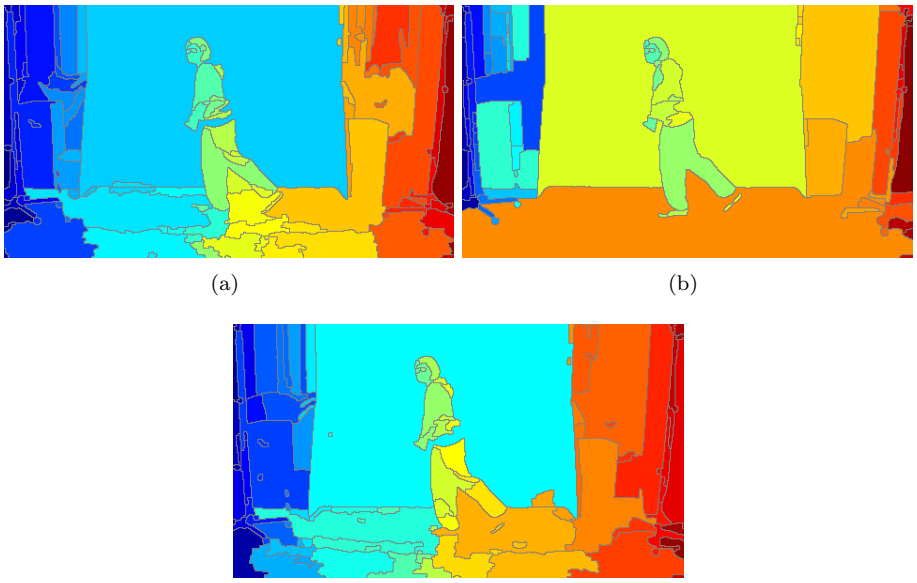
we are capable of modelling this behaviour, an accurate object match between images can be made. Once images are matched, the deformation field can be calculated and the interpolation computed.

For this first step we will make use of the Watershed algorithm. As explained in the previous chapter[1], this segmentation process will divide the image into homogeneous regions with a set of edge pixels delimiting each one of them called *edglets*. It is of our interest that the defined regions are the most similar possible between both images. This way, the correspondences will be computed more accurately. In order to make this possible, special attention must be paid to the image simplification process. Even if both input images represent a same scene, differentiated by some few angle degrees or by a fraction of a second, noise components make that same objects differ meaningfully at pixel level. Therefore, because of the nature of the Watershed algorithm, without a careful image simplification, segmentation can vary just too much between the two images, making the matching process an impossible task. Figure 2.3 shows an example of this possible problem.

Different image simplification methods may be used. Figure 2.4[2] shows some

[1] Section 1.2

[2] The adaptive noise-removal filter was applied using Matlab function `wiener2.m` with parameters $M = 3, N = 3$



(a) (b)

(c)

Figure 2.4: Watershed image segmentation using different image simplification procedures: (a) opening by reconstruction filter, (b) anisotropic diffusion followed by an opening by reconstruction filter, (c) adaptive noise-removal filtering followed by an opening by reconstruction filter.

examples of how the Watershed algorithm offers different results depending on variations in the simplification stage. Morphological operation filters are helpful tools for this kind of necessities. For this work an *opening by reconstruction* filter was employed. This technique *flattens* the image by eliminating local[3] minima and maxima; homogeneous regions are simplified, and therefore we get rid of an important amount of noise that may alter the object matching.

2.2 Edglet matching

Once the segmentation is done using the Watershed algorithm, the next step is to match the edglets between both images. Euclidean coordinates for each and every one of the edglets are found, and with this information the *shape context descriptors*[4] are computed for the whole set of them. For the interest of this work, with respect to determining the shape context histograms for each edglet, 8 bins were used for $log(r)$ and 14 bins for θ (refer to Figure 1.3(c) to get a clear idea of what the histogram bins are).

Histograms for each edglet are afterwards utilized for the computation of matching costs described by Equation 1.8. Since costs for every possible edglet matching between images is calculated, we obtain a matrix of size $N \times N$, being N the number of edglets from the image which has the most. To square the

[3]The *local* aspect will be given by the size of the structuring element employed in the morphological operations.

[4]The shape context descriptor is explained in detail in Section 1.3

matrix, in case of a difference in the number of edglets in each image (normal case), *dummy* pixels are used for the padding. These *dummy* pixels may be later used as matches for outliers.

The best matching combination will be that which offers the lowest cost. To know which one it is, the *matching matrix* is subjected to the Auction algorithm[5]. A first *assignment matrix* is obtained, which is then submitted to several processes in order to be optimized.

In practice, not all possible edglet matchings are considered. To reduce computation time, knowing that an object can *move* only up to a certain distance from one perspective (or time instant) to another, the search for each edglet's matching is limited to a determined surrounding area where it may be possible to find it. This *searching area* will depend, in the case of a change in perspective, on the angle difference between cameras, and the distance from these to the object. In the case of a time difference between images, the searching area will depend on the speed of the moving objects in the scene. Since this is not an easy task and the intention is not knowing the exact limits, we consider that using our own criteria is enough for determining a *reasonable* size of the searching area. For the aim of this work a squared area of 50×50 pixels was used, with the pixel to be matched in the center.

2.2.1 Sub-imaging

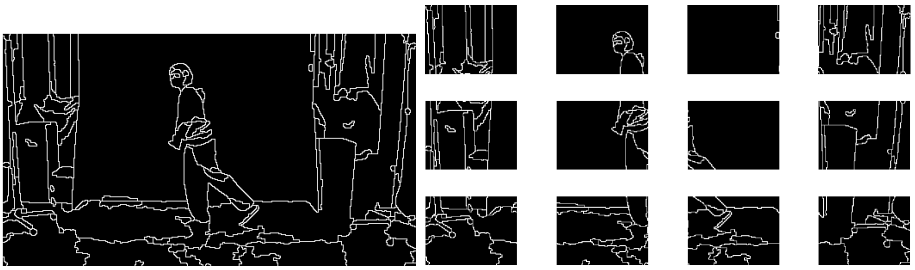
Depending on the scene and on the size of the image, between 2,000 and 20,000 of the pixels are edglets. This means that the matching matrix will have a considerable size, enough to represent a problem (in a computation time aspect) when submitted to the Auction algorithm.

To mitigate this inconvenience, after the edglets are detected, the image is divided into several smaller images called *sub-images* which are analysed separately. This is a delicate procedure considering that, even though the Auction algorithm is computed locally in each sub-image and that an assignment matrix is obtained for each one of them, the possible matchings for each edglet in the sub-image must take into account the global set of edglets. This has to be considered particularly for the case of edglets near the edges of each sub-image, whose match may be located in the adjacent sub-image. To deal with this problem, half-superimposed sub-images are also used. This way, we consider up to four possible matches for each edglet (one for each sub-image to which it may belong to) and assure that the partitioning of the global image does not represent an issue in order to find the best match. Of course, this means that a later study must be done in order to decide which of the different matches is the correct one.

With all, sub-imaging represents an important computation time saving. If, for example, the image is divided into 12 sub-images (Figure 2.5) plus another 8 that act as the superimposed ones, the Auction algorithm computation time (for the whole image) is reduced from a factor of N^2 to $20 \times (N/12)^2$ (being N the order of the matrix) which translates into a huge saving, considering that we are dealing with matrices with orders of several thousands[6].

[5] The Auction algorithm is explained in detail in Section 1.4

[6] For example, in the case of having 10,000 edglets, the time factor reduces from 100×10^6 to less than 14×10^6.



(a) (b)

Figure 2.5: Sub-imaging example. (a) Edglets computed from the original image, taken from the Watershed algorithm result, (b) 12 sub-image division.

Besides finding the best matching for each edglet between the superimposed sub-images, the fact of performing local analysis instead of a single global one involves dealing with some other issue whose nature and solution are explained in the next section.

2.3 Matching optimization

Since matches are found from each one of the several sub-images into which the whole image is divided, a one-to-one assignment cannot be guaranteed. Some of the edglets from the second image will have more than one assignment, and therefore, some of them will have none. These are called *bad assignments*. To deal with them the images are put into an iterative process which basically consists of repeating the following two steps:

1. The edglets which have *good* assignments, i.e. one-to-one assignments, are found and removed from both images.

2. The *new* image (without the good assignments) is then processed again with the Auction algorithm, using the same original cost matrix, but with bigger sub-images.

The procedure is repeated until the image is no longer divided into sub-images and analysed globally. Up to this point, enough good assignments have been removed and the number of edglets is sufficiently low for the Auction algorithm not to last *too much*. For this work, four iterations are done before finding a global one-to-one assignment.

2.4 Local homography computation

The purpose of finding the edglet matching is to use this information to compute local homographies. In Section 1.1 we defined a homography as the projective transformation that takes each point $\mathbf{x}_i$ in $\mathbb{P}^2$ to its corresponding $\mathbf{x}'_i$ likewise in $\mathbb{P}^2$. For our purpose, the points $\mathbf{x}_i$ and $\mathbf{x}'_i$ are points in the two images, respectively, each image being considered as a projective plane $\mathbb{P}^2$.

We now talk about *local* homographies because we will compute a projective transformation for each image segment. Each segment (homogeneous regions calculated with the Watershed algorithm) and its corresponding homography is what Stich et al. [4] define as a *translet*.

It is important to consider that because of occlusions and other "noise" factors, the assignment matrix will contain between 20 and 40% of outliers. These must be removed from the set of matchings that will be used for the homography computation. I will now present the whole process for the computation of the homographies, including some *filtering* procedures to get rid of the previously mentioned outliers. Remember that these are local homographies, so the following steps are taken for each one of the homogeneous regions:

- **Cost filter**. As said in Section 1.3 when a difference in the number of edglets between the two images exists, the cost matrix is squared by padding it with "dummy" pixels that act as possible occlusions. The cost for matching with these pixels will always be higher than a certain threshold ϵ_d defined in the *cost filter*. These "dummy" matchings, together with the rest of matchings that overpass cost ϵ_d are rejected (see Figure 2.6(b)).

- **RANSAC filter**. With the remaining matches, the RANSAC algorithm[7] is applied. Remember that this algorithm includes some random processes; this means that we will not always obtain the same results from the same set of assignments. Figure 2.6(c) shows the remaining assignments after cheap and RANSAC filtering.

- **First homography computation**. Using the matches left, after applying the first two filters, a first local homography is computed:

$$\mathbf{H}\mathbf{x}_i = \mathbf{x}'_i,$$

  where $\mathbf{x}_i$ corresponds to the remaining edglets from the first image, and $\mathbf{x}'_i$ to the set of remaining edglets from the second image.

- **Distance filter and second homography computation**. Any pixel $\mathbf{H}\mathbf{x}_i$ that lies farther than 3 pixels away from its corresponding $\mathbf{x}'_i$ will be filtered out, and the matrix $\mathbf{H}$ computed again (second local homography). This is called the *distance filter*.

- **Possible extra iteration**. Considering that for different RANSAC filtering, we may obtain different results, an extra iteration of the whole process (from cost filtering to distance filtering), may be applied again to those regions which reach a determined percentage of outliers (specified by the user), or whose inliers are not enough to calculate $\mathbf{H}$. It is a trade-off between computation time and the effort for obtaining the most amount of translets possible. It is important to point out that when the matrix $\mathbf{H}$ is not found in the first iteration, only in about 5% of the cases the algorithm will succeed in finding it in a second iteration.

[7] Refer to Section 1.5 for more information about this algorithm

(a) (b)

(c)

Figure 2.6: Example of edglet matching for a specific homogeneous region; in this case the face of the dancer. Blue crosses correspond to edglets in the first image, and yellow crosses correspond to the edglets that have been assigned to them from the second image. Correspondences have been represented with green lines. (a) Initial assignments, (b) after cheap filter has been applied, (c) after cheap filter and RANSAC have been applied.

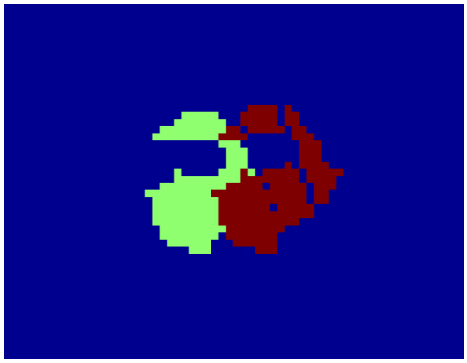


Figure 2.7: Example of the transformation of a determined region in the first image (greenish area) into the synthesized region (redish area) by means of the homography. These areas correspond to the face of the dancer from previous examples shown.

2.5 Selective translet merging

As detailed in Section 1.1, at least four point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ have to be specified in order to fully constrain $\mathbf{H}$. Match filtering may lead to a situation in which not enough correspondences remain for the computation of matrix $\mathbf{H}$. This problem is addressed by a selective translet merging. Every translet that does not succeed in having enough inliers for the second computation of its homography, is merged with the neighbouring translet with which it shares the highest number of edglets. This can be done because one same edglet may be part of up to four sets of edglets that delimit different translets (most frequently of only two).

Once the translets are merged, the one that does not count with a transformation matrix $\mathbf{H}$ is automatically provided with the one from its neighbouring translet. This way it is guaranteed that every homogeneous region has a corresponding transformation matrix.

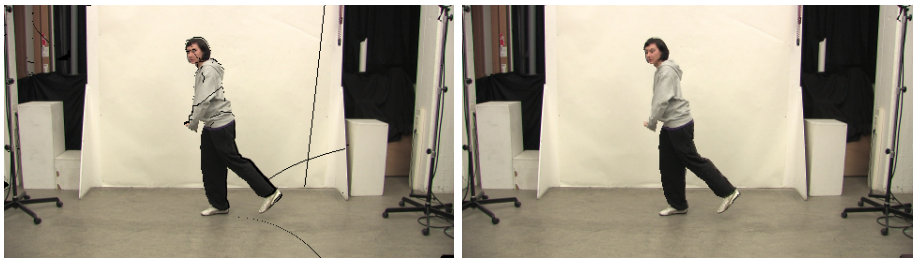
2.6 Deformation vector field and interpolation

The deformation vector field will be the tool that allows making the whole transformation of the first image and synthesizing the different viewpoint. This is done on a per-pixel basis. Since translets partition the image, each pixel in the image is uniquely associated with a translet t. The deformation vector for a pixel $\mathbf{x}$ is thus computed from the translet's homography H_t as

$$d(\mathbf{x}) = H_t \cdot \mathbf{x} - \mathbf{x}. \tag{2.1}$$

Figure 2.6 shows an example of how the deformation vectors work for each pixel. The synthesized image may contain *cracks* due to the deformation of each translet.

This deformation field will therefore synthesize the second image (alternative viewpoint or time instant). In order to generate the desired **interpolation**, the



(a) (b)

Figure 2.8: Once the interpolated image has been computed, *black* areas may appear due to occlusions or translet "cracks" (a). To solve this issue, the Inpainting algorithm is employed (b).

deformation vectors must be scaled depending on the position (in time or space) desired by the user and added to their corresponding pixels $\mathbf{x}_i$:

$$\mathbf{x}_{int} = \mathbf{x} + (a \cdot d(\mathbf{x})) \,, \tag{2.2}$$

where a is the scalable constant.

If, for example, the interpolated result is desired to be at a mid-position, a must be equal to 0.5. A value of $a = 1$ will synthesize the viewpoint from the original alternative perspective, i.e. synthesizing the second image.

2.7 Inpainting

In Section 1.6 I mentioned that one of the toughest issues to deal with, when computing correspondences, are occlusions. When having a different perspective of a scene, or when an object in this scene moves, occlusions inevitably occur. These areas in the image (behind a moving arm for example), in which edglet matching cannot exist since they can be seen from only one of the viewpoints (or time instants), have to be filled with information taken from either two images, or by a post-processing stage carried out in the synthesized one. We opted for this second option applying the Inpainting algorithm[8] directly in the interpolated image.

Missing regions due to occlusions and other kind of imperfections, as those of the translet cracks, are fixed using the Inpainting algorithm. Figure 2.8 shows an example of the performance of this strategy. The example is the result of a time-interpolated image. Notice in the figure that occlusions happen mainly around the object that is moving; in this case the dancer.

[8]More information about Inpainting can be found in Section 1.6.

Chapter 3

Results

In the following chapter the objective is to show the results obtained from the whole algorithm designed which, of course, takes into account the series of tools presented in the two previous chapters. Errors and inaccuracies in the results will be treated with special emphasis in order to understand their origin and to be fixed by anyone who possibly takes the staff to give continuity to this work. The first three sections are dedicated to show interpolation results first in space, then in time and finally in both, space and time, done simultaneously. The last section is devoted to numerically evaluate the results.

3.1 Space interpolation

Space interpolation refers to the interpolation of two images obtained from the same scene, but from different perspectives. As mentioned in Chapter 2, the angular distance between the cameras that capture the images should not exceed 10 degrees.

In the following pages I will show some figures with some information that precedes the final interpolation result. These various images will help us to better expose a judgement on the final output by visually analysing each step of the algorithm. Let us proceed:

- **Figure 3.1** shows the two input images for this example. Observe, at the edges of each image, the zones of the scene that are not visible in both images because of the switch of perspective (e.g. the left white column in 3.1(a) is invisible in 3.1(b)). These areas, together with occlusions, will represent the biggest challenge for the interpolation process. Occlusions, as described in previous chapters, are those areas in the image that cannot be seen from the alternative viewpoint because they are covered by other objects (e.g. the right side of the dancer's face in 3.1(b) cannot be seen in 3.1(a), covered by the rest of the face because of the change in perspective).

- **Figure 3.2** displays the Watershed results. These images contain the edglet information and the segments that will define the translets. Every white pixel surrounding the homogeneous regions represents an edglet. The objective will be to match every edglet in image 3.2(a) with an edglet in image 3.2(b). If more edglets exist in the first image, those which do not

find any *low cost* match will be matched with *dummy* pixels that represent possible occlusions or other kind of outliers.

The Watershed algorithm is not a linear system, therefore any slight difference, between stereo images in the supposed homogeneous regions, may result in important differences in the algorithm output. Even if images from Figure 3.1 may seem identical in certain areas, at pixel level differences exist mainly because of unavoidable noise which comes from the very moment the images are taken (i.e. from the video camera itself), to any post-processing or manipulation of them (e.g. any compression or resizing procedures).

These differences can be clearly observed on the upper left part of the images. Even though original images show a clear similarity in the area behind the white column left to the white screen, and which the correspondences between both images could be pointed out easily, the algorithm fails in segmenting equally. The problem arises more clearly when less contrast exists between regions. This phenomenon, as will be shown later, will drive to some significant errors, based on the simple fact that edglets from the first image will not be able to find good matches.

Alternatively, we have very similar segmentations in the area of the dancer, for example. In these cases, edglets will have no issue in finding accurate matches.

- **Figure 3.3** shows the edglets from the two images. Each edglet has a different value, represented in these images as different colors. A delimited area (depending on the angle distance between cameras and the distance from these to the closest object) around each edglet in 3.3(a) is defined, and costs in order to match them with every edglet from 3.3(b) inside this area, are computed. Based on these costs, by means of the Auction Algorithm (explained in Section 1.4), matches are found for each edglet. Image 3.3(c) shows where each edglet from 3.3(a) would move, depending on the computed matches. This image, even if it does not exactly tell about the correspondences, it does help in giving an idea (thanks to the colors of the eglets) of the behaviour of the matching process. Matches that exceed a user-determined cost, are not shown in the image.

- **Figure 3.4** represents the merging process. Image 3.4(a) shows how some translets have merged according to the criterion explained in Section 2.5. Those translets which have *too many* outliers, and therefore which cannot compute their corresponding transformation matrix $\mathbf{H}$, will merge with the selected neighbour translet. These merging processes are very likely to happen (when performing space interpolation) at the edges of the images, where accurate object correspondences cannot be made. In this example the number of translets has reduced from around 90 to approximately 30.

  Image 3.4(b) shows the segmentation of the first image once the merging has been done and the edglets added to their corresponding translet. Edglets will add to the translet which represents a *better* transformation, i.e. where its average distance $||\mathbf{H}_t\mathbf{e}_i - \mathbf{e}'_j||$ is lower.

- **Figure 3.5** displays the deformation vector fields. Image 3.5(a) represents the deformation in the x axis and 3.5(b) in the y axis. Deformation is

computed in a per-pixel level, i.e. each pixel takes the transformation matrix **H** that has been calculated for the translet to which it is part of, and determines its new position in the alternative viewpoint. These two images are quite illustrative to show the particular areas which have a bad determination in their transformation. Particularly, image 3.5(b) helps in this way by highlighting pixels that move distances out of normal ranges for this example (e.g. redish and bluish areas that represent 60 and -60 pixels of transfer respectively).

- **Figure 3.6** are the final results. Image 3.6(a) shows the results once the total deformation field has been applied to the first image. Pixels have moved to their new positions according to the deformation vectors computed using the corresponding transformation matrix **H**. As it is evident, *blank* areas exist where no pixels have moved. These areas are due to disocclusions or to errors in the transformation computation.

  Image 3.6(b) shows the final result once the Inpainting algorithm has been applied. *Cracks* due to the translet transformation are corrected very well without leaving any trace. Also, missing regions due to disocclusions around the dancer are treated in a very plausible way according to this work's motivation: in the perception manner.

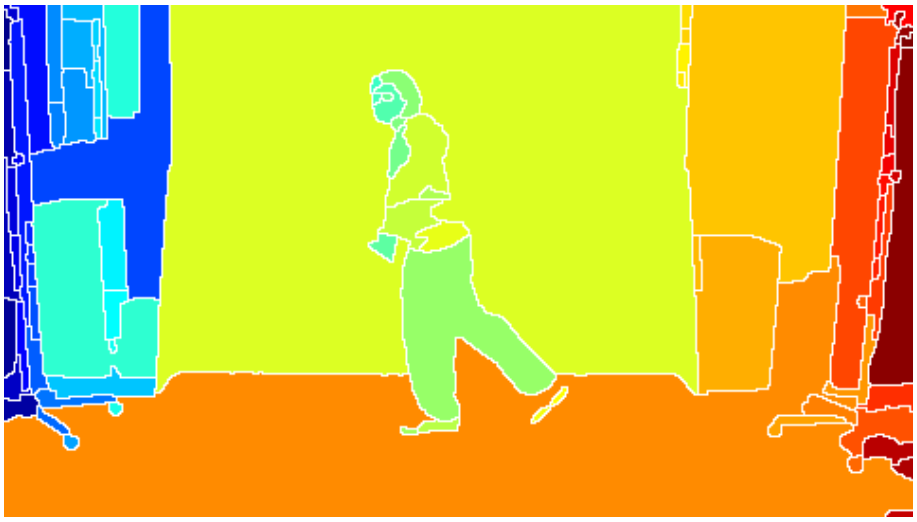
  However, areas which had been anticipated as problematic, in the Watershed computation and the deformation field images, have noticeable errors. The white column in the left side of the image is a clear example of the inaccuracy in the estimation of the transformation process. Other than this and some other close regions, the final image has achieved plausible results, especially with respect to the *important* object in the scene which is the dancer.

(a)

(b)

Figure 3.1: Original stereo images. Input images for the space interpolation process; I will refer to them as first image (a), and second image (b).

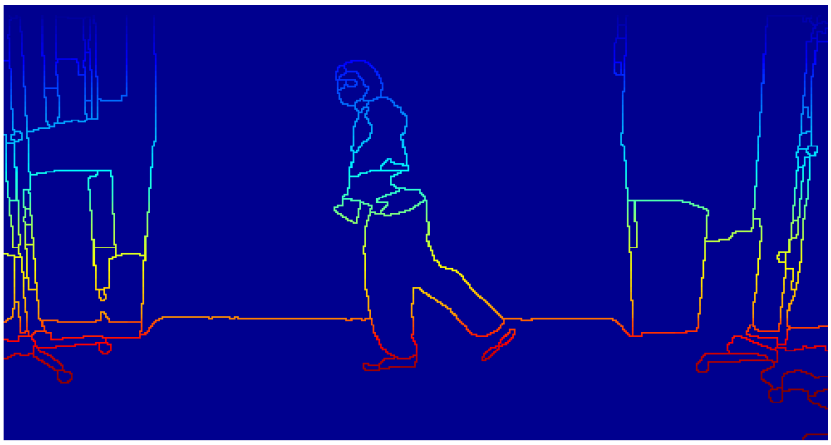
(a)



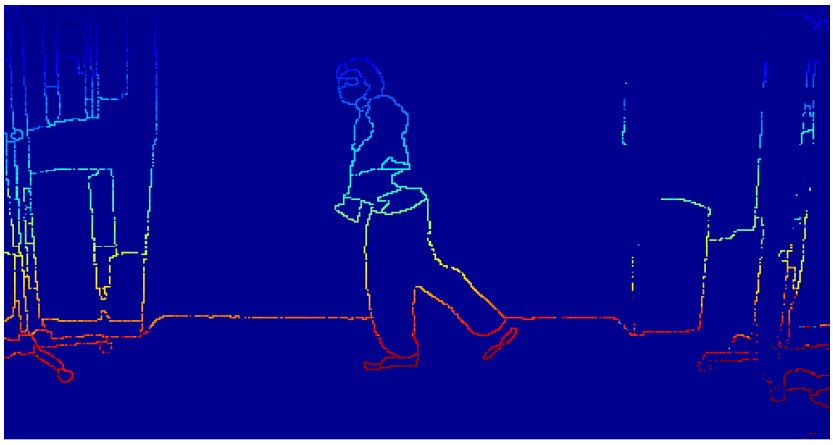
(b)

Figure 3.2: Watershed images. Results from images from Figure 3.1 when processed by the Watershed algorithm.

(a)



(b)



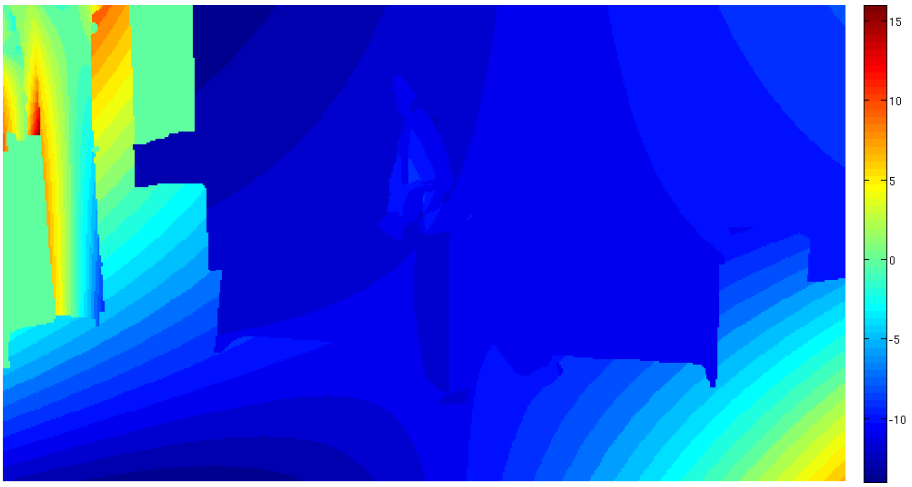
(c)

Figure 3.3: Edge correspondence. (a) and (b) show the edglets from the first and second original images respectively, and (c) shows where has each edglet from the first image moved to the second according to their matches.

(a)

(b)

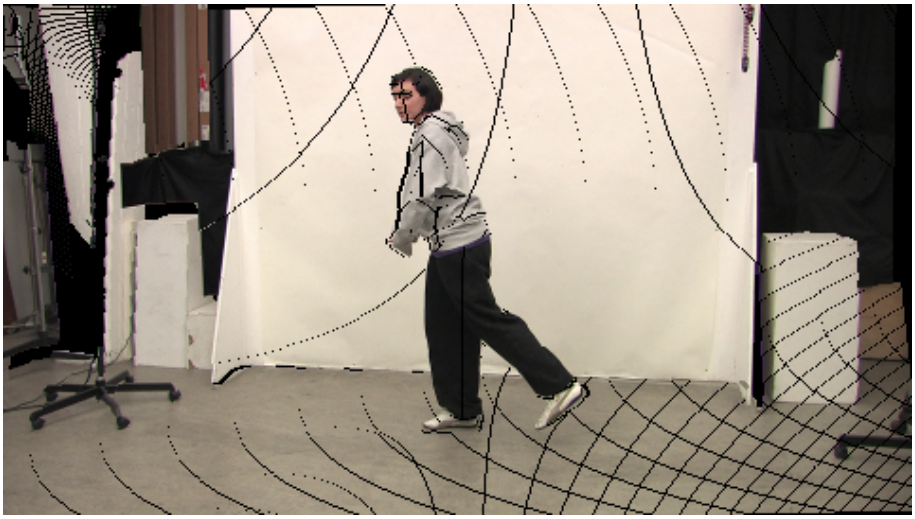
Figure 3.4: Translet merging. These two images show the results after merging the translets that have a considerable number of outliers. Image (b) has included the edglets into the homogeneous regions so they can be processed with their respective transformation matrix **H**.



(a)

(b)

Figure 3.5: Deformation vector fields. (a) and (b) show the deformation fields for the x and y axis respectively. Scale is represented in number of pixels.



(a)

(b)

Figure 3.6: Final interpolation results. Image (a) shows the output before the inpainting algorithm has been applied. Image (b) shows the final result once the inpainting algorithm has been applied.

3.2 Time interpolation

The following images show the interpolation process for two images from different consecutive time instants. The video was taken at a rate of 25 frames per second, so the period between both images is reduced to 40 ms. Generating time interpolation images allows us to synthesize slow motion video effects.

As we will see, it is noticeable that in a time interpolation task, the problem reduces mainly to obtain the accurate transformation information in the objects which are moving (in this case, for example, the background does not move). This is different from the space interpolation process where, since a change in perspective exists, every object in the scene *moves* and a deformation computation is essential for the whole image.

Previous Space Interpolation section has described the general characteristics of each of the algorithm steps as well as the particularities for the space case. In this section I will only concentrate on pointing out and describing the particularities for the time interpolation case, and comparing them, whenever it helps the analysis, with the results shown in the preceding section. Let us analyse the results:

- **Figure 3.7** refers to the two input images. These have been taken in consecutive time instants. Notice that the differences between images reduce to the movement of the dancer; the background, of course, does not move.

- The background does not move, but this does not mean that the objects in it are *identical* between the two input images. This can be seen in **Figure 3.8**, where the Watershed algorithm results offer a slightly different segmentation in the background. Once again, as stated in the previous section, this is due to noise acquired from the moment the images are generated with the video camera and if any other kind of later image manipulation was done afterwards.

- **Figure 3.9(a)** displays the result after translet merging, once the edlets have also been added to their corresponding translets. It is clearly evident that the merging process is almost unnecessary for this case, for only about 3 pair of translets have been merged in this example. Compared to the two thirds of translets having been merged in the space interpolation case, it represents a significant difference. This is merely due to the similarity in the image segmentation between both images done with the Watershed algorithm.

- **Figures 3.9(b)** and **3.9(c)** show the deformation vector fields. Notice that in this case, except for one translet computed inaccurately, movement concentrates in the dancer's regions. The rest of the image maintains motionless since it only acts as a still background.

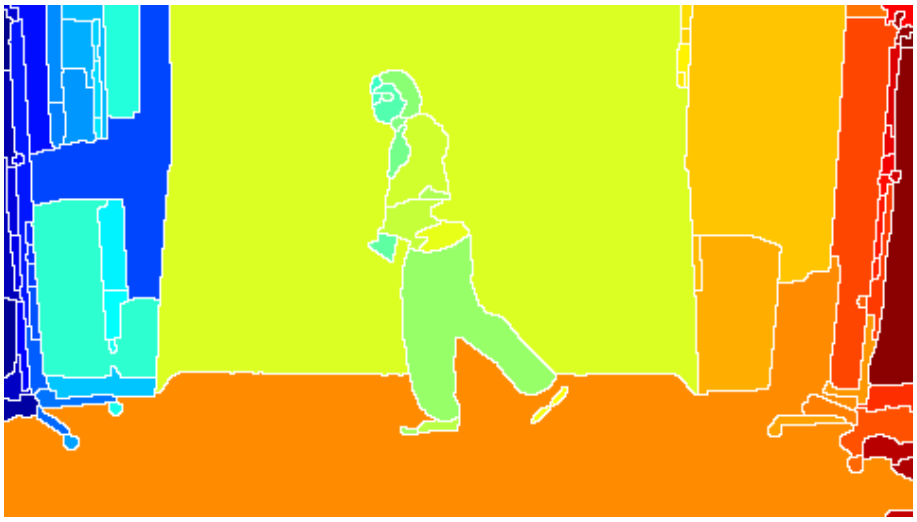
- **Figure 3.10** shows the results before and after applying the Inpainting algorithm. As it was expected, the missing regions are limited to the areas around the moving object (dancer) due to disocclusions. The final image has very plausible results. We have an interpolated image that allows a slow motion effect when submitted to video sequence. The translet that

had been detected in the deformation vector field as a possible error has no important consequences in the final interpolated image.

(a)

(b)

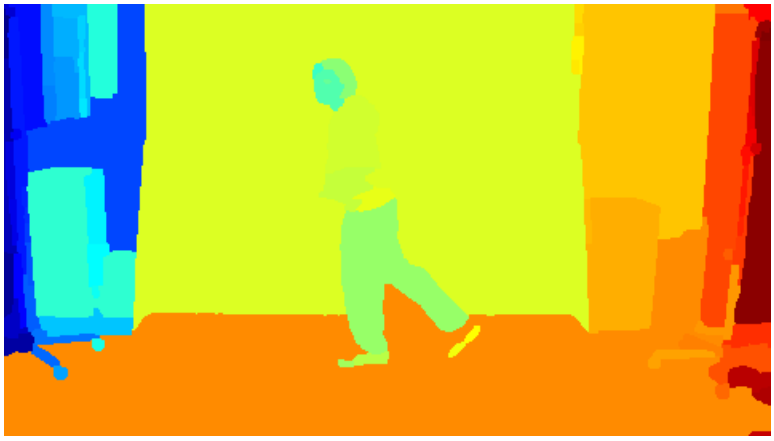
Figure 3.7: Original time consecutive images. Input images for the time interpolation process.



(a)

(b)

Figure 3.8: Watershed images. Results from images from Figure 3.7 when processed by the Watershed algorithm.



(a)

(b)

(c)

Figure 3.9: Translet merging and deformation fields. Image (a) is the selective translet merging result; images (b) and (c) represent the deformation vector fields in the x and y axis respectively. Scale is represented in number of pixels.

(a)

(b)

Figure 3.10: Final interpolation results. Image (a) shows the output before the inpainting algorithm has been applied. Image (b) shows the final result once the inpainting algorithm has been applied.

3.3 Space and Time interpolation

So far we have discussed space interpolation and time interpolation separately. In this section I present the case where interpolation is computed in both, time and space. Once again, the challenge is to achieve an accurate matching of the background, since the viewpoint will change, but also objects in the scene will have movement. As in the previous section, I will only mention the particularities for this special case. For general descriptions on the results please remit to the analysis done for space interpolation in Section 3.1.

- **Figure 3.11** displays input images for the space and time interpolation case. Cameras are separated no more than 10 degrees and the time difference between frames is equal to 40 ms.
- **Figure 3.12** shows the Watershed algorithm results. Notice that, as in the case of space interpolation, some areas have been segmented differently one from the other. This will lead to problems for edglet matching in those specific regions.
- **Figure 3.13(a)** is the selective merging result. One particularity of this result is that the background, which does not have any moving object between time frames, has merged into practically only two translets. As we will see in the final results, the two minor translets that have not merged into the main background translets, will show some inconsistencies.
- **Figures 3.13(b)** and **3.13(c)** are the deformation field vectors. These images are quite interesting since they clearly show the difference between both camera positions. The deformation fields tell how the movement of the cameras must be in order to navigate from one viewpoint to the other.

  Image 3.13(b) represents the deformation in the x axis. Notice that the background has higher values when moving away from the center. This has total sense considering that the focus of both cameras coincide in the dancer in the middle of the image, and therefore any object farther than her will *move* faster when changing the perspective.

  Image 3.13(c) represents the deformation in the y axis. This image is even more interesting. It clearly shows that rotation exists in the difference of position between viewpoints. The behaviour of the colors in the image (i.e. the distance each pixel moves from one perspective to the other), tells us that a rotation in the plane perpendicular to the floor exists when moving from one camera position to the other. This can be seen also in the x deformation field (image 3.13(b)), since also the values are not constant from top to bottom, and therefore existing also a slight rotation in the plane parallel to the floor. However, in this second case the rotation is more evident.

  Also, in both deformation fields, the movement of the dancer, due to the interpolation done in time, can be seen in the middle of the images. The particularity of the region at the right side of the graphs, where the value is zero in the two outputs, is because the algorithm has been programmed in a way that, in an extreme case, it totally discards the display of those regions where the error in the calculation of the transformation process is

bigger than a user-determined threshold. This threshold normally represents a limit where the display of that particular region would notoriously affect the perception of a natural interpolation.

- **Figure 3.14** are the final results, before and after the Inpainting algorithm has been applied. Missing regions at the edges of the image are visible as well as some other errors such as translet *cracks* that could not be fixed with the Inpainting due to *fold-overs*, i.e. overlapped regions. However, in the central area of the image, we obtain good interpolation results that lead to plausible object movement.

In the next section, where another case of time interpolation is analysed, we will have the opportunity to compare the results with ground truth by interpolating images A and C and comparing the result with image B from a consecutive image sequence in time $A < B < C$. This is rather difficult to do in space because of the inaccuracy on the information on the exact position of the cameras.

(a)

(b)

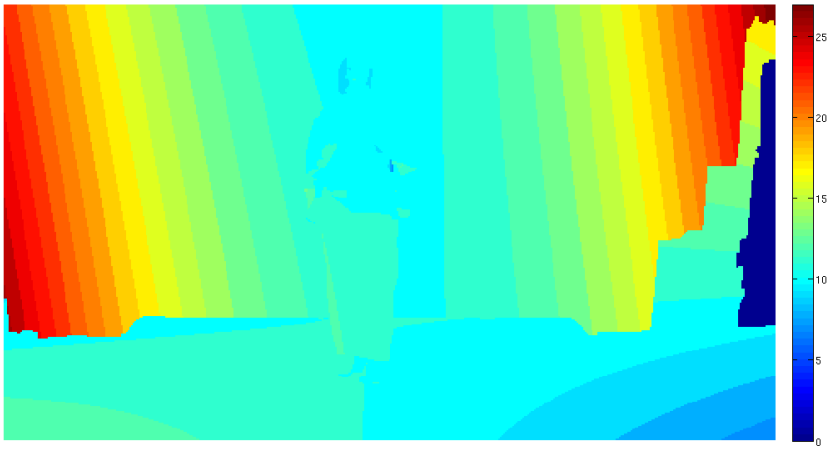
Figure 3.11: Original stereo and time consecutive images. Input images for the time interpolation process.

(a)

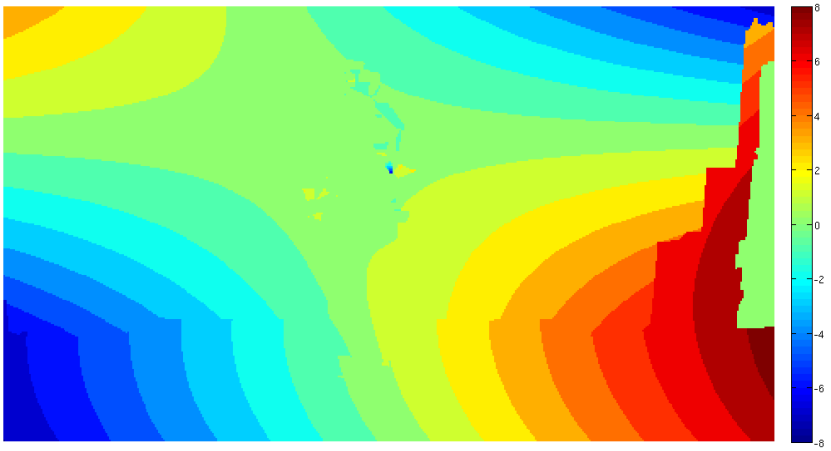
(b)

Figure 3.12: Watershed images. Results from images from Figure 3.11 when processed by the Watershed algorithm.

(a)

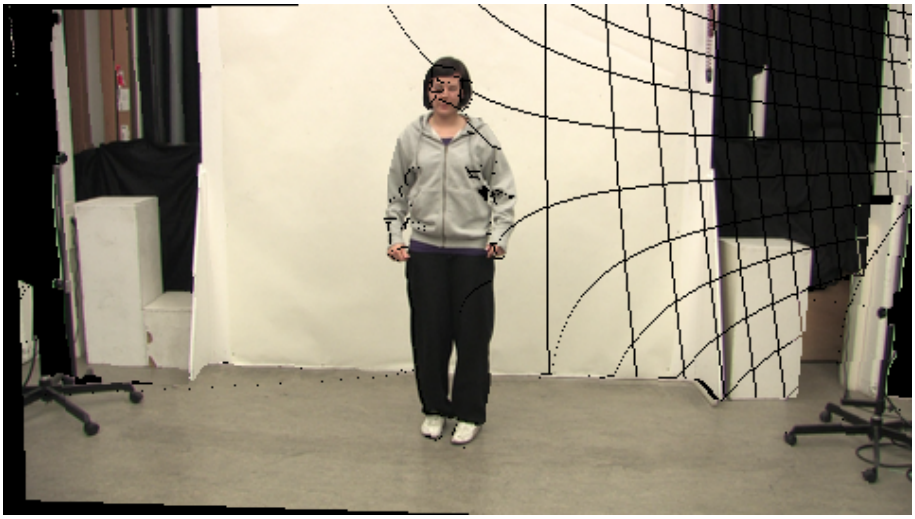


(b)



(c)

Figure 3.13: Translet merging and deformation fields. Image (a) is the selective translet merging result; images (b) and (c) represent the deformation vector fields in the x and y axis respectively. Scale is represented in number of pixels.



(a)

(b)

Figure 3.14: Final space and time interpolation results. Image (a) shows the output before the inpainting algorithm has been applied. Image (b) shows the final result once the inpainting algorithm has been applied.

3.4 Evaluation of the results

Considering that the interpolation strategies are perception-motivated, results are quite difficult to evaluate. The natural way of doing so would be by means of user studies that would, in fact, evaluate the aimed characteristic of the results: plausibility to human eyes. This, of course, would have required more human and time resources which were out of scope considering the objectives of this work.

A quantitative way for evaluating the *quality* of the results is computing the PSNR (peak signal-to-noise ratio) between ground-truth and the interpolation result. PSNR is usually used as a measure of fidelity of compressed video or image signals. It is defined as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{max^2}{MSE} \right), \tag{3.1}$$

where the MSE (mean squared error) is defined as:

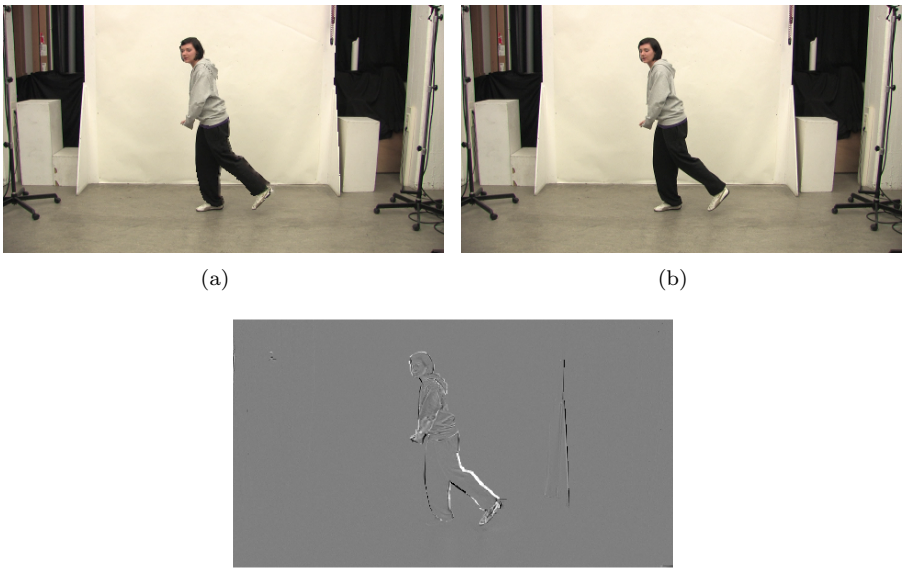
$$MSE = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} \left[I(i,j) - K(i,j) \right]^2 . \tag{3.2}$$

These equations apply for two images I and K of size $m \times n$, where one of them is considered as the noisy approximation; in our case, the interpolated image. max is the maximum possible value of the pixels. Since pixels are represented using 8 bits, $max = 255$.

The PSNR is not a very accurate evaluation method for our experiments. This can be understood if we think of an object in the interpolated image that has moved some few pixels less than the position of it in the ground-truth image. This difference, while being completely imperceptible for the human eye (for perception), will significantly affect the PNSR.

Nevertheless the PSNR values may give us an idea of the performance of the algorithm. When submitting images 3.15(a) and 3.15(b) to the time interpolation strategy, we obtained a PSNR equal to 30 dB. In image compression good quality is considered above 45 dB and bad quality under 30 dB. However, the PSNR tends to be less representative of the error near 30 dB and more representative near 45 dB, where the error is minimum. Image 3.15(c) helps adding substance to this number. Error concentrates in the periphery of the moving object. This is important to be clarified since, as mentioned before, *small* object displacements may be insignificant for our interest and not for the PSNR measurements.

An interesting phenomenon occurs in this example which deserves attention: Between both images, besides other parts of the body, the leg of the dancer moves. This movement results in missing regions because of the disoccluded area behind the leg. Missing regions are solved using the Inpainting algorithm; however, in this case it does not perform very well. This can be noticed in error image 3.15(c) where a white stripe follows the contour of the dancer's leg. The problem lies in the edglets' decision for merging with a specific translet. When the Watershed algorithm does the segmentation process, regions are separated by lines of pixels (edglets) which may clearly correspond to one of the regions they are separating. This is more evident when a high contrast exists between adjacent regions. In our example, edglets were part of the dark region (the leg),



(a) (b)

(c)

Figure 3.15: Final result and difference to ground truth. (a) is the final interpolation result, (b) is the real in-between image, and (c) represents the difference between the interpolation result and (b).

but when the decision had to be made, they were merged into the clear region (white screen). Therefore, in the result, the missing region was surrounded by *leg pixels* and filled inaccurately with dark pixels with the Inpainting algorithm instead of filling them with the background screen pixels. In the next chapter (Future Improvements) I will talk more about this issue and possible ways of solving it.

The graph in Figure 3.16 shows the PSNR results of 25 interpolated images taken from a video sequence. Values oscillate between 26 and 37 dB, where the lowest values were obtained from instants where objects in the scene have moved faster as it was expected. The average value is 31.2 dB, with a standard deviation equal to 3.

It is important to consider that for these experiments, interpolation has been done for non-consecutive time images. As in the rest of the experiments, image 3.15(b) is two time instants later than 3.15(a). This has been done so the result could be compared with the available actual in-between image. Since the difference between images is larger than a normal case, higher errors are expected.

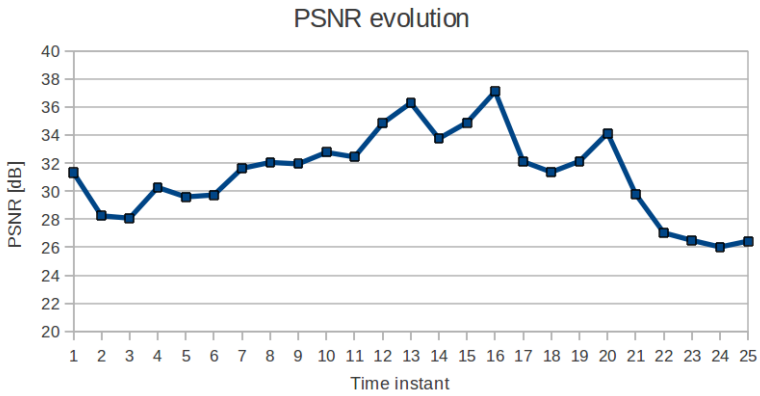


Figure 3.16: Interpolation PSNR. These values have been obtained from 25 time interpolation images.

Chapter 4

Future Improvements

This brief chapter is dedicated to any student who may be interested in giving continuity to the work presented in this thesis. The idea is to offer guidelines wherein to concentrate any future contributions in order not to duplicate efforts and to accelerate possible improvements. I hope that readers find this information helpful. The following are the points which I considered to be the immediate issues to be resolved.

1. The first thing to consider for future improvements is the evaluation of the performance of Watershed algorithm as the segmentation technique. One of the advantages of this method is that the segmentation process defines, besides the homogeneous areas, the edglets as limits of these areas. In Stich et al. [4], the procedure is different. The definition of edglets and homogeneous areas is done separately; the first is done by using edge detection methods, particularly in their case the Compass operator ([10]), and the latter with a segmentation strategy different from the Watershed ([11]). This means that the way of relating edglets with translets changes between their method and ours.

   The issue with the Watershed algorithm is, as stated in Chapter 3, its high sensitivity to noise, which forces to rely on a previous simplification stage which, in turn, has plenty of difficulties in assuring that the images to be interpolated will be segmented *sufficiently* similar. This segmentation similarity is crucial for the performance of the whole interpolation algorithm.

   Either efforts are focused in finding a methodology for an accurate segmentation using the Watershed algorithm, which basically means concentrating on the simplification stage, or maybe trying with different segmentation strategies.

2. Finding accurate dense correspondences between the two images is a fundamental step in the interpolation algorithm. Stich et al. [4] suggest an extra step after the edglet matching determination procedure to optimize the correspondences: user interaction. In an ideal case this should not be necessary; actually one of the objectives of this strategy was to avoid user interaction. However, it may be interesting to take a look at the Klose et al. article titled "Flowlab - an interactive tool for editing dense image

correspondences" ([16]), where an interactive tool that allows the user to modify and correct dense correspondence maps between the two given images is presented. Supervised correspondence matching may be worth it if the time the user consumes in doing so is not counterproductive.

3. The selective translet merging is another crucial step in the algorithm. Up to now, the merging condition is limited to the number of edglets both translets share. In other words, a *bad* translet will merge with the *good* translet around it with which it shares the most number of edglets. I consider this condition not to be the best way to make the decision. A better option, and it would need some experimental stage to prove that it really improves the algorithm, would be to consider gray-level similarity (less contrast) between regions to determine merging. Another option could also be texture analysis. It has to be considered that, theoretically, merging must be limited to those regions which are part of the same object or that share similar flow between images.

4. Once the deformation field has been computed, the synthesize of the interpolated image is done at a pixel level, but translet by translet. Up to now, there is no criterion that determines the order in which each translet is put into the *canvas*. This order has an impact on the result, considering that fold-overs (superimposed pixels) exist. Which translet should be in front of the other? That is not an easy question to answer. The objective is then to know the distance of the objects from the camera and this way knowing which translet should be above when fold-overs exist.

   When changing perspective, objects in the scene will *move* faster according to the distance from the rotation center defined by the movement between the two cameras (the point where their optical axes coincide). If both cameras look parallel at the scene, optical axes coincide in the infinite and therefore, distance from the cameras is inversely proportional to the speed of the objects.

   So, if the speed of the translets is used to determine their proximity to the cameras and this way be able to solve fold-overs, extensive information about the position of the cameras and their optical characteristics is needed. Even then, proximity is a difficult measurement, for a fast moving object could be in front or behind the camera rotation center. In the examples used in the previous chapter, the rotation center is aimed to be found in the dancer, therefore any object closer or farther from her will move faster and fold-overs would be difficult to overcome.

   One first quick solution is to decide putting the *fastest* translet in the top, since optical axes tend to be parallel and therefore coinciding many times farther than any object in the scene. But it is important to know that it is not a general solution.

5. One problem that we faced when displaying the results was that translet cracks were not able to be fixed by the Inpainting algorithm (which performs very well in this matter) when facing fold-overs. Inpainting acts on missing regions, so any translet crack that falls over another translet will not be taken into account to be fixed. A possible solution to this issue is to apply the Inpainting algorithm each time a translet is *placed* in the

interpolated image. This way, if fold-overs exist, the translet cracks will already be solved.

6. In Section 3.4 a problem related with edglet merging was encountered. At a certain point of the interpolation algorithm, after dense correspondences have been computed, edglets must merge with translets so they can obtain its transformation matrix $\mathbf{H}$ and define their deformation vectors. Up to now edglets will choose the translet where its average distance $||\mathbf{H}_t\mathbf{e}_i - \mathbf{e}'_j||$ is lower. This leads to inaccuracies in some occasions as it was pointed out from Figure 3.15(c). Since it is difficult to know beforehand to which region each edglet really belongs to (it will depend on the gradient computation technique among other factors), one possible solution is to change the edglet merging criterion to gray-level similarity (as I proposed also for translet merging).

7. For the time interpolation case, if the moving objects cover a relatively small area of the scene (as it happens in the example shown in Section 3.2), dense correspondence computation could be limited to the moving objects. This means that motionless background areas, considering that they do not change between images, would be directly taken from either image without submitting it to the whole interpolation algorithm. This way inaccuracies could be avoided in these regions. This may not be an easy task, but I considered it to be an interesting mechanism for time interpolation optimization. Computing the deformation vectors for regions that are known to be immobile in advance seems to be an unnecessary waste of energy.

8. Finally, another aspect which future work should pay attention to is in the algorithm computation time. My programming skills are not specially sharp, so I am sure that the code could be optimized to run faster. Some parts of the code spend much more time than others and my intention is to point out those which last too long and that I consider could be improved.

   Two parts of the code must be the first to be treated if less computation time wants to be consumed. One is the Auction algorithm function (`auction.m`) and the other is the cost matching computation function (`costs_matching.m`). The time these two functions spend is considerably large to invest some effort in optimizing them.

Conclusions

The goal of this work was to develop an alternative to the dependence of mere hardware resources for the creation of video navigation in space and time, where viewpoints are not limited to the position of the cameras, and therefore offering better navigation flexibility. Instead of the need of a wide video camera infrastructure to cover every desired viewpoint of the scene, just a few cameras are distributed along the observer plane. Video navigation is then generated with post-processing methods applied to the images taken from the video sequences. This allows a greater flexibility of the navigation, where viewpoints are not limited to the position of the cameras.

The algorithm has not yet reached the capacity of an imperceptible replacement of real ground-truth images. However, I am firmly convinced that the followed scheme has the potential to entirely achieve the aimed goal. The results obtained in the interpolated images are quite good, under the understanding of this work as a first phase for a wider project. Chapter 4 points out the strategies to be carried out in order to considerably improve the interpolation process.

Since video navigation is the aimed application for this algorithm, it has been motivated to respond to perception plausibility. As mentioned in Chapter 3, a representative evaluation of the results offers a challenge under this condition; the employment of user studies should be considered for the evaluation at future stages.

Video navigation tests have been carried out using our approach. The appearance of some *artifacts* in the interpolated images does not allow a totally smooth navigation. This happens especially in the space interpolation case, in the areas at the edges of the images. In the case of time interpolation, image sequences offer quite good *slow motion* effects. Almost no artifacts are detected in this case.

The application of the algorithm in video navigation has been done merely by placing the interpolated images between the original images of the sequence. This is a very robust approach; some other considerations have to be taken into account to optimize the video navigation experience. Lipski et al. [5] have worked to accurately apply the interpolated results for this matter in their article "Virtual Video Camera: Image-Based Viewpoint Navigation Through Space and Time". They introduce a concept known as *spacetime tetrahedralization* to explain the mechanism for the synthesizing of the video effects. It is made clear that the process is not only a matter of uniformly inserting the interpolated im-

ages in the sequence. Also, the article "High Resolution Image Correspondences for Video Post-Production" citeLipski10hires is an interesting contribution for the optimization of the correspondences when working with high resolution images. As I stated before, an accurate edglet matching is the fundamental link to the whole algorithm.

Although totally plausible interpolated images have not yet been achieved, and that still, from a perspective point of view, real images cannot be fully replaced at this phase of the developed algorithm, this work has shown that the use of image processing tools as an alternative for the deployment of extremely sophisticated video camera infrastructures to generate video navigation experiences is feasible. And this leads to drastically lower costs.

I encourage future students to give continuity to this project.

References

[1] J. R. Casas, F. Marqués, and P. Salembier, "Handouts from the image and video processing course." MERIT master, Signal Theory and Communications Department, UPC, Fall semester 2010.

[2] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 509 –522, apr 2002.

[3] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, (New York, NY, USA), pp. 417–424, ACM Press/Addison-Wesley Publishing Co., 2000.

[4] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor, "Perception-motivated interpolation of image sequences," *ACM Transactions on Applied Perception (TAP)*, vol. 8, no. 2, pp. 1–25, 2011. http://doi.acm.org/10.1145/1870076.1870079.

[5] C. Lipski, C. Linz, K. Berger, A. Sellent, and M. Magnor, "Virtual video camera: Image-based viewpoint navigation through space and time," *Computer Graphics Forum*, vol. 29, no. 8, pp. 2555–2568, 2010.

[6] C. Lipski, C. Linz, T. Neumann, M. Wacker, and M. Magnor, "High resolution image correspondences for video post-production," in *Proc. European Conference on Visual Media Production (CVMP) 2010*, vol. 7, (Los Alamitos, CA, USA), pp. 33–39, IEEE Computer Society, Nov. 2010. http://doi.ieeecomputersociety.org/10.1109/CVMP.2010.12.

[7] T. Beier and S. Neely, "Feature-based image metamorphosis," *SIGGRAPH Comput. Graph.*, vol. 26, pp. 35–42, July 1992.

[8] P. Vangorp, J. Laurijssen, and P. Dutré, "The influence of shape on the perception of material reflectance," *ACM Trans. Graph.*, vol. 26, July 2007.

[9] G. Ramanarayanan, K. Bala, and J. A. Ferwerda, "Perception of complex aggregates," *ACM Trans. Graph.*, vol. 27, pp. 60:1–60:10, August 2008.

[10] M. Ruzon and C. Tomasi, "Color edge detection with the compass operator," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2, pp. 2 vol. (xxiii+637+663), 1999.

[11] P. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, pp. 167–181, 2004. 10.1023/B:VISI.0000022288.19776.77.

[12] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[13] D. P. Bertsekas, "Auction algorithms for network flow problems: A tutorial introduction," *Computational Optimization and Applications*, vol. 1, pp. 7–66, 1992.

[14] M. Zuliani, "Ransac for dummies." 2011.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[16] F. Klose, K. Ruhl, C. Lipski, and M. Magnor, "Flowlab - an interactive tool for editing dense image correspondences," in *Proc. European Conference on Visual Media Production (CVMP) 2011*, Aug. 2011.