University of Cagliari

PhD School of Mathematics
and Scientific Computing

Perception and Motion

Use of Computer Vision to solve Geometry Processing problems

Author:
Stefano Marras

Supervisor:
Prof. Riccardo Scateni

Abstract

Computer vision and geometry processing are often see as two different and, in a certain sense, distant fields: the first one works on two-dimensional data, while the other needs three dimensional information. But are 2D and 3D data really disconnected?
Think about the human vision: each eye captures patterns of light, that are then used by the brain in order to reconstruct the perception of the observed scene. In a similar way, if the eye detects a variation in the patterns of light, we are able to understand that the scene is not static; therefore, we're able to perceive the *motion* of one or more object in the scene.
In this work, we'll show how the perception of the 2D motion can be used in order to solve two significant problems, both dealing with three-dimensional data. In the first part, we'll show how the so-called *optical flow*, representing the observed motion, can be used to estimate the alignment error of a set of digital cameras looking to the same object. In the second part, we'll see how the detected 2D motion of an object can be used to better understand its underlying geometric structure by means of detecting its *rigid parts* and the way they are connected.

Contents

Introduction **iii**

1 Background **1**
1.1 The acquisition of three dimensional shapes 1
1.2 The Bundle Adjustment problem 5
1.2.1 The Camera Model 6
1.2.2 Error Modeling 8
1.2.3 Numerical optimization 11
1.2.4 Related works 12
1.3 The computation of the Optical Flow 14
1.3.1 Definitions 14
1.3.2 Local methods 17
1.3.3 Global methods 27

2 Bundle adjustment via Optical Flow computation **35**
2.1 Overview 36
2.2 Implementation 40
2.3 Estimating the Egomotion 42
2.4 Preliminary results 46
2.5 Limitations and known issues 47
2.6 Improvements and future works 51

3 Motion-based mesh segmentation **55**
3.1 The mesh segmentation problem 55
3.2 Motion-based Mesh Segmentation 56
3.3 Algorithm 60
3.3.1 Augmented Silhouette Extraction 61
3.3.2 1D analysis 64
3.3.3 Mesh Partitioning 66
3.4 GPU parallelization 72
3.5 Experimental Results 73
3.6 Limitations 74

4 Conclusions **77**
4.1 Bundle Adjustment 77
4.2 Motion-based mesh segmentation 78
4.3 Final Remarks 79

A Controlled and adaptive mesh zippering **81**
A.1 Motivation 81
A.2 Background 81
A.3 Algorithm overview 82
A.3.1 Border Erosion 83
A.3.2 Clipping 84
A.3.3 Cleaning 87
A.4 Experimental results 87
A.5 Final remarks 88

Bibliography **93**

Introduction

The expression *Computer Graphics* is usually referred to the process of creating and manipulating images using the computer. Even if computer graphics exists since 1960s, only in the last two decades it has become a significant field of study and research, thanks to applications such as medical aid, automatic surveillance, robotics, but also video-games and movies. However, the expression Computer Graphics doesn't fit very well the different types of applications and purposes of this field.
We can distinguish between two different main areas:

- **Computer Vision**
- **Geometry Processing**

In this work, we'll show that computer vision and geometry processing may not be so distant as one can think. In the first part, we will study if a common computer vision technique (namely the *optical flow*), that uses the perceived motion of the objects in a scene, is suitable to solve a typical geometry processing task such as the camera bundle adjustment, while in the second part, we'll show how the *perception of the motion* can be used to perform a geometry processing task such as the *rigid-part segmentation* of a deforming shape.

Computer Vision has the aim to estimate properties, geometric and dynamic, of the 3-D world from one or more digital images [168]. Using a less formal and more poetic definition, Computer Vision makes the computer able to *see*. Of course, we are not referring to the physical device that handles the acquisition of the images (the camera); instead, we are referring to the ability of the machine to correctly *understand* the image that the computer has acquired, and eventually use this information in order to take some action. For example, a robot, equipped with one or more small cameras, can navigate through a room, avoiding obstacles, using only the information of the surrounding environment or even in an open space [24, 68, 99, 130, 153, 161, 166, 184], using information obtained by the camera or combining them with other information obtained by different devices. Most of these works try to emulate the vision of the insects, particularly of bees, which is probably the simplest biological vision system. Other fields of computer vision try to emulate the human vision, in order to accomplish different goals, as for the reconstruction of an object from

a series of images, or the detection of a specific person in a crowd, and so on.

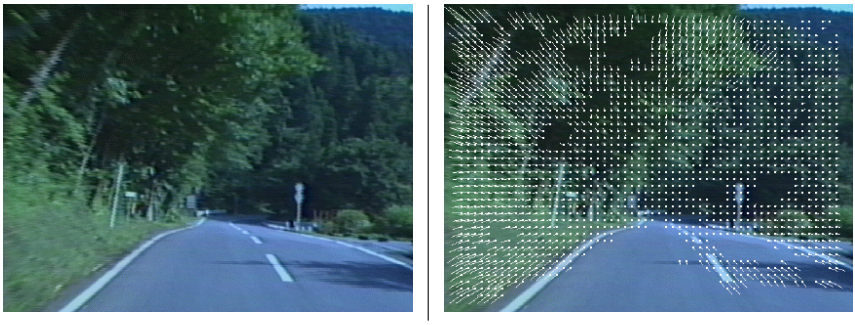

Figure 1: An example of computer vision application: optical flow for road navigation.

Digital Geometry Processing is a relatively new field of computer science, related to both graphics and geometry, focused on the efficient analysis and manipulation of three dimensional geometric data [27]. Typical operations include surface reconstruction from point samples, filtering operations for noise removal, geometry analysis, shape simplification, geometric modeling, interactive design or even understanding the *underlying structure* of the object itself [27, 97, 105, 117, 169]. All those applications and techniques have in common the fact that they work almost exclusively on the geometric information encoded in the shape, such as the position of the points in the space, or the normals of the points, without taking into account other information, such as color information or motion. The reason behind the importance of geometry processing is the fact that digital objects (so-called *polygon meshes*) have become increasingly popular in recent years, and are nowadays used intensively in many areas of computer graphics (computer-aided geometric design, computer games, movie production). The geometry processing techniques overlap, in some way, the **shape analysis** field, that is, the field aiming to analyze geometric shapes, usually performing some statistical analysis on the shape itself [151, 156].
Those two fields, apparently different, may work together in order to solve several problems. This is the case of the 3D reconstruction using stereo vision from images as described in [121] and [102]: given a pair of images representing the same object from two slightly different points of view, one can use the correspondence of the significant points in the two images in order to infer the position of the point in the three dimensional space by triangulation; the results can be improved adding information about the position of the cameras where the images have been taken. The *stereo reconstruction* is one of the few and classic example of synergy between these two fields, but there are also different problems, such as *image-to-geometry registration*, or *shape from motion segmentation*, that are usually solved in other ways, can be successfully solved combining techniques and algorithms from both computer vision and geometry

processing, as it will be shown in the remaining of this work. Before continuing, we need to spend some more words about the idea of *shape and motion*, which links all the section of the work here presented, and about the *cultural heritage*, which is the field of interested where this work should be placed.

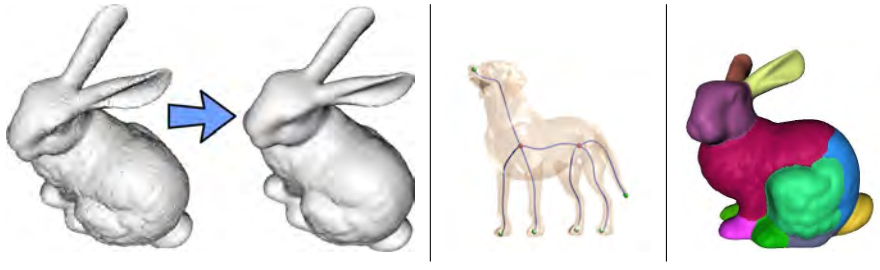

Figure 2: An example of geometry processing application (mesh denoising, on the left) and two examples of shape analysis (skeletonization in the middle, segmentation on the right). Both fields require to manipulate the three-dimensional data representing a shape.

Shape and motion

When a human being observes a scene, the light from the environment hits its retina and generates a number of signals that the human brain translate to a retinal image. When there's a motion between the observer and the scene, the moving pattern of lights falls upon the retina, creating the *perception of the motion* of the scene. The same principle holds for synthetic non-human devices, such as cameras and video-cameras: the three dimensional scene is captured and discretized as a two-dimensional image, and the motion of the scene is captured by a number of different images. The easiest way to perceive the motion of a scene is to compare two or more images of the same scene, taken in different times, in order to capture the *absolute* motion of the scene, or from different viewpoints, in order to capture the *relative* motion between the observer and the scene. The disparity between images can be represented as a vector field, describing the motion of the components of the scene, and can provide a number of information about the structure of both scene and motion. Computing how the image changes when the observer moves in a given manner relative to the scene of a specified geometry is not a big deal, but the inverse problem of inferring the structure of both the scene and the motion from the apparent motion of the image is not a trivial task. In the next chapter, the theoretical foundations of the optical flow will be introduced and developed, and it will be clear how information derived from apparent motion can be used to reconstruct motion and shape structure.

Cultural Heritage

The field of *digital cultural heritage* is one of the applications of Computer Graphics more specifically, it's the branch of the Computer Graphics that deals with the *acquisition, processing and visualization of human artifacts having some kind of historical and cultural value*; see [66]. Typically, the aim of DCH process is to obtain a digital three-dimensional model of the artifact, that has to be as more detailed and accurate as possible, in order to allow to the user to perform reliable analysis and measurement. The definition of digital models includes a large number of artifacts, from Greek vases, to Pompei's graffiti [13] to the statues of the Renaissance [50] [150], but also a number of large-scale models, such as the digital representation of churches, palaces, or even entire artistic sites (see [39], [41], [22], [62], [14], [42] for examples). The possible applications of DCH are the automatic classification of an artifact (see [61]), its virtual inspection and restoration, that can be a useful aid for the real restoration process (such as [43], [17], [63], [40]), or the creation of large databases of artistic objects (such as [56]), that can be used to create virtual museums [48]. Part of this work has been developed in cooperation with the Visual Computing Lab, one of the state-of-the-art laboratory in this particular field of research.

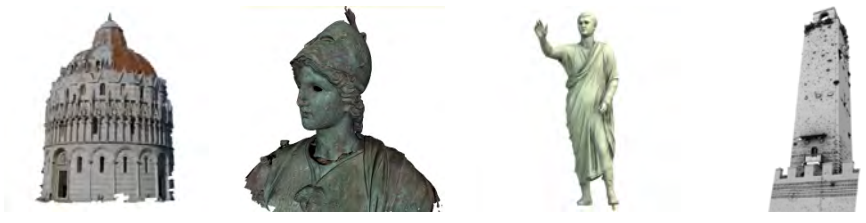

Figure 3: Some examples of applications of Cultural Heritage.

Chapter 1

Background

The goal of this work is to use techniques from Computer Vision in order to solve (or, at least, try to solve) problems that are strictly related to the field of Geometry Processing and Shape analysis. Particularly, this work will focus on two different problems:

- **Image-to-geometry registration**
- **Shape segmentation from motion**

In order to better understand the problems of the image-to-geometry registration, we propose, in the first section of this chapter, a brief survey of the techniques for the acquisition of three dimensional shapes; then, in the remaining part of this chapter, we will focus with more detail on the problem of the registration; finally, we will analyze the Computer Vision techniques used for solving these problems, that is, the *optical flow*. An introduction to the generic shape segmentation problem is instead presented in chapter 3.

1.1 The acquisition of three dimensional shapes

The acquisition of three dimensional shapes is the process that aims to create a digital version of a real, existing object. It differs from other techniques like CAD modeling or procedural modeling in the sense that the digital shape is creating starting from real data (the existing object) and not from scratch (using some CAD software for example) or from a mathematical description of the shape. This process of acquisition is carried out by a device called **3D scanner**; 3D scanner is a device that can acquire information about the spatial position of a three dimensional point (see Figure 1.1). Even if exist some scanner that actually touch the object in order to collect these data, most of the 3D scanners obtain the information about the points without any actual contact with the object; this technique is generally cheaper, faster and more efficient that using a contact scanner, so in this work we'll focus on the issues

Figure 1.1: 3D scanner devices.

related to the use of non-contact 3D scanners.

The first step of the pipeline of the digitalization of a real three dimensional object is the **acquisition of the position of the object's surface points**. This first step has the aim of create a number of array of distance values, usually called **range images** or **range maps**; each range map can be seen as a $m \times n$ grid of distances (range points) that describes a surface in such a way that each point has three spatial component, two defined by the indices of the point in the grid, and the third value which is the depth associated with that specific element. The way to obtain a single range map may vary, depending on the algorithm or device used in the process. One of the most used and common technique uses the so-called *time-of-flight* scanners: the device emits pulse of light, then measures the time needed for the light to be reflected by the surface of the object and then be caught by a special detector; knowing the speed of light and the round-trip time, the device is able to compute the distance of a number of surface points of the object. Usually, those devices use infrared light or laser LIDAR (Light Detection and Ranging), depending on the size and the distance of the object. Alternatively, it's possible to create a depth-value map by *passive stereo matching*: two cameras, placed in different position, see the object from two different viewpoints; if we are able to identify precisely the same point on the different images obtained by the two cameras, then it's possible to compute the exact position of the point simply using the information related to the position of the cameras (baseline and angles); in order to create a dense range map, we need to identify correctly a large number of corresponding point, which leads to several problems (noise, occlusion, different light conditions and so on). This technique tends to be too much prone to errors, but it can be improved using the *active stereo matching*: in this case, instead of having two cameras observing the object, we will have one fixed camera and a special light source emitting several black and white pattern on the object surface over time; identifying the patterns, it's possible to compute the distance associate with a specific pixel of the image. In this way, known as *structured light* scanning, we can improve the quality of the result in terms of number of samples, precision running time of the process. Similarly,

instead of using the structured light, we can use a *laser scanning* technique: a device project a laser pattern onto the object, and the camera compute the depth of the points using the ray-plane intersection. It's a common thought that structured light and laser scanners produce the best results. All those techniques produce a single depth map; in order to create a digital model from the real object, we need to create a number of depth maps, capturing depth information all around the object, in a way that is not so different from taking pictures of the entire surface of the object: we place the scanner in several points, and collect related depth data (see Figure 1.2 for an example of several range maps). Usually, the earliest stage of the acquisition pipeline plans how to place the scanners or cameras around the object, in order to cover the entire surface, avoiding over-sampling (which can lead to noise) or under-sampling (which can result in hole on the surface). However, objects with peculiar properties of the surface (for example, object made by glass) or of the geometry (object with concavities) are often difficult to acquire and digitalize.
After the completion of the acquisition step, what we have is a number of different range maps; in order to reconstruct the object, these range maps need to be **aligned** in a common reference system, in order to create a single point cloud that can be thought as a first approximation of the shape of the object. The alignment is not a trivial task: of course, if one knows in advance the correspondences between points in the different range maps, it's straightforward to compute the relative roto-translation that brings the maps to be aligned, but usually those correspondences are not known. In order to solve the problem, some software asks the user to manually set the correspondences between points in different range maps, and then, starting from at least four correspondences, the roto-translation is computed. Also, automatic techniques has been developed; particularly, one of the most used is the **Iterative Closest Point** (**ICP**) algorithm, developed by Besl and McKay [21]: starting from a sub-sample of points of one range maps, find the closest points on the second one, and then compute the rotation and translation that minimize the distance between the maps. The algorithm proceed iteratively until no significant improvement has been made. The techniques of Besl and McKay has been improved since 1992, as can be seen in [147], [152] and [80]; however, the main idea behind the work still stands. Also this approach can be extended to an arbitrary number of range maps, solving a global minimization problem, in order to perform a simultaneous alignment of all the maps instead of working locally on single pairs, with the significant improvement of spreading the alignment error all over the maps, avoiding to accumulate it.

The third and final step of the digitalization process has the goal of creating an approximation of the surface from the point cloud created in the previous step (Figure 1.2). This problem can be addressed in several ways, depending on the desired results and the used technique. First of all, we need to distinguish between *explicit* reconstruction methods, that use directly the data encoded in the scanned points, and *implicit* reconstruction methods, that use the scanned data as a guidance for creating the surface of the digital shape. In the first case,

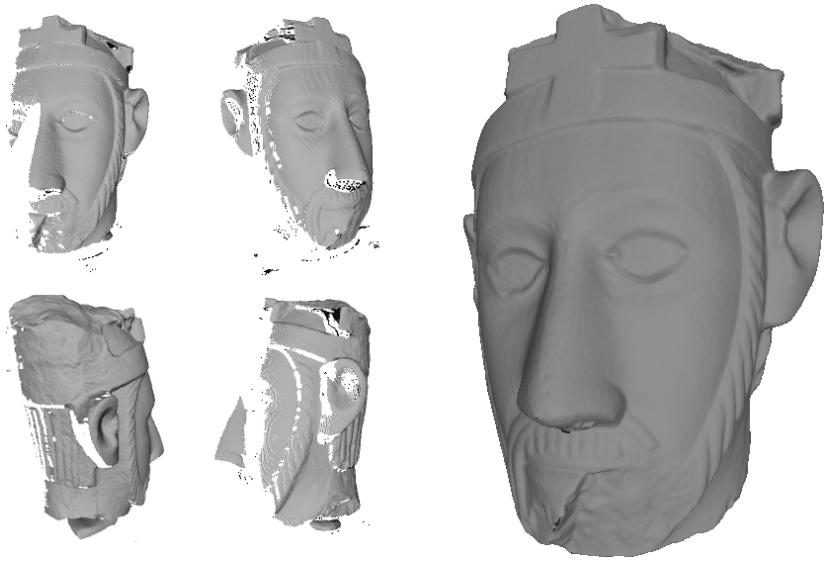

Figure 1.2: Four different range maps, obtained scanning a real object, and the digital shape obtained by their merge.

the points of each range map are used to create a polygonal (typical triangular) approximation of the surface, using algorithms like the Ball-Pivoting [20], the α shape [69], the active snake [177] or the Delaunay triangulation; then, the different approximated surfaces are merged into a single object, using the alignment information obtained in the previous step. This process is called *zippering* and has the advantage of use the real scanned data, allowing to obtain a realistic and detailed model, preserve the regular structure of the range maps, and doesn't need particular auxiliary data structure. We refer to the appendix A for more details on the final stage of the acquisition pipeline and particularly on the zippering algorithm. Unfortunately, it's a process subject to a number of problem, especially if the data are noisy or incomplete. On the other hand, implicit reconstruction methods are more robust to noise, but doesn't guarantee a real faithfulness to the real object: small details can be lost, holes may be filled and so on. The implicit reconstruction techniques uses the entire point cloud to create an approximation of the object surface, usually minimizing some distance function in order to obtain a surface that is as closest as possible to the scanned points. Probably, the most used algorithm is the well-known Marching Cube technique [117] developed by Lorensen and Cline in 1987, then improved during years (see for example [45,97,132]), which produces a 2-manifold discrete surface, but is computationally expensive, and, as already pointed out, there is no guarantee that the result will be the desired one (it can result in over-tessellation or in highly irregular meshes). Other methods work directly on the ranges maps (e.g. the volumetric approach proposed by Curless and Levoy [59] or the Poisson surface reconstruction [105]) but suffer

of the same problems of the Marching Cubes algorithm.
As can be seen throughout the section, the acquisition pipeline presents several problems, and usually the final result needs to be refined with some post-processing phase, such as de-noising or remeshing, in order to create a visually nice and useful shape. At the moment, a fully automatic acquisition process is impossible to achieve, since the user is usually involved in the first two steps of the process.

1.2 The Bundle Adjustment problem

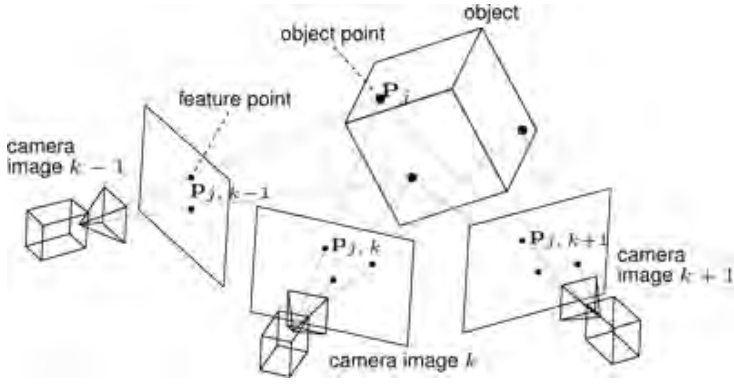

Figure 1.3: An object, characterized by several points $\mathbf{P}_i$, is seen by different cameras. The Bundle adjustment problem has the aim to determine the position of each camera by means of the correspondences between feature points $\mathbf{p}_{i,k}$. See also Figure 2.1.

The acquisition process is able to capture only **geometric information** of the object. To create a realistic and detailed digital version of a real three dimensional object, we usually need to attach to the geometry also information related to the visual appearance of the object. Assuming that we want to create a digital model of some particular ancient Greek vase (Figure 1.2: the geometric information is not enough, because the most of the artistic and cultural value of the object comes from the painting on the surface of the vase itself.

In order to collect information about color and texture, the user has to take a number of pictures of the real object, in order to capture the entire surface, and then project the color information from the images to the surface of the digital object. Obviously, to achieve a good result, we need to know the exact position where the picture has been taken, which is an information usually hard to retrieve. In literature, this process is referred as the **bundle adjustment** problem. More formally, the bundle adjustment is the problem of refining a visual reconstruction to produce both optimal 3D structure and viewing parameter (*camera pose* and *camera calibration*) estimates [167]. In our case, since we already have the 3D structure obtained by the scanning

Figure 1.4: The same object (in this case, a vase) captured from several points of view, and a digital representation encoding only the geometry on the object. In order to enhance the quality of the digital model, we need to add the color and texture information to the geometry.

process, we focus on the optimal estimation of the viewing parameter; *optimal* means that the camera parameters estimates are found by minimizing some cost function that quantifies the model fitting error. The name *bundle* refers to the bundles of light rays that leave the 3D features of the image and then converge on each camera center, which are *adjusted* optimally with respect to image features and camera position.

1.2.1 The Camera Model

Before discussing in detail the bundle adjustment problem formulation, we need to spend some words about the geometric aspect of the image formation and the camera parameters involved in the process. We refer to these information as the **camera model**. Here we will give a brief overview, with focus on the elements of interest, and refer to [168] for a detailed description.

In the following, we refer to the image 1.5 in order to describe the camera model. The most common geometric model of a camera is called **perspective** or **pinhole** model: it basically consists of a plane π, called *image plane*, and a 3D point $\mathbf{O}$ called *center* or *focus of projection*. The distance between π and $\mathbf{O}$ is the *focal length* f. The line passing by $\mathbf{O}$ perpendicular to π is the *optical axis*; its intercept with π is the *image center* (point $\mathbf{O}_1$). As shown in the image, the camera has its own reference system, called *camera frame*, centered in $\mathbf{O}$ and having the optical axis as Z-axis. Let a point $\mathbf{P}(X, T, Z)$ in the camera frame; its projection $\mathbf{p}_1(x, y)$ on the image plane is computed as follows:

$$x = f\frac{X}{Z},\ y = f\frac{Y}{Z}. \tag{1.1}$$

Aside from the perspective camera model, we need to introduce other kinds of information, allowing to connect the camera position with the *world reference frame*, and to translate the coordinates of an image plane point to *pixel coordinates*, which are the only coordinates directly available from the image. In vision, those data are referred as *extrinsic* and *intrinsic* parameters:

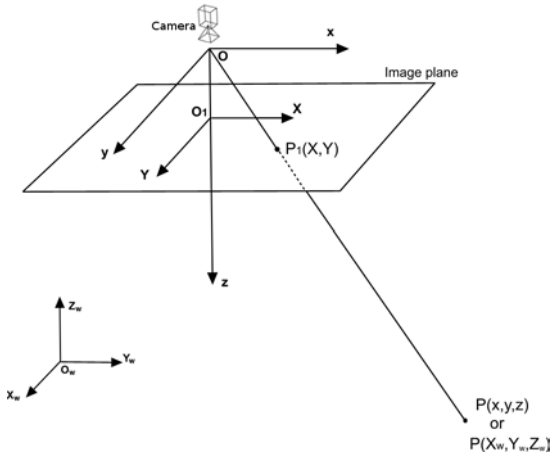

Figure 1.5: The perspective camera model

Extrinsic parameters define the location and the orientation of the camera reference frame with respect to a know world reference frame.

Intrinsic parameters link the pixel coordinates of an image point with the corresponding coordinates in the camera reference frame.

We previously mentioned that the camera has its own reference frame; however, this frame is usually unknown, and a common problem is determining the location and the orientation of the camera frame with respect to the world reference frame; often, this goal has to be achieved using only image information. We define the extrinsic parameters as any set of geometric parameters that identify uniquely the transformation between the camera reference frame and the known world reference frame. Typically, the transformation between the two reference frames is described as a combination of a 3D vector $\mathbf{T}$, describing the relative positions of the origins of the two reference frames, and an orthogonal 3×3 matrix $\mathbf{R}$ that encodes the *rotation* that brings the corresponding axes of the two frames onto each other. The relation between the coordinates of a certain point $\mathbf{P}$ expressed in world and camera frame by $\mathbf{P}_w$ and $\mathbf{P}_c$ respectively, can be written as

$$\mathbf{P}_c = \mathbf{R}(\mathbf{P}_w - \mathbf{T}). \tag{1.2}$$

The intrinsic parameters are the set of parameters that characterize the *optical*, *geometric* and *digital* characteristics of the camera. Particularly, we need to know the focal length f, which encodes the information related to the perspective projection, the image center (o_x, o_y) and the pixel size, (s_x, s_y), expressed in millimeters, which encodes the information related to the transformation between camera frame and image space. For the sake of completeness, in some

cases also the radial distortions of the lens, k_1 and k_2, are considered as an intrinsic parameter, but in most works they may be simply ignored. A point on the image plane $p \equiv (x, y)$ can be thought as the point $p \equiv (x, y, f)$ in the camera reference frame; this point corresponds to a certain pixel $p_i \equiv (x_i m, y_i m)$ of the image I. The transformation between camera and image frame coordinates is described in the following

$$x = -(x_i m - o_x)s_x, \; y = -(y_i m - o_y)s_y. \tag{1.3}$$

The estimation of the parameters of a single camera, usually referred as the **camera calibration problem**, is a well-established procedure, that can be trivially accomplished by means of some predefined calibration pattern (usually a chessboard) that allows to exactly determine a number of correspondences between the real world known coordinates of the pattern, and the coordinates of the camera frame; from these correspondences, a linear system is build and solved in order to find both extrinsic and intrinsic parameters (see [168] for a more detailed description of the problem and its solution).
A different and more complex situation arises when we need to calibrate a number of cameras that are observing the same scene; in this case, we need to estimate the position of all the cameras with respect to the common world frame, and, consequently, with respect to all the other cameras. After that we correctly establish the position of all the cameras, the information encoded in the images can be used to reconstruct the geometry of the scene, or to enhance the digital shape projecting onto its surface color and texture information. As we stated in the very beginning, this process of the parameters estimation of all the cameras capturing a scene is usually referred as the **bundle adjustment** problem.

1.2.2 Error Modeling

The bundle adjustment procedure usually require to know in advance a number of correspondences between pixels in different images (e.g.: two pixels representing the same 3D point). These correspondences may be manually selected by the user that specifies the corresponding pixels, or may be computed using some feature extraction and matching algorithm (see, for examples, [15, 55, 129]); in the following, we assume that these features are already extracted and matched in a number suitable for our purposes. Figure 1.6 show an example of feature matching between two images

Assume that a three dimensional scene is modeled by individual static 3D features $\mathbf{X}_p$, $p = 1...n$, imaged in m shots with ; basically, we are assuming that we know, for a number of real 3D points pictured in the images, the corresponding pixels in m images. Also, let $\mathbf{P}_i$, $i = 1...m$ be the camera poses and internal calibration parameters (one set for each camera), and let $\mathbf{C}_c$, $c = 1...k$ be the calibration parameters constant across several images. As a matter of fact, the number of cameras is usually lower than the number of images, and the intrinsic parameters don't vary, so we can assume that there's

Figure 1.6: Feature matching between two of images of Figure 1.2. Each green line connect a pixel in the left image with the corresponding pixel on the right image. The feature extraction and matching algorithm is provided by the OpenCV library [28].

a set of parameters that can be considered constant. We are given uncertain measurements $\overline{x}_{ip}$ of some subset of the possible image features x_{ip}, which is the real image feature $\mathbf{X}_p$ in image i. For each observation $\overline{x}_{ip}$ we assume to have a predictive model $x_{ip} = x(\mathbf{C}_c, \mathbf{P}_i, \mathbf{X}_p)$ that tells us where the $\mathbf{X}_p$ feature will be located in the image taken from a camera having $\mathbf{C}_c$ and $\mathbf{P}_i$ as parameters. We can then derive a feature prediction error:

$$\Delta x_{ip} = \overline{x}_{ip} - x(\mathbf{C}_c, \mathbf{P}_i, \mathbf{X}_p). \tag{1.4}$$

The equation 1.4 represents the measurement error between the real observed image feature and the predicted one. The obvious goal is find the parameters that minimize the error, that is, the camera parameters that best estimate the position of the feature points.

Usually, the process starts with some initial parameter approximate estimate, and then continues optimizing a non-linear *cost function*, representing the *total prediction error*, over a large non-linear parameter space. The estimates of both image features and camera parameters can be thought as a large *state vector* `x`, while the cost function is a generic `f(x)` depending from Δx_{ip}. The search for the optimum is usually an iterative process: at the i-th iteration, the estimate `x` gets updated of a quantity δ`x` in order to move towards the optimum value. Of course, the goodness of the solution will depend on the accuracy and correctness of the predictive model, and on the optimization

strategy chosen. In the remaining part of the chapter, some strategies will be presented and discussed, focusing more on the modeling part than on the numerical aspects; a detailed description of the numerical aspect is given in [167]. One of the most basic parameter estimation methods if the **nonlinear least squares**. Assume that we have vectors of observations $\overline{\mathtt{z}}_\mathtt{i}$ predicted by a model $\mathtt{z}_\mathtt{i} = \mathtt{z}_\mathtt{i}(\mathtt{x})$ where $\mathtt{x}$ is a vector of model parameters. Then, using the nonlinear least squares, we search for the parameter values that minimize the **weighted Sum of Squared Error** (SSE) cost function:

$$\mathtt{f}(\mathtt{x}) = \frac{1}{2}\sum_i \Delta\mathtt{z}_\mathtt{i}(\mathtt{x})^\intercal \mathtt{W}_\mathtt{i} \Delta\mathtt{z}_\mathtt{i}(\mathtt{x}), \quad \Delta\mathtt{z}_\mathtt{i}(\mathtt{x}) = \overline{\mathtt{z}}_\mathtt{i} - \mathtt{z}_\mathtt{i}(\mathtt{x}) \tag{1.5}$$

In 1.5, $\Delta\mathtt{z}_\mathtt{i}(\mathtt{x})$ represent the feature prediction error and $\mathtt{W}_\mathtt{i}$ is the weight matrix (symmetric, positive definite). The main advantages of this method are that optimization methods based on LS solvers allow to stably handle also ill-conditioned problems, and its insensitivity to the natural boundaries between observations; the drawback is a high sensitivity to outliers, which makes this basic method unreliable and often unsuitable.

For a more robust estimates, we need to choose a different model such that even if one (or more) of the observation is affected by some measurement error (e.g. mismatching correspondences), the global estimation won't. The basic idea is that, if a observation introduces some noise, it can be down-weighted, or even removed from the process of estimation of the parameters; similarly, if a group of observation introduces noise, then all the observations of the group should be treated in the same way, in order to preserve data that are correct. We can model this behavior modifying the original least squares formulation in order to write the error using a **robustified least squares** approach. The observations are partitioned into *independent groups* $\overline{\mathtt{z}}_\mathtt{i}$, where independent means that each group can be down-weighted or even rejected independently of all the other groups. The way we should partition the observation into different groups depends from the type of errors that affects the observations (wrong or missing feature correspondences between images, for example). Each group of observation contributes an independent term $\mathtt{f}_\mathtt{i}(\mathtt{x}|\overline{\mathtt{z}}_\mathtt{i})$ to the total cost function; each term is modeled in a way depending on the expected distribution of inliers and outliers of the group of observations. The error is now modeling as

$$\mathtt{f}(\mathtt{x}) = \frac{1}{2}\sum_\mathtt{i} \rho_\mathtt{i}(\Delta\mathtt{z}_\mathtt{i}(\mathtt{x})^\intercal \mathtt{W}_\mathtt{i} \Delta\mathtt{z}_\mathtt{i}(\mathtt{x})), \quad \Delta\mathtt{z}_\mathtt{i}(\mathtt{x}) = \overline{\mathtt{z}}_\mathtt{i} - \mathtt{z}_\mathtt{i}(\mathtt{x}) \tag{1.6}$$

Here, $\rho_i(s)$ is a weighting function, that can be linear, sub-linear and so on, depending on how one wants to model the behavior of the function. Obviously, if $\rho_i(s) = s$, then the function becomes the common weighted SSE; ideally, $\rho_i(s)$ should be defined such that $\rho_i(s) = \inf$ for the outliers. In [167] Triggs suggests to use radial distributions to model the independent terms associated with each group of observations. Both weighted SSE and robustified LS can

be used to model the errors related to geometric image features (point-to-point correspondence between images), but they are also suitable for modeling the intensity-based image patches matching.

1.2.3 Numerical optimization

Finally, after choosing a suitable quality metric modeled in some way, we need to optimize it, minimizing (or maximizing, depending on the model). In this section we'll have a brief discussion about the suitable methods to perform the optimization, without entering into details since it's not the purpose of this work, and also since, in our final approach, described in the following chapter, we choose to don't use any numerical optimization, applying instead a more *heuristic* approach.
The basic problem, as previously stated, is to find the parameters $\mathtt{x}$ that minimize the error function $\mathtt{f}(\mathtt{x})$. Starting from an initial estimate $\mathtt{x}$, we need to find the displacement $\delta\mathtt{x}$, represented as a vector in the space of the parameters $\mathtt{x}$, such that $\mathtt{x}+\delta\mathtt{x}$ locally minimize (or at least, reduces) the cost function $\mathtt{f}$. Usually, the function cost is approximated using a local model for the function (e.g. its Taylor expansion), and then the minimum of the approximation is found; even if we have no guarantee that the minimum of the approximation will also be the minimum of $\mathtt{f}$, in most cases this process will lead to the desired result.
The basic approach, as we mentioned before, uses the Taylor expansion of the function:

$$\mathtt{f}(\mathtt{x}+\delta\mathtt{x}) \approx \mathtt{f}(\mathtt{x}) + \mathtt{g}^\mathtt{T}\delta\mathtt{x} + \frac{1}{2}\delta\mathtt{x}^\mathtt{T}\mathtt{H}\delta\mathtt{x} \tag{1.7}$$

where $\mathtt{g}$ is the gradient vector defined as $\mathtt{g} \equiv \frac{\mathtt{df}}{\mathtt{dx}}(\mathtt{x})$ and H is the Hessian matrix $\mathtt{H} \equiv \frac{\mathtt{d}^2\mathtt{f}}{\mathtt{dx}^2}(\mathtt{x})$, that we can assume to be positive definite. The model has a unique global minimum, that can be found by means of the *Newton's method*. First, we define the Newton step as $\delta\mathtt{x} = -\mathtt{H}^{-1}\mathtt{g}$, then we estimated the new function value as $\mathtt{f}(\mathtt{x}+\delta\mathtt{x}) \approx \mathtt{f}(\mathtt{x}) - \frac{1}{2}\delta\mathtt{x}^\mathtt{T}\mathtt{H}\delta = \mathtt{f}(\mathtt{x}) - \frac{1}{2}\mathtt{g}^\mathtt{T}\mathtt{H}^{-1}\mathtt{g}$; iterating, the method converges to the optimum of the function $\mathtt{f}$, with quadratic asymptotic convergence. This conceptually easy method has actually some drawbacks: large computational cost in solving the Newton step equations, that takes $\mathcal{O}(n^3)$ for a dense $n \times n$ system, the non-trivial computation of the Hessian matrix for non-trivial cost function, possible failures (function converging to saddle point, inaccurate prediction for large displacements). We can enforce the method by forcing the Newton step to follow a descent direction that significantly reduces the value of the function, introducing some form of step control, a strategy adopted by the damped Newton methods, the trust region methods and the Levenberg-Marquardt methods.
When the error is modeled using a weighted SSE, as seen before, then the gradient and the Hessian can be expressed in terms of its Jacobian matrix $\mathtt{J} \equiv \frac{\mathtt{dz}}{\mathtt{dx}}$:

$$g \equiv \frac{df}{dx} = \Delta z^T W J, \quad H \equiv \frac{d^2 f}{dx^2} = J^T W J + \sum_i (\Delta z^T W)_i \frac{d^2 z_i}{dx^2} \tag{1.8}$$

The second term of the Hessian can be dropped, in order to obtain the Gauss-Newton approximation, that leads to the Gauss-Newton equation:

$$(J^T W J)\delta x = -J^T W \delta z \tag{1.9}$$

This method is particularly common when dealing with nonlinear least squares; it solves most of the issues of the Newton method for well-parameterized bundle problems, and it's also very accurate. However, since in most cases we have to deal with outliers, noise and incorrect data, we saw before that the SSE needs to be robustified introducing a ρ function; in the same way, the derivative of ρ must be introduced in the gradient and in the Hessian definition, in order to obtain a robustified Gauss-Newton approximation. Obviously, we can add some constraints to the formulation of the problem, if seeded; in this case, the problem becomes a constrained minimization problem: find the minimum of $f(x)$ under the constraints $c(x)$. The formulation of both the problem and the solution can be revised using the sequential quadratic programming technique, or using the linearized constraints to eliminate some of the variables of the problem, obtaining an unconstrained reduced problem to optimize easily.
Minimizing a cost function expressing the error of a bundle adjustment problem has also a number of issues directly related to the implementation. Probably, the first and most important issue is that most of the equations contains matrix that have to be inverted in order to compute the solution; this is a procedure that is numerically too expansive and has to be avoided as much as possible, preferring instead some particular decomposition (Cholesky, LU...) depending from the structure of the problem (e.g. a sparse matrix, or a triangular matrix and so on). The good choice can improve both speed and precision of the algorithm, while a bad choice can severely affect the result of the computation. Also the scale of the problem, and its preconditioning (the choice of which parameters to use) influence the performance of the algorithm, because the wrong scale can result, for example, in an ill-conditioned Hessian matrix.

1.2.4 Related works

The problem of the bundle adjustment was first tackled in photogrammetry and aerial cartography, in the late 50's, by Brown [31], that first used the definition of bundle adjustment. Initially the method used correspondences between photos captured by calibrated camera that lead to the construction of a linear system solved by Gaussian elimination. The original work has then been improved during years, changing the model of the camera (introducing distortion parameters, for example) and the calibration process, as seen in [32], [33] and [34]. Subsequently, the problem of the bundle adjustment has been also investigated in order to reconstruct 3D objects or 3D scenes from a sequence

of images (a number of pictures, for example or a video sequence, that lead to the so-called *structure from motion reconstruction*). Initially, a naive technique was a good choice: given a small set of images with manually extracted corresponding features, finding the position of the cameras required a reasonable amount of time. However, when the size of the dataset started to grow (more photos, or photos with higher resolution), the method become unsuitable, and other approaches have been developed, even if the idea behind most of them is basically the same, that is, defining some cost function expressing the error in the estimation of the parameters, and then minimize it in some way. The main differences between different work are usually the formulation of the function and the technique adopted to minimize it. A number of works tries to define the function in a way that its minimization is as cheap as possible: for example, [119], such as [167], explore the theoretical aspects of the problem, and propose one (or more, in the second case) efficient solution to it. [119] for example proposes the so-called Levenberg-Marquardt algorithm, while [118] proposes as alternative the dog leg algorithm; also [38] and [1] propose several improvement from a numerical and computational point of view; also [101] propose some improvements due to the use of the BLAS3 library.
Of course, there's also a number of other algorithms trying to revise the main approach. For example, [138] suggests a divide-and-conquer algorithm, optimizing small subset of data in parallel and then use these partial alignment to obtain the global optimization, in order to recreate a 3D scene and also use an out-of-core policy that allows to deal with large scale data. [158] introduce a hierarchical approach to the problem: the set of the images is divided into segments such that the feature points can be tracked across each segment; then, bundle adjustment is performed on each segment, in order to create a partial 3D reconstruction of the scene; finally, the resulting models are merged into a common reference frame. Here the novelty is the hierarchical approach: not a large problem derived from all the observations, but smaller problem (and, of course, easier to solve) and then the merging of the solutions. The possible drawback is a propagation of the error during the merging phase, that needs to be handled properly, as seen in [73]. A selective approach can be found in [133] where the 3D structure of a scene is extracted from a video sequence; in this case, each frame of the video is analyzed, and bundle adjustment is performed only on significant frames (*keyframes*) avoiding to add to the problem redundant data (while, for example, [70] perform a new optimization step for each frame of the sequence). [162] proposes instead an incremental approach: each image examined is added to an existing solution and the optimization step is performed adding to the solution only the not redundant information encoded in the new image. For the sake of completeness, we also mention the work by Steedly et al. [163] presenting an heuristic technique based on spectral analysis, the statistical modeling approach described in [72], and the work by Holmes, Murray et al. [96] that introduces a relative frame representation is introduced in spite of the global common coordinate system; bundle adjustment is then performed in this particular frame.
This work shares also some parts with the image-to-geometry registration tech-

nique. The image-geometry registration problem consists of determining the parameters of the camera model used to map the 3D model onto the image plane. The main difference with the bundle adjustment is that, in this particular case, we work on a single image (that is, a single camera and a single set of parameters). In [182], a survey of the issue is presented; other approaches are presented in [57, 64, 77, 113, 122].
In the following, we'll address the problem of the bundle adjustment from a different point of view, defining a new error function cost, and showing that the minimization may be achieved using a technique borrowed from the computer vision field, the computation of the so-called *optical flow*. Section 1.3 describes the formulation of the optical flow problem and presents several different approaches to its computation. Chapter 2 describes, in detail, how this technique can be used to solve the bundle adjustment problem.

1.3 The computation of the Optical Flow

The **Optical Flow computation** is one of the fundamental problem in Computer Vision. The **optical flow** is an approximation of the image motion, defined by Gibson [85] as *the moving patterns of light falling upon the retina*, and by Beauchemin [18] as *the projection of velocities of 3D surface points onto the image plane of a visual sensor*. Basically, given a sequence of two or more image, the optical flow is a convenient way to describe the motion of the objects of the pictured scene; each object describe a *trajectory* which is usually represented as a 2D vector on the image plane. This section is organized as follows: first, we will give a brief introduction to the problem, with particular focus on the classic definitions; then we will describe three of the techniques used for computing the optical flow, try to focus on their advantage and drawbacks. A comparison between the presented techniques, with special focus on our final goal, is proposed in chapter 2.

1.3.1 Definitions

In order to compute the optical flow we have to make some initial assumption, as stated first by Horn and Schunck [98]. Particularly, we assume that the intensity of local time-varying image regions is approximately constant under motion for at least a short duration. Formally, we can express this hypothesis as

$$I(x,t) \approx I(x+\delta x, t+\delta t) \tag{1.10}$$

where δx is the spatial displacement of the image region at (x,t) after time δt. The left-hand side of 1.10 can be expanded in a Taylor series, in order to obtain the following:

$$I(x,t) = I(x,t) + \nabla I \cdot \delta x + \delta t I_t + O^2 \tag{1.11}$$

where $\nabla I = (I_x, I_y)$ is the image spatial intensity gradient, and I_t is the temporal first order derivative. Starting from 1.11, removing the O^2, which represents

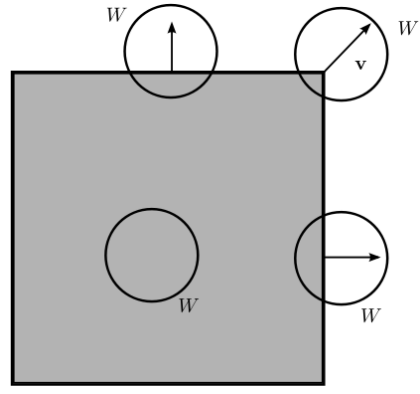

Figure 1.7: The aperture problem. Suppose that the gray square is moving in the **v** direction, but we observe its motion through the W window. Depending from the position of W, the estimation of the motion change significantly: if W is inside the square, no motion is detected, due to the poor texture; on the edges, we can estimate motion only in the gradient direction, while the motion can be estimated correctly only if W capture the moving corner.

the second and higher order terms and can be assumed as negligible, then subtracting $I(x,t)$ on both sides and finally dividing by δt, we obtain

$$\nabla I \cdot \mathbf{v} + I_t = 0 \tag{1.12}$$

where $v \equiv (u,v)$ is the image velocity. The equation 1.12 is usually known as the **optical flow constraint equation** and it's the starting point for a large number of solving techniques.
It's straightforward to notice that our goal is to find the components of the motion u and v, so we have two different unknowns but only one linear equation to find them: that means that the problem is *ill-posed*, and one can estimate only the motion component in the direction of the local gradient of the image intensity function. We refer to this issue as the *aperture problem* [172](Figure 1.7). Following the aperture problem, we can conclude that the optical flow can be estimated using the constraint equation only at locations where there is sufficient *intensity structure*, that is, we can estimate correctly the optical flow only in image regions with enough texture details. In order to estimate correctly and reliably the optical flow in a certain region of the image, we also need that other main conditions are satisfied: uniform illumination over the temporal sequence, Lambertian surface reflectance and pure translation parallel to the image plane. Typically, it's impossible to achieve all of these conditions, but the more we get closest to satisfy them, the better results we obtain. Obviously, *good results* means that the optical flow will be a good approximation of the image motion.

Anyway, there is a number of situations where it's impossible to compute a good optical flow. Regions with poor or no texture are basically useless, since

the spatial gradient is almost zero and doesn't contain any useful information; regions with highlights or shadows violates the constraint equation, and may not be handled correctly (eventually, the equation can be used as a first approximation, to be refined with other techniques). Other issues include occlusions, due to objects suddenly appearing in the sequence, and non conventional objects, such as transparent ones, which are difficult to handle and generally lead to incorrect solutions. Also, when computing the optical flow of a sequence of images, one could need to have a quality measure for the estimation of the flow; one of the problems is how to compute the reliability of the flow. Different confidence measure has been proposed; a very simply measure (which surprisingly is also quite robust and good) has been proposed in [159] and use the smallest eigenvalue of a least-squares matrix. Other quality measures have been proposed in [136], [145], [174], [76], [173], [8] and [2]. In the following part of this section, we will analyze some of the techniques used to compute the optical flow between two or more images, with special regards to the techniques that allow to compute a so-called *dense* flow, that is, a flow vector is computed for each pixel of the reference image; techniques based on feature-tracking, as the Kalman Filter [55, 175], won't be covered.
In literature, a huge number of different methods computing the optical flow of a sequence of images have been proposed; we briefly will give on overview of some general aspects of the different approaches before discuss some of them in details. Usually, the methods differ for several aspects, including the *strategy* used to compute the displacement (global approach vs local approach), the *information* used (intensity, local derivatives, frequency domain, statistical inference and so on) and the *scale* (single resolution vs hierarchical approach). The images, results and timing here presented are courtesy of the Middlebury evaluation dataset [12].
Global methods usually compute a dense flow minimizing some function defined over the entire image, while **local methods** works locally on single regions of the image or, in some cases, even on single pixels. Usually, global methods leads to smoother results, since they are more robust to noise in images; moreover, thanks to the possibility to add easily regulator terms to the objective function (constraints of smoothness or rigid-motion), it's generally easier to achieve the continuity of the motion. Unfortunately, global approaches have also major drawbacks: usually, it's impossible handle small displacement, since the global approach tends to privilege large shifts, so results may not be accurate; secondly, this type of technique is quite expensive in terms of time and space, and is usually not suitable for real-time applications; they may also experimented issues when working on high-definition images. On the other side, local methods are faster and cheaper to implement and to run, and are usually *highly parallelizable*, since the motion of one region is usually computed without any knowledge of the other motions in the image. Working on small regions allows to achieve a better precision (pixel or subpixel precision), but it has the disadvantage of making the result to be more sensitive to noise, especially when working on small sized regions of interest. Also, in some cases it's impossible to correctly estimate a dense flow using only local methods, since some regions

(as we already mentioned) may not have enough texture information. Hybrid approaches have been proposed in literature like in [35, 37], with good results. Also, there also other approaches that cannot be univoquely classified as local or global; for example, the so-called SIFT flow [112] based on the detection and matching of dense feature points, computed over the sequence of the image using the *Significant Invariant Feature Transform*, or the *Bayesian* Optical Flow [178], that uses *a-posteriori* probability to compute real-time accurate optical flow.
While in the early days of compute vision optical flow were computed using only one scale of resolution, as in [98], at the moment almost every method works in a **hierarchical** *coarse-to-fine* way, which is a smart way to approach and, in some cases, simplify the problem, at the cost of spending a little bit more time and space. In order to work in a hierarchical way, each original image is used as a starting point for a Gaussian pyramid composed by the same image re-sampled at different resolutions, then OF computation is performed on low resolution images. Displacement computed at a certain level is the used as initial estimate for performing the computation at a higher resolution image. The flow gets refined, iteration after iteration, level after level, until the process finally reaches the finest level and the flow is computed on the original images. This process is useful especially for the estimation of large motions, but, as stated for the global methods, could result in a lack of precision, even if we can partially get around this problem using a quality measure and recomputing explicitly the flow to the finest resolution if the previous estimate is not enough good. This approach is used particularly for local methods, as seen for example in [8, 19, 36, 94, 126, 127].

In the following sections, we'll briefly review some techniques for the computation of the optical flow, belonging to different type of approaches: first we'll see a *differential* method (usually referred as the **Lucas-Kanade algorithm**) and its evolution, followed by the description of a *correlation* method, the **block matching algorithm**; the third approach proposed is the **energy minimization** approach, originally proposed by Horn and Schunk. The first two techniques can be classified as local techniques, while the last one is usually labeled as global; all of them adopt a hierarchical approach.

1.3.2 Local methods

The Lucas-Kanade algorithm

The first technique presented here is based on the use of the *spatial* and *temporal first-order derivatives* of the image intensities, has been proposed by Lucas and Kanade [121] and it's considered one of the first **differential methods** for the computation of the optical flow. In the following, we assume that the image domain is continuous and differential in both space and time. Since each image is a discretization of a real scene, the first-order derivatives are actually approximation, computed by simple subtractions of values of adjacent pixel, or by using adequate convolution masks (Prewitt, Sobel or Roberts

operators [86]). Differential techniques are strongly based on equation 1.12: global methods use 1.12 plus some other constraint (e.g. smoothness regularization) in order to guarantee that the compute optical flow will be dense over the entire image, while local methods use normal velocity information in local neighborhoods of a pixel in order to perform a LS minimization and find the best fit for v. Using an oversimplification, we can say that global methods are local methods where the neighborhoods of a single pixel is the entire image.
Assume that the constraint equation 1.12 yields a good approximation of the normal component of the motion field, and that the motion field itself can be well approximated using a constant vector field within any small patch of the image plane. Then, for each $\mathbf{p}_i$ within a *small* $R \times R$ patch Q, we can write

$$(\nabla I)^\intercal \mathbf{v} + I_t = 0 \tag{1.13}$$

Spatial and temporal derivatives of the image brightness are computed at $\mathbf{p}_1, \mathbf{p}_2 \cdots \mathbf{p}_{R\times R}$, that is, in the local neighborhood of central pixel $\mathbf{p}_i$. Usually, a *small* patch Q has a size of 5×5 or 7×7 pixel, in order to capture enough information to estimate the motion of its central pixel (Figure 1.8). The optical flow can be estimated within Q by finding the constant vector $\overline{\mathbf{v}}$ minimizing the functional

$$\Psi[\mathbf{v}] = \sum \mathbf{p}_i \in Q[(\nabla I)^\intercal \mathbf{v} + I_t]^2. \tag{1.14}$$

The solution to the LS problem 1.14 can be found easily by solving the linear system:

$$A^\intercal A \mathbf{v} = A^\intercal \mathbf{b} \tag{1.15}$$

A is a column vector containing the spatial derivatives (I_x, I_y) of the pixels belonging to Q:

$$A = \begin{bmatrix} I_x(\mathbf{p}_1) & I_y(\mathbf{p}_1) \\ I_x(\mathbf{p}_2) & I_y(\mathbf{p}_2) \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ I_x(\mathbf{p}_{R\times R}) & I_y(\mathbf{p}_{R\times R}) \end{bmatrix} \tag{1.16}$$

The vector $\mathbf{b}$ contains the temporal derivatives of the pixels belonging to Q, obtained applying some derivative filter to the image of the sequence:

$$\mathbf{b} = \begin{bmatrix} I_t(\mathbf{p}_1) \\ I_t(\mathbf{p}_2) \\ \cdot \\ \cdot \\ \cdot \\ I_t(\mathbf{p}_{R\times R}) \end{bmatrix} \tag{1.17}$$

The least square solution of 1.15 is computed as follows:

$$\overline{\mathbf{v}} = (A^\intercal A)^{-1} A^\intercal \mathbf{b} \tag{1.18}$$

Figure 1.8: Two patches (or region) Q_0 and Q_1, selected on an input image, both having 7×7 pixels size. The patch Q_1 contain enough information to estimate the motion of its pixels, while the patch Q_0 as poor texture, and therefore will not produce reliable results.

Computing the approximation of motion is the key-point of the algorithm. Typically, given a time-varying sequence of n images $I_1, ...I_n$ each image is filtered using both a *spatial Gaussian filter* ($\sigma = 1.5$ pixels) and a *temporal Gaussian filter* ($\sigma = 1.5$ frames). Then, for each pixels of each image of the sequence, A and $\mathbf{b}$ are computed, and the displacement vector $\overline{\mathbf{v}}$ is obtained using 1.18. Depending from the size of the temporal filter, some images could not be correctly processed (for example, the first and the last k images, if the temporal filter has size of $2k + 1$).

The basic version of the Lucas-Kanade tracker can be improved in several way; for example, assigning a weight to each pixel of the window, in order to give more importance to the pixels of the patch that are nearest to the central one $\mathbf{p}$, or carrying in the computation information about derivatives of the first or second order, in order to keep track not only of the color but also of the variance of the light of the scene. In the following, we briefly describe some of the possible improvements to the algorithm.
A *naive* implementation of the algorithm it's an easy goal to achieve. Computing $\overline{\mathbf{v}}$ it's easier than what it seems from 1.18; in fact, it's straightforward to see that $A^\intercal A$ is non-other than a 2×2 matrix computed as

$$A^\intercal A = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{1.19}$$

while $A^\intercal \mathbf{b}$ can be computed as

$$A^\intercal \mathbf{b} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{1.20}$$

In the end, in order to find the vector $\overline{\mathbf{v}}$ representing the motion of patch Q, we need to build and invert a 2×2 matrix and multiply it by a 2D vector; it's a reasonable small amount of computations that doesn't require large time or space. Also, the computation of the motion of a given pixel $\mathbf{p}$ is completely unrelated with the motion computed for a different pixel $\mathbf{q}$, and that means that this process can be easily parallelized and distributed over a number of processing units, each of them working independently from the others. Two examples of GPU-distributed implementation can be found in [126, 127]: in both cases, the algorithm is accelerated using the CUDA architecture [142].
Another valuable feature is the fact that the algorithm provides also an easy way to estimate the goodness of the computed flow. The matrix computed in 1.19 is, in fact, a **Harris matrix**, typically used for corner detection in images [92]. The eigenvalues λ_1, λ_2 of the Harris matrix computed over the neighborhood of a pixel $\mathbf{p}$ provide information about the saliency of $\mathbf{p}$:

- if $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$, then $\mathbf{p}$ has no feature of interest; this is the case of the pixels belonging to flat or poor textured regions (see the Q_0 patch in Figure 1.8);
- if $\lambda_1 \approx 0$ and λ_2 has a large positive value, then $\mathbf{p}$ belongs to an *edge* of the image;
- if both λ_1 and λ_2 have large positive value, then $\mathbf{p}$ is a corner pixel (see the Q_1 patch in Figure 1.8).

We can add this information to the Lucas-Kanade framework: if at least one of the eigenvalues of the Harris matrix is ≈ 0, we cannot track the pixel in the image sequence, since it doesn't contain enough information, while if both λ_1 and λ_2 are larger then zero, we can reasonably assume that the patch contains interesting feature and we can then compute a good estimation of the motion of its central pixel. Finally, we can add the computation of the eigenvalues of $A^\intercal A$ to our framework, providing at the end of the computation, alongside with the estimation of a motion vector $\overline{\mathbf{v}}$ for pixel $\mathbf{p}$, a **quality measure**, given by the *smallest eigenvalue* of $A^\intercal A$, eventually normalized in the range $[0, 1]$. This measure can be extremely useful in order to perform operation such as flow filtering or flow re-sampling, since it allows to discard not-reliable vectors and, in some cases, even correct their values using the best ones.

Unfortunately, the Lucas-Kanade algorithm has several drawbacks. Like almost every other local technique, it is sensitive to noise in the images: even one or two noisy pixels in the region of interest may dramatically affect the estimate of the motion for that region, even if the value of the quality measure is high. Also, the results depend on the size of the neighborhood. Let's say that, typically, a 5×5 or a 7×7 squared window contains enough information to correctly detect the motion. However, problems can arise when dealing with complex textures, because a window with bigger size is needed; but the larger the window, less local is the computation, and the motion could result in a poor estimate. A naive implementation like the one that has been described

previously is subject to problems related scene illumination, such as poor illumination, or difference of illumination between images in the sequence (and we'll see that those problems are significant especially when one has to deal with images acquired from different points of view).

However, it's possible to arrange several improvement to the original technique, in order to achieve better results. It's possible to improve the smoothness of the flow applying a common *Gaussian filter* on the flow, following the example of the image processing technique: since we expect that groups of pixels corresponding to a common objects move rigidly, we can use the filter to correct the direction of spurious pixels. We could also improve this technique using a *weighted Gaussian filter*, where the weight of each pixel is obtained from the eigenvalues of 1.20, locally computed on each patch. Even better, since Gaussian filter tends to smooth the boundaries of the objects and of their motion, we can apply a different filter, like the *bilateral filter*, in order to correct the flow computed on spurious pixels while preserving the rigidness of the boundaries. If we are following a hierarchical approach, we can simply add these new steps at the end of each iteration. Doing so, we are not changing the idea behind the computation of the flow; we are just correcting the error in the low level in order to improve the results that will be achieved in the next iteration. Other improvements focus more on the speed-up of the algorithm ([126, 127]). There are also several approaches that make one step back, and try to change the original formulation of the problem. For example, Trucco [168] proposes a weighted Lucas-Kanade algorithm, with both spatial and temporal Gaussian smoothing on the sequence of image. The work [137] proposes a *revised* definition of the optical flow and of the original brightness constraint equation, in order to integrate radiometric and geometric information about the scene. In this formulation, the unknowns of the problems are not only δx and δy, representing the motion of the pixel during time, but also δm and δc, two parameters related to the radiometric transformation of the scene and the smoothness of the motion respectively. The computation of the optical flow information related to a pixel p then becomes the solution of the following equation:

$$\sum_W \begin{bmatrix} I_x^2 & I_x I_y & -I_x I & -I_x \\ I_x I_y & I_y^2 & -I_y I & -I_y \\ -I_x I & -I_x & I^2 & I \\ -I_x & -I_y & I & 1 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \\ \delta m \\ \delta c \end{bmatrix} = \sum_W \begin{bmatrix} \delta - I_x I_t \\ \delta - I_y I_t \\ \delta I I_t \\ \delta I_t \end{bmatrix} \tag{1.21}$$

where I_x and I_y are the first order discrete spatial derivatives, computed over the window W, I_t represents the temporal derivatives and I is the brightness value of the pixel. The equation 1.21 is solved, for less than particular cases (such as poor texturing regions, where the 4×4 matrix is nearly singular), and the solution contains information about the motion vector $(\delta x, \delta y)$ that will be more reliable whenever δm and δc are small. However, our experimental results show that the improvement in the quality of the output is not

Algorithm	Running time	Endpoint error	Angular error
Pyramidal LK	12s	0.3961	13.96
GPU LK [126]	0.35s	0.4572	12.15
FOLKI [108]	1.4s	0.2956	10.56
PGAM-LK [3]	0.37s	0.3759	11.86

Table 1.1: Comparison between the algorithm presented in Figure 1.10. For each algorithm we provide the running time, the average endpoint error (that is, the distance between the estimated destination pixel and the correct one) expressed in pixels, and the average angle error (that is, the angle in space between the estimated flow vector and the real one) expressed in degrees.

significant, compared to the introduced computation overhead. We conclude this section showing a selection of different examples of the implementation of the algorithm in 1.10; the comparison between their running time and average error is shown in 1.1.

The block matching algorithm

Another popular local technique, used to compute the displacement vectors between two or more image, is the so-called **block matching** (or *window matching*) algorithm. This technique is particularly suitable for object tracking, surveillance system, obstacle detection, but also video compression, autonomous navigation of a robot and so on ([9, 71, 82–84, 91]). This technique belongs to the **correlation-based methods**, since the focus of the algorithm is the analysis of the gray level (or color) pattern around the point of interest, and the search for the most similar pattern in the successive image. Formally, first we define a *block* (or a *window*) $B_i(p)$ around the pixel p of the image I_i as a $n \times m$ window of pixels centered in p, we consider the blocks B_j on a different image I_j, obtained shifting the coordinates of $p \equiv (x_p, y_p)$ by means of two values i and j, with $-\Delta < i < \Delta$ and $-\Delta < j < \Delta$. The estimated displacement of p is given by the shift corresponding to the minimum of a distance function f, or the maximum of a correlation measure c, between the intensity pattern in two corresponding blocks (Figure 1.11). In other words, given the block $B_i(p)$, we search for the block $B_j(q)$ that is *more similar* to B_i in image I_j. Of course, the implicit assumption is that the gray level pattern is constant in the image sequence, as already stated for the Lucas-Kanade technique, and that the block contains enough texture information for a good estimation of the motion. Obviously, there's a number of issues in this definition of the problem:

- the size of the block B_i: a small block could result in an uncorrect displacement detection, since the information is computed on too few pixels, while a large block could perform better but slower, since it involves a larger number of elements;
- there's no unique definition of the distance/similarity measure (even if this, as a matter of fact, can be turned into an advantage, since may give

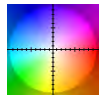

Figure 1.9: Color code for flow visualization (HSV model) borrowed from [12]. The direction of the flow is represented by the hue of the color and the magnitude by the value.

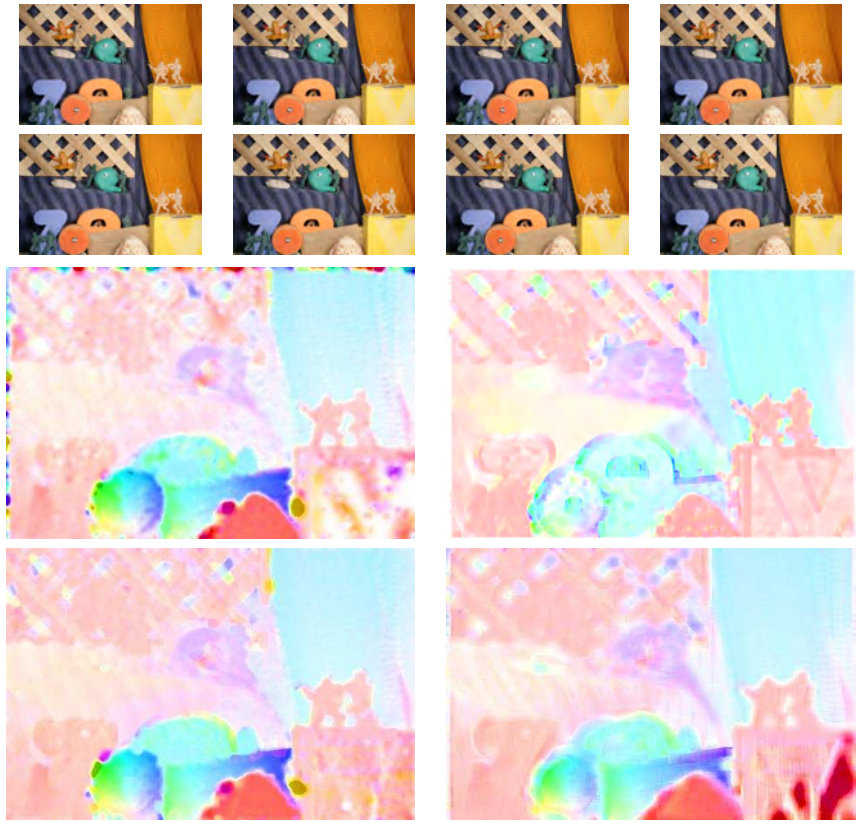

Figure 1.10: Some examples of different implementation of the Lucas-Kanade algorithm, performing on the same dataset of images. The first rows show the original dataset. The other images present, in order, the output of a naive implementation of the pyramidal-LK algorithm, the output of the work proposed in [126], the result of the algorithm proposed in [108], and finally the output of the algorithm [3]. Color code in Figure 1.9.

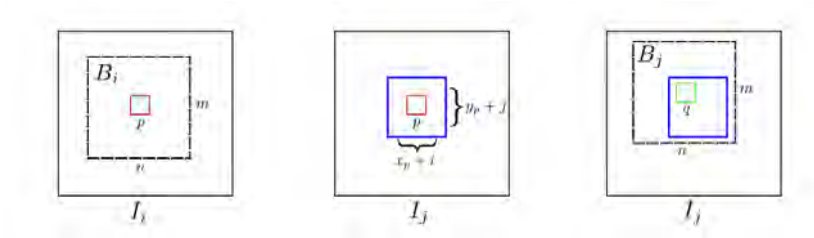

Figure 1.11: The block matching related to a pixel $p \equiv (x_p, y_p)$ in image I_i. In the leftmost image, the pixel p and its $n \times m$ block B_i; in the middle, the search region obtained shifting p of (i, j) in the image I_j; in the rightmost, the block B_j, same size of B_i, centered in the pixel $q \in I_j$ and belonging to the search region.

more versatility to the technique);

- the size of the shifting is not uniquely defined (a smaller Δ performs less comparisons, since it takes into account a smaller number of blocks, but could result in an inaccurate detection, while a larger Δ can heavily affect the computation);

- the motion between two blocks may not be represented by means of integer shift values, and, as a result, we should perform non-integer subsampling on the image, in order to perform a more accurate analysis.

Usually, this kind of algorithm performs better on good matchable feature points, such as corner points; if a correspondence to a feature point p is found, then the algorithm has a good chance to estimate the motion of p with high precision (even sub-pixel motion); the drawback is that, similarly to the Lucas-Kanade technique, it could be difficult to obtain an accurate dense flow computed over all the pixels of the images, since the feature points are usually sparse over each image (a problem that we already deal with, in the previous section). Most of the applications using correlation-based methods doesn't need a dense flow, preferring instead to compute a reliable flow only on those points that are considered of interest. Consider, for example, an object tracking algorithm: it's enough, for the application, to reliably track the boundary of one specific object in the scene. The ease of the implementation, its cheap computational cost, and its versatility are major advantage of this algorithm; also, the technique can be easily extended in order to adapt it to a large range of different situations, to achieve properties like robustness to noise or to illumination changes.
The selection of the distance/similarity measure is obviously the first aspect to analyze when working with the block matching technique, since choosing a specific measure instead of a different one will surely affect the results, sometimes even dramatically. Let assume that we need to compare two squared windows, $N \times N$ pixels each, the first one centered in (x_i, y_i) and belonging to

image I_0m and the second one centered in (x_j, y_j) and belonging to I_1. The simplest choice it just to sum the values of the **absolute difference** of the corresponding pixels (SAD):

$$SAD(B_i, B_j) = \sum_{k,h=-N/2}^{N/2} |I_0(x_i + k, y_i + h) - I_1(x_j + k, y_j + h)|.$$

As usually happens, the simplest way is not the better one, and, in fact, the SAD is usually prone to errors and is not a reliable measure. Results get better if we switch from the SAD to the **sum of squared difference** (SSD):

$$SSD(B_i, B_j) = \sum_{k,h=-N/2}^{N/2} (I_0(x_i + k, y_i + h) - I_1(x_j + k, y_j + h))^2.$$

This similarity measure performs well under the assumption that the gray value is locally constant. Whenever this assumption does not hold, we need to use a different measure that takes into account also the gray-level variation, using the global average intensity values (respectively, i_0 and i_1) in order to adjust the measure:

$$ZSSD(B_i, B_j) = \sum_{k,h=-N/2}^{N/2} (I_0(x_i + k, y_i + h) - i_0 - I_1(x_j + k, y_j + h) - i_1)^2$$

$$LSSD(B_i, B_j) = \sum_{k,h=-N/2}^{N/2} (I_0(x_i + k, y_i + h) - i_0/i_1 - I_1(x_j + k, y_j + h) - i_1)^2$$

Other measure used especially when dealing with video compression are the *Mean Absolute Difference* (MAD) and the *Mean Squared Error* (MSE):

$$MAD(B_i, B_j) = \frac{1}{N^2} sum_{k,h=-N/2}^{N/2} |I_0(x_i + k, y_i + h) - I_1(x_j + k, y_j + h)|$$

$$MSE(B_i, B_j) = \frac{1}{N^2} sum_{k,h=-N/2}^{N/2} (I_0(x_i + k, y_i + h) - I_1(x_j + k, y_j + h))^2.$$

These different measures compute similarity between B_i and B_j using only gray level values of the images, but sometimes, in the computation, we need to add some terms related to the spatial derivatives of the pixels. Being D_{x_0} and D_{y_0} the discrete derivatives of image I_0, and D_{x_1} and D_{y_1} the ones of I_1, we could compute the similarity as:

$$DSSD(B_i, B_j) = \sum_{k,h=-N/2}^{N/2} \Big((I_0(x_i + k, y_i + h) - I_1(x_j + k, y_j + h))^2 +$$

$$+ (D_{x_0}(x_i + k, y_i + h) - D_{x_1}(x_j + k, y_j + h))^2 +$$

$$+(D_{y_0}(x_i+k,y_i+h)-D_{y_1}(x_j+k,y_j+h))^2\Big)$$

The extension to the SAD is straightforward. Adding two terms for the spatial derivatives allows to compute similarity even if the assumption of brightness constancy doesn't hold. One can also chooses to weight each term of the equation, either uniformly (each term has a $\frac{1}{3}$ weight) or giving a largest weight to the image gray values (for example, giving a weight of 0.5 to the first term of the equation, and 0.25 to both the others term). We can also add another weight, related to the position of each pixel in the block: pixels that are nearer to the center should have a weight that is larger than the one of the pixel nearer to the boundary of the block, possibly assigning the weights in a Gaussian fashion.
Eventually, we could also use some statistical correlation measure like the *Cross-Correlation*, or (even better) the *Normalized Cross-Correlation*, that measures the correlation between two blocks, that is, how much the data of the first block *depends* from the data in the second one. In general, there isn't a measure that performs better than the others in every situation, even if the SSD and its variants usually lead to better results; the choice depends from the characteristic of the data and from the desired goal.
Other technical details, like the size of the window or the computation of sub-pixel flow, haven't been accurately dealt in literature, and the optimization of the algorithm is usually performed differently on different applications. However, those aspects have been partially investigated in [16,81,155,160]. For the definition of a quality measure of the estimated flow, we remind the reader to the previous section.
Like any other local method, the block matching technique can be used together with a hierarchical approach, especially when dealing with large displacements that should be computationally too heavy to handle when working on the original resolution, and that could also introduce noise in the solution. Initially, the block matching is performed on coarse resolution images, and the resulting flow is used as starting point for the computation to a finer level; when switching from a level to a finer one, we could also adjust parameters like the size of the block, or the Δ value that determines how many blocks we test. This is, by far, the most common approach to the block matching technique; the improvements to this basic approaches has been proposed in a large number of works. The work by Anandan [8] is based on a Laplacian images pyramid, and uses a coarse-to-fine SSD-based matching strategy that also detects sub-pixels motion with a confidence measure derived from the principal curvature of the SSD surface. The approach by Singh [160], similarly, uses the SSD minimization combined with a two-stages computation: the first stage is the estimation of the displacement, obtained by a combination of SSD values, computed on triples of adjacent high-pass filtered images, and a probability distribution R_c defined for each pixel, obtained from the SSD surface; the second stage is the propagation of the estimation to the finer levels using a neighborhood smoothness constraint, achieved by minimizing an appropriate functional. Baglietto et al. [11] proposed a parallelized version of the block matching, interesting especially from the implementation point of view. The *Full Search Algorithm*

or *exhaustive search* compares one block of the first image with all the blocks of the second image, with the obvious drawback of being quite expensive. The *Three Step Search* and its evolution [111] use a predefined pattern in order to search and evaluate the best block matching. An extension of this algorithm is the so-called *Simple and Efficient Search* [120] that, following the idea that the motion can be represented as a unimodal surface, there can't be two minimum in two different directions, and therefore the search pattern can be restricted to a search in one of four quadrants. *Four Step Search* [146] and *Diamond Search* [47, 180, 181] use respectively a square-pattern and a diamond-pattern in their search; first they use a larger pattern, to locate the region of interest, and then they refine the search using a smaller pattern. The *Adaptive Rood Pattern Search* [139] performs a fast search based on statistic information extracted by a video sequence. We should finally mention the work by Okutomi and Kanade [144] that uses probability values as weights in the block matching estimation, the work by Chen [46] that revises the definition of the problem and the choice of the measure in order to optimize the performances of the algorithm, the work by Nillius et al. [140] that proposes the using of the NCC and of the Walsh transforms in order to fast compute the optical flow, and finally the work of Patras [145] that, instead of improving the algorithm, propose a metric to evaluate the goodness of the estimated motion field, using a probabilistic framework and assigning a confidence measure to each estimated displacement using the a-posteriori probability. In general, block matching turns out to be a good trade-off in terms of both result and computational cost; also, its versatility makes it a powerful algorithm, since one needs only to define the similarity measure and the size of the window of interest.
We present some results of this techniques in Figure 1.12; tests are performed on the same dataset but with different parameters. It's easy to notice that taking into account derivatives performs better than using only color information (less noise). In our test, we find that a good trade-off in terms of time and result is achieved by means of a 9×9 search window, combined with a hierarchical approach and a Gaussian filtering on the flow at the end of each iteration.

1.3.3 Global methods

The energy-minimization framework and its evolution

The impossibility to compute a dense flow on the whole image can be, in some applications, a major drawback. In a certain sense, this is a problem shared with the Lucas-Kanade tracker and, in general, with the local techniques: poor textured or occluded regions lead to unreliable results and, in general, incorrect estimates. This is mainly due to the fact that the estimated motion is computed locally, using only information related to a neighbors of the region of interest of a given pixel. Sometimes, even if it's impossible to compute correctly the displacement, it's possible to reasonably estimate it observing the motion of the other region of the image. In this case, we assume that 1.12 still holds everywhere on the image, but also we assume that the motion of each object of

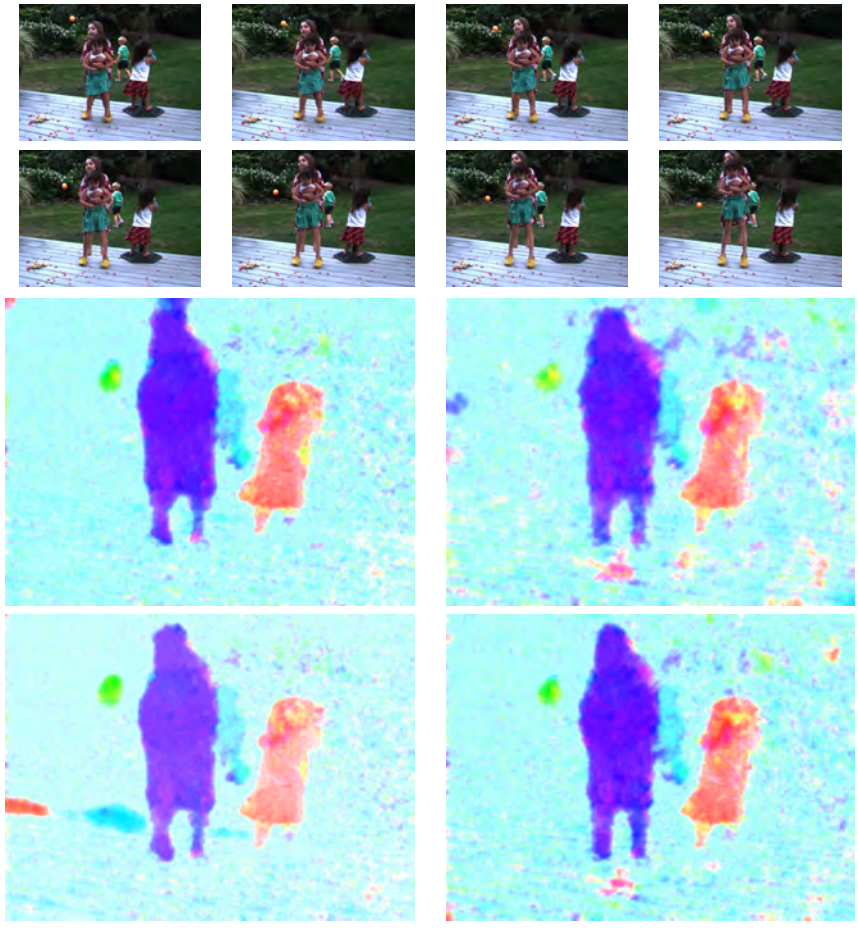

Figure 1.12: The block matching algorithm applied on the same sequence (the Backyard dataset, in the first two rows). On the left column, we present result of the DSSD measure, while on the right column we presents the result of the common SSD measure. Test are performed using a hierarchical approach and with a block window of 5×5 size (third row) and a 9×9 windows (last row). Running time is about 0.5s for each image, since the algorithm has been implemented using the CUDA GPU acceleration. The average endpoint error in the less-noisy case is 1.4763 pixels, with an average angular error of 6.75 degrees.

the scene can be approximate with a *rigid* and *smooth* movement. The motion is then estimated globally, taking into account the information related to the estimates of the motion of all the pixels, in order to preserve the smoothness of the final result. Typically, global techniques made an explicit use of the brightness constraint equation 1.12 in the formulation of the problem, combining it with one or more *regularization term* encoding the constraints that we want to preserve in the computation; in this way, they define a *functional* which is then minimized over a domain of interest (usually, the entire image domain). In the first basic version of this global technique, proposed by Horn and Schunk [98], the regularization term is a *smoothness constraint*, requiring a slowly varying optical flow field. The reason for the constraint is conceptually easy to understand: if two pixels belong to the same object of a scene, and we are assuming that each object moves rigidly, then their motions should be almost identical. It's uncorrect to say that each motion should be computed only locally, since, ideally, the correct estimation of the motion of few, significant pixels of the objects should indicate the direction of the motion of the entire object. Formally, equation 1.12 is combined with the smoothness constraint to define an error functional:

$$\int_D \Big((\nabla I \cdot \mathbf{v} + I_t)^2 + \lambda^2 tr((\nabla \mathbf{v})^\intercal (\nabla \mathbf{v})) \Big) d\mathbf{x}. \tag{1.22}$$

D represents the domain of interest, while $\mathbf{v}$, as seen before, represents the two-dimensional motion field of the domain of interest; if the domain of interest is the image I then a vector $\mathbf{v} = (u, v)$ is defined for each pixel $\in I$. Our goal is to find the field $\mathbf{v}$ that minimizes the functional. As a matter of fact, minimizing the functional means minimizing each term of the functional: minimizing the first term implies that the brightness constraint equation is satisfied, while minimizing the second term has the effect that smoothness is preserved, since the second term introduce a penalization for abrupt changes of direction of the motion field vectors.
Since we are working on digital images, that are, for definition, a digitalization of the real world, we usually cannot compute exactly the field $\mathbf{v}$ of 1.22; instead, what we can do is to build a linear system having n equation and $2n$ unknowns (one equation for each pixel of the image, and two unknowns for each pixel, representing the motion components u and v); each equation is obtained expanded the equation 1.22 and isolating the knowns, in order to achieve a formulation similar to $A\mathbf{v} = b$. The field $\mathbf{v}$ is then computed as an approximation of the solution of the linear system, via least squares solver, or SVD decomposition, or iteratively and so on, usually depending on the properties of the equations. The resulting optical flow field will be, in general, smoother than an optical flow field computed using local methods; unfortunately, there is some major drawbacks. Particularly, the first drawback is related to the high computational cost of the global techniques. As a matter of fact, we need to build a system having an equation for each pixel; even for an image with medium resolution (e.g. $1024 \times 768 = 786342$ pixels) the size of the linear system is huge, and the procedure for the minimization using LS or SVD decomposition could be quite expensive; the situation gets complicated when we have to work on high

resolution images (e.g. $2500 \times 1600 = 4$ Megapixels). Another drawback is that global techniques produce results that are better in terms of visualization and understanding of the motion, but that can be less accurate than a local estimation; for example, small motion could be lost due to the smoothness constraint. In general, this technique capture very well large displacement and motion of entire objects, but may perform poorly when we need to detect small movements (e.g. subpixel precision) occurring in a well-localized region of interest.
The basic approach of Horn and Schunk that led to the formulation of 1.22 it's the first one and it's, someway, too naive; the brightness constancy equation turned out to be unrealistic in many case, such as the *single motion assumption* (the assumption that there's only one moving object in the scene). In a way that is similar to what we saw for the local technique, the original formulation has been revised in literature, in order to adapt the method to a large range of different problems. The main idea stays the same: defining and minimizing some functional formed by a number of constraint, encoding the characteristic of the scene that we are observing. In the years, a number of regularization terms different from the original used in 1.22 have been proposed, starting from Nagel and Enkelmann [135] introducing the *oriented smoothness constraint*, forcing the vector field to change only in direction characterized by small variation of the gray level; in this way, boundaries of the objects, characterized by high variation of gray level, are detected and their motion is correctly estimate. Their work has then been improved in [7] with the reformulation of an energy functional that is invariant under brightness changes. Black and Anandan [25] modify the formulation of 1.22 introducing two new constraints, the *spatial coherence constraint* and the *temporal coherence constraint*; in this way, it becomes possible to track multiple objects, in the same sequence, assuming that they are moving rigidly and their velocity is constant during time. Subsequently, in [26] they propose to relax the single motion assumption in order to better and robustly estimate the flow, even if there are several moving objects in the scene, preserving the discontinuities between the motion of different objects. Other formulations of 1.22 have been discussed, among others, in [74], that proposes the segmentation of the motion field in order to guarantee a coherent motion for each object of the scene, [6] that models and solves the problem as a partial differential equation, [134], where Mukawa proposes a regularization term that takes into account not only the smoothness constraint, but also other constraints related to the lighting scene effects, assuming to have a single moving object in a scene with a light source, and in [51], by Cohen, that proposes to minimize a non-quadratic functional, by means of solving iteratively a system of nonlinear differential equation, in order to preserve the discontinuities of the optical flow on the boundaries of moving objects.
One of the most significant improvements of the energy minimization framework has been proposed in 2004 by Brox et al. [36]; they propose a global variational model, derived from the original 1.22. The assumption is that optical flow constraint 1.12 usually does not hold, at least in real-case scenarios,

but the variations in the gray level could be helpful to determine the displacement vector, assuming that also these variations are invariant in the sequence. This criterion is referred as the **gradient constancy assumption**:

$$\nabla I(x, y, t) = \nabla I(x + u, y + v, t + 1) \tag{1.23}$$

where $\nabla = (\partial_x, \partial_y)$ denotes the spatial gradient of the image and $\mathbf{v} = (u, v)$. This constraint joins the original brightness constancy equation in order to create the first terms of the new global equation:

$$E_{Data}(u, v) = \int_\Omega \Psi\Big(|I(\mathbf{x} + \mathbf{w}) - I(\mathbf{x})|^2 + \gamma|\nabla I(\mathbf{x} + \mathbf{w}) - \nabla I(\mathbf{x})|^2\Big)\mathbf{dx} \tag{1.24}$$

where Ψ is an increasing concave function that penalize outliers and leads to a robust formulation. This term represents both brightness and gradient constancy assumptions and therefore will replace the first term of 1.22. Then, like in [98], we need to define ad add a second term, representing the optical flow smoothness constraint, in both spatial and temporal sense. The authors propose a smoothness energy written as

$$E_{Smooth}(u, v) = \int_\Omega \Psi\Big(|\nabla_3 u|^2 + |\nabla_3 v|^2\Big)\mathbf{dx} \tag{1.25}$$

where ∇_3 represents the spatio-temporal gradient, but it can be eventually replace by the simple spatial gradient. Finally, the total energy is the weighted sum between the two terms:

$$E(u, v) = E_{Data} + \alpha E_{Smooth} \tag{1.26}$$

The weight $\alpha > 0$ is a regularization parameter, and the domain of interest Ω is the whole image. Obviously, the main goal is to find the optical flow field that minimizes E. The problem is solved using a coarse-to-fine approach that, level after level, converge to a minimum of a function that is also the minimum of the original energy function. For an exhaustive discussion about the formulation of the minimization problem and its numerical solution, we refer to the original work [36]; in this section, we discuss about the characteristic and the formulation of the problem. This approach has been further developed by the same authors; in their work [35] they successfully combine this technique with an image segmentation algorithm in order to achieve a piecewise smooth optical flow. The energy formulation keeps into account also the different segments of the image, their correspondences and the possible deformation of each patch.
The energy minimization model has been carefully analyzed and revised by Black, Sun and Roth in [165], where they discuss the *secrets* of the computation of optical flow, giving some interesting suggestion for both formulating the problem (e.g. choosing carefully the penalty term in the energy formulation) and implementing a solution (pre-processing step, coarse-to-fine strategy, interpolation methods and so on).
Generally, the energy minimization framework is a powerful and versatile technique to compute the optical flow. As already seen, the global equation can

Algorithm	Running time	Endpoint error	Angular error
Horn-Schunck [98]	49s	1.5156	9.1655
Black and Anandan [25]	328s	1.1145	6.4547
Brox *et al.* [36]	18s	0.9135	4.6235
2D-CLG [3]	844s	1.3845	6.2945

Table 1.2: Comparison between the algorithms presented in Figure 1.13. For each algorithm we provide the running time, the average endpoint error (that is, the distance between the estimated destination pixel and the correct one) expressed in pixels, and the average angle error (that is, the angle in space between the estimated flow vector and the real one) expressed in degrees.

be adapted to a large number of situations, changing the terms definition or defining and adding some more constraint, according to the properties of the scene and the desired results. A major advantage is its robustness against noise, the ability to overcome problems of poor texturing and, in some cases, dealing with occlusions and illumination changes that usually are problematic when working on local spatial information. As we already mentioned, the majors drawback are the possible loss of precision (small disparities may be poorly estimated or even completely lost, because of the smoothness constraint forcing large regions of the image to have a coherent flow) and the high computational cost usually required by these techniques, especially when we need to solve large linear system (or, depending from the functional definition, even nonlinear system); the discretization of the problem and the implementation of its solution could be also a drawback, compared with the simplicity of implementing a local technique. Sometimes, it's also possible to use an hybrid approach: a global method could easily detect with good precision large displacements of the objects of the scene, and the resulting flow can be then locally refined using some local technique, in order to preserve also small motions and details. An example is proposed in [37] where the two classic techniques, developed by Lucas-Kanade and Horn-Schunck, are combined in order to define an energy functional that is then minimized locally on the regions of interest of the image. Some results of global techniques are presented in Figure 1.13, while timings and error evaluation are presented in table 1.2. It is easy to notice that the resulting flows are usually quite smooth, since they capture large displacement of the objects in the scene, but they may fail to capture small movements. A visual comparison between global and local techniques, performing on the same test-case, is shown in Figure 1.14, while errors and times are summarized in table 1.3.

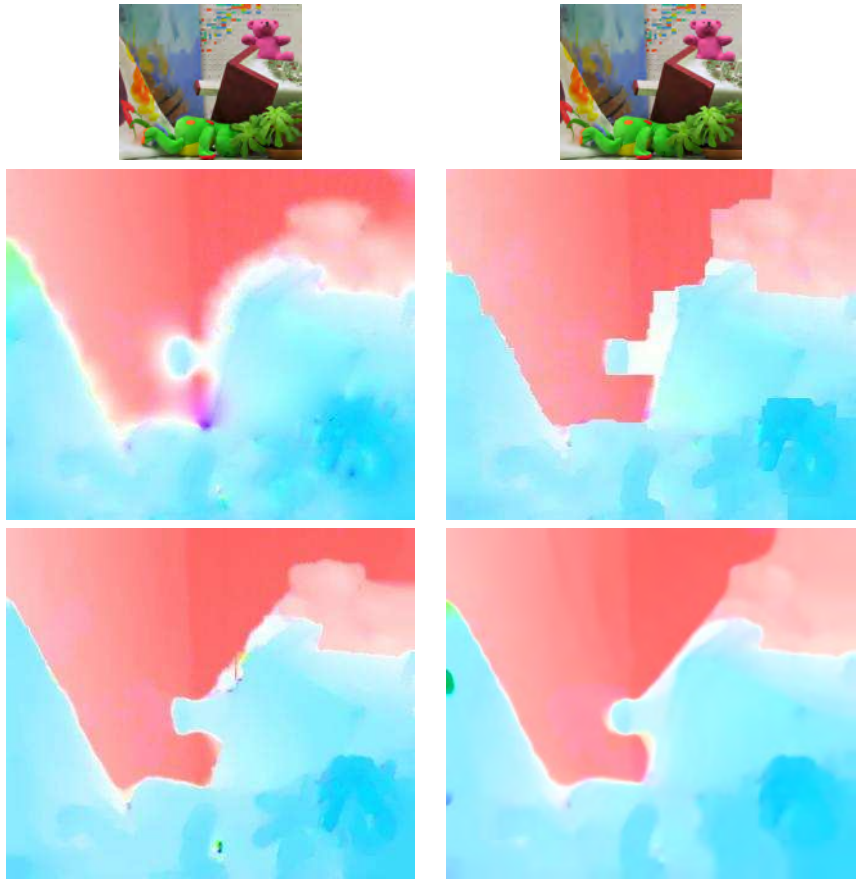

Figure 1.13: Optical flow computed on a 2-frames test case. We present the results of the original Horn-Schunck algorithm [98], the algorithm by Black and Anandan [25], the Brox method [36] and the hybrid approach described in [37]. It's easy to notice that the global motion of the scene is captured in a smooth fashion. Small displacements, such the ones related to the head of the green character, may experience loss of precision in the computation. Color code in Figure 1.9.

	FOLKI [108]			Brox [36]		
Dataset	Time	Endpoint	Angular	Time	Endpoint	Angular
Grove	1.27s	1.5360	6.1660	17.6s	1.1045	3.7938
Mequon	1.42s	1.5261	20.961	17.3s	0.2731	3.7230
Schefflera	1.31s	1.2358	17.656	18.1s	0.3922	4.9722
Yosemite	0.87s	0.2659	4.6756	14.6s	0.1040	2.2211

Table 1.3: Comparison between the FOLKI algorithm [108] and the Brox method [36] related to Figure 1.14. The local method is faster, but has generally an higher error due to the poor results on flat regions. The global method has a smaller error, but is significantly slower.

Figure 1.14: Visual comparison between the FOLKI algorithm [108] (Lucas-Kanade tracker with GPU parallelization) and the Brox method [36] on several datasets. Color code in Figure 1.9.

Chapter 2

Bundle adjustment via Optical Flow computation

The work presented here is inspired by the one proposed in [64]; in their work, DellePiane *et al.* use the computation of optical flow to determine the disparities between two images of the same scene, and then use this information to create a texture for the digital model of the scene, obtained warping the original images. It's probably the first time that a computer vision technique has been used to partially solve a geometry processing typical problem, that is, improving the projection of color and texture information from the images to the digital surface; however, it's someway correct to say that their approach is hybrid, since the optical flow is used to modify the images, shifting pixel position to best fit the geometry of the object, so it actually doesn't affect the geometry of the scene or the camera placement.
The purpose of this work is to understand if the computer vision of the optical flow may be used to solve other problems usually related to the geometry processing. Particularly, in this chapter we investigate the use of the optical flow in the bundle adjustment process. The idea is to use this technique to find the best placement for our camera, in order to project the color and texture information on the digital surface removing the artifacts (such as blurring or ghosting, Figure 2.1) and eventually modifying the underlying geometry of the digital surface, according to the observed displacements.
The chapter is subdivided as follows: first, an overview of the approach is given, followed by the description of its naive implementation; we then make a brief digression in order to explain how to move the cameras according to the optical flow (determining the *egomotion* of the camera) and show the preliminary results. The last two sections describe the limitations of the approach and discuss several possible improvements to it.

2.1 Overview

Assume that two cameras, C_0 and C_1, placed into two different points, are looking to the same object from two different points of view; we capture two images of the object, I_0 and I_1. Assume also that C_0 and C_1 are placed such that I_0 and I_1 overlap in some part, so that there is a part of the object that is visible in both images (see Figure 2.1). Assume that both cameras capture a certain point Q of the object. Casting a ray r_0 from the center of the camera C_0, directed to Q, we can easily find the intersection q_0 between r_0 and I_0, that is, the projection of Q onto the image plane of camera C_0. In the same way, we can cast a ray r_1 from C_1 and find the point q_1 on the image plane I_1.

Figure 2.1: The same object (in this case, a vase) captured from two different viewpoints, and a digital representation encoding only the geometry on the object. Our goal is to find the correct alignment of the cameras in order to project the color and texture information from the image onto the surface. See also Figure 1.2 for a similar case.

Suppose now to have a point P of the real object, captured in both the images I_0 and I_1, and suppose that we can univoquely identify the pixels p_0 and p_1 corresponding to P. Ideally, if we cast a ray r passing through C_0 and p_0, hitting the object and then going back to the camera C_1, we should find that the intersection between I_1 and r is exactly the point p_1. What actually happens is that the intersection point is not p_1, but it's instead a different pixel p'_1, slightly shifted with respect to p_1. This happens because of the discretization of the process of image acquisition, since the continuous signal (the captured scene) has to be discretized and quantized in order to obtain a digital image, introducing some (usually small) error. Since the pixel p_0 has fixed size and integer coordinates in the image reference frame, it may not correspond to the actual point identified by the intersection between r_0 and I_0, so the point hit on the surface may not be the actual P point but a nearest P' point; then, casting a ray from P' to C_1 introduce another error, that lead to the identification of the pixel p'_1 instead of p_1. Assume now that we have two images I_0 and I_1, and

Figure 2.2: A ghosting artifact due to incorrect alignment of the cameras.

a digital reconstruction of the real object, obtained using a 3D scanner; we have now another level of error introduced in the process, since also the acquisition of the object is affected by some quantization and discretization error. The effect is that, if we want to project the color information from the pixels p_0 and p_1 to the surface, then the rays r_0 and r_1 will probably hit the digital surface into two different points, even if they correspond to the same real three dimensional point P; the re-projection of the colors back to the cameras is affected by a possibly significant error that needs to be reduced, in order to be able to add color and texture information to the object without creating artifacts on the surface (Figure 2.1), due to the fact that different pixels may project different color information onto the same point of the digital object. The image 2.1 will help better understand the process and the errors here described.

Ideally, what we should achieve, in order to reduce the error, is that projecting color information from p_0 to the surface and then back-projecting the information to C_1, the intersection point p'_1 is as close as possible to the actual p_1 pixel, corresponding to the same 3D real point P of p_0. In other words, we want to minimize the length of the displacement vector $\vec{\mathbf{v}} = p'_1 - p_1$, and we can do this in two ways:

- moving the camera C_1 in order to make p_1 and p'_1 overlap;
- modifying the geometry of the object, moving an existing point (or adding a new one) in order to affect the re-projection of the information to C_1 and make p_1 and p'_1 overlap.

Obviously, the solution isn't so simple: as a matter of fact, moving the camera or modifying the geometry may reduce the length of vector $\vec{\mathbf{v}}$ related to

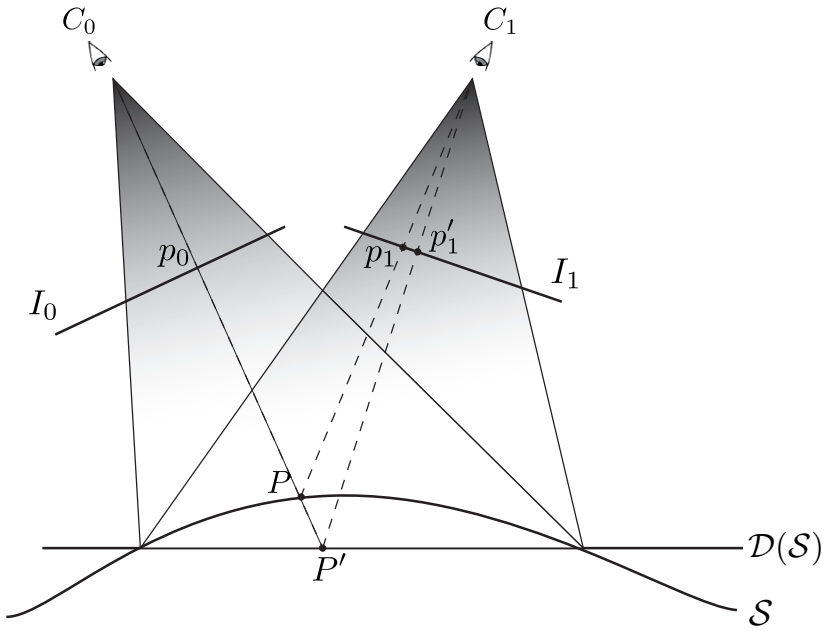

Figure 2.3: The alignment error. Images I_0 and I_1 capture the same point P, lying on the surface $\mathcal{S}$, corresponding to pixels p_0 and p_1. However, the discretization of the surfaces $\mathcal{D}(\mathcal{S})$ introduce some error. If we project information from p_0 onto the digital surface $\mathcal{D}(\mathcal{S})$ and then back-project onto I_1, we find a pixel p'_1 different from p_1. Our goal is to get p_1 and p'_1 as close as possible. The vector $\vec{p_1 p'_1}$ represents the displacement of the back-projection.

P, but may also introduce new error,related to pixels corresponding to different points of the object. We need to take in account *all the displacement vectors* involved into the the process: we don't project a single pixel p_0, but, obviously, we project the *entire image* I_0 onto the surface of the object, and then back-projecting onto the image plane I_1. We can discard all the information related to pixels that doesn't project directly onto the object (the background) and to pixels that project onto parts of the surface that aren't visible from C_1 (occlusions). We then obtain a number of displacement vectors $\vec{\mathbf{v}^i}$, each of them representing the displacement between the pixel p_1^i and the back-projection of its corresponding pixel, p_0^i, on image I_0. The goal is now to minimize the following:

$$\sum_i ||\vec{\mathbf{v}_i}||, \ i \ projected \ from \ I_0 \ to \ I_1 \tag{2.1}$$

Explicitly, we want find the camera alignment that reduces the global displacement as much as possible; eventually, after the alignment, we can further reduce the error acting directly on the digital surface of the object.
Even if the analogy is maybe already clear, it's time to bring into the problem

the optical flow technique. Let I_0 be the image captured by C_0; we project all the pixels of I_0 onto the digital object, and then back-project it onto the image plane of the camera C_1. We call then B_1^0 the image formed onto the image plane of C_1 by the back-projection, minus the background pixels and the occluded ones; the indices indicates that the images is obtained the image I_0 onto the image plane of C_1. We can computed the displacement vectors $\vec{v_i}$ simply computing the optical flow between B_1^0 and I_1: ideally, the flow should be null, but actually, for each pixel we will have an estimate of its displacement. This is straightforward: pixel p_1 corresponds to pixel p_1', so it implies that they share a number of information (color, gradient, behavior of the neighbors), so applying an optical flow technique it's equal to identify, for each pixel, the position of the corresponding one, and the displacement between them. The goal is finding the camera alignment that minimizes the optical flow between the back-projected image B_1^0 and the actual, image I_1 captured by C_1. In this way, the color information should be projected onto the surface smoothly and reducing the risk of artifacts.
And how can we minimize the flow? Again, the answer is in the optical flow, that is not only a measure of the displacement, but can also be used in order to figure out how the camera has to be moved to achieve null flow. To clarify this point, consider B_1^0 and I_1 as two images of the same scene, took from a moving into two different instant t_0 and t_1; which is the motion described by the camera between t_0 and t_1? The answer to this question also gives us the trajectory that the camera has to describe in order to minimize the flow, and this trajectory can be computed using the optical flow. The trajectory of the camera C is known as the *egomotion* of C. We'll see later, in more detail, how to estimate the egomotion from the optical flow field of a sequence of images.
However, we usually don't have only a single pair of images; in order to cover the entire surface of an object, we usually need to take a number of images, depending from the size of the object. So let assume to have n different images, taken from n different points of view; we need to minimize the sum of the displacement vectors of each possible pair of images. For each couple I_i and I_j, we take the back-projected image B_j^i and the compute the displacement with respect to I_j. Assume that $\mathbf{v}_i^{\vec{j,k}}$ represent the displacement vector for the pixel p_i from image I_j to image I_k; then, the global error can be expressed as

$$\sum_{j=1}^{n} \sum_{k=j+1}^{n} \sum_{i \in I_j} ||\mathbf{v}_i^{\vec{j,k}}|| \tag{2.2}$$

that is, the sum of all the possible displacement vectors involved in the process. Of course, not every pair of images contributes to the error; couple of images that has no overlap parts, for example, has no displacement vector to minimize and they can be discarded. Notice that, since aligning I_i to I_j and aligning I_j to I_i is equivalent, we took it into account only once (this explain the behavior of indices j and k). Also, we assume to have an initial configuration with all the cameras C_i placed in a good position, in the sense they are more or less pointed to the object, so that projection and back-projection will produce

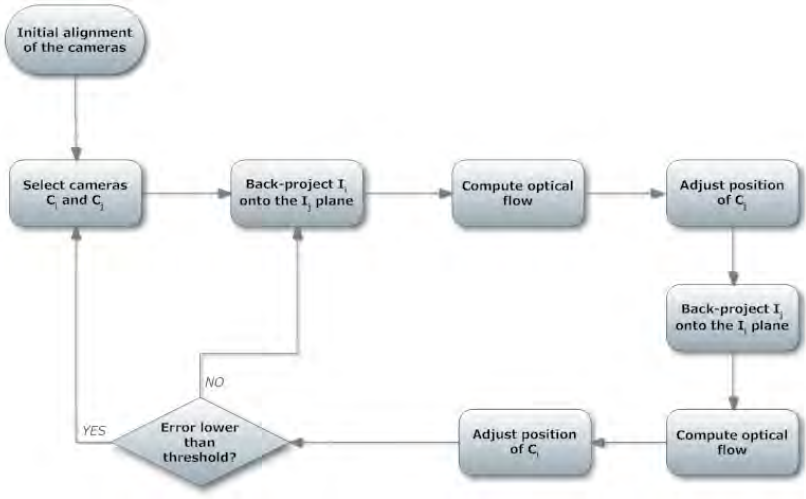

Figure 2.4: Flowchart of the iterative alignment algorithm. Alignment is performed iteratively on each pair of overlapping cameras.

reliable results. We can assume to have an initial configuration obtained using one of the algorithms previously described in the chapter 1.2, that has to be refined in order to achieve the best possible precision and coherence, reminding that the main goal is to align the cameras to project the color information to an existing digital shape (and, if possible, improving the detail of its geometry); stereo reconstruction from images is not taken into account. A recap of the algorithm is presented in Figure 2.1

2.2 Implementation

The preliminary results presented here are obtained using a synthetic framework, built in order to verify the goodness and the correctness of the idea behind the algorithm. In the 2.5 we will explain which are the problematic related to the real application of this technique and how to overcome them.
The system consists of two synthetic pin-hole cameras C_0 and C_1, implemented using the OpenGL library, placed in a common coordinates frame, a digital object, acquired using one of the techniques decried in 1.1, and two synthetic images I_0 and I_1 representing the texture of the object. The images are realized with the goal to achieve maximum accuracy in the computation of the optical flow, reducing the possible ambiguities; each pixel has a unique color and unique partial spatial derivatives. Each camera has its own extrinsic and intrinsic parameters with the exception of the the radial distortion coefficient; we also assume that cameras are already calibrated, so that intrinsic parameters (focal length, aspect ratio and image center) are known in advance, and both cameras are looking to the object. The system setup is sketched in Fig-

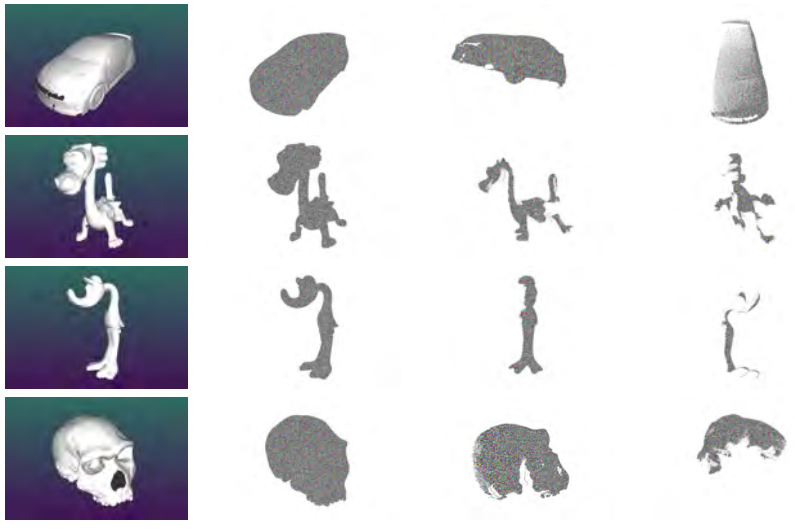

Figure 2.5: Synthetic datasets. Each dataset is created starting with the digital shape; then, two cameras C_0 and C_1 are positioned. One camera, let's say C_0, is taken as reference camera; we create the synthetic image I_0 such that it projects perfectly onto the shape. The back-projection of the newly created image on the other camera image plane is taken as reference image I_1 for C_1. In the image, we show on the leftmost column the digital shape, then the reference texture for camera C_0 and two reference images for two different camera positions.

ure 2.1, while the synthetic dataset composed by two texture images and the digital shape is shown in Figure 2.5

Each cameras takes a reference image of the object as shown in Figure 2.5. Then, the position of camera C_1 is randomly or manually perturbed in order to achieve a worse alignment. We run the algorithm until it converges and the position of C_1 is adjusted coherently with the information of the optical flow between the back-projection of I_0 and the reference texture I_1. In other words, the algorithm has the aim to correct the position of the misplaced camera trying to move it back to its original position corresponding to its reference image. We start projecting the I_0 image from the camera position C_0 onto the object and then back-projecting onto the image plane of camera C_1, obtaining the B_1^0 image; this step is accomplished thanks to the OpenGL library and the graphics card hardware. We compute the optical flow between the back-projected image B_1^0 (what the camera currently sees) and the reference image I_1 (what the camera should see, if correctly placed) using the GPU accelerated implementation of the classic Lucas-Kanade approach, described in [126]. An example of the resulting flow is shown in Figure 2.6; here, the camera C_1 is characterized by a rigid translation from its original position. It's straightforward to notice that

Figure 2.6: The original reference image for the camera C_1, on the left; in the middle, the image B_1^0 obtained back-projecting the reference image from camera C_0 onto the image plane of C_1; on the right, the optical flow expressing the displacement between B_1^0 and C_1. Color code in Figure 1.9.

the direction of the displacement vector is more or less the same for all of the pixels; the main difference is the magnitude of the displacement. This is mainly due to both the geometry and the perception of the scene: points that are far from camera are usually characterized by small displacements, while points close to the camera that moves in the same way are characterized by a larger displacement. However, for each pixel we know exactly what is the displacement between the real pixel of the image and its back-projection: we know its magnitude and its direction. The magnitude gives an indication of the error of the alignment in this specific pixel but, if we take into account both direction and magnitude of the flow, we can understand the motion that the camera should perform in order to reduce the displacement itself. If, for example, we observe that all the pixels are characterized by displacement vector having the same direction and the same magnitude, we can assume that we need to apply a rigid translation to the camera, in the opposing direction of the flow, in order to move the real image and to fit them with the back-projected one. Unfortunately, it's quite unusual for every pixel of the image to be characterized by the same displacement; commonly, vectors follows different direction. This is mainly due to the fact that the camera could move along a line, rotate around an axis and also zooming in/out, even if we can consider this movement as a particular case of motion along a line (specifically, the $Z - axis$ of the camera reference frame). However, we need a way to estimate the motion that the camera has to do in order to minimize the flow. The inspiration for the next step comes from the branches of the robotics that deals with the automatic navigation.

2.3 Estimating the Egomotion

Assume that we've been given the original image I_1, the back-projected image B_1^0, and the optical flow OF, represented as a vector field, describing the displacement between a pixel $p_{i,1}^0$ and its I_1 corresponding pixel $p_{i,1}$. The displacement is represented by the vector v_i: ideally, if we move $p^0 i, 1$ of a quantity equal to $||v_i||$ along the direction of v_i itself, the flow between the two pixels becomes zero. But we don't want to modify ($warp$) the I_1 image in order to

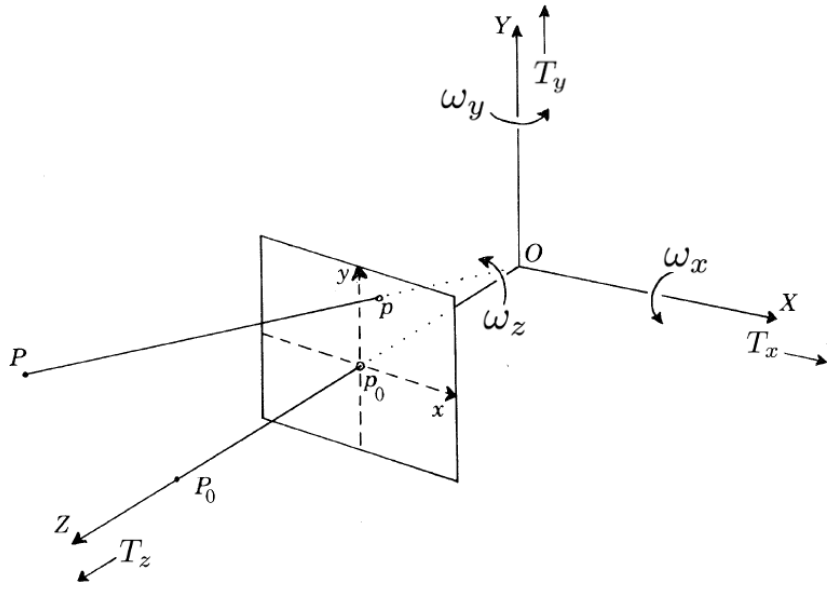

Figure 2.7: Reference frame $OXYZ$. The observer O is placed in the origin and is looking in the direction of the Z axis. The observer may move by means of a rigid motion, formed by a translation T and a rotation R.

fit the back-projection; also, we need to take into account that different pixel could be characterized by different displacements. So, the question now is: how can we minimize the flow? how we should move the image I_1 in order to make them coincide with B_1^0 as much as possible?
Suppose to have a static scene, and to take a video of this scene from a moving camera; we can then extract a number of frame F_0, F_1....F_n from the video sequence. If we compute the optical flow between F_i and F_{i-1}, what we compute is the *apparent motion* computed by the objects in the scene in the interval $[t_{i-1}, ti]$; *apparent* because the motion is actually performed by the camera itself. In robotics, the real-time computation of the optical flow has been largely use to estimate the so-called egomotion, for the automatic navigation of vehicles, robots and so on. In this section we briefly introduce the concept of ego-motion and the technique used to estimate it from a sequence of frames; then, we'll see how the egomotion estimation can be applied to our framework. The estimation of egomotion is a problem directly related with the perception of the motion of a scene, and its interpretation; this problem have been investigated since the 50's (see for example [85], that first defines the optical flow as the moving pattern of light that fall upon the retina), since there's an obvious relation between the perception of the motion and the understanding of a scene (see also Ullman and Marr, [123, 170, 171]). However only in the 80's, with the developing of the robotics, this technique has been applied to automatic navi-

gation system, with the aim of building a vehicle, equipped with one or more cameras, that should be able to move autonomously inside an environment, using the information coming from the observation of the surrounding to detect its own position and eventually modify its current motion to avoid collision, without any *a priori* knowledge of the geometry of the scene. In order to evaluate its own motion and take a decision, the system has to analyze the motion perceived by its visual sensor, and use it to understand both the surrounding environment and its own motion.
The relation between the perceived motion and the *instantaneous egomotion* has been first explored in [116] by Longuet and Higgins; various application are presented in [23, 67, 68, 83, 148, 149, 179, 183]. The main idea behind all these works is the same: computing the apparent motion of the scene by means of the optical flow between consecutive frames of the captured video sequence, and use it to compute the instant egomotion of the object and eventually change the trajectory. The optical flow has to be computed in a reasonable amount of time (ideally, it should be computed real-time) with usually limited hardware resources; it's necessary to find a reasonable trade-off between accuracy of the flow and computation time. For our purpose, we tend to privilege accuracy over fast computation time, (even if, of course, we try to keep the running time as lowest as possible), so we focus on the extraction of the information related to the ego-motion.
Suppose to have a moving monocular observer (e.g., a pin-hole camera) O, and assume that it's the center of a reference frame $OXYZ$ whose Z-axis coincide with the view direction of O (see Figure 2.7). Assume that the observer O moves rigidly with respect to the scene: its motion can be divided into the combination of a translation T and a rotation R. The translation is characterized by three components (T_x, T_y, T_z) encoding the motion in the three direction, while rotation is defined by a rotation $R = \omega \times \rho$, where ω represent the angular velocity and ρ the instantaneous radius of curvature. For the sake of simplicity, we can neglect the radius of curvature, and represent the rotational part of the motion using only its angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$. Let also $P \equiv (X_p, Y_p, Z_p)$ be a point observed by O. Then, the relative motion between O and P can be described as

$$\mathbf{V} = -\mathbf{T} - \omega \times P \tag{2.3}$$

In components:

$$\begin{cases} V_x = -T_x - \omega_y Z_p + \omega_z Y_p \\ V_y = -T_y - \omega_z X_p + \omega_x Z_p \\ V_z = -T_z - \omega_x Y_p + \omega_y X_p \end{cases} \tag{2.4}$$

From the camera model described in 1.2.1 we know that the projection $p \equiv (x_p, y_p)$ of P onto the image plane of the camera (the digital equivalent of the *retinal image*) is computed, using perspective, as:

$$x_p = f\frac{X_p}{Z_p}, y_p = f\frac{Y_p}{Z_p} \tag{2.5}$$

where f is the focal length of the camera.
The motion of p on the image plane is the optical flow (u, v) describing the apparent motion of P, can be written using 2.5 and 2.4:

$$\mathbf{v} = (u, v) = f\frac{Z\mathbf{V} - V_z\mathbf{P}}{Z^2} \tag{2.6}$$

We refer to 2.6 as the *basic equations of the motion field.* We can write 2.6 in components:

$$\mathbf{v} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{T_z x - T x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy}{f} - \frac{\omega_y x^2}{f} \\ \frac{T_z y - T y f}{Z} - \omega_x f + \omega_z x + \frac{\omega_y xy}{f} - \frac{\omega_x y^2}{f} \end{pmatrix} \tag{2.7}$$

Notice that the motion field can be seen as the sum of two components, one depending on the translation only and one on the rotation only. We can finally write 2.7 in a compact notation that emphasize these two components:

$$\mathbf{v} = \begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{Z}\begin{pmatrix} -f & 0 & x \\ 0 & -f & y \end{pmatrix} T + \frac{1}{f}\begin{pmatrix} xy & -(f^2 + x^2) & fy \\ (f^2 + x^2) & -xy & -fx \end{pmatrix} R \tag{2.8}$$

This equations gives us the relation between the moving camera, a static point in the space and its apparent motion detected by the camera, e.g the optical flow. We can use these equations into several situations: for example, if we took a number of pictures of an object, from different know viewpoints, and then we compute the optical flow, we could use the displacement vectors and the positions of the viewpoints in order to reconstruct the geometry of the scene (up to a scale factor, depending on Z), or we can use the optical flow to estimate the motion of the camera (up to a scale factor) and computing the collision time with some object of the scene, and so on. Some example can be fount in the work by Giachetti et al. [83], that handle mobile navigation based on the optical flow of a video captured by a camera, or the one by Beyeler [23] that uses the optical flow captured by a camera to handle the takeoff and landing of an helicopter.
Let's go back to our case. We have the back-projected image B_i^j and the real image I_j, and we computed the optical flow between them. What we need to know is how to move the camera C_j in order to minimize the flow, so the unknowns of our problem are, obviously, the translation T and the rotation R. Instead, we know a number of displacement vectors $\mathbf{v}$, each of ones indicating the motion described by a single pixel of B_i^j. Referring to 2.8, for each $\mathbf{v}$ we know the focal length and the coordinates of a point (x, y) on the image plane, but since we know also the geometry of the scene, we can easily obtain the depth value Z associated to each pixel of our image. The relation that we need to compute T and R from the optical flow and the geometry is then

$$\begin{pmatrix} -f & 0 & x & xy & -(f^2 + x^2) & fy \end{pmatrix} \begin{pmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = u \tag{2.9}$$

$$\begin{pmatrix} 0 & -f & y & (f^2+x^2) & -xy & -fx \end{pmatrix} \begin{pmatrix} T_x \\ T_y \\ T_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = v \qquad (2.10)$$

that leads to the formulation of a linear system having 6 unknowns (the components of T and R) and consisting of two equations for each displacement vector correctly estimated from the images. Since we have 6 unknowns, we need at least 6 equations, that is, 3 correspondences. Since we usually have a large number of correspondences, the system is usually *overdetermined*, and its solution is computed by means of least-square minimization. Notice that, after solving the system, it's possible that we need to rearrange the three components of the rotation R in order to obtain three orthogonal vectors. At this point, we know the translation T and a rotation R that we need to apply to the camera in order to minimize the flow. In order to avoid errors due to large displacement, we constrain the magnitude of both T and R, so the process could be eventually repeated for several time before convergence. In this case, we'll say that the procedure converges when the global error (the sum of the norm of the displacement vectors) does not decrease after moving the camera, that is, no optimization is possible.
This part usually represent one of the bottle-neck of the entire algorithm. Since we're dealing with high-resolution images, we have a large number of correspondence vectors; the effect is that the LS minimization involves a $N \times 6$ matrix where N is quite huge (for a 640×480 image, the number of correspondences vector is over 300000) and computing the approximate solution requires a large amount of time. We'll see in section 2.6 how we can deal with this problem.

2.4 Preliminary results

In this section, we'll briefly show some preliminary results, obtained on a series of synthetic dataset already described in section 2.2 and pictured in Figure 2.5. Each test case consists of one digital shape and two synthetic images of the shape, simulating two photos of the object captured from two different viewpoints, having a resolution of 640×480 pixels. Each image is projected onto the surface and back to the other camera, then the optical flow is computed using CUDA acceleration. From the optical flow, we build the linear system described at the end of section 2.3 and solve it using the Eigen library [89]. For each test case we report in table 2.1 the initial error (the sum of the flow vectors), the number of iteration needed to converge, the total running time in seconds and the final global error.

	Initial error	Iterations	Time	Final error	Error/pixel
Car	1480610	5	37.868s	12671	0.46
Dragon	505234	6	23.431s	1380	0.0966
Flamingo	186766	5	21.35s	2430	0.2383
Skull	3050110	4	39.774s	157704	5.007

Table 2.1: Table related to several preliminary results of the algorithm for the bundle adjustment via optical flow. In the test we adjust the position of one camera using the other camera as reference. Flow error is expressed in pixel; all the images have a 640×480 resolution, but we work only on the subset of texture pixels. It can be notice that on objects characterized by a smooth surface, the alignment leads to good result, with a small error for pixel, while for a more complex object like the skull, the alignment may have a higher residual error.

2.5 Limitations and known issues

In the previous sections, we described the idea and detail the implementation behind the algorithm for the bundle adjustment; however, in section 2.4 we have been able to present only some preliminary results confirming the goodness of the idea. Obviously, the main answer is now why we wasn't able to produce results using a real test-case, with real high-resolution images and a complex and detailed digital shape. Actually, at the present moment, there are several issues and limitations that haven't made possible to obtain satisfactory result on real data. We'll review these known problems, explaining their effect on the algorithm, while in the next section we'll describe possible ways to overcome them.
The first problem is related to the type of data on which the algorithm must perform: the images and the digital model. The images are usually high-resolution pictures of some artifacts (as stated in the introduction). The images are taken from different points of view; the effect is that, usually, the lighting is different from one point to another, so even if two images shares some part of the object, the information related to the color of this part may be different from one image to the other. This is true especially when it's not possible to take pictures in a controlled lighting environment (large artifacts, building, statues and so on). In this case, the problem is that the selected optical flow technique should be robust to lighting changes, but introducing this requirements has some drawback. The obvious choice is to use some global technique, in order to obtain a smooth and dense flow, taking into account information related to the color and the derivatives of each pixels, enforced by some other smoothness constraint. However, since we're working on high-resolution images (3 or even 4 Megapixel for each image), building and solving a linear system could be problematic due to the size of the problem. Suppose to have two pictures with a resolution of 3000×2000 pixels; if even only half of the pixel would be considered as interesting elements, the system to be solved should have about six millions equations (2 for each pixels) and the solution should consist of

Figure 2.8: The real *Skull* dataset. A number of images cover the entire surface of the object; our goal is to place the cameras in order to project the color information onto the geometry of a digital shape. Due to several problems (the texture of the skull is almost flat, the illumination slightly changes from one image to the other) our algorithm is not completely suitable to work on this type of data.

about three millions of displacement vectors. Even if, of course, it's technically possible to solve a system of this size, time and memory consumption are extremely high; probably, this choice is not the best one if we want for the algorithm to perform in a reasonable amount of time. Also, global techniques allow to detect large displacement, but they often fail when dealing with small disparities; in our case, even if it's reasonable to assume that all the pixels are characterized by similar displacement (since the motion of the camera is unique), the magnitude of each vector may be different, depending on the depth value of the original point. The obvious choice seems to implement a local technique for the computation of the optical flow, but also this choice has several drawbacks. The first one is related to the type of data we're working on.

If the object pictured in our images has poor or flat texture (see for example the skull images 2.8), computing a reliable and robust dense flow is quite complicated, since there's only a small subset of pixels whose displacement can be estimated correctly; most of the pixels belong to flat region and they will be characterized by null displacement vector, with the result that also the global error function could be biased. Secondarily, local techniques are often prone to errors, so some displacement could be estimated incorrectly; a number of bad estimated displacements could affect the solution of the linear system, and the error could propagate iteration after iteration. In our experiment, we test several techniques:

- classic **block matching** algorithm, with multi-scale Gaussian pyramid of images and taking into account both color and temporal derivatives. Results were good for regions with high texture, quite poor for the rest of the image; the quality of the results depends on several parameters like the size of the block of interest, the (optional) sub-pixel precision related to interpolation, the weights assigned to derivatives and color. We enforce the result with some Gaussian smooth filter in order to smooth the

flow without losing information on small displacement. Block matching has been implemented on CPU, since the number and the time of the operations makes it hard to be adapted for a GPU implementation: as a consequence, times are proportional to the resolution of the input images.

- original **Lucas-Kanade tracker**, implemented on the GPU as described in [126]. Time, even for high-resolution images, is sensibly small than the block matching approach. However, due the problems related to the lighting condition, the color information often doesn't match between a pair of images, and the result is usually unsatisfactory and not suitable for our purpose, even performing some pre-processing step on the images to deal with illumination issues (e.g. re-equalization of the image, removal of lighting artifacts). The quality of the results is affected by parameters like the number of level of the Gaussian pyramid and the size of the window of interest. We also tested the implementation of [108] provided by the authors, without experiencing significant improvements.

- **revised Lucas-Kanade tracker**, presented in [137], in order to deal with illumination changes and other artifacts. Results were poor in terms of both time and quality of the resulting flow; large displacement weren't correctly estimated, since most of them were detected as small displacement affected by illumination changes. Small displacement are estimated slightly better the original Lucas-Kanade tracker.

- **SIFT flow** [112], implemented using MATLAB© framework. Good quality and precision in the result, but large amount of time (> 30 mins for each couple of images) and space required to perform the computation makes them unsuitable (and unstable on certain machines).

- **Energy minimization** proposed by Brox [36], implemented using the MATLAB© framework. In this case, the flow was able to detect a smooth, common large displacement related to all the pixels involved in the process in about 7 minutes for an high-resolution image, but the lack of precision for pixel characterized by small movements affect the estimation of T and R, that are estimated incorrectly (typically, the estimated movement is larger than it should be) and the effect is that the algorithm converges without finding a good minimum: the global final error is still too large.

- **Farneback's algorithm**, detailed in [141] and provided by the OpenCV library [28]. The results are affected by same problem of the local trackers: the algorithm is not able to deal with illumination changes in the sequence of images.

It can be notice that every technique we tested has both advantages and disadvantages; the best solution will probably be a hybrid technique; we will describe two possible approaches in the next section.
In general, however, the whole pipeline requires a running time that is significantly higher than the state-of-the-art methods for solving the same problem,

which makes it, at the moment, not suitable. So another factor to take into account is it's computational cost: the computation of optical flow between a pair of image with $N \times M$ pixels can require a large amount of time; the same consideration holds for building and solving the linear system, whose size is directly related to the resolution of the image. Iterating this process requires a huge amount of time, that needs to be reduced in some way.
Also, since we have introduced this aspect, the solution of the linear system could be affected by numerical error in some cases, especially when the observed displacement vectors aren't reliable, and this usually has two main consequences:

- the estimated values of T and R are incorrect, and has the effect of moving the cameras in a completely wrong position, such that even iterating the procedure, it's impossible to recover its correct position;

- the algorithm may converge at a local minimum of the global function error 2.2 instead of its global minimum (a problem common to lots of heuristic approaches).

Both problems can dramatically affect the results of the algorithm, since at the end of the iterations the camera is placed in a position where, sometimes, it can't even see the scene.
Speaking of local minimum that doesn't allow to recover a good positioning of the cameras, let assume to have two camera C_0 and C_1 with an initial poor alignment. Suppose to project image I_0 from C_0 onto the digital object: if the initial alignment is quite poor, the color information projected onto the object hit the surface in a wrong way. For example, they could be shifted respect to the real object. We then back-project this information onto C_1 and use the obtained image to compute the displacement. The result could be that we find a rigid motion to apply to C_1 that brings I_1 to coincide with a wrong back-projected image. In other word, the risk is that moving the camera, we'll find a position well-fitting for wrong data, without having any way to distinguish this situation from the expected, correct one. This error affect also the distribution of the alignment across the different cameras. Assuming that we first align cameras C_0 and C_1, then C_1 and C_2 and so on. If the alignment between C_0 and C_1 carries some error, due to one of the previously listed reasons, or simply because it's practical impossible to achieve a perfect alignment; the obvious effect is that this error will affect the computation of C_1 and C_2, whose alignment will, again, be affected by some error, that is accumulated in the global error function, and so on. The accumulation of the error may introduce difficulties into the alignment of the last cameras; ideally, it should be instead distributed over the different cameras, and we should also find a way to evaluate and eventually correct the initial positioning of the cameras, in order to be sure that the error affecting the alignment is as small as possible.
The next section presents some hypothetical solutions to these problem. Final considerations about this chapter are proposed in section 4.1.

2.6 Improvements and future works

The idea behind the algorithm described so far is someway correct, since, under certain assumptions and conditions, we have the empirical proof that it can be used to reduce the global error alignment. However, as we just explained, there is a number of issues and limitations that must be some way overcome if we want to make the algorithm robust, versatile and reliable. In the following, we discuss several possible improvements and their expected impact on the algorithm.
The first possible improvement comes from the observation that we know exactly the geometry of the scene (the digital object) and the current position of the cameras in the common reference frame. As a consequence, we can add to the framework also the information provided by the **epipolar geometry** of the scene. The epipolar geometry is the geometry describing a stereo system composed by two cameras, placed in two different points, and a generic three dimensional point P seen by both cameras. We briefly recall some of the basic concepts of the epipolar geometry in order to see how them will fit into our framework; for a more detailed description of the topic, we refer to the book by Trucco and Verri [168].

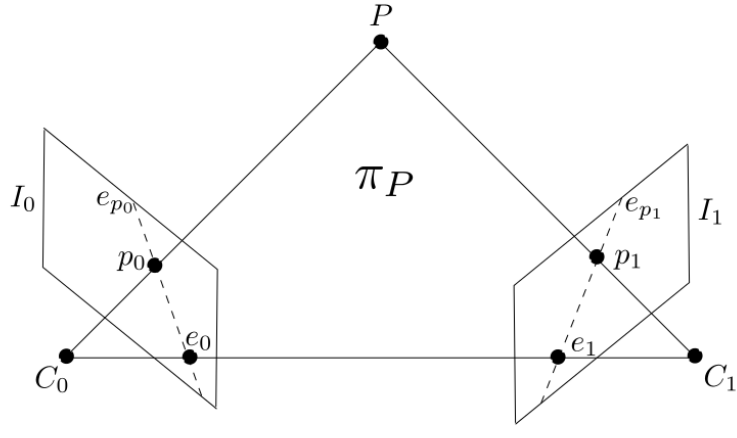

Figure 2.9: Essential elements of epipolar geometry: cameras C_0 and C_1, a point P and its projections p_0 and p_1 on the image planes, the epipolar plane π_P, the epipoles e_0 and e_1 and the epipolar lines e_{p_0} and e_{p_1}.

Given a pairs of stereo cameras C_0 and C_1, and a point P in a 3-D space, we call *epipolar plane* the plane π_P passing through P and the center of projection of the cameras (Figure 2.9). The lines where π_P intersects the image planes are called *conjugates epipolar lines* (or simply epipolar lines); the im-

age in one camera of the projection center of the other one is called *epipole*. From this data, and knowing the position of a number of 3D points on the two image planes, we can, for example, estimate the roto-translation that brings one camera onto the other; we can also estimate extrinsic and intrinsics parameter of the cameras, up to a scale factor, using the so-called Eight-point algorithm [115]. However, we are more interesting in the fact that the epipolar geometry could be used to correct and make easier the computation of the flow and also the camera adjustment. Starting with a certain point P and its two corresponding pixels p_0 and p_1 (as pictured in 2.1); we already know that the back-projection of p_0, p'_1, could be different from p_1, and that their displacement is represented as an optical flow vector. If we examined the optical flow vector $v = p'_1 - p_1$ with respect to the epipolar line e_{p_1} related to the point P, we could write v as the sum of two vectors v_o, v_e, the first one perpendicular to e_{p_1} that we call *orthogonal flow*, and the other one corresponding to the projection of v onto e_{p_1} that we call *epipolar flow*. Moving the camera in order to minimize the orthogonal flow, the estimation of the displacement between p_1 and p'_1 becomes easier, since it becomes a simple search on a 1D line, that is, the epipolar line related to the point P. So the first improvement is to move the cameras in order to minimize the orthogonal flow over the entire image, using a procedure similar to the one used for the rectification of the image [78]. After this first adjustment, we can easily compute the flow, and use it to move the camera, but we can also simplify this step, since we now know that, in order to reduce the flow, we can constrain the translation T on the epipolar plane related to P, and we can constrain the rotation R to be around the normal of π_P. Computing the solution of the system can be, in a certain sense, simplified by these assumptions.

The problem of computing the optical flow in a reliable and (possibly) fast way is another main problem that has to be faced. We already stated that, due to the type of data we have to work on, we need to choose for a method able to handle with illumination changes and to estimate sub-pixel flow, but we cannot apply global techniques directly on the full-resolution images due to an extreme computational cost. We propose an hybrid approach to the problem, that can be summarized in using a global technique on a sub-sampled pair of image in order to estimate the large displacements (the highest levels of the Gaussian pyramid of images), and then perform the computation on the high-resolution images (lowest levels) using some local technique. Both procedure should be able to deal with lighting changes, so they must carry on the derivatives into their computations, and the estimation of the flow with sub-pixel precision can be achieved by means of image interpolation. The two selected techniques could be the flow computed using the Brox energy minimization [35], for the low-resolution images, and an *ad-hoc* block matching technique. The global technique, performed on low resolution images, is less expensive, and using the block matching technique on the full-resolution images should preserve the finest level of detail. Eventually, we can add a third step to this pipeline, that is, after the computation of the low-resolution global flow, use the estimated flow to compute a first approximate roto-translation, apply it to the camera,

and then perform the high-resolution block matching on the images obtained by the new cameras configuration. In this way, large displacement computed from low resolution images will be used to move cameras in order to remove the largest vectors, leaving only small displacement that will be estimated by means of the local technique.
Another essential feature that we need to add to our algorithm is an estimation of the quality (and, consequently, of the reliability) of each displacement vector computed. We need a way to distinguish between poor estimated vectors and good ones. After performing tests using different quality measure, we found out that, in this case, we can estimate the quality using the eigenvalues of the Harris matrix 1.19, as described in section 1.3.2. The eigenvalues gives a measure of the reliability of the vector in both the directions; assigning a quality to each vector allows to improve the efficiency and the stability of the linear system built using 2.9 and 2.10, reducing the number of entries: we could select only a subset of pixels of the original image, characterized by the highest quality values, and then use only them to build a smaller linear system, that could be solved faster and in a more robust way, since all the values are good and we don't have to deal with noise, incorrect observations and so on. Eventually, when the quality is good for one direction of the vector only, we can introduce in the system only partial information deriving from the u or v component of the vector. Eventually, the computational cost of solving the linear system could be reduced using techniques like the GPU acceleration, that doesn't reduce the computational complexity but allows to obtain results in a smaller amount of time.
Finally, we deal also with the accumulation of the error, re-defining the pipeline described in Figure 2.1. Instead of working on C_0 and C_1, then C_1 and C_2 and so on, we partition the set using a balanced binary tree. Each pair of leaves of the tree is a camera, and alignment is performed in a bottom-up feature. We first align leaves with the same father; ideally, we should align C_0 and C_1, then C_2 and C_3 and so on; there is no overlap between the pairs. Then, we visit the upper level of the tree: now each node has two children, corresponding to two pairs of aligned cameras. We then select one camera from each pair in order to obtain a new pair of cameras having the possible maximum overlap, and use this new pair to compute the alignment of the two children, eventually moving the cameras that are already aligned, if needed.
At the end of the alignment process, the error will unlikely be zero; there will be residual value that cannot be decreased simply adjusting the position of the cameras. In this case, the alternative way to decrease the global error is to modify the underlying geometry of the digital object, moving or adding vertices to the object in order to affect the back-projection of an image onto the other and, as a consequence, reducing the displacement vectors and minimizing the global error. This technique performs similarly to the stereo reconstruction techniques [168] that allow to recreate the geometry of a scene from a set of images, with the difference that, in our case, the geometry already exists and we already know the correspondences between the pixels, thanks to the optical flow. The position of the new pixel could be estimated finding the (approxi-

mate) intersection between the line passing through the camera center C_i and the pixel p_i, and the line passing through C_j and the pixel p_j corresponding to p_i (that is, the pixel such that the back-projection of p_i is characterized by a displacement vector ending in p_j). In this way, we should achieve the goal of improving both the camera alignment and the object geometry, making it more detailed and precise.
However, only a part of the described improvements has been massively tested, especially the ones related to the choice of the optical flow technique to be used; most of them are discussed also to point out that, even solving some of the problem detailed in the previous technique, the resulting technique will probably be affected by other, different problems, and in general it's required to increase the complexity of the framework. We'll remind the reader to section 4.1 for final consideration about the work presented in this chapter.

Chapter 3

Motion-based mesh segmentation

3.1 The mesh segmentation problem

The problem of partitioning a three dimensional object into meaningful parts, usually known as *mesh segmentation*, is a challenging problem, with a large number of applications in geometry processing, shape analysis, shape compression and so on. The main idea behind the mesh segmentation is to compute some value on the elements of the three-dimensional mesh (faces, edges or vertices) and then define some measure (curvature, distance, shape-diameter [106], diffusion distance [30, 53] and so on) in order to find similar elements and finally to partition the mesh in several component, each of them is characterized by similar elements and it's distinct from the other parts. However, since the notion of "meaningful parts" is ambiguous, there's not a unique solution to the problem. In literature, a number of different partitioning methods has been proposed. One of the fundamental work in mesh partitioning has been proposed by Katz and Tal ([104]); in this work, each face of the mesh is characterized using geodesic distance from all the other faces, then a number of significant face is selected and used as *seeds* for a *clustering* algorithm. The idea of classifying the element using some measure and then perform partitioning using some clustering technique, such as *adaptive k-means* clustering, or

fuzzy clustering, or *mean-shift clustering* and so on, become in the years the standard way to proceed in order to solve the problem. The main advantages of this approach are its versatility, since it's possible to use the very same approach with different similarity measure, and the fact that it leads to results quite good in terms of visual appearance of the different parts. Unfortunately there may be also some significant drawbacks; for example, in [104], to reduce the time complexity of the algorithm, the authors build a matrix containing the distances between pairs of vertices, called *all-pair shortest path* matrix (or simply *APSP*), increasing the expensiveness in terms of space and time; also, the dependence from parameters does not allow to achieve a fully automatic technique, and it's necessary to spend some time on tuning the parameters. The correct tuning of the parameters can be difficult (and, in a certain sense, frustrating), and it's quite rare to find a combination of parameters performing well for every type of object. However, during the years several improvement of [104] have been proposed (see for example [103], that use parallelization on graphics card to optimize the algorithm) alongside with completely different approaches, such as [79] or [10] that perform a hierarchical clustering on the mesh: starting from the initial cluster (that is, the whole object), smaller parts of the object are identified and divided, then each part is, in turn, examined and subdivided into smaller pieces, until no more splitting is possible. Cohen-Steiner *et al.* [52] propose a different k-means clustering approach, based on the variational properties of the shape. Another interesting technique proposed by Lai *et al.* [107], derived from the engineering, uses the *random walks*; basically, for each face, we compute the probability of reaching all the other faces of the mesh, using a probability measure obtained as combination of both geodesic distance and discrete curvature. A face F_a has high probability of reaching the face F_b if the face F_a it's close to F_b in terms of geodesic distance, and the path between F_a and F_b doesn't pass through significant changes in mesh curvature. The main advantages of this technique are the ease of implementation and a low computational cost, but unfortunately, it's not so versatile as the previous described techniques, and it's affected, in some cases, by numerical instability. *Mesh scissoring* (Lee *et al.*, [110]) identifies small groups of significant vertices on the mesh (that is, vertices where the measure value changes significantly) and use them as starting points to detect the boundary between relevant parts; then, the boundaries are computed and refined using the adaptive snakes technique, borrowed from the image processing field. For a detailed analysis of these techniques, we refer to the survey by Shamir [156].

3.2 Motion-based Mesh Segmentation

In recent years, a different task rose: the segmentation of a three-dimensional object, based not on a single, static mesh, but on a given set of different poses (deformations) of the object, with known point-to-point correspondences. This task is usually referred as *motion-based mesh segmentation*, *rigid-part mesh segmentation*, or *dynamic mesh segmentation*; the main goal is to partition the

mesh into parts that move coherently over the different poses. Such segmentation problems have important applications in computer vision [54, 93, 128, 151, 154], graphics [60, 100, 109], mechanical engineering, and biomechanics [5, 44]. The main difference with the classic mesh segmentation is that, in this particular case, the measure characterizing each element of the mesh comes from a *temporal sequence*, and not from a single static mesh. For example, instead of using discrete Gaussian curvature computed on some vertex v, we compute the variance of the curvature on the same vertex during the sequence, or the average value, or, again, the maximum difference between curvature value along the different poses. The elements of the mesh are classified using some measure that reflects the behavior of the elements over time. Notice that the motion-based segmentation could lead to a partitioning different from the segmentation computed on the static object. Assume that we are trying to segment the mesh of an animal, and suppose that, in the different poses, one leg stays put; then, that particular leg will be detected as a single rigid part of the object, even if intuitively it should be probably split in two parts, at knee height, since there is no motion information in that specific part of the shape. Of course, hybrid approaches are possible, and information derived from static analysis of the original object can be combined with the ones coming from the temporal sequence, but the achieved results does not really solve the problem.

Before going further, we need to formally define the problem of the motion-based segmentation. Let $\mathcal{K} = (X, E, T)$ be an abstract simplicial complex with a set of n nodes $X = \{x_1, \ldots, x_n\}$, a set of edges E, and a set of triangles T, such that any edge $e = (i, j) \in E$ between nodes x_i and x_j has exactly two adjacent triangles $t, t' \in T$. Any mapping $\Phi\colon X \to \mathbb{R}^3$ that associates with each node $x_i \in X$ a 3D vertex v_i then yields a manifold triangle mesh $M = (\Phi, \mathcal{K})$ with *geometry* $V = \Phi(X) = \{v_1, \ldots, v_n\}$ and *topology* $\mathcal{K}$. Suppose we are given N manifold triangle meshes $M_k = (\Phi_k, \mathcal{K})$, $k = 1, \ldots, N$ with the same topology, which represent N different poses of some shape. We assume that all poses can be segmented consistently into l *rigid parts* and m *deformable joints*. Then there exists a decomposition of the set X into disjoint sets $S_1, \ldots, S_l$ and $J_1, \ldots, J_m$, such that the corresponding patches of the meshes M_k with vertices $\Phi_k(S_i)$ and $\Phi_k(J_j)$ are related by suitable rigid or non-rigid transformations, respectively. The main goal of motion-based segmentation is to find the rigid parts S_i of the shape, using only the information given by the N meshes $M_1, \ldots, M_N$.
This problem can be tackled in two opposite ways:

- detect the deformation applied to each node, and then group them into cluster locally characterized by the same rigid deformation;
- find the nodes where the deformation occurs the most, classify them as joint regions J_j and then extract the rigid part (actually, in most of the approaches, the joint regions J_j are neglected and their nodes distributed to the nearest rigid part of the shape)

The assumption of the known point-to-point correspondence may appear to

be too restrictive, but it's actually a pretty common and reasonable requirement in this kind of problem.
James and Twigg [100] try to solve the problem of rigid part segmentation using the first type of approach. Assuming to have the temporal sequence of n different poses $M_1, ...M_n$. For each face T_k, the algorithm estimate a trajectory obtained as the concatenation of $n-1$ significant rotation of T_k in the sequence. Each rotation is then projected into a 9-dimensional space, and finally a clustering is performed in order to detect the parts of the object characterized by similar movement during time. The rigid parts are then used to create a skeletal representation of the object, where rigid parts corresponds to the *bones* and non-rigid parts corresponds to the *joints* between bones. As a matter of fact, the algorithm extracts, from the sequence of poses, a sequence of skeletons encoding the deformation and the motion of the shape in the sequence. A conceptually similar approach has been proposed by Lee *et al.* in [109]. In their work, face clustering is performed as the partition of the *weighted dual graph* $\mathcal{G}$ of a mesh, where the weights of the arcs are computed using both deformation and geometric information, in order to obtain the connected components representing the rigid parts of the shape. A different approach has been proposed in [60] by de Aguiar *et al.* In this work, the detection of the rigid parts is just a single step in a most extensive algorithm; however, de Aguiar had the intuition of classifying the vertices by examining how a certain vertex v moves with respect to its neighbors. A special *affinity matrix*, encoding the relationship between a vertex and its neighbors, is built, and then spectral clustering is performed on the matrix itself. Notice that, differently from all the other works here described, [60] is the only one to perform clustering on the vertices instead of the faces. Rosman et al. [151] represent motion as a group-valued function on the shape and determine a segmentation by using a total variation-like functional.

The approach of Wuhrer and Brunton [176] solves the problem as follows. Given the meshes, it computes the *dual graph* $\mathcal{G}$ of the common simplicial complex $\mathcal{K}$, in which a node corresponds to a triangle of the original mesh, and an edge connects two adjacent triangles. Each edge of $\mathcal{G}$ is then weighted by the maximum variation of its corresponding dihedral angles across all poses. In this way, dual edges with small weights correspond to rigid parts of the mesh, while dual edges with large weights correspond to deformable joints. To determine the l rigid parts of the shape, the dual graph $\mathcal{G}$ is partitioned by computing its minimum spanning tree, and then cutting off the dual edges with the l largest weights. The number of segments is usually unknown and can be computed at runtime. The possibility of *over-segmentation* requires to perform an additional merging step in order to obtain reasonably sized segments. The complexity of the first step of the algorithm is $\mathcal{O}(N^2 n + n \log n)$, while the merging step, which consists of the reinsertion of some of the edges into the spanning tree, has $\mathcal{O}(n \log n)$ complexity. Two examples of motion-based segmentations are presented in Figure 3.2.

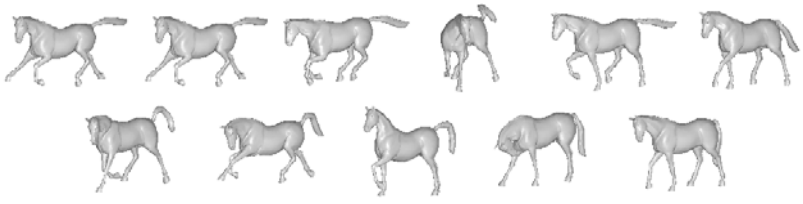

Figure 3.1: The horse test case, consisting of 10 different poses of the same shape.

Figure 3.2: Two examples of the motion-based segmentation performed on the horse test case (Figure 3.1). On the left side of the image, segmentation computed by [151], on the right the one computed by [176].

The work here presented differs from all the previous approach, since it uses information from the visual perception of the different poses of the shape to detect the parts where the deformation occurs. The main motivation for our approach comes from the following observation. Assume that we are given a curve going through the rigid parts S and S' and a joint J connecting them. Then, if the shape is articulated at this joint, we will observe the two parts of the curve undergoing a rigid transformation. As a result, the curvature of the curve at the joint will change, but will remain constant at the rigid parts (see Figure 3.3). Given a set of curves that cover all rigid parts of the articulated shape and observing them in different poses, we will be able to segment the shape according to its motion.

This idea is rooted in Marr and Ullman (see [170], [171] and [123]) and the perception of the neural motion. In [170], Ullman proves that if a body is rigid, we can recover its three-dimensional structure from only three frames, up to a reflection. Starting from this theorem, and assuming that we are dealing with objects that are *locally rigid*, it's possible to formulate the so-called *rigidity assumption*, stating that *any set of elements undergoing a two-dimensional transformation that has a unique interpretation as a rigid body moving in space is caused by such a body in motion and hence should be interpreted as such.* The rigidity constraint also helps to distinguish between objects moving in the same scene. Finally, if we consider an object as a collection of smaller object, each of them moving rigidly (corresponding to the rigid parts), we can take advantage of the previously mentioned assumption to detect motion of the shape using

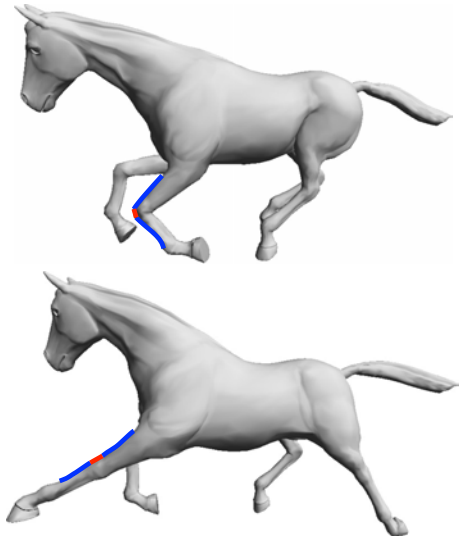

Figure 3.3: Motivation for our approach: shown are two poses of the horse shape and a curve passing through two articulated parts (upper and bottom part of the leg) connected by a joint (knee). The curve can be easily segmented into rigid parts (blue).

only the orthographic projections of its boundaries.

3.3 Algorithm

Our approach to the shape segmentation from motion aims to identify the joints of the shape, and then use this information to detect the rigid parts of the object. The algorithm can be subdivided in three main step:

1. extract a set of 1D curves, named *augmented silhouettes*, from different shape poses and different viewpoints;

2. analyze each curve using motion information, in order to detect where the deformation occurs the most

3. combine the analysis of the 1D curves to obtain the final segmentation of the object.

The algorithm is summarized in Figure 3.4. As we see in the following, while understanding the first two step is straightforward, finding a way to combine the monodimensional information is a not trivial task . In the following sections, we will describe in detail each of the steps, providing several unsuitable approaches for the third step, pointing out their weakness, and finally showing how the clustering can be efficiently performed.

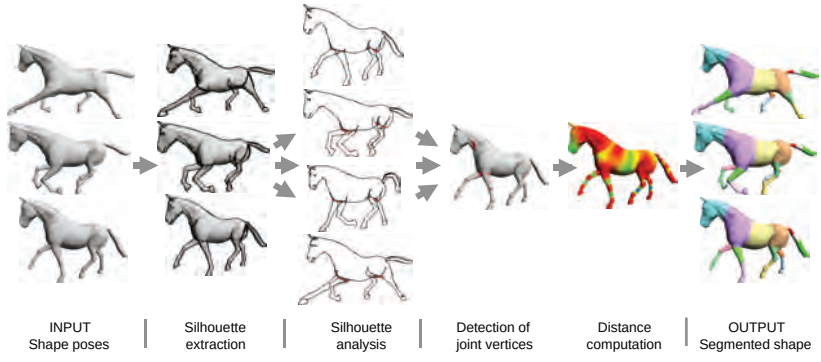

Figure 3.4: The different stages of the proposed motion-based segmentation algorithm.

3.3.1 Augmented Silhouette Extraction

Though in principle our approach can work with any set of curves, we use a specific data structure that contains information about the perceptively relevant edges of the shape, referred to as the *augmented silhouette*; the first step of the algorithm is to build this data structure, containing information about the *perceptually significant* shape edges.

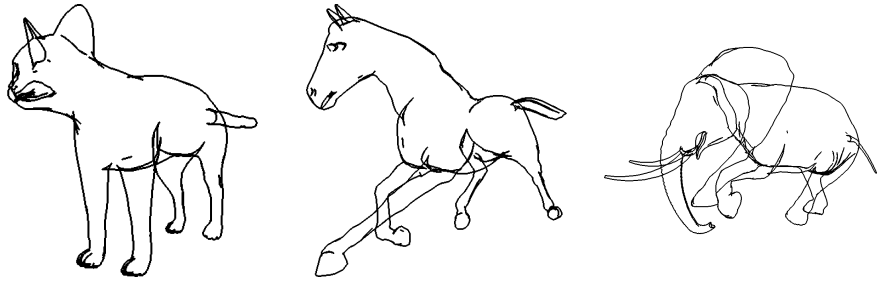

Figure 3.5: Augmented silhouettes extracted from several shapes.

Given K view points $\mathcal{V}_1, \ldots, \mathcal{V}_K$, the augmented silhouette $\mathcal{S}_{j,k} = (X, E_{j,k})$ of mesh M_j with respect to view point $\mathcal{V}_k$ is a graph with nodes X and edges $E_{j,k} \subset E$. An edge $(i_1, i_2) \in E$ is an element of $E_{j,k}$ if and only if the corresponding mesh edge $[v_{j,i_1}, v_{j,i_2}]$ is *perceptively relevant*, that is, exactly one of the two adjacent faces is visible from view point $\mathcal{V}_k$. Since we do not take occlusion into account, the augmented silhouette usually contains more than just the silhouette edges of M_j with respect to $\mathcal{V}_k$, hence the name (see Figure 3.6 and 3.5).

The classification of edges into perceptually relevant and irrelevant is simple and can be done in parallel for different edges and for different view points. We first process all triangles of M_j in parallel and determine the signs of the dot products between each triangle normal and the viewing directions of the view

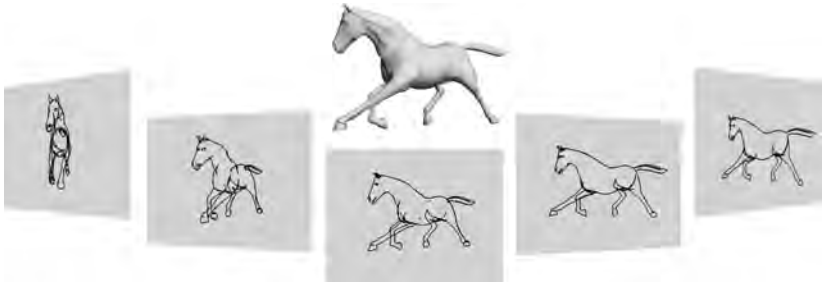

Figure 3.6: Examples of several augmented silhouettes extracted from a 3D shape of the horse for different view points.

points $\mathcal{V}_k$. Then we process all edges of M_j, compare the signs of the adjacent triangles, and classify an edge as perceptively relevant if and only if the signs are different (see Figure 3.7).

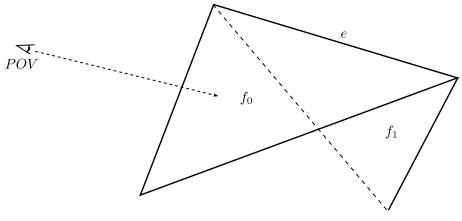

Figure 3.7: A perceptually significant edge. The face f_0 is visible from the viewpoint, but its adjacent face f_1 is not; the shared edge e is then detected as significant.

The main advantage due to picking perceptually relevant edges without taking into account the occlusions is that we are able to consider a large number of edges belonging to a larger number of part of the shape, so we actually need less silhouettes to detect all the rigid parts of the shape. In Figure 3.8 we propose a comparison between a silhouette and and augmented silhouette of the same object.

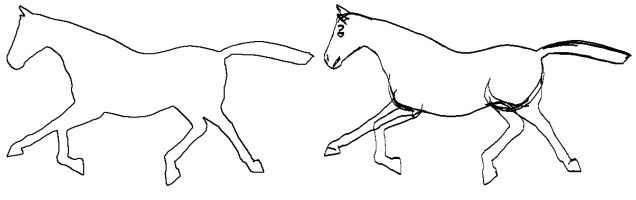

Figure 3.8: A standard silhouette and an augmented one, captured from the same point of view.

The algorithm that summarize the extraction of the augmented silhouette

is described in *AugmentedSilhouette*.

Algorithm *AugmentedSilhouette*
1. **Input:** a three-dimensional mesh S, a point of view POV.
2. **Output:** a graph containing the perceptually relevant edges of S.
3. Initialize an empty graph G
4. **for** each edge e_i of S
5. **do** get incident faces f_{i_0} and f_{i_1}
6. **if** *is_visible*$(f_{i_0}$,$POV) \oplus$ *is_visible*$(f_{i_1}$,$POV)$
7. **then** get the vertices v_{i_0} and v_{i_1} of e_i
8. **if** v_{i_0} is not linked with G
9. **then** add a node n_{i_0} to G linked to v_{i_0}
10. **else** get the node n_{i_0} corresponding to v_{i_0}.
11. **if** v_{i_1} is not linked with G
12. **then** add a node n_{i_1} to G linked to v_{i_1}
13. **else** get the node n_{i_1} corresponding to v_{i_1}.
14. Add an arc connecting n_{i_0} and n_{i_1} to G and link it to e_i
15.

In our experiments we use $K = 25$ view points, with positions taken from a subdivided icosahedron that circumscribes all poses and viewing directions towards the center of this icosahedron. Moreover, we use only K' randomly chosen augmented silhouettes from all KN possibilities and found a value of K' between 50 and 100 to be sufficient for correctly identifying the rigid and deformable parts. We also found out that increasing the number of POVs doesn't affect significantly the final results, since the distribution of the edges over the silhouettes stays more or less unchanged: most of the vertices will appear in just a few silhouettes, while only a small subset of them frequently appears (see Figure 3.9).

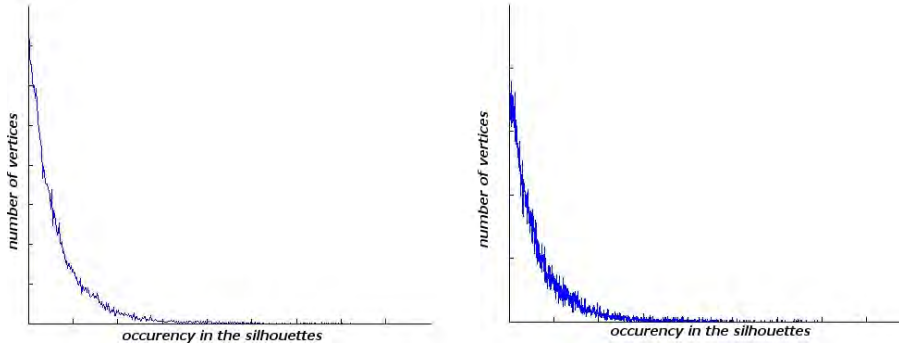

Figure 3.9: Distribution of the edges of the same deformation sequence over the silhouettes. Pictured on the left, the distribution of the edges over 420 silhouettes; on the right, the distribution over 1680 silhouettes.

Typically, each augmented silhouette contains only $\mathcal{O}(\sqrt{n})$ edges and nodes, so the overall size of all silhouettes is on the order of $K'\sqrt{n}$, which is significantly

less than the Nn vertices from all N meshes, because $K' < KN \ll N\sqrt{n}$, especially for large meshes.

3.3.2 1D analysis

At this point, we have collected a number of augmented silhouettes data structure, each of them linked to the elements of the three-dimensional object. Now, in order to better understand how the shape moves during the temporal sequence, we analyze the way the edges of the augmented silhouettes moves along the different poses. As a matter of fact, there's a direct correlation between the movements of the silhouettes and the movements of the objects, since it's quite obvious to notice that the movement of a silhouette edge during time is the result of the movement of the corresponding edge of the object, and vice-versa (see Figure 3.3); ergo, to each movement of the three dimensional model corresponds a variation of the silhouette. More precisely, if the move occurs in a particular vertex v of the shape, it will be visible at least in one silhouette containing a node linked to v. The task becomes now to detect the subset of vertices involved into the deformation, labeling them as *significant vertices* In order to do so, we consider each of the K' selected augmented silhouettes from the first step separately.
Suppose that x_i is a node with neighboring nodes x_{i_1} and x_{i_2} in such an augmented silhouette $\mathcal{S}_{j,k}$. All three nodes have corresponding 3D vertices $v_{l,i} = \Phi_l(x_i)$, $v_{l,i_1} = \Phi_l(x_{i_1})$, $v_{l,i_2} = \Phi_l(x_{i_2})$ in each of the N meshes M_l and related 2D vertices $w_{l,i}, w_{l,i_1}, w_{l,i_2}$ in the projection with respect to view point $\mathcal{V}_k$. We then compute the angles

$$\alpha_l = \sphericalangle(w_{l,i_1} - w_{l,i}, w_{l,i_2} - w_{l,i})$$

between the two edges incident to w_{l_i} in the 2D projections of the N poses and assign the maximal difference

$$\max_{l=1,\dots,N} \alpha_l - \min_{l=1,\dots,N} \alpha_l$$

of these angles as a *deformation weight* to the node x_i. In this way, the nodes which are likely to correspond to joint vertices of the shape are the ones with the largest weights. The deformation measure is similar to the one proposed in [176] with the difference that we analyze dihedral angles between edges instead of angles between faces. This computation is not carried out for nodes x_i with less than two neighbors in $\mathcal{S}_{j,k}$, and for nodes with more than two neighbors we compute the difference between the maximum and the minimum of all angles formed by any pair of adjacent edges.

After assigning a deformation weight to each node, we further analyze the augmented silhouette and identify the *significant* nodes by iterative adaptive clustering (more details in algorithm *AdaptiveClustering*).

Algorithm *AdaptiveClustering*

1. **Input:** an augmented silhouette S and the deformation values def of its vertices.
2. **Output:** the augmented silhouette S, where each vertex is labeled as significant or not significant.
3. $seed_0 := 0$, $seed_1 := max\{def(v)|visavertexofS\}$,
4. $C_0 = \emptyset$, $C_1 = \emptyset$
5. **while** C_0 and C_1 does not stay unchanged
6. **do for** each vertex v of S
7. **if** $|def(v) - seed_0| < |def(v) - seed_1|$
8. **then** add v to C_0
9. **else** add v to C_1
10. $seed_0 := avg(def(v_i)), v_i \in C_0$
11. $seed_1 := avg(def(v_j)), v_j \in C_1$
12. Label all the vertices of C_1 as significant and all the others as not significant.

Typically, the process requires 6 or 7 iterations to converge; at the end, cluster C_1 contains the significant vertices of the augmented silhouette. This process does not require any parameter and leads to better results than using a simple thresholding, since it's less sensitive to noise and tends to mark as significant only few vertices, highlighting the parts of the silhouette where most of the deformation occurs. Note that a node $x \in X$ may be classified differently in each of the K' augmented silhouettes: either as significant, as insignificant, or not classified at all, if it has less than two neighbors; this obviously depends from the fact that x could be characterized by different neighbors in different silhouettes. In Figure 3.10, we highlighted in red the significant vertices of the silhouettes pictured in Figure 3.5.

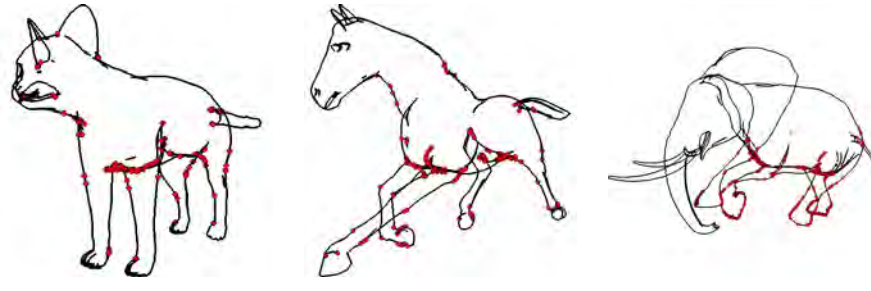

Figure 3.10: Significant vertices (marked in red) of the augmented silhouettes of Figure 3.5.

As we compute N angles for each pair of adjacent edges in each augmented silhouette, the overall complexity of computing all deformation weights is on the order of $K'N\sqrt{n} < K^2N\sqrt{n}$, which is significantly less than the $\mathcal{O}(N^2n)$ operations needed in the algorithm of Wuhrer and Brunton to compute the weights for the edges of the dual graph. The additional computational cost for the clustering processes is negligible, as it is only on the order of $K'\sqrt{n}$.

3.3.3 Mesh Partitioning

After we collect the information from the 1D data structure, we need to step back to the original geometry of the shape, and work directly on its original element in order to find the clusters. This part is the less intuitive, since it wasn't immediately clear how the data could be used to produce the clustering. In the following, we exploit three unsatisfactory approaches, inspired to the random walks segmentation, the hierarchical face clustering approach and the Katz-Tal approach, briefly discussing the ideas behind each of them, with focus on advantages and limitations. Finally, we describe a fourth approach that performs well in terms of both quality of the segmentation and running time of the algorithm.

Random Walks

The first approach we tried to partitioning the elements of the mesh is based on the *random walks* technique, borrowed from the image processing [87] and first used in [107] for static mesh segmentation by Lai *et al.* The idea behind the basic algorithm is quite simple: given a face of the mesh T_i, and a significant faces (or *seed*) $\mathbf{T}_k$, what's the probability that a random path $\mathbf{P}_i$ starting from T_i will pass through $\mathbf{T}_k$? And, given a number of seeds $\mathbf{T}_1, ...\mathbf{T}_n$, which seed has the highest probability of being reached first by path $\mathbf{P}_i$? Obviously, T_i will belong to the cluster C_k whose seed $\mathbf{T}_k$ has the highest probability of being reached by $\mathbf{P}_i$. In order to compute the probability of the random walks, each edge has to be weighted using a *probability measure*: high-weighted edges have good chances to be the ones where the walks pass through, while edges with low probability will block the path, forcing it to pass through a different edge. Therefore, for each edge e of face T_k, we define a probability value $p_{i,k}$, such that $\sum_{i=1}^{3} p_{i,k} = 1$. The probability that the path $\mathbf{P}_i$ starting from face T_i reach the face T_k, $k \neq i$, is defined as

$$P^k(T_i) = \sum_{j=1}^{3} p_{j,k} P^k(T_{i,j}). \tag{3.1}$$

Here, $T_{i,j}$ is one of the faces adjacent to T_i. The probability that a path starting from T_i reach T_i itself is 1. Fixing a number of seed, and using 3.1, we can build a large sparse linear system whose solution is the vector containing the probabilities to reach the seed $\mathbf{T}_k$ randomly walking from each face T_i. Of course, the whole idea requires to careful assign the probabilities values to the edges, in order to give advantage to path satisfying some particular conditions: for example, if one wants to privilege paths that don't pass through discontinuities in the shape, then probably he will obtain better results if he assign high probabilities value to edges where the transition from one face to the other is smooth, and low probability in presence of a discontinuity (e.g. concave dihedral angles), according to the *minima rule* for mesh segmentation [95].
In our cases, the extension is straightforward: we want that a random walks starting from face T_i pass through edges characterized by small deformation

value, avoiding to pass through high deformation regions. The idea, in fact, is that a seed should be reached only by paths starting from faces that are in the same rigid parts, that is, paths that don't pass through high-deformation regions. First thing that needs to done is then to estimate, in some way, the amount of deformation for each edge $e \in E$. We simply stated that, given an edge e_i, its deformation is computed as the average of the deformation value of its vertices v_k and v_j, computed as described in section 3.3.2. This value can be multiply by the number of times that v appears on the silhouette (occurrences vector) in order to give higher importance to vertices that appear frequently and are more reliable. We assign probabilities to each edge, keeping in mind that edges characterized by null deformation are shared between faces that must be part of the same cluster and therefore must have low probability. Finally, we build and solve a sparse linear system in order to find, for each face, the probability to belong to each cluster.
Unfortunately, this approach turned out to be not suitable for several reasons:

1. the number of seeds has to be known at the beginning of the computation, which is a condition that usually cannot be satisfied.

2. unreliability of the edge deformation value: as a matter of fact, we compute the measure as the average of vertices deformation, but since the deformation of a vertex may be the consequence of a deformation related to different edges, we have no guarantee that this is a good estimate for the edge deformation measure. The edge could actually stay unchanged and be characterized by a high deformation value even if deformation occurs only in its neighborhood.

3. numerical instability, that leads this technique to perform well for a limited number of seed, and to poor result for more than 7-8 seeds, results. Also, connectivity of clusters is not guarantee, and in some cases results could be classified as nonsense (e.g., a seed could belong to a cluster different from its own).

However, it's quite reasonable to think that, if we correctly characterize each edge $e \in E$ with a reliable deformation measure, this algorithm could lead to better result, even if we still need to overcome the numerical instability; however it's not the best choice for our purpose.

Hierarchical face clustering

A different (and better) approach is inspired to the hierarchical face clustering technique, first proposed in [79] for static mesh segmentation. Hierarchical clustering is a greedy approach to the mesh clustering process, that reminds of the region growing approach. Following the description by [156], the hierarchical face clustering technique can be summarized as described in *FaceHierarchical-Clustering*.

We need to define a measure of merging priority, in order to populate Q, and a test condition, in order to distinguish between valid and not valid merging. The priority is computed in the same way of the previous approach: the

Algorithm *FaceHierarchicalClustering*
1. Initialize a priority queue Q of merging candidates.
2. Insert all pairs of adjacent faces in Q
3. **while** Q is not empty
4. **do** get the next merging candidate (u, v) from Q **if** (u, v) can be merged
5. **then** merge (u, v) into w
6. insert all new merging candidates involving w to Q
7.

deformation of an edge e is taken as the average deformation of its vertices, in order to assign high priority to edges characterized by small deformation value. That follow the intuition that, if there's no deformation between two adjacent faces T_i and T_j, then they likely belong to the same cluster, and therefore the merging candidate (T_i, T_j) must have high value, in order to merge these two faces as soon as possible. Faces adjacent to an existing cluster should be added to the cluster if there's no significant deformation on the shared boundary edge. On the other hand, if the deformation of e is quite high, then probably the two adjacent faces will belong to different rigid parts of the shape and therefore they should not be clustered together. The merging condition can be tested using a simple threshold on the deformation value.
This approach leads to better results than the previous one; it preserves connectivity of the clusters, it doesn't need to know *apriori* the number of significant clusters, and it's also quite easy to understand and implement. However there is also some drawback, in addition to the not completely reliable measure (as already explained before). We pointed out other two problematic features:

- in most of the cases, deformation doesn't occur sharply on a single edge, but it's spread over a small region, affecting a number of faces and edges. As a result, edges where deformation occurs may have a measure lower than the threshold, and their incident faces maybe clustered together even if, conceptually, they should belong to different parts of the object.

- it frequently happens that boundaries between shapes are not smooth, for the same reason we mentioned before; some faces is clustered with a part of the object even if it could be clustered with a different one. In fact, sometimes the difference of the deformation values of edges incident to the same face is zero, and the face is add to the first cluster that comes out from the priority queue, even if it could (and sometimes should) be clustered differently.

In order to overcome these two limitations, the measure should be computed locally on group of edges or faces, even if the size of the neighborhood usually cannot be determined a priori, and also a post-processing boundary refinement step could be added, in order to smooth the boundaries between clusters, defining some global optimization term involving not only deformation values, but also geometrical properties of the object.

Vertex clustering based on Dijkstra's shortest path

As it can be seen from the previous two approach, one of the conceptual error was the attempt to characterize edges or faces of the mesh using a deformation value computed on the vertices. The obvious way to solve this problem is to switch from a face clustering to a vertex clustering. In this way we also be more faithful to the definition of the problem introduced in 3.2. We then present a third approach, that provides better results, partially inspired from the work by Katz et Tal [104], with the main difference that, instead of performing face clustering, we perform vertex clustering on the shape. The main idea behind this approach is to identify the significant vertices on the original shape, starting from the 1D analysis, and use them in order to locate the seeds of the segmentation and to adjust the size of each cluster. Conceptually, these significant vertices should be the boundaries between adjacent rigid parts.
The first step of this approach requires to locate the significant vertices on the shape. For each node $x \in X$, we count the number of times that it has been classified as significant over all the silhouettes, and call this number the *saliency* of the node. After assigning these values to the vertices, we perform an adaptive clustering on the vertices, in a similar fashion of the algorithm *AdaptiveClustering*, with the obvious exception that the clustering is performed on the three-dimensional shape. In this way, after a number of iterations, we are able to identify the regions of the shape where deformation occurs, corresponding to the clusters of significant vertices (the joints). These vertices will be used to detect the position of the seeds and the boundaries of the clusters; they will be assigned to the clusters only at the end of the procedure. In order to identify both the seeds and the elements of the clusters, we add to the geometric information, approximating the geodesic distance between vertices of the original shape, the information coming from the 1D curve.
Assume to have correctly detected the significant vertices on the silhouette S. Conceptually, they identify the regions where the deformation occurs the most and, at the same time, separate the regions moving rigidly. We can detect the chains of vertices corresponding to the rigid parts simply cutting the silhouette graph at the significant vertices and taking the remaining connected components. If two vertices x_i, x_j belong to the same rigid part in each silhouette, then it's reasonable to assume that they belong to the same rigid part of the shape. We summarize this relation between pairs of vertices build a $n \times n$ matrix where the (i, j) entry represent the number of times that x_i and x_j belong to the same cluster in the silhouette. We call this matrix *affinity matrix*. It's straightforward to notice that the affinity matrix is sparse and symmetric, so it can be build and stored with low computational cost.
The geometric information is encoded in a pre-computed *all-pair shortest path* (*APSP*) matrix containing, for each couple of vertices x_i and x_j, the approximation of their geodesic distance computed performing the Dijkstra's shortest path algorithm over a reference pose. Each not significant vertex x is then characterize by a value $d(x)$ representing its minimum geodesic distance from a significant vertex. The vertex with the highest distance value is selected as

the first seed of the clustering step.
In order to evaluate if a vertex x_j belongs to the same cluster of the seed x_i we compute a *distance value*:

$$distance(x_j, x_i) = (\alpha\ geodesic(x_j, x_i)) + (\frac{\beta}{affinity(x_j, x_i)}) \tag{3.2}$$

Here α and β are two normalized user-defined parameters such that $\alpha + \beta = 1$. If the vertex x_j has a small distance value respect to seed x_i, it means that its geodesic distance from x_i is quite small, or that x_j and x_i are frequently clustered in the same rigid parts of the silhouettes; in both cases, it's reasonable to assume that x_j and x_i belong to the same cluster of the shape. We define a threshold t related to the seed x_i as

$$t = (\alpha\ d(x_i)) + \epsilon \tag{3.3}$$

Here α is the same parameter of 3.2 and ϵ is, again, user defined. In other words, if the distance between x_j and the seed x_i is lower than t, then x_j is closer than the nearest significant vertex, and therefore we assign x_j to the cluster whose seed is x_i.
After that a cluster has been populated, we update the values of $d(x_j)$ for all the not clustered vertices, computed as the minimum distance from a significant or a clustered vertex. Then, a new seed is selected an a new clustering iteration is performed. The algorithms stops when one of these conditions is met:

1. the number of clusters is equal to the number of cluster desired by the user;
2. at least k the vertices has been clustered (usually $k \approx 80\%$ of n).

Usually, at this point there's a number of vertices that haven't been classified; those vertices are simply assigned to the nearest cluster. Following the common approach, we also assign the significant vertices to the nearest cluster, instead of explicitly creating some joints cluster.
This approach, whose results is shown in Figure 3.11, is surely better then previous one; it keeps track of both 2D and 3D information, since it merges silhouettes segmentation and geometry of the object. However, it's still not good: depends from several parameters, it requires a large amount of time and space, and the results are not completely satisfactory. This is mainly due to the poor performance of the approximation of the geodesic distance using Dijkstra's shortest path algorithm; also, the need of an explicit *APSP* matrix makes problematic to deal with large-sized meshes (e.g. more than 100000 vertices). In the next section we'll finally show how these problem can be overcome, and how we can finally obtain good quality segmentation in a reasonable amount of time which is about half the time needed by the state-of-the art-methods.

Vertex clustering based on Diffusion Distance

Our final approach moves from premises that are similar to the ones of the clustering based on Dijkstra's algorithm. Again, For each node $x \in X$, we count

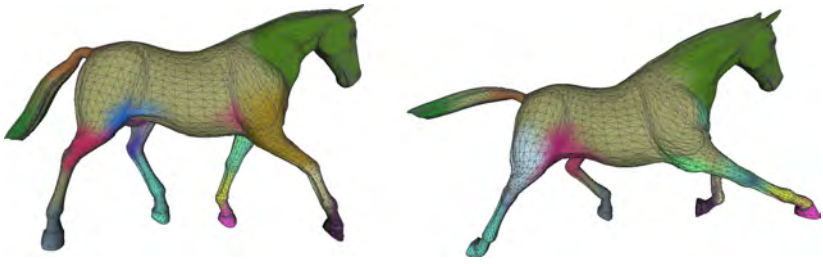

Figure 3.11: A result of the vertex clustering method based on Dijkstra's shortest path algorithm. Results are better than the previous approaches, but computational cost is significantly high.

the number of times that it has been classified as significant (*saliency*), then we identify the nodes in X that correspond to the non-rigid joint regions of the shape by simply thresholding on the saliency, selecting the nodes that appear to be significant in at least a certain percentage of all augmented silhouettes (usually 10% to 20%). The final segmentation is found by first computing the distances of all nodes to these selected nodes and then growing clusters from the local maxima of this distance function.

The main difference is that, to determine the distance between two nodes x and x', we consider the corresponding 3D vertices v and v' in one of the meshes and compute the *diffusion distance* [30, 53]

$$d_t(v, v') = \sum_{i \geq 0} e^{-2\lambda_i t}(\phi_i(v) - \phi_i(v'))^2, \tag{3.4}$$

where ϕ_i are the eigenfunctions and λ_i the eigenvalues of the discretized Laplace-Beltrami operator Δ. As this distance is intrinsic and invariant to isometric shape deformations, it does not matter which of the N meshes we consider for the actual computation. This distance is more reliable than the geodesic approximation, and it's more suitable for our purpose.

In our experiments, we use the *cotangent weight* discretization [131] of the form $\Delta f = A^{-1}Wf$, where f is a function defined on the vertices of the mesh, represented by a vector of size N, A is a diagonal matrix of size $N \times N$ that contains the area elements of each vertex, and W is a $N \times N$ zero-mean matrix with elements

$$w_{ij} = \begin{cases} -(\cot \beta_{ij} + \cot \gamma_{ij})/2, & (i,j) \in E, \\ -\sum_{k \neq i} w_{ik}, & i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Here, β_{ij} and γ_{ij} denote the angles opposite the edge $[v_i, v_j]$ in the two triangles sharing this edge. The eigenfunctions and eigenvalues of Δ are found by solving the generalized eigendecomposition problem $W\phi_i = \lambda_i A\phi_i$. The

parameter t controls the scale of the diffusion distance and is selected based on the shape diameter. Since the coefficients $e^{-2\lambda_i t}$ decay fast, in practice it is enough to use the first 5 to 10 eigenpairs to accurately approximate the diffusion distance (3.4).

As can be seen in Figure 3.4 (fifth column) and in Figure 3.12, this isolated connected subsets of X that are characterized by similar distance values and correspond to the rigid parts of the shape. We then perform a region growing clustering in order to build the different segments of the shape, picking as seeds the local maxima of the distance function and adding all connected vertices with a distance value less than some threshold. The clustering stops when at least 80% of the vertices have been clustered. The remaining vertices are finally assigned to the nearest cluster. The resulting segmentation is shown in Figure 3.4 (sixth column).

Figure 3.12: The *diffusion distance* from the significant vertices, computed on the horse shape. Red parts correspond to the joints between rigid parts, that are characterized by yellow/green/blu colors.

The time complexity of this segmentation step is $\mathcal{O}(n^2)$, due to the pairwise distance computation between vertices, which is quite heavy for large meshes. However, section 3.4 explains how the runtime can be drastically reduced thanks to parallelization of the algorithm on the GPU.

3.4 GPU parallelization

Our algorithm is highly parallelizable and can be efficiently implemented on SIMD-type processors. In our implementation, we parallelize part of the algorithm on the GPU, using the CUDA architecture [142], while other parts are parallelized on multi-core CPUs.

In the first stage of the algorithm, the extraction of the silhouettes is parallelized first launching a CUDA thread for each face of the mesh, testing its visibility and storing the results in a boolean grid, then launching a CUDA thread for each edge of the mesh and, using the vector previously populated, testing if the edge is significant. In this way, we can decrease the running time

up to 75% of the time needed on a single core CPU. Furthermore, each viewpoint can be processed independently from the others.
In the second stage, the analysis of the curves can be distributed on multiple processing units (e.g., multiple CPU cores), since each silhouette is examined separately from the others. It's not possible to use the CUDA architecture since the operations that has to be performed aren't suitable to be parallelized in a SIMD fashion. Each graph is stored using the data structure provided by the LEMON graph library, while the saliency values are stored in an Eigen sparse vector.
In the third stage, the computation of the diffusion distance between a vertex v and all other vertices in V is executed in separate CUDA threads. We use this feature to speed up the computation of the minimum distance from the selected vertices of V: we compute the distance between each selected vertex and the other vertices in V and then pick, for each vertex, the minimum distance value. If s is the number of selected vertices, we need to update the minimum distance of each vertex s times, and the update is performed in parallel for each vertex in V. Using the same logic, we accelerate also the final part of the algorithm, that is, assigning significant vertices to the nearest cluster.

3.5 Experimental Results

In this section, we present several results of our algorithm and comparisons with other methods. The tests have been performed on an Intel QuadCore Q9550 2.83GHz with 4GB onboard memory and using CUDA parallelization on an NVIDIA GeForce GTX-260 graphics card. We use the LEMON graph library to store the augmented silhouettes. Shapes are provided by the Aim@Shape repository and by the Sumner shape dataset [164]. Running times of the algorithm are summarized in table 3.1.

Figure 3.13 shows a comparison between our method, the method of Rosman et al. [151], and the results from Wuhrer and Brunton [176], applied to the same input poses of the horse model with approximately 8 000 vertices. The segmentation is computed using 50 silhouettes, randomly picked from 10 poses. The segmentation is consistent with the shape articulations and correctly captures the rigid parts of the object, except for the hoofs. The boundaries of the rigid parts are generally smoother than the ones detected by the other algorithms. We also do not detect small patches, as Rosman et al. [151], since they do not correspond to any visually perceived motion.

Figure 3.14 shows the cat model, segmented using 50 different silhouettes and 8 input poses. In both tests, the runtime of the algorithm is slightly lower then [176], but the difference becomes more significant with the increase of the number of vertices of the input shapes, especially thanks to the massive parallelization. In the elephant model (see Figure 3.15) with approximately 40 0000 vertices and segmented using 100 silhouettes, the legs are clearly separated from the rest of the body which is almost static. The motion of the front legs is visible and well captured, while the hind legs moves rigidly, as also detected

Figure 3.13: Comparison between our method (left), Rosman et al. [151] (middle), and Brunton and Wuhrer [176] (right). Tests are performed on the horse dataset shown in Figure 3.1.

by Wuhrer and Brunton [176]. The runtime for this test is 16.3 secs, and thus significantly smaller than the 46 secs needed by Wuhrer and Brunton to obtain a comparable segmentation.

The largest test case is the armadillo model (see Figure 3.16) with 166 000 vertices. Segmentation is performed using 5 input poses and capturing 125 silhouettes; the run-time is 144 secs. The result of this test case is discussed in section 3.6.

	n	N	K	Step 1	Step 2	Step 3	total
cat	7207	8	50	1.02	0.51	0.773	2.3
horse	8431	10	50	0.898	0.55	0.643	2.1
elephant	42321	10	100	7.602	4.081	4.58	16.3
armadillo	165954	5	125	79.36	32.446	32.149	144

Table 3.1: Different shapes used in our experiments and runtime (in sec) of different stages of the algorithm. Step 1 is the extraction of the augmented silhouettes, step 2 is the 1D analysis and step 3 is the 2D clustering based on diffusion distance.

3.6 Limitations

There are several main limitations to our algorithm. The first one is that we don't have the guarantee that the selected viewpoints will capture every motion of the shape, even if this is an outside chance. On the other hand, in most cases we use more silhouettes than the one that we actually need, introducing redundancy and noise in data; a smarter selection of the silhouettes could affect the total running time and the results.
As can be seen in Figure 3.16 (the Armadillo model), some of the rigid parts are not clearly detected. The motion of the left knee, for example, is not captured, because the surface deformation is distributed over a large number of vertices and we cannot distinguish clearly where the motion occurs during the analysis of the augmented silhouettes. This is by far the main limitation of our approach: since we work computing the deformation value of each single vertex, we are not able to estimate the deformation of a subset of vertices (a

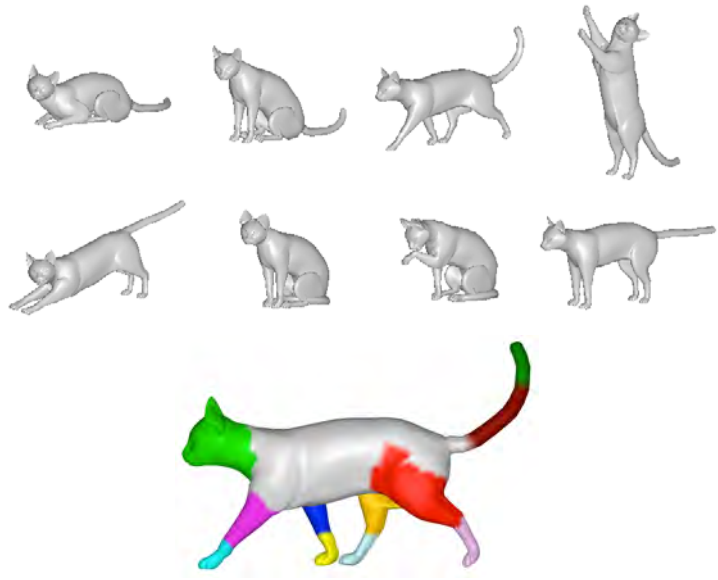

Figure 3.14: Segmentation of the cat shape using our method.

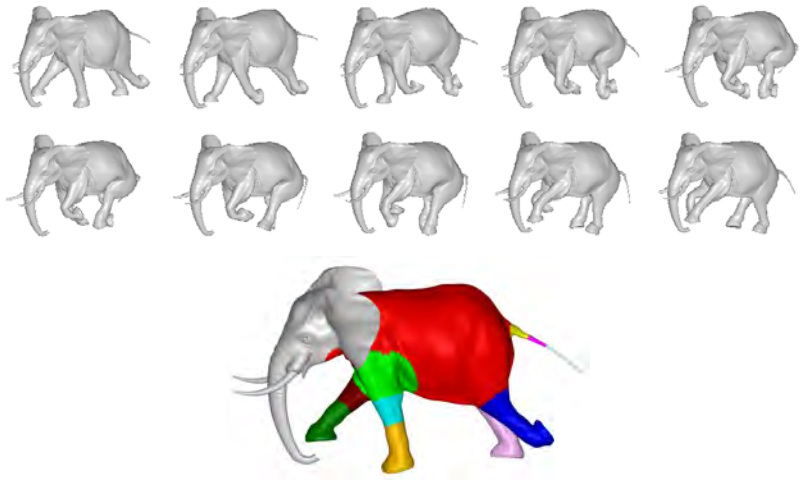

Figure 3.15: Segmentation of the galloping elephant shape using our method.

Figure 3.16: Segmentation of the armadillo shape using our method.

region). If the deformation is spread over a number of vertices, the deformation value of each single vertex is usually small, but the global deformation of the region is significantly high. However,our approach is still able to capture the general structure of the object without resulting in over-segmentation as in the method of Wuhrer and Brunton [176] and it runs in 144 secs, compared to 349 secs of [176].
For the final remarks on this work and a discussion about possible future improvements, we refer to the chapter 4.2.

Chapter 4

Conclusions

4.1 Bundle Adjustment

In the chapter 2, we presented a possible, novel approach to the problem of the image-to-geometry registration that take advantage of the optical flow computation in order to estimate the alignment error and to compute the rigid motion that we must apply to the cameras in order to minimize the error. As previously stated, at the present moment the algorithm performs well only on synthetic dataset, using ad-hoc generated images allowing to compute a dense and reliable optical flow, starting with a good initial alignment. The experimental tests on these datasets show that, under several constraining assumption, it's possible to reliably estimate the alignment error and correct it until we reach its global minimum. We also perform a number of experimental tests on real datasets, consisting of sequences of high-resolution images of a real object and a detailed digital model of the object itself. In these cases, the results, as already mentioned, was unsatisfactory, due to the large number of limitations of the system. Even the implementation of possible solutions to some issues didn't produce satisfactory results, due to lack of precision or increase of amount of time.
The obvious question is: *how far can we go with this method?* Also, can this method actually work *better than the existing ones*? Unfortunately the answer isn't so obvious. The synthetic test cases proves empirically that the theoretical idea is somehow correct, as seen before, even if the cost, in terms of running time and spatial resources, is quite high. After all, we are, in a certain sense, pushing to the limits the classic technique of adjusting camera position from correspondences between images, with the difference that, in our case, each pixel of the image has a corresponding one in the other. However, passing from synthetic to real cases means to add complexity to the problem, affecting both results and computational cost. Most of the assumption we made in the previous test case (dense and reliable flow, initial good alignment and so on) may not hold anymore, and even if we'll be able to solve the problem (computing the flow, align the cameras etc) we don't have the guarantee that the

achieved result will be better that the state-of-the-art method. Moreover, it's quite difficult to conciliate good results and good timing, since usually good results requires much precision and, as consequence, higher times. A system with good results but high timing could be, again, not suitable, since most state-of-the-art algorithm performs in a short amount of time. Relaxing the constraint could be a partial solution (e.g.: using images are captured in the same illumination condition), but the risk is to lose generality and novelty; if, for example, we choose to drop some vectors in the optical flow and focus only on the good quality one, the technique will become not-so-different from the standard bundle adjustment methods based on correspondences between images. To summarize, the idea is correct and, under some assumption, will stand, but it will be quite hard to make it suitable for real applications.
The work presented in [126], describing an implementation of the Lucas-Kanade optical flow techinque on GPU, has been developed as partial result of this part of the thesis.

4.2 Motion-based mesh segmentation

In the chapter 3 of this thesis, we introduced a method for motion-based 3D shape segmentation. Our method is based on extracting 1D silhouette curves from the shape, segmenting them into rigid parts, and then merging the 1D segmentation information to obtain the shape segmentation. Basically we followed the way that human beings observe the motion of the objects: detecting the motion from the 2D perceived image (the silhouette) and then use this information to infer the structure of the object. We found out that the results were comparable to the state of the art in terms of aesthetic quality. The method is also computationally efficient and highly parallelizable (especially on graphics hardware), which makes its running time significantly lower than the one from the state-of-the-art methods, that also have an higher computational complexity.
However, as one may notice by reading sections 3.5 and 3.6, there is a number of cases where our approach is not able to produce result of the expected quality (the Armadillo case, for example). In general, there is a number of possible improvements that we can add to our algorithm, in order to deal with these particular cases.
First of all, the algorithm will benefit from a *smarter choice* of the viewpoints, in order to capture the entire motion of the shape with only a minimal set of silhouettes. *Smarter* means that each silhouette should contain significant information about the motion, reducing the redundancy of data as much as possible. Obviously, since we cannot make any assumption about the motion of the shape, it's not possible to know *a priori* where the cameras should be placed in order to extract the silhouettes. However, we can try to place the cameras as a consequence of the spatial alignment of the object, for example performing the PCA on a reference mesh and then placing the camera subsequently. Alternatively, we can start placing only one camera, then observing the motion from this first camera and choosing a position of the camera that

allows to capture *new* motion information (that is, motion occurring in vertices different from the ones captured from the previous viewpoint). Eventually, one could assign to each silhouette a weight, based on the captured motion, in order to discard redundant or noisy data; the weight may be computed as function of the edge normals and the direction of view, or as a function of the number of times that an edge has been detected as perceptually relevant in the other silhouettes. However, the most urgent improvement is related to the *non-local*, that is, deformation spread across a number of connected vertices. In this case, the deformation occurs in a *region* and not in a single vertex, so our method is not completely suitable, since we cannot locate precisely the significant vertices. Our algorithm should be able to handle this situation, eventually using an incremental measure of vertices deformation that takes into account not only the local variation of the dihedral angle, but also the deformation of the angles of the connected vertices, weighting the angles in a Gaussian fashion. In this way, each vertex should be characterized by a deformation value expressing the behavior of both the vertex and its neighborhood. Finally, different distance measures, such as geodesics, could be used instead of the diffusion distance, especially if the computation becomes too expensive; in this way we can save time and space related to the computation of the eigenvalues of the Laplacian matrix, that can be expensive for large meshes. However, this is the less urgent improvement, since this computation is performed once on only the reference object, and the resulting values are simply inserted in the framework together with the different poses.
The work presented in this part of the thesis has been conditionally accepted to the GMP 2012 conference [125].

4.3 Final Remarks

The work here presented has the aim to investigate the relation between the perception of the motion and the geometry of a scene, and its possible applications. Human beings are able to infer the characteristics of an object by simply looking at it; the projection of the object on the retinas, combined with the capability of the brain to perform automatically a stereo reconstruction, allows us to create a description of the observed shape. In the same way, the perception of the motion, corresponding to the perception of the changes in a scene, allows us to understand how the scene is composed and how the different objects (or different parts of the same object) behave.
In our work, we explored two possible applications:

- The use of a perceived (but, as a matter of fact, not-existing) motion to align a set of images to the geometry of a scene;
- The use of a perceived (and real) motion to understand how an object moves/deforms, or, more generally, how it changes during time.

In the first case, speaking of perceived motion is somehow misleading, since we introduce the optical flow as a measurement of the alignment error, but ac-

tually there is no real motion to be perceived in the scene. However, using this trick and treating the displacement as the motion perceived by a moving camera, it is theoretically possible to determine the exact position of each camera with respect to the observed geometry. At the moment, however, drawbacks are more than the advantages; as we already pointed out, the goodness and the validity of the idea are not in doubt, since the issues are mostly related to achieving a robust and possibly fast implementation of the alignment pipeline. We can say that, in this case, the perception of this *fake* motion is a possible solution to the problem, but not necessarily the best one.
Instead, the second application of the perceived motion shows definitely good results. Observing a moving object from a number of different viewpoints gives the observer some important clues on how the object changes during time and, subsequently, which the different parts of the objects are, how they move and the way they are connected. We showed that this can be done by simply taking into account the projections of the deforming shape onto a number of different image planes (corresponding to a number of different viewpoints) and examining how the obtained silhouettes change during time. In this sense, the perceived motion corresponds to a *real* motion of the observed object, and it is helpful to understand the structure of the shape. In general, the way we perceive an object or a scene can be very useful in order to understand its static structure and properties, as pointed out by Marr and Ullman [123] and as proved by experimental results by Livesu et al. [114] that uses the static silhouettes of the object in order to reconstruct its topological skeleton. Our work shows that the way the perception changes during time can be a significant help in order to understand how the structure of the observed object changes (that is, which parts move and which ones are static).
The perception of a shape may be used also in order to better understand and describe an object; examples of possible future applications may be mesh segmentation from static silhouettes, reconstruction from the perception of a point cloud ([90]), and so on. In the author's opinion, the perception may be used in order to describe the properties, static or dynamic, of a shape. There's a number of characteristics (the curvature, the shape-diameter, the medial axis and so on) that may be inferred by simply looking at a object from different vewipoints, without precisely knowing its underlying structure. Working on perceived data should allow to move the problem from 3D to 2D, since we basically work on images; the obvious advantage is a reduction of the complexity in the analysis, partially counterbalanced by an overhead related to the combination of the data derived from the 2D analysis, similarly to our work on the motion-based segmentation. However, in the author's opinion, a straightforward development of this work should be the application of the perception to the field of the shape analysis (*Perceptual Shape Analysis*), in order to better understand if we can use the visual appearance of an object to describe it and eventually to compare it with other shapes.

Appendix A

Controlled and adaptive mesh zippering

During the developing of this work, we had to deal also with problems not directly related with perception and motion. Particularly, when working on shape reconstruction by means of range scanners, we had to deal with the problem of merging range maps produced by different devices, with different precision and so on. We then developed a novel algorithm for merging the range maps, named *adaptive mesh zippering*. In this appendix we'll present this novel algorithm, published as [124] and presented during the GRAPP conference, May 2010.

A.1 Motivation

In section 1.1 we already examined the different stages of the 3D acquisition pipeline. However, thanks to the improving in the technology for digitizing real objects, these steps have been refined in several regards, and even if the 3D acquisition pipeline can be considered as a stable process, further improvements still can be done (for example in terms of computation time). In this section we'll focus on the process of merging aligned range maps, the last step of the acquisition pipeline. The revisitation of the mesh zippering as a mesh merging algorithm is mainly motivated by the fact that, thanks to the improvement of both the software and the hardware involved in the 3D scanning pipeline, the quality of acquisition and range maps alignment may allow to zip them together and to obtain the same result as with more costly and sophisticated methods.

A.2 Background

We recall from section 1.1 that a *range map* is an image where each pixel stores a depth value, which means that information contained in a range map is a set

of points in 3D space. Given the regular distribution of the samples in the image space, it is possible to triangulate them to obtain a triangle mesh for each range map.
An early mesh merging approach, called *zippering* [169], consists in stitching together the pairs of triangulations that overlap in 3D space by eliminating the redundant faces in the overlapping regions and then adjusting the mesh connectivity locally. Other approaches work by triangulating union of the point sets, like the Ball Pivoting [20], which consists of rolling an imaginary ball on the point sets and creating a triangle for each triplet of points supporting the ball. This approach is strictly connected to the idea of α-shapes [69], which may be thought as a ball pivoting where the ball may appear simultaneously everywhere outside the scanned volume. In other approaches, like in [177], active contours are used to deform a mesh to fit the sample points. Note that in these cases the vertices of the final mesh do not coincide with the initial samples. In fact, the reconstruction of the final representation is also a way to filter the input data to reduce noise and discard outliers.
Imperfections and noise produced by the scanning devices is one of the reasons for the success of *volumetric methods*, which convert the range maps in a volumetric domain (a discrete distance field), well suited for filtering and merging operations. The Space Carving proposed in [58] uses range maps and line of sight of the scanner to determine a configuration of the empty voxels consistent with all the range images. Many of the improvements to volumetric methods consist of enhanced methods for estimating distances and normals of the surface. The method proposed in [97] uses local estimation of tangent planes to obtain a signed distance function. More recent approaches uses MLS local approximations of the surface to address continuity of the distance function [4, 157] which may be then visualized by sampling the isosurface with points, using contouring techniques [75] to build a polygon mesh or ray tracing it at rendering time [88]. A quadratic local approximation scheme has been combined with a hierarchical space subdivision that allows the reconstructions of the surface for large number of points in [143]. In the same flavor, the Poisson Surface Reconstruction algorithm [105] formulates the problem of defining the surface from a set of point as a Poisson problem, for which a least square solution can be efficiently found at different scales.

A.3 Algorithm overview

Our zippering algorithm consists of the same steps as the original version presented in [169], that we summarizes in the following. Given two meshes A and B that partially overlap, the zippering algorithm proceeds in three phases:

- **Border Erosion**. Remove faces from the border of the patches to minimize data redundancy.
- **Clip a patch against the other**. The border of one of the two range maps is projected on the other range map, the faces intersected by projec-

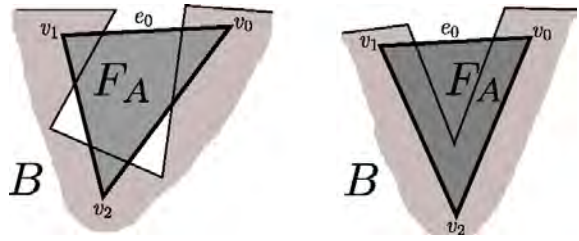

Figure A.1: False Positive

tion of A's border are re-triangulated in order to create a new consistent connectivity.

- **Cleaning**. Improve the quality of the mesh in the region when re-triangulation occurred.

These phases are revisited to extend the zippering to support a user-defined criterion to eliminate the data redundancy and to robustly support the zippering of patches with very different resolution.

A.3.1 Border Erosion

In the original zippering algorithm [169] a triangular face is said to be *redundant* if its 3 vertices project on the surface of the other patch. As it is shown in Figure A.1 this process may easily create holes of roughly the size of the face, that will be eventually triangulated after the zippering process. However, there are situations in which we cannot make the assumption that range maps to be zippered are similar in terms of size and number of polygons, for example because more than one device is used to scan the surface and the two produce range maps with different granularity. This is quite common in scanning Cultural Heritage artifacts, where there are portions of the surface that cannot be accessed with the device used and are subsequently covered with a different one [65].

Therefore we compute the distance between a face of a patch and the other patch by using a uniform sampling of the face, in order to guarantee a bounded size of the holes that we may create. We say that a point *projects* on the patch if it is closer than a given threshold to the patch and the closest point is not on a border edge (see Figure A.2). Our algorithm classifies a face of patch A as redundant if all the samples project on the patch B (and vice versa).

The meaning of eliminating redundant faces is to redefine the border of the patches, in other terms to establish a frontier to divide the region where faces of the mesh A are taken as valid data from the region where faces of the mesh B are taken. Figure A.3 shows three different frontiers for the same pair of range maps. In the original algorithm a single patch is chosen as the one containing redundant information and therefore only its faces are possibly

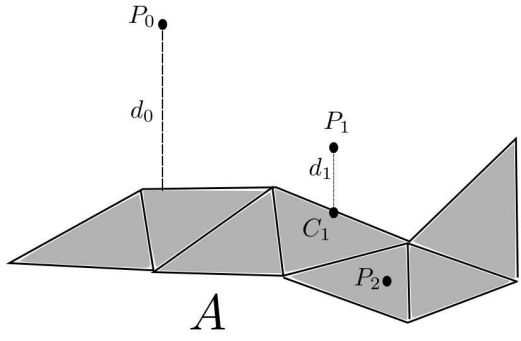

Figure A.2: Three points and a patch A: distance d_0 between P_0 and A is greater than a given threshold ε, so P_0 does not project on A; distance d_1 between P_1 and A is lower than ε, but the closest point C_1 is on a border edge of A, so also P_1 does not project on A; finally, P_2 project on A, since its distance from A is lower than ε and closest point does not lie on border of A.

redundant. Conversely, our algorithm chooses to test redundant faces on the base of a *quality* value that we may use, for example, to preserve faces with lower estimated acquisition error, or with a better aspect ratio.

Figure A.4 sketches the erosion algorithm. The queue Q stores the faces on the border of the two patches in ascending order of quality; the face with lowest quality will be the first one of the queue. Starting from the first face in the queue, we check all faces, testing redundancy. Each face is tested once, and then it is removed from the queue; if the face is redundant, then it will be also removed from the original mesh, and its neighbors will be added to Q. The process stops when Q is empty, and all the faces involved in the process have been tested.

A.3.2 Clipping

At this point, the redundant faces have been removed from A and B and we are in a situation where there is an overlap between the two meshes so that every face on the border of a patch is only partially overlapping the other patch.

Our clipping algorithm consist of two steps:

1. refine the faces on the border of the patch A until no border edge projects on more than one face of B

2. remove from each face of B the part covered by a projection and re-triangulate the remain.

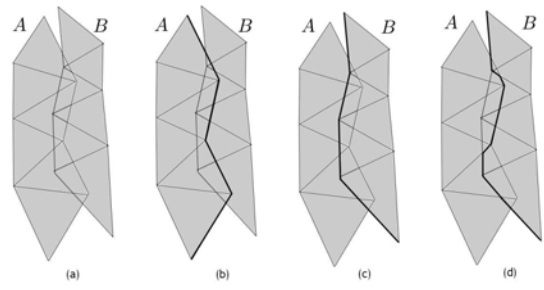

Figure A.3: Two patches (a), labeled as A and B, and possible frontiers between them; the frontier can be made of border faces from A (b) or B (c), or it can be mixed with part of border from A and part from B (d).

```
Erosion(Patch A, patch B){
Q = BorderOf(A) + BorderOf(B);
while(!queue.empty()){
     Face f = Q.pop();
     if( Redundant(f,OtherPatch(f))){
       RemoveFromThePatch(f);
       queue += Adjacents(f);
     }
    }
}
```

Figure A.4: The erosion algorithm selects the lowest priority face until there is redundancy of data.

Refinement

The first step if carried out as follows. We keep a queue B_A of faces of A to be processed and initialize it with the border faces of A. Then the following steps are applied until the queue is empty.

Remove a face F_A from B_A, project its two border vertices, named v_0 and v_1, on the surface of B. Let F_0 and F_1 be the faces of B where projections of v_0 and v_1 lie, and let e_0 the border edge between v_0 and v_1. Then one of the followings holds:

- F_0 **and** F_1 **are the same face**, in this case, the projection of e_0 lies completely inside the face, we associate the face F_A with F_0 (see Figure A.5.(a)).

- F_0 **is adjacent to** F_1**, that is, they share an edge; we name this edge** e_S, in this case we create a vertex on the edge e_S positioned at the closest point of e_S to e_0 and replace the face F_A with two faces as shown in Figure A.5.(b), which are inserted in B_A.

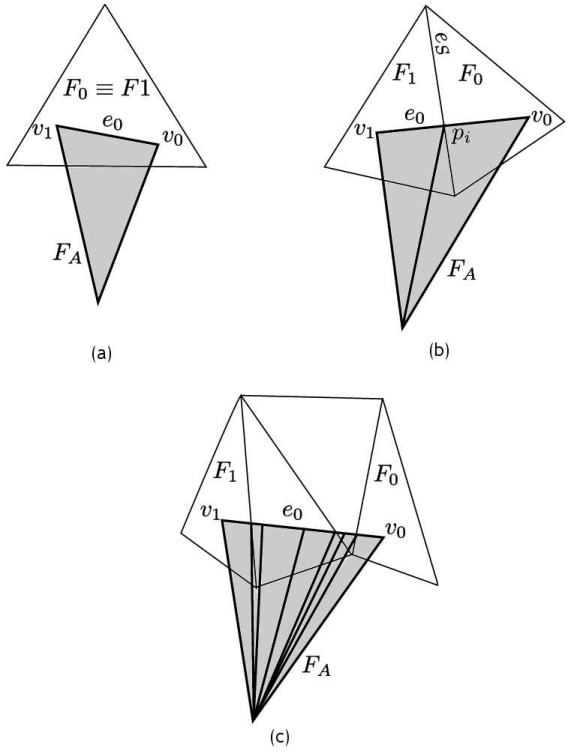

Figure A.5: (a) projection of both vertices lie in one face; (b) in two adjacent faces. (c) projections lie on non adjacent faces: the border face is recursively split.

- F_0 **is not adjacent to** F_1, in this case we simply split the face F_A along the middle point of the border edge and insert the resulting faces in B_A (see Figure A.5.(c).

- **One of the two vertices does not project on B** (see Figure A.6.(a)), in this case we simply consider the non projecting vertex, say it is v_0 as projecting in an ideal face of patch B passing through v_0 and process the faces as explained in the previous cases.

- **None of the vertices project**, in which case there is no action to take (see Figure A.6.(b)).

In other terms, we refine the faces until the first case is verified (see Figure A.5.(c)). Note that the last two cases always happen when only a portion of one patch overlaps the others, while may not be encountered when using a patch to close a hole.

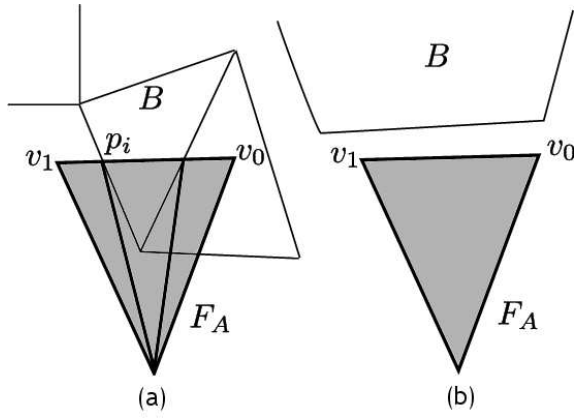

Figure A.6: Two additional case: vertex v_1 is not projectable on mesh, so edge is split on point p_i, then only segment from v_0 to p_i is projected on B; both v_0 and v_1 are not projectable, so F_A will not be modified.

Removal and Re-triangulation

At this point each of the projecting faces of A will be associated with a face of B. What we need to do is to decompose each face of B in a part *clipped* by the projection of the faces of A and a part to be re-triangulated accordingly to such projections. As shown in Figure A.7 for each face F_B to clip we will have one or more polylines with ending vertices exactly on the edges of F_B, which allow to separate the face in non-clipped and clipped part. Then we only need to discard the clipped parts and to re-triangulate the remaining polygon. Note that in rare situations we can also have more than one remaining polygon for the same face (see Figure A.7.(c)).

A.3.3 Cleaning

The quality of the faces is an important part of any reconstruction techniques. In this sense the zippering bears a clear disadvantage since it uses a constrained triangulation. On the other hand, we know exactly the region of the resulting mesh where the quality of the faces tends to be poor and may selectively filter those parts. Furthermore, we know that the poorly shaped faces are mostly the result of almost planar subdivision (during the refinement step) or planar re-triangulations (the final part of clipping).

A.4 Experimental results

In this section, we present some results of the experiments conducted. For each result we also gave information about time consumed by each stage of the algorithm. All of the experiments were performed on a Inter Quad-Core Q9550 2.83GHz equipped with a NVidia GeForce GTX 260 and 4,00 GB of on-board

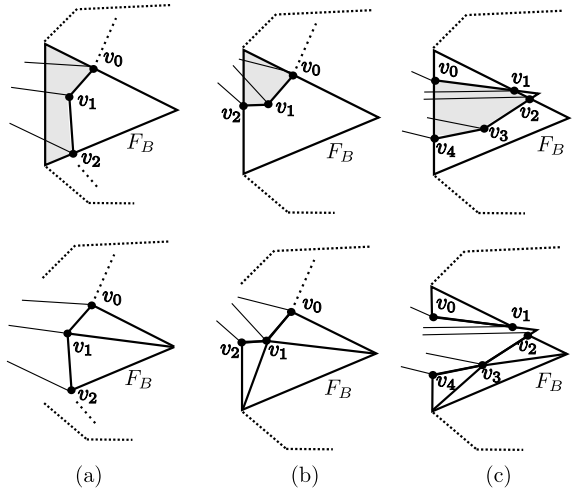

Figure A.7: Examples of face clipping. The polygon shaded in gray is to be removed, the rest of face F_B will be re-triangulated.

memory. Our algorithm has been developed as a plug-in for the MeshLab system [49]. Figure A.10 shows a model of a car with a large missing part and the patch that covers the missing part (top row). We applied our algorithm with three different criteria: by preserving the faces of the model, by preserving the faces of the patch and by weighting the faces on the base of their distance from the border, so obtaining a clipping frontier in the middle of the previous two. Table A.1 reports the timing for the three cases.

The most interesting experiment is the comparison with the Poisson Surface Reconstruction Algorithm [105] to merge two range maps with $2.5M$ triangles each. The range maps, obtained with a Breukmann Smartscan laser scanner [29], are very dense (100 samples per millimeter) and very well aligned. Figure A.9 shows two range maps merged using PSR (left) and the result or our zippering algorithm (right). The bottom of the Figure shows the Hausdorff distance between the two results mapped as false color on the mesh. As expected, the result is essentially the same and the sewing regions are not noticeable. However, our zippering algorithm took 1m27s seconds to complete against the 6m16s of the Poisson Surface Reconstruction. Note that the PSR is done with the original source code provided by the authors and in the same hardware setup.

A.5 Final remarks

The improved version of the zippering algorithm we've just presented extends the original version, by enabling enhanced control over the redundancy of data

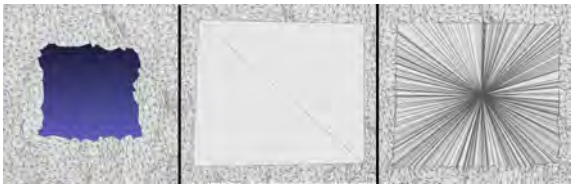

Figure A.8: Test case. A hole in a dense mesh (left) is covered with with a quad (center) and the algorithm produces a conformal mesh.

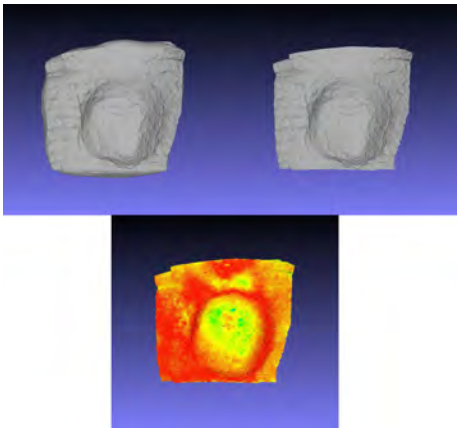

Figure A.9: Poisson reconstruction, on the left, and zippered range maps on the right. The bottom of the figure shows Hausdorff distance between the two results mapped to a ramp between red ($0m$) and blue($0.001m$).

Criteria	Erosion	Clipping	Cleaning
Preserving model's face	938 ms	1018 ms	79 ms
Preserving patch's face	1016 ms	4454 ms	89 ms
Using distance from border	962 ms	1050 ms	90 ms

Table A.1: Result table for experiment shown in Figure A.10

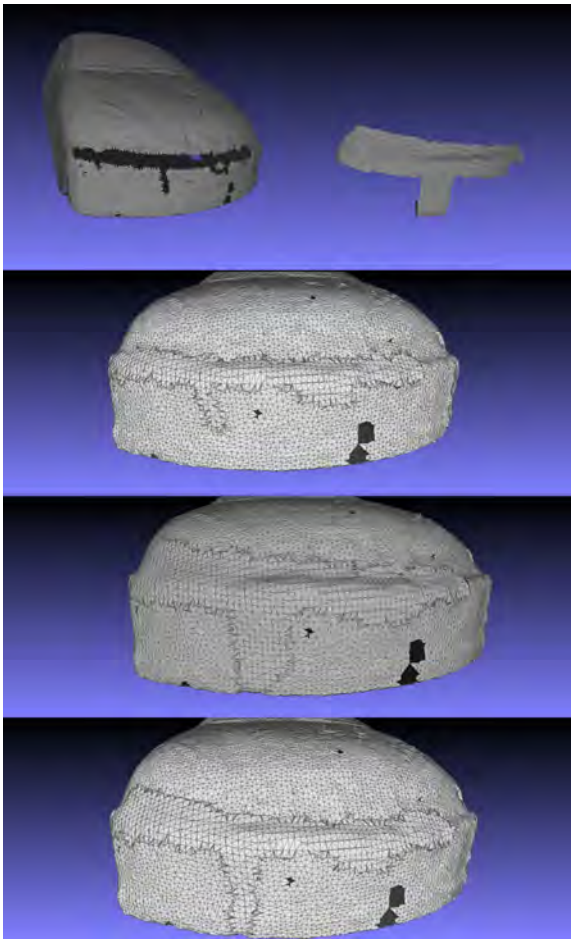

Figure A.10: A large hole on the surface of a mesh and zippering with three different criteria: faces from mesh are preserved, then faces from patch are preserved and, finally, faces are weighted using their distance from border.

and supporting merging of meshes with very different granularity. This allows the user to select which data save and which discard, and we can easily extended the procedure in order to manage an arbitrary number of range maps (two or more patches covering a hole, two or more overlapping range maps and so on). We propose this algorithm as an alternative to the other reconstruction algorithm since it's quite efficient, results aren't affect from precision loss and the user has the chance to control the way the merging is performed.

Bibliography

[1] Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. Building rome in a day. In *Proc. of International Conference on Computer Vision* (2009). 13

[2] Ajit, S. *Optic flow computation: a unified perspective*. IEEE Computer Society Press, 1992. 16

[3] Alba, A., Arce-Santana, E., and Rivera, M. Optical flow estimation with prior models obtained from phase correlation. In *Advances in Visual Computing*, vol. 6453 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 417–426. 22, 23, 32

[4] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., and Silva, C. T. Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01* (2001), IEEE Computer Society, pp. 21–28. 82

[5] Alexander, E. J., Bregler, C., and Andriacchi, T. P. Non-rigid modeling of body segments for improved skeletal motion estimation. *Computational Modeling Engineering Science 4* (2003), 351–364. 57

[6] Alvarez, L., Esclarin, J., Lefebure, M., and Sánchez, J. A pde model for computing the optical flow. In *Proceedings of CEDYA XVI* (1999). 30

[7] Alvarez, L., Weickert, J., and Sánchez, J. Reliable estimation of dense optical flow fields with large displacements. *Int. J. Comput. Vision 39* (August 2000), 41–56. 30

[8] Anandan, P. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision 2* (1989), 283–310. 16, 17, 26

[9] Ancona, N., and Poggio, T. Optical flow from 1d correlation: Application to a simple time-to-crash detector. In *Computer Vision, 1993. Proceedings., Fourth International Conference on* (may 1993), pp. 209 –214. 22

[10] Attene, M., Falcidieno, B., and Spagnuolo, M. M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22* (2006), 181–193. 56

[11] Baglietto, P., Maresca, M., Migliaro, A., and Migliardi, M. Parallel implementation of the full search block matching algorithm for motion estimation. *Proceedings The International Conference on Application Specific Array Processors* (1995), 182–192. 26

[12] Baker, S., Roth, S., Scharstein, D., Black, M. J., Lewis, J., and Szeliski, R. A database and evaluation methodology for optical flow. *Computer Vision, IEEE International Conference on 0* (2007), 1–8. 16, 23

[13] Balzani, M., Callieri, M., Fabbri, M., Fasano, A., Montani, C., Pingi, P., Santopuoli, N., Scopigno, R., Uccelli, F., and Varone, A. Digital representation and multimodal presentation of archeological graffiti at pompei. In *VAST'04* (2004), pp. 93–103. vi

[14] Balzani, M., Uccelli, F., Scopigno, R., and Montani, C. La cattedrale di pisa nella piazza dei miracoli: un rilievo 3d per l'integrazione con i sistemi informativi di documentazione storica e di restauro. In *Restauro 2006: Salone dell'Arte del Restauro e della Conservazione dei Beni Culturali e Ambientali*. Acropoli srl, 2006. Ferrara, Aprile 2006. vi

[15] Bar-Shalom, Y., and Li, X.-R. *Multitarget-Multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995. 8

[16] Barron, J., Fleet, D. J., Beauchemin, S. S., and Burkitt, T. Performance of optical flow techniques. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92., 1992 IEEE Computer Society Conference on* (jun 1992), pp. 236 –242. 26

[17] Bathow, C., Breuckmann, B., Callieri, M., Corsini, M., Dellepiane, M., Dercks, U., Scopigno, R., and Sigismondi, R. Documenting and monitoring small fractures on michelangelo's david. In *CAA 2010 Conference Proc.* (2010), F. J. Melero, Ed. vi

[18] Beauchemin, S. S., and Barron, J. L. The computation of optical flow. *ACM Comput. Surv. 27* (September 1995), 433–466. 14

[19] Bergen, J., Anandan, P., Hanna, K., and Hingorani, R. Hierarchical model-based motion estimation. In *Computer Vision - ECCV 92*, vol. 588 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1992, pp. 237–252. 17

[20] Bernardini, F., Mittleman, J., Rushmeier, H., Silva, C., and Taubin, G. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5* (1999), 349–359. 4, 82

[21] Besl, P., and McKay, N. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14* (1992), 239–256. 3

[22] Besora, I., Brunet, P., Callieri, M., Chica, A., Corsini, M., Dellepiane, M., Morales, D., Moyés, J., Ranzuglia, G., and Scopigno, R. Portalada: A virtual reconstruction of the entrance of the ripoll monastery. In *3DPVT08: Fourth International Symposium on 3D Data Processing, Visualization and Transmission* (June 2008), pp. 89–96. vi

[23] Beyeler, A., and Floreano, D. optiPilot : control of take-off and landing using optic flow. *Perspective* (2009). 44, 45

[24] Bianco, G., Cassinis, R., Rizzi, A., Adami, N., and Mosna, P. A bee-inspired robot visual homing method. In *Advanced Mobile Robots, 1997. Proceedings., Second EUROMICRO workshop on* (oct 1997), pp. 141 –146. iii

[25] Black, M., and Anandan, P. Robust dynamic motion estimation over time. In *Proceedings CVPR '91, IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (jun 1991), pp. 296 –302. 30, 32, 33

[26] Black, M. J., and Anandan, P. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Comput. Vis. Image Underst. 63* (January 1996), 75–104. 30

[27] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Lévy, B. *Polygon Mesh Processing*. AK Peters / CRC Press, Sept. 2010. iv

[28] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000). 9, 49

[29] Breuckmann. Breuckmann smartscan. `http://www.breuckmann.com`. 88

[30] Bronstein, M. M., and Bronstein, A. M. Shape recognition with spectral distances. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 5 (May 2011), 1065–1071. 55, 71

[31] Brown, D. *A solution to the general problem of multiple station analytical stereotriangulation*. D. Brown Associates, Inc., 1958. 12

[32] Brown, D. Close range camera calibration. *Photogrammetric Engineering* (1971). 12

[33] Brown, D. Calibration of close range cameras. *Int. Archives Photogrammetry* (1972). 12

[34] Brown, D. The bundle adjustment - progress and prospects. *Int. Archives Photogrammetry* (1976). 12

[35] Brox, T., Bregler, C., and Malik, J. Large displacement optical flow. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 0* (2009), 41–48. 17, 31, 52

[36] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision - ECCV 2004*, T. Pajdla and J. Matas, Eds., vol. 3024 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 25–36. 17, 30, 31, 32, 33, 34, 49

[37] Bruhn, A., Weickert, J., and Schnörr, C. Lucas/kanade meets horn/schunck: combining local and global optic flow methods. *Int. J. Comput. Vision 61* (February 2005), 211–231. 17, 32, 33

[38] Byröd, M., and Åström, K. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *Proc. of The 20th British Machine Vision Conference* (2009). 13

[39] Callieri, M., Chica, A., Dellepiane, M., Besora, I., Corsini, M., Moyés, J., Ranzuglia, G., Scopigno, R., and Brunet, P. Multiscale acquisition and presentation of very large artifacts: The case of portalada. *ACM Journ. on Computing and Cultural heritage 4*, 4 (April 2011). vi

[40] Callieri, M., Cignoni, P., Ganovelli, F., Impoco, G., Montani, C., Pingi, P., Ponchio, F., and Scopigno, R. Visualization and 3d data processing in david's restoration. *IEEE Computer Graphics & Applications 24* (2004), 16–21. vi

[41] Callieri, M., Corsini, M., Girardi, M., Padovani, C., Pagni, A., Pasquinelli, G., and Scopigno, R. The rognosa tower in san gimignano: Digital acquisition and structural analysis. In *Proceedings of The Tenth International Conference on Computational Structures Technology (CST2010)* (2010). vi

[42] Callieri, M., Debevec, P., Pair, J., and Scopigno, R. Realistic realtime illumination of complex environment for immersive systems a case study: the parthenon. In *SPIE International Symposium on Optical Metrology, 13-17 June 2005, Munich, Germany* (2005). vi

[43] Callieri, M., Dellepiane, M., and Scopigno, R. *La madonna di Pietranico - Storia, restauro e ricostruzione di un'opera in terracotta.* Edizioni ZIP, 2011, pp. 74–82. vi

[44] Carman, A. B., and Milburn, P. D. Determining rigid body transformation parameters from ill-conditioned spatial marker co-ordinates. *Journal of Biomechanics 39*, 10 (2006), 1778–1786. 57

[45] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. Reconstruction

and representation of 3d objects with radial basis functions. In *Computer Graphics (SIGGRAPH 01 Conf. Proc.), pages 67-76. ACM SIGGRAPH* (2001), Springer. 4

[46] Chen, Z. Efficient block matching algorithm for motion estimation. *International Journal of Signal Processing* (2009), 133–137. 27

[47] Cheung, C.-H., and Po, L.-M. A novel cross-diamond search algorithm for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Techn. 12*, 12 (2002), 1168–1177. 27

[48] Ciabatti, E., Cignoni, P., Montani, C., and Scopigno, R. Towards a distributed 3d virtual museum. In *Proceedings of the working conference on Advanced visual interfaces* (New York, NY, USA, 1998), AVI '98, ACM, pp. 264–266. vi

[49] Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. MeshLab: an Open-Source Mesh Processing Tool. In *Sixth Eurographics Italian Chapter Conference* (2008), pp. 129–136. 88

[50] Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R. Acquisition and management of digital 3d models of statues. In *Proceedings of 3rd International Congress on "Science and Technology for the Safeguard of Cultural Heritage in the Mediterranean Basin* (2002). vi

[51] Cohen, I. *Nonlinear Variational Method for Optical Flow Computation*, vol. 1. Citeseer, 1993, pp. 523–530. 30

[52] Cohen-steiner, D., and Duke, U. Variational shape approximation. *ACM Trans. Graph 23* (2004), 905–914. 56

[53] Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America 102*, 21 (May 2005), 7426–7431. 55, 71

[54] Comport, A. I., Marchand, É., and Chaumette, F. Complex articulated object tracking. In *Articulated Motion and Deformable Objects*, vol. 3179 of *Lecture Notes in Computer Science*. Springer, 2004, pp. 189–201. 57

[55] Cooper, W. S. Use of optimal estimation theory, in particular the kalman filter, in data analysis and signal processing. *Review of Scientific Instruments 57*, 11 (nov 1986), 2862 –2869. 8, 16

[56] Corsini, M., Dellepiane, M., Dercks, U., Ponchio, F., Callieri, M., Keultjes, D., Marinello, A., Sigismondi, R., Scopigno, R., and Wolf, G. Cenobium - putting together the romanesque cloister

capitals of the mediterranean region. *BAR International Series 2118* (2010), 189–194. vi

[57] Corsini, M., Dellepiane, M., Ponchio, F., and Scopigno, R. Image-to-geometry registration: a mutual information method exploiting illumination-related geometric properties. *Computer Graphics Forum 28*, 7 (2009), 1755–1764. 14

[58] Curless, B. From range scans to 3D models. *Computer Graphics 33*, 4 (Nov. 1999), 38–41. 82

[59] Curless, B., and Levoy, M. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 303–312. 4

[60] de Aguiar, E., Theobalt, C., Thrun, S., and Seidel, H.-P. Automatic Conversion of Mesh Animations into Skeleton-based Animations. In *Computer Graphics Forum (Proc. Eurographics EG'08)* (Hersonissos, Crete, Greece, 4 2008), vol. 27. 57, 58

[61] Dellepiane, M., Callieri, M., Fondersmith, M., Cignoni, P., and Scopigno, R. Using 3d scanning to analyze a proposal for the attribution of a bronze horse to leonardo da vinci. In *The 8th International Symposium on VAST International Symposium on Virtual Reality, Archaeology and Cultural Heritage* (Nov 2007), Eurographics, pp. 117–124. vi

[62] Dellepiane, M., Callieri, M., Paribeni, E., Sorge, E., Sulfaro, N., Marianelli, V., and Scopigno, R. Multiple uses of 3d scanning for the valorization of an artistic site: the case of luni. In *Sixth Eurographics Italian Chapter Conference* (2008), Eurographics, Eurographics, pp. 7–14. vi

[63] Dellepiane, M., Callieri, M., Ponchio, F., and Scopigno, R. Mapping highly detailed color information on extremely dense 3d models: the case of david's restoration. *Computer Graphics Forum 27*, 8 (2008), 2178–2187. vi

[64] Dellepiane, M., Marroquim, R., Callieri, M., Cignoni, P., and Scopigno, R. Flow-based local optimization for image-to-geometry projection. *IEEE Transaction on Visualization and Computer Graphics Online first* (2011). 14, 35

[65] Dellepiane, M., Venturi, A., and Scopigno, R. Image guided reconstruction of un-sampled data: a coherent filling for uncomplete cultural heritage models. In *IEEE Workshop on eHeritage and Digital Art Preservation* (2009), IEEE, p. In press. 83

[66] Dellepiane, M., Venturi, A., and Scopigno, R. Image guided reconstruction of un-sampled data: A filling technique for cultural heritage models. *Int. J. Comput. Vision 94* (August 2011), 2–11. vi

[67] Desouza, G., and Kak, A. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 2 (2002), 237–267. 44

[68] Dev, A., Krose, B., and Groen, F. Navigation of a mobile robot on the temporal development of the optic flow. *Development 131*, 2 (Dec. 2003), 361–375. iii, 44

[69] Edelsbrunner, H., and Mücke, E. P. Three-Dimensional Alpha Shapes. *ACM Transactions on Graphics 13* (1994), 43–72. 4, 82

[70] Engels, C., Stewénius, H., and Nistér, D. Bundle adjustment rules. In *In Photogrammetric Computer Vision* (2006). 13

[71] Enkelmann, W. Obstacle detection by evaluation of optical flow fields from image sequences. In *Computer Vision ECCV 90*, O. Faugeras, Ed., vol. 427 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1990, pp. 134–138. 22

[72] Eudes, A., and Lhuillier, M. Error propagations for local bundle adjustment. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 0* (2009), 2411–2418. 13

[73] Farenzena, M., Fusiello, A., and Gherardi, R. Structure-and-motion pipeline on a hierarchical cluster tree. *Proceedings of the IEEE International Workshop on 3-D Digital Imaging and Modeling* (2009). 13

[74] Farneback, G. Very high accuracy velocity estimation using orientation tensors, parametric motion, and simultaneous segmentation of the motion field. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* (2001), vol. 1, pp. 171 –177. 30

[75] Fiorin, V., Cignoni, P., and Scopigno, R. Out-of-core MLS reconstruction. In *Proc. of he Ninth IASTED International Conference on Computer Graphics and Imaging (CGIM)* (2007), pp. 27–34. 82

[76] Fleet, D. J., and Jepson, A. D. Computation of component image velocity from local phase information. *International Journal of Computer Vision 5* (1990), 77–104. 10.1007/BF00056772. 16

[77] Franken, T., Dellepiane, M., Ganovelli, F., Cignoni, P., Montani, C., and Scopigno, R. Minimizing user intervention in registering 2d images to 3d models. *The Visual Computer 21*, 8-10 (sep 2005), 619–628. Special Issues for Pacific Graphics 2005. 14

[78] Fusiello, A., and Irsara, L. Quasi-euclidean uncalibrated epipolar rectification. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on* (dec. 2008), pp. 1 –4. 52

[79] Garland, M., Willmott, A., and Heckbert, P. S. Hierarchical face clustering on polygonal surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (New York, NY, USA, 2001), I3D '01, ACM, pp. 49–58. 56, 67

[80] Gelfand, N., Ikemoto, L., Rusinkiewicz, S., and Levoy, M. Geometrically stable sampling for the icp algorithm. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on* (oct. 2003), pp. 260 – 267. 3

[81] Giachetti, a. Matching techniques to compute image motion. *Image and Vision Computing 18*, 3 (Feb. 2000), 247–260. 26

[82] Giachetti, A., Campani, M., Sanni, R., and Succi, A. The recovery of optical flow for intelligent cruise control. In *Intelligent Vehicles '94 Symposium, Proceedings of the* (oct. 1994), pp. 91 – 96. 22

[83] Giachetti, A., Campani, M., Torre, V., and CRS, C. The use of optical flow for road navigation. *IEEE transactions on robotics and automation 14*, 1 (1998), 34–48. 22, 44, 45

[84] Giachetti, A., Gigli, G., and Torre, V. Computer assisted analysis of echocardiographic image sequences. In *Proceedings of ICIAP Sanremo* (1995), Springer LNCS, pp. 258–264. 22

[85] Gibson, J. J. *The Perception of the Visual World*. Houghton Mifflin, Boston, MA, 1950. 14, 43

[86] Gonzalez, R. C., and Woods, R. E. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008. 18

[87] Grady, L. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28* (2006), 1768–1783. 66

[88] Guennebaud, G., and Gross, M. Algebraic point set surfaces. *ACM Transactions on Graphics 26* (July 2007), 23. 82

[89] Guennebaud, G., Jacob, B., et al. Eigen 3.0. `http://eigen.tuxfamily.org`, 2010. 46

[90] Guggeri, F., Scateni, R., and Pajarola, R. Depth carving for raw point clouds reconstruction. *submitted to Eurographics 2012 short papers*. 80

[91] Gyaourova, A., Kamath, C., and Cheung, S. Block matching for object tracking. *Techincal report for US Department of Energy (US)* (2003). 22

[92] Harris, C., and Stephens, M. A combined corner and edge detector. In *Alvey vision conference* (1988), vol. 15, Manchester, UK, pp. 147–151. 20

[93] Hauberg, S., Sommer, S., and Pedersen, K. S. Gaussian-like spatial priors for articulated tracking. In *Computer Vision – ECCV 2010, Part I*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., vol. 6311 of *Lecture Notes in Computer Science*. Springer, 2010, pp. 425–437. 57

[94] Heeger, D. J. Optical flow using spatiotemporal filters. *International Journal of Computer Vision 1* (1988), 279–302. 10.1007/BF00133568. 17

[95] Hoffman, D., Richards, W., Pentl, A., Rubin, J., and Scheuhammer, J. Parts of recognition. *Cognition 18* (1983), 65–96. 66

[96] Holmes, S., Sibley, G., Klein, G., and Murray, D. W. A relative frame representation for fixed-time bundle adjustment in sfm. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation* (Piscataway, NJ, USA, 2009), ICRA'09, IEEE Press, pp. 2631–2636. 13

[97] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuelzle, W. Surface reconstruction from unorganized points. In *ACM Computer Graphics Proc., Annual Conference Series, (SIGGRAPH 92)* (1992), pp. 71–78. iv, 4, 82

[98] Horn, B. K. P., and Schunck, B. G. Determining optical flow. *Artificial Intelligence 17* (1981), 185–203. 14, 17, 29, 31, 32, 33

[99] Iida, F. Biologically inspired visual odometer for navigation of a flying robot. *Robotics and Autonomous Systems 44* (2002), 2003. iii

[100] James, D. L., and Twigg, C. D. Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3 (Aug. 2005). 57, 58

[101] Jeong, Y., Nister, D., Steedly, D., Szeliski, R., and Kweon, I.-S. Pushing the envelope of modern methods for bundle adjustment. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 0* (2010), 1474–1481. 13

[102] Kanade, T., and Okutomi, M. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16* (1994), 920–932. iv

[103] Katz, G., and Jr, J. K. All-pairs shortest-paths for large graphs on the GPU. *Proceedings of the 23rd ACM SIGGRAPH* (2008). 56

[104] Katz, S., and Tal, A. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22* (July 2003), 954–961. 55, 56, 69

[105] Kazhdan, M., Bolitho, M., and Hoppe, H. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), SGP '06, Eurographics Association, pp. 61–70. iv, 4, 82, 88

[106] Kovacic, M., Guggeri, F., Marras, S., and Scateni, R. Fast approximation of the shape diameter function. *Proceedings of GraVisMa 2010 5* (2010), 65–72. 55

[107] Lai, Y.-K., Hu, S.-M., Martin, R. R., and Rosin, P. L. Rapid and effective segmentation of 3D models using random walks. *Computer Aided Geometric Design 26*, 6 (Aug. 2009), 665–679. 56, 66

[108] Le Besnerais, G., and Champagnat, F. Dense optical flow by iterative local window registration. In *IEEE International Conference on Image Processing 2005* (2005), IEEE, pp. I–137. 22, 23, 33, 34, 49

[109] Lee, T.-Y., Wang, Y.-S., and Chen, T.-G. Segmenting a deforming mesh into near-rigid components. *Vis. Comput. 22* (September 2006), 729–739. 57, 58

[110] Lee, Y., Lee, S., Shamir, a., Cohenor, D., and Seidel, H. Mesh scissoring with minima rule and part salience. *Computer Aided Geometric Design 22*, 5 (July 2005), 444–465. 56

[111] Li, R., Zeng, B., and Liou, M. L. A new three-step search algorithm for block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on 4*, 4 (1994), 438–442. 27

[112] Liu, C., Yuen, J., Torralba, A., Sivic, J., and Freeman, W. T. Sift flow: Dense correspondence across different scenes. In *Proceedings of the 10th European Conference on Computer Vision: Part III* (Berlin, Heidelberg, 2008), ECCV '08, Springer-Verlag, pp. 28–42. 17, 49

[113] Liu, L., and Stamos, I. Multiview geometry for texture mapping 2d images onto 3d range data. In *In CVPR'06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2006), IEEE Computer Society, pp. 2293–2300. 14

[114] Livesu, M., Guggeri, F., and Scateni, R. Reconstructing the curve-skeleton of 3d shapes using the visual hull. *submitted to IEEE Transactions on Visualization and Computer Graphics*. 80

[115] Longuet-Higgins, H. C. A computer algorithm for reconstructing a scene from two projections. In *Readings in computer vision: issues, problems, principles, and paradigms*. M. Kaufmann Publishers, San Francisco, CA, USA, 1987, pp. 61–62. 52

[116] Longuet-Higgins, H. C., and Prazdny, K. The Interpretation of a Moving Retinal Image. *Royal Society of London Proceedings Series B 208* (July 1980), 385–397. 44

[117] Lorensen, W. E., and Cline, H. E. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21* (August 1987), 163–169. iv, 4

[118] Lourakis, M. I. A., and Argyros, A. A. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2* (Washington, DC, USA, 2005), ICCV '05, IEEE Computer Society, pp. 1526–1531. 13

[119] Lourakis, M. I. A., Lourakis, M. I. A., Argyros, A. A., and Argyros, A. A. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Tech. rep., Institute of Computer Science-FORTH, Heraklion, 2004. 13

[120] Lu, J., and Liou, M. A simple and efficient search algorithm for block-matching motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on 7*, 2 (April 1997), 429–433. 27

[121] Lucas, B. D., and Kanade, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2* (San Francisco, CA, USA, 1981), Morgan Kaufmann Publishers Inc., pp. 674–679. iv, 17

[122] Maes, F., Vandermeulen, D., and Suetens, P. Comparative evaluation of multiresolution optimization strategies for multimodality image registration by maximization of mutual information. *Medical Image Analysis 3*, 4 (1999), 373 – 386. 14

[123] Marr, D., Ullman, S., and Poggio, T. *Vision: A Computational Investigation Into the Human Representation and Processing of Visual Information*. MIT Press, 2010. 43, 59, 80

[124] Marras, S., Ganovelli, F., Cignoni, P., Scateni, R., and Scopigno, R. Controlled and adaptive mesh zippering. In *Proceedings of GRAPP - International Conference in Computer Graphics Theory and Applications* (2010). 81

[125] Marras, S., Hormann, K., Bronstein, M., Scateni, R., and Scopigno, R. Motion-based segmentation based on augmented silhouettes. In *In proceedings of GMP* (2012). 79

[126] Marras, S., Mura, C., Gobbetti, E., Scateni, R., and Scopigno, R. Two examples of gpgpu acceleration of memory-intensive algorithms.

In *In proceedings of EuroGraphics Italian Chapter* (2010). 17, 20, 21, 22, 23, 41, 49, 78

[127] Marzat, J., Dumortier, Y., and Ducrot, A. Real-time Dense and Accurate Parallel Optical Flow using CUDA. In *International Workshop on Computer Vision and Its Application to Image Media Processing* (Tokyo, Japon, 2008). 17, 20, 21

[128] Mateus, D., Horaud, R., Knossow, D., Cuzzolin, F., and Boyer, E. Articulated shape matching using laplacian eigenfunctions and unsupervised point registration. In *Proceedings of CVPR* (Anchorage, June 2008), pp. 1–8. 57

[129] Maybeck, P. S. *Stochastic models, estimation and control. Volume I.* Press, Academic, 1979. 8

[130] Mccarthy, C., and Barnes, N. Performance of optical flow techniques for indoor navigation with a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation* (2004), pp. 5093–5098. iii

[131] Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, H.-C. Hege and K. Polthier, Eds., Mathematics and Visualization. Springer, 2003, pp. 35–57. 71

[132] Montani, C., Scateni, R., and Scopigno, R. Discretized marching cubes. In *Proceedings of the conference on Visualization '94* (Los Alamitos, CA, USA, 1994), VIS '94, IEEE Computer Society Press, pp. 281–287. 4

[133] Mouragnon, E., Lhuillier, M., M., D., Dekeyser, F., and Sayd, P. Real time localization and 3d reconstruction. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 363–370. 13

[134] Mukawa, N. Estimation of shape, reflection coefficients and illuminant direction from image sequences. In *Computer Vision, 1990. Proceedings, Third International Conference on* (dec 1990), pp. 507 –512. 30

[135] Nagel, H.-H., and Enkelmann, W. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-8*, 5 (sept. 1986), 565 –593. 30

[136] Ndhlovu, T., and Nicolls, F. An alternative confidence measure for local-matching stereo algorithms. In *Proceedings of RobMech* (2009). 16

[137] Negahdaripour, S. Revised definition of optical flow: Integration of radiometric and geometric cues for dynamic scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell. 20* (September 1998), 961–979. 21, 49

[138] Ni, K., Steedly, D., and Dellaert, F. Out-of-core bundle adjustment for large-scale 3d reconstruction. 13

[139] Nie, Y., and Ma, K.-K. Adaptive rood pattern search for fast block-matching motion estimation. *IEEE Transactions on Image Processing 11*, 12 (2002), 1442–1449. 27

[140] Nillius, P., and Eklundh, J. Fast block matching with normalized cross-correlation using walsh transforms. *Laboratory, Stockholm, Sweden, Tech. Rep. TRITA* (2002). 27

[141] Nordberg, K., and Farnebäck, G. A framework for estimation of orientation and velocity. In *Proc. IEEE Intl. Conf. on Image Processing* (2003), pp. 3–57. 49

[142] NVIDIA Corporation. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007. 20, 72

[143] Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., and Seidel, H.-P. Multi-level partition of unity implicits. *ACM Transactions on Graphics 22* (2003), 463–470. 82

[144] Okutomi, M., and Kanade, T. A locally adaptive window for signal matching. *International Journal of Computer Vision 7* (1992), 143–162. 27

[145] Patras, I. Confidence measures for block matching motion estimation. In *in International Conference on Image Processing. IEEE* (2002), pp. 277–280. 16, 27

[146] Po, L.-M., and Ma, W.-C. A novel four-step search algorithm for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol 6* (1996), 313–317. 27

[147] Pulli, K. Multiview registration for large data sets. In *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on* (1999), pp. 160 –168. 3

[148] Raudies, F., and Neumann, H. An Efficient Linear Method for the Estimation of Ego-Motion from Optical Flow. *Pattern Recognition* (2009), 11–20. 44

[149] Roberts, R., Potthast, C., and Dellaert, F. Learning general optical flow subspaces for egomotion estimation and detection of motion anomalies. *2009 IEEE Conference on Computer Vision and Pattern Recognition* (June 2009), 57–64. 44

[150] Rocchini, C., Cignoni, P., Montani, C., Pingi, P., Scopigno, R., Fontana, R., Greco, M., Pampaloni, E., Pezzati, L., Cygielman, M., Giachetti, R., Gori, G., Miccio, M., and Pecchioli, R. 3d scanning the minerva of arezzo. In *IN ICHIM'2001 CONF. PROC., VOL.2* (2001), pp. 265–272. vi

[151] Rosman, G., Bronstein, M. M., Bronstein, A. M., Wolf, A., and Kimmel, R. Group-valued regularization framework for motion segmentation of dynamic non-rigid shapes. *International Journal of Computer Vision* (2011), 1–12. iv, 57, 58, 59, 73, 74

[152] Rusinkiewicz, S., and Levoy, M. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on* (2001), pp. 145 –152. 3

[153] Santos-Victor, J., Sandini, G., Curotto, F., and Garibaldi, S. Divergent stereo for robot navigation: Learning from bees. iii

[154] Sapp, B., Toshev, A., and Taskar, B. Cascaded models for articulated pose estimation. In *Computer Vision – ECCV 2010, Part II*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., vol. 6312 of *Lecture Notes in Computer Science*. Springer, 2010, pp. 406–420. 57

[155] Scharstein, D., and Szeliski, R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision 47* (April 2002), 7–42. 26

[156] Shamir, A. A survey on Mesh Segmentation Techniques. *Computer Graphics Forum* (2008). iv, 56, 67

[157] Shen, C., O'Brien, J. F., and Shewchuk, J. R. Interpolating and approximating implicit surfaces from polygon soup. *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05* (2005), 204. 82

[158] Shum, H.-Y., Ke, Q., and Zhang, Z. Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In *Proc. of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'99)* (June 1999). 13

[159] Simoncelli, E. P., Adelson, E. H., and Heeger, D. J. Probability distributions of optical flow. In *Proc. Conf. Computer vision and Pattern recognition* (1991), IEEE Computer Society, pp. 310–315. 16

[160] Singh, A., and Allen, P. Image-flow computation: An estimation-theoretic framework and a unified perspective. *CVGIP: Image Understanding 56*, 2 (1992), 152 – 177. 26

[161] Srinivasan, M. V., Chahl, J. S., Weber, K., Venkatesh, S., Nagle, M. G., and Zhang, S.-W. Robot navigation inspired by principles of insect vision. *Robotics and Autonomous Systems* (1999), 203–216. iii

[162] Steedly, D., and Essa, I. Propagation of innovative information in non-linear least-squares structure from motion. *Computer Vision, IEEE International Conference on 2* (2001), 223. 13

[163] Steedly, D., Essa, I., and Delleart, F. Spectral partitioning for structure from motion. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2* (Washington, DC, USA, 2003), ICCV '03, IEEE Computer Society, pp. 996–. 13

[164] Sumner, R. W., and Popović, J. Deformation transfer for triangle meshes. In *ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, pp. 399–405. 73

[165] Sun, D., Roth, S., and Black, M. Secrets of optical flow estimation and their principles. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (june 2010), pp. 2432 –2439. 31

[166] Taylor, C., and Kriegman, D. Vision-based motion planning and exploration algorithms for mobile robots. In *Workshop on the Algorithmic Foundations of Robotics* (1999), pp. 69–83. iii

[167] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice* (London, UK, 2000), ICCV '99, Springer-Verlag, pp. 298–372. 5, 10, 13

[168] Trucco, E., and Verri, A. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998. iii, 6, 8, 21, 51, 53

[169] Turk, G., and Levoy, M. Zippered polygon meshes from range images. In *SIGGraph-94* (1994), pp. 311–318. iv, 82, 83

[170] Ullman, S. Filling-in the gaps: The shape of subjective contours and a model for their generation. *Biological Cybernetics 25* (1976), 1–6. 10.1007/BF00337043. 43, 59

[171] Ullman, S. On visual detection of light sources. *Biological Cybernetics 21* (1976), 205–211. 10.1007/BF00344165. 43, 59

[172] Ullman, S. *The Interpretation of Visual Motion*. MIT Press, 1979. 15

[173] Uras, S., Girosi, F., Verri, A., and Torre, V. A computational approach to motion perception. *Biological Cybernetics 60* (1988), 79–87. 10.1007/BF00202895. 16

[174] Waxman, A. M., Wu, J., and Bergholm, F. Convected activation profiles and the measurement of visual motion. In *IEEE Proceedings of CVPR* (1988), pp. 717–723. 16

[175] Welch, G., and Bishop, G. An introduction to the kalman filter. Tech. rep., University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995. 16

[176] Wuhrer, S., and Brunton, A. Segmenting animated objects into near-rigid components. *The Visual Computer 26* (2010), 147–155. 10.1007/s00371-009-0394-5. 58, 59, 64, 73, 74, 76

[177] Ye, D., and Hong, Q. Abstract 2.5d active contour for surface reconstruction. In *Proceedings of VMV 2003* (2003). 4, 82

[178] Zelek, J. S. Bayesian real-time optical flow. In *Proc. of 15th Int. Conf. Vision Interface* (2002), pp. 310–315. 17

[179] Zhang, Z., Cui, P., and Cui, H. Recovery of egomotion from optical flow with large motion based on subspace method. In *2006 IEEE International Conference on Robotics and Biomimetics* (2006), IEEE, pp. 555–560. 44

[180] Zhu, S., and Ma, K.-K. A new diamond search algorithm for fast block-matching motion estimation. *Image Processing, IEEE Transactions on 9*, 2 (2000), 287–290. 27

[181] Zhu, S., Tian, J., Shen, X., and Belloulata, K. A new cross-diamond search algorithm for fast block motion estimation. In *Proceedings of the 16th IEEE international conference on Image processing* (Piscataway, NJ, USA, 2009), ICIP'09, IEEE Press, pp. 1561–1564. 27

[182] Zitova, B. Image registration methods: a survey. *Image and Vision Computing 21*, 11 (Oct. 2003), 977–1000. 14

[183] Zufferey, J.-C., Beyeler, A., and Floreano, D. Optic Flow to Steer and Avoid Collisions in 3D. *Review Literature And Arts Of The Americas* (2009), 73–86. 44

[184] Zufferey, J.-C., and Floreano, D. Fly-inspired visual steering of an ultralight indoor aircraft. *IEEE Transactions on Robotics 22* (2006), 137–146. iii