CAMERA MATCHING

IN COMPUTER GRAPHICS

Master's Thesis

Vass Gergely

in 2003

Dept. of Control Engineering and Information Technology

Budapest University of Technology and Economics

M Ű E G Y E T E M 1 7 8 2

Alulírott Vass Gergely, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem.

. . . .

0.1 Abstract

Scientists of the field of image processing and machine vision has developed many algorithms to extract useful information from images and image sequences. *Camera matching* is one of the most important and well researched problem concerning computer vision. The main task of camera matching is to calculate the camera parameters – position, orientation and internal parameters of the projection – based on the *image sequences* captured by some unknown imaging device. Prior to the evaluation of camera parameters the *lens distortion* has to be removed from the images. The camera information is of key importance in computer graphics when integrating 3D elements into live action footage. In this thesis the theoretical background and the practical use of camera matching and lens distortion is discussed, while focusing on their aspects concerning computer graphics.

Based on the experiences from the implementation of a lens distortion removing and applying tool, some improvements of the existing algorithms are proposed. The basic algorithms of camera matching have been also implemented.

0.2 Összefoglaló

A képfeldolgozással és gépi látással foglalkozó kutatók számos algoritmust fejlesztettek ki különböző információk kinyerésére álló- ill. mozgóképekből. A kamera illesztés az egyik legfontosabb és legrégebben kutatott ilyen probléma. A kamera illesztése során egy képszekvencia alapján határozzuk meg az eredeti, ismeretlen kamera külső és belső – orientáció, pozíció, fókusz távolság stb. – paramétereit. A feladat megoldásakor a képen látható lencsetorzítást is ki kell küszöbölni. Az eredeti kameráról kinyert információ nélkülözhetetlen a számítógépes grafikával előállított elemek valós felvételekbe történő illesztésekor. Dolgozatomban ismertetem a kamera illesztés és a lencse torzítás korrekciójának elméleti hátterét és gyakorlati alkalmazását, különös tekintettel a számítógépes grafikával kapcsolatos alkalmazásra. A lencsetorzítást kezelő eszköz fejlesztése során szerzett tapasztalatok alapján javaslatot teszek az ismert módszerek kibővítésére. Munkám részeként a kamera illesztés alapvető algoritmusait is implementáltam.

Contents

Chapter 1

Introduction

1.1 Camera matching

Scientists of the field of *image processing* and *machine vision* has developed many algorithms to extract useful information from images and image sequences. The applications of such results include pattern and face recognition, medical analysis, super resolution and robot navigation. *Camera matching* is one of the most important and well researched problem concerning machine vision. The main task of camera matching is to calculate the camera parameters – basically position, orientation and field of view – based on the image sequences captured by some unknown imaging device. This information might be useful for example to navigate agents in real or virtual environments.

While image processing deals with existing images *computer graphics* [SK99] applications focus on synthesizing new images. In case of 3D graphics virtual photographs are generated from virtual models, in a way that the resulting pictures look like real photographs. The quality of these images depend on the plausibility of the rendering algorithm and the quality of the virtual model. The virtual model – including geometry, textures, lights, cameras and animation – might be built by modelers or acquired using some external device. There are many substantially different ways to acquire geometry, texture and animation. Some of these methods require rather expensive hardware, while others – like image based solutions – need only some low-end hand held camera.

The combination of image based algorithms like feature tracking, camera matching and texture extraction makes it possible to build high quality models, based on some footage taken by simple camera of real objects or sets (see figures 1.1 to 1.4). These technologies seem to be the primary tool of model, texture and motion acquisition in the future, since they do not require any spacial equipment[1]. In this thesis the theory of camera matching and it's current and future use in computer graphics is examined.

1.2 The focus of this thesis

A huge number of papers, books and articles have been published about camera matching algorithms, thus all mathematical problems have been examined thoroughly. In this thesis a general overview is given about the mathematical background based on the literature, and the practice of camera matching in computer graphics is introduced. Since the majority of the camera matching solutions – especially the calibration processes – are extremely sensitive to the error in the acquired data and to the user's input on some unknown camera parameters, many cases they do not provide any useful camera information. The implemented basic algorithms might serve as a basis for further research to build a robust camera matcher.

The issue of lens distortion is also discussed in great detail. This rather simple mathematical problem causes many trouble to the users of camera matching softwares, since lens distortion introduces significant error in the reconstruction. However, the proposed solutions often do not meet the need of the industry utilizing computer graphics and camera matching. In this work a simple way of semi-automatic calibration and a very fast method for applying distortion is described. A very basic method is also proposed that helps the integration of CG and real images with distortion.

1.3 Structure of this document

In chapter 2 we look at the related literature of camera matching and give a basic explanation of all the math behind the problem. At first the notation of projective geometry

[1]Note: Today a low-end video camera is not considered special equipment.

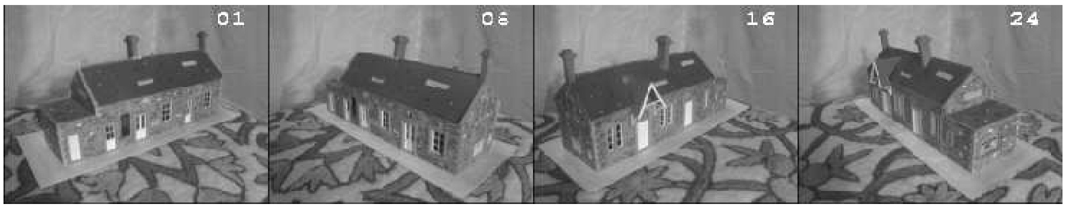

Figure 1.1: The input of the procedure is an image sequence.

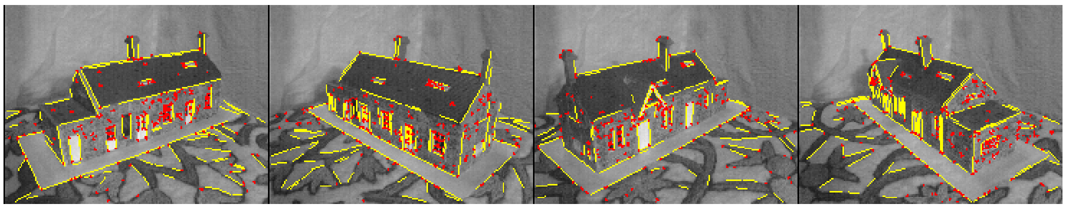

Figure 1.2: Features are detected – points or lines – and matched over image pairs or triplets.

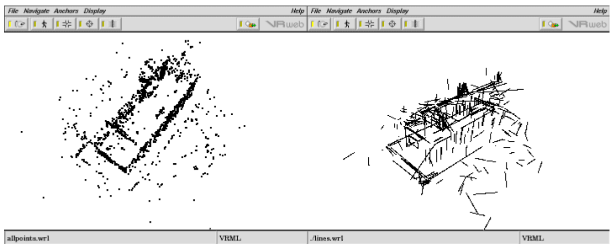

Figure 1.3: Based on the matched tokens the motion of the camera can be calculated. Using triangulation the 3D position of the tracked features can be evaluated as well.

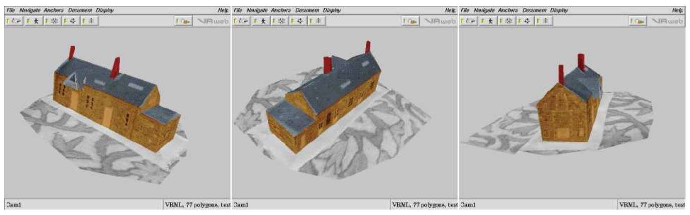

Figure 1.4: Some algorithms are capable to build 3D textured models based on the recovered structure. (Images by P. Torr)

is given, which is used to define the camera model used in computer vision. To find the unknown parameters of the camera geometric constrains of multiple vision are examined. The methods to extract the solution from token[2] matches and geometric constrains are also discussed.

In chapter 3 the issue of lens distortion is examined. This harmful effect has to be removed from the source footage in order to successfully extract the desired data. Detailed explanation is given about the different lens distortion models and calibration techniques. In this chapter not only the literature is summarized, but new results are also presented related to lens distortion. The issue of applying distortion is also discussed, a new, and very fast method is proposed.

Chapter 4 discusses the practical issues of implementing the algorithms of lens distortion removal and camera matching. First a short description is given about the implemented lens plug-in and its way of working. A brief description is given about the commonly used robust token extraction method, and about the fast, non-automatic external tracker used for testing the implemented program. Finally the internal pipe-line is presented of the implemented camera matching system.

The current use of camera matching for computer graphics is discussed in chapter 5. Detailed explanation is given about the typical way of working in post production studios utilizing camera matching softwares for integrating computer graphics into live action footage. With a real Hollywood example every step of the pipe-line is illustrated.

Lastly in chapter 6 the new researches about the possible future applications are outlined. These results might be the basis of further developments of the system. The conclusions and experiments related to the work done are discussed, and at the end the possible future developments are summarized.

[2]It is not possible for most of the cases to use every pixel of a digital image in complex calculations. A token is some identified element or attribute of the picture – like tracked points or lines – which contains possibly important information. These tokens are used in the calculation, thus the huge amount of data building up the images does not affect the speed of the algorithms.

Chapter 2

Overview of the problem

2.1 Mathematical representation and notation

The process of finding the unknown camera parameters is usually based on point – or line – correspondences over multiple images. In order to derive the equations and constraints used later in the thesis, some mathematical definitions should be given. The framework to describe 3D points, image points and projections is the *projective geometry*. A short introduction to this field is written by Stan Birchfield [Bir98] while some more detailed but still tutorial like explanation is given in [HZ00] and in the appendix of [MZ92]. For a more formal approach refer to [Spr64]. As a general reference for the theory of camera models and multiple view geometry [HZ00] is strongly recommended.

Projective geometry

Points in the 2D and 3D *Euclidean geometry* are represented by $(x, y)^\top \in \mathbf{R^2}$ and $(x, y, z)^\top \in \mathbf{R^3}$ vectors respectively. In the 2D and 3D projective spaces homogeneous vectors $\mathrm{x} = (\mathrm{x}, \mathrm{y}, \mathrm{w})^\top \in \mathcal{P}^2$ and $\mathrm{X} = (\mathrm{X}, \mathrm{Y}, \mathrm{Z}, \mathrm{W})^\top \in \mathcal{P}^3$ are used. To transform the homogeneous coordinates back to Euclidean the following formulas should be used: $(x, y) = (\mathrm{x}/\mathrm{w}, \mathrm{y}/\mathrm{w})$ and $(x, y, z) = (\mathrm{X}/\mathrm{W}, \mathrm{Y}/\mathrm{W}, \mathrm{Z}/\mathrm{W})$. The points whose fourth homogeneous coordinate is 0 are called *ideal points*.

Lines can be also represented by homogeneous vectors. The 2D line $ax + by + c = 0$ is equivalent to the homogeneous 3-vector $\mathrm{l} = (a, b, c)^\top \in \mathcal{P}^2$. Note that $\mathrm{l} =$

$(a, b, c)^\top$ and $\mathrm{l} \cdot k = (a \cdot k, b \cdot k, c \cdot k)^\top$ represents the same line. Also for vectors of points $\mathrm{x} = \mathrm{x} \cdot k$, thus for homogeneous quantities '=' indicates equality always up to a non-zero scale factor.

It easy to see that in projective geometry the point x lies on the line l if and only if $\mathrm{x}^\top \mathrm{L} = 0$. The equation for calculating the intersection of two lines is $\mathrm{p} = \mathrm{l}_1 \times \mathrm{l}_2$ and the line connecting two points is $\mathrm{l} = \mathrm{p}_1 \times \mathrm{p}_2$.

3D to 2D camera projection

The transformation of the camera projects the 3D points to the 2D projective plane. This transformation is described by a 3×4 matrix P on homogeneous vectors:

$$\mathrm{x} = \mathrm{PX} \tag{2.1}$$

where x is homogeneous 3-vector and X is a homogeneous 4-vector. If the matrix P represents a simple perspective transformation the matrix elements are the following:

$$\mathrm{P}_{\text{perspective}} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

where f is the *focus distance*(see figure 2.1).

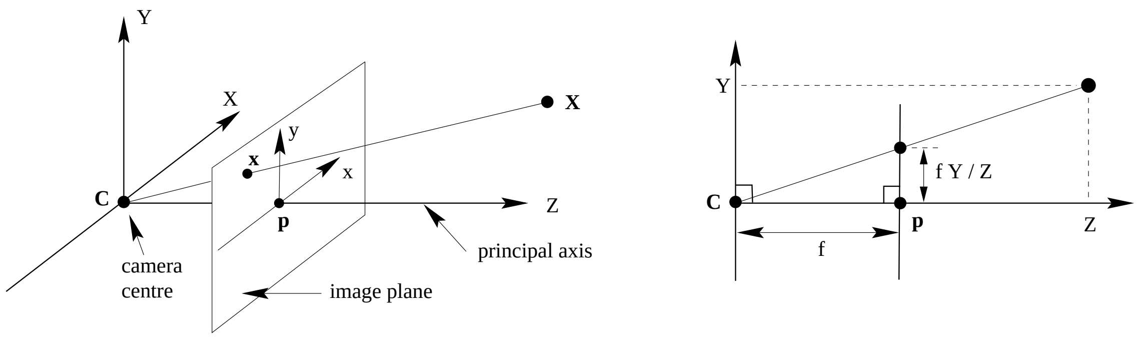

Figure 2.1: The principal point p is at the intersection of the principal axis and the image plane. The focus distance is the distance between the camera center C and the image plane. (Image from [HZ00])

An arbitrary camera projection matrix looks like this:

$$\mathrm{P} = \mathrm{P}_{\text{internal}}\mathrm{P}_{\text{perspective}}\mathrm{P}_{\text{external}} = \begin{bmatrix} k_u & k_c & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_u & -\alpha_u \cot\theta & u_0 \\ 0 & \alpha_v / \sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\,\mathrm{I} \,|\, 0\,] \begin{bmatrix} R & t \end{bmatrix} = \mathrm{AD} \tag{2.3}$$

where α_u and α_v are the scale factors of the image plane, θ is the skew, the point (u_0, v_0) is the principal point[1], R is the 3×3 rotation matrix, and t is the 3×1 translation vector. The matrix A contains the so called *internal parameters* while D contains the *external parameters*. For better understanding, the external parameters define the 3D position of the camera, while the internal parameters define how the camera projects the 3D points to the 2D plane. Considering real cameras – like film or CCD cameras – the external parameters are up to the photographer's position and the internal parameters usually depend on only the focus distance. Almost all cameras have unit aspect ratio[2] and no skew.

The main task of camera matching is to recover all the internal and external parameters of the camera. Where the internal parameters are known the camera is called *calibrated camera*. In this case only the 3D position of the camera – or the world's position relative to the camera – must be recovered from the images. If the internal parameters are not known the camera may be calibrated in a separate step – using some special calibration process and a known calibration object. In the field of robotics this approach has been preferred, since a custom camera rig – or stereo rig – can be calibrated very accurately. In 1992 it was shown in [Fau92] that using

[1]The principal plane is the plane through the camera center parallel to the image plane. The principal axis is the line passing through the camera center, with direction perpendicular to the principal plane. The principal point is the image point of the intersection of the principal line and the image plane. This is usually in the center of the image.

[2]Which means the pixels are squared pixels and no anamorphic lens is used. If this is not the case the images should be corrected manually prior to camera matching.

uncalibrated camera – where the internal parameters are not known – the *Euclidean* 3D reconstruction of the points is not possible. This means that based on arbitrary uncalibrated views only projective reconstruction is possible (See figure 2.3).

Later, in 1992 it has been also proposed in [Har92] that the calibration of the focal lengths – which is in most of the cases the only changing internal parameter of the camera – can be done based on only the point correspondences which is called *self calibration*. Using this method the Euclidean reconstruction is possible.

Essential matrix

The information about the internal and external parameters of the camera cannot be extracted from single images with unknown structure[3]. Using two or more images the self-calibration and the 3D reconstruction is possible. Most algorithms are based on correspondences of two images recorded from slightly different positions, these images are called *stereo images*. The first attempt to make use of such stereo images was in 1981 [LH81]. Based on a set of image correspondences the relative placement (external parameters) and the focal lengths (the major internal parameter) can be derived of the two cameras, and that is all the information we may deduce [Har92].

Suppose we have two pictures taken from different positions from the 3D point M, and it projects onto the two image planes at m_1 and m_2. If we suppose the camera is *calibrated* the points m_1 and m_2 are given with respect to the camera's coordinate frame, and the projection matrix of the camera consists of only the rotational and translational elements. The mathematical relationship between the two image points is expressed by the *epipolar constraint*. It says that the vector from the first camera's optical center to the first imaged point, the vector from the second optical center to the second imaged point, and the vector from the optical center to the other are coplanar (figure 2.2).

This relationship can be expressed by the following formula:

[3]Unknown structure means we have no information about the geometry of the recorded scene.

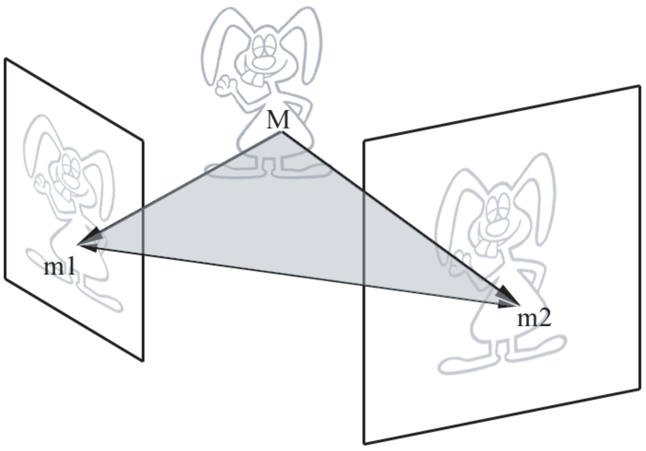

Figure 2.2: The epipolar constraint.

$$m_2{}^T(t \times Rm_1) = 0 \tag{2.4}$$

where R and t capture the rotation and translation between the two camera's co-ordinate frames. If we do not use the notation $t = [\, a\; b\; c\,]^T$ but define the matrix $T = \begin{bmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{bmatrix}$ the equation 2.4 turns out to be simpler:

$$m_2{}^T(TRm_1) = m_2{}^T E m_1 = 0 \tag{2.5}$$

The matrix $E = TR$ is called the *essential matrix* [FLM92] [Zha96]. If we find the essential matrix five parameters can be derived: three for the rotation and two for the direction of translation[4]. The essential matrix has two constraints:

1. the determinant has to be zero, thus the matrix is singular
2. its two non-zero singular values has to be equal.

[4]The magnitude cannot be recovered due to the depth/speed ambiguity

Fundamental matrix

The fundamental matrix describes the geometric relationship between the matching image points, however, it deals with uncalibrated cameras as well. The internal parameters of the two cameras are stored in the matrices A_1 and A_2 (equation 2.3). Transforming the points m_1 and m_2 with these matrices we get $\overline{m}_1$ and $\overline{m}_2$:

$$\begin{aligned}\overline{m}_1 &= A_1 m_1\\ \overline{m}_2 &= A_2 m_2\end{aligned}$$

The epipolar constraint is valid for the points $\overline{m}_1$ and $\overline{m}_2$ just like in the calibrated case (equation 2.4), since we compensated for the internal parameters. This yields the following equation:

$$\begin{aligned}({A_2}^{-1}\overline{m}_2)^T(t \times R{A_1}^{-1}\overline{m}_1) &= 0\\ {\overline{m}_2}^T {A_2}^T (t \times R{A_1}^{-1}\overline{m}_1) &= 0 \qquad (2.6)\\ {\overline{m}_2}^T F \overline{m}_1 &= 0 \qquad (2.7)\end{aligned}$$

The matrix $F = {A_2}^{-T} E {A_1}^{-1}$ is called the *fundamental matrix* [LF96]. F has 7 degrees of freedom and is a rank two matrix, thus it has two non-zero singular values. The 7 degrees of freedom is enough for deriving the 5 parameters of the relative translation and rotation, while the two extra parameters help to find the focal lengths[5] of the two cameras [Har92]. These are the only parameters that can be extracted from the fundamental matrix.

Trifocal tensor

If more than two views are considered even more complex constrains can be defined for point and line correspondences [BP] [FM95]. However, the use of more than three images is computationally too intensive. The constrain for matching points and lines of three images, or *image triplets* is called the *trifocal tensor* T_{ijk} [PBZ96]. This $3 \times 3 \times 3$ homogeneous tensor encapsulates the trilinear relations

[5]This is true only if there is no skew and the aspect ratio is one over all images.

for corresponding points and lines in three images. Using the tensor a point can be transfered to a third image from correspondences in the first and the second:

$$x''_l = x'_i \sum_{k=1}^{k=3} x_k \mathrm{T}_{kjl} - x'_j \sum_{k=1}^{k=3} x_k \mathrm{T}_{kil}$$

for all $i, j = 1...3$. Similarly, a line can be transferred as

$$l_i = \sum_{j=1}^{j=3} \sum_{k=1}^{k=3} l'_j l''_k \mathrm{T}_{ijk}$$

ie. the same tensor can be used for transfering both lines and points.

2.2 Finding the fundamental matrix

finding) In order to implement a fast camera matching algorithm – and for the sake of simplicity – the aim of this thesis is not to find the trifocal tensor but the fundamental matrix. There are several different approaches to calculate the fundamental matrices based on a set of point matches [HZ00]. The definition of the fundamental matrix is the following:

$$\mathrm{x}' \, \mathrm{F} \, \mathrm{x} = 0 \tag{2.8}$$

where $\mathrm{x} = (x, y, 1)^T$ and $\mathrm{x}' = (x', y', 1)^T$ are the matching points. Each point match give rise to one linear equation in the unknown entries of F. Specifically, the equation corresponding to a pair of points $(x, y, 1)$ and $(x', y', 1)$ is

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0 \tag{2.9}$$

From a set of n point correspondences, we obtain a set of linear equations in the form:

$$\mathrm{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathrm{f} = 0 \tag{2.10}$$

This is a homogeneous set of equations, thus f can only be determined up to scale. If the rank of A is exactly 8 there is one unique solution (up to scale).

If the data is not exact – which is *always* the case in real applications for computer graphics – the rank of A might be 9. In this case one finds a least-squares solution for f, which can be obtained by the *singular value decomposition* (SVD) of A. The SVD of a general $M \times N$ matrix M is:

$$M = U \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \vdots & \\ & & & s_N \end{bmatrix} V^T \tag{2.11}$$

The elements of the diagonal matrix are the *singular values*, for which $s_1 \geq s_2 \geq \ldots \geq s_N$, U is a $M \times N$ column-orthogonal matrix and V is the transpose of a $N \times N$ orthogonal matrix. The details of the mathematics concerning the SVD is out of the scope of this thesis, for further discussion refer to [WPF92] where the implementation of the algorithm is also presented in C.

The least-squares solution for f is the singular vector corresponding to the smallest singular value of A. That is, the *last column of* V in the SVD decomposition $\mathrm{A} = \mathrm{UDV^T}$. The solution vector found in this way minimizes $\|\mathrm{Af}\|$, which is our goal (see equation 2.10). The fundamental matrix can be built from the vector f:

$$\mathrm{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \tag{2.12}$$

This mathematical tool is the essence of the most basic and well known procedure called *normalized 8-point algorithm* proposed in 1981 [LH81] to compute the fundamental matrix.

2.2.1 The normalized 8-point algorithm

The key to success with the normalized 8-point algorithm – which is based on the singular value decomposition of the matrix built based on equation 2.10 – is proper normalization of the input data before constructing the equations. The suggested normalization is a very simple transformation:

- The image has to be translated so that the centroid of the considered points is at the origin.
- The image has to be scaled so that the RMS distance of the points from the origin is equal to $\sqrt{2}$.

This simple transformation of the points in the image before formulating the linear equations leads to a enormous improvement in the conditioning of the problem, thus the result will be much more stable. The added complexity of the algorithm is insignificant.

An important property of the fundamental matrix is that it is singular – in fact of rank 2 – and most applications based on the fundamental matrix rely on this fact, as well. However, the matrix F found by the SVD described above will not in general have rank 2, thus it has to be corrected[6]. The simplest way to enforce singularity is to use the SVD (again). In particular, let $\mathrm{F} = \mathrm{UDV^T}$ be the SVD of F, where D is the matrix $\mathrm{diag}(s_1, s_2, s_3)$ satisfying $s_1 \geq s_2 \geq s_3$. Then $\mathrm{F'} = \mathrm{U\ diag(s_1, s_2, 0)\ V^T}$ is a good solution, since it minimizes the Frobenius norm $\|\mathrm{F} - \mathrm{F'}\|$.

The steps of the normalized 8-point algorithm:

1. **Normalization**

   Transform the image coordinates according to $\mathrm{\hat{x}_i = T\ x_i}$ and $\mathrm{\hat{x}'_i = T'\ x'_i}$, where T and $\mathrm{T'}$ are the normalizing transformations consisting of translation and scaling.

2. **Linear solution**

   Determine $\widehat{\mathrm{F}}$ from the singular vector corresponding to the smallest singular value of $\widehat{\mathrm{A}}$ (using the SVD), where $\widehat{\mathrm{A}}$ is composed from the matches $\widehat{x}_i \leftrightarrow \widehat{x}'_i$ as defined in 2.10.

3. **Correction**

   Replace $\widehat{\mathrm{F}}$ by $\widehat{\mathrm{F}}'$ such that $\mathrm{det}\widehat{\mathrm{F}}' = 0$ using the SVD again and zeroing out the

[6]Note that all the algorithms described here to compute the fundamental matrix differ only in the way of this correction of enforcing singularity. The initial estimate of F based on the SVD is the first step of every algorithm. The only difference is how sophisticatedly the correction is done.

smallest singular value.[7].

4. **Denormalization**

   Set $\mathrm{F} = \mathrm{T}'^{\mathrm{T}}\ \widehat{\mathrm{F}}'\ \mathrm{T}$ to compensate the normalization. Matrix F is the fundamental matrix corresponding to the original data $\mathrm{x_i} \leftrightarrow \mathrm{x'_i}$.

2.2.2 The algebraic minimization algorithm

Although the normalized 8-point algorithm is very fast and rapid, it is numerically not optimal, since the entries of F have not equal importance in the relationship between corresponding points, which is not considered in the algorithm. This means that there might be a F' matrix that is singular just like F computed by the normalized 8-point algorithm, but is a better solution: $||\mathrm{Af}'|| < ||\mathrm{Af}||$. The algebraic minimization algorithm tries to find this *singular* F' matrix while minimizing $||\mathrm{Af}'||$ during an iterative process. For more details refer to the literature, for example [HZ00].

2.2.3 The Gold Standard method

The Gold Standard method is one of the most sophisticated algorithms to find the fundamental matrix, since it uses a more complex error model, which fits better the problem. It tries to find the Maximum Likelihood estimate of the fundamental matrix, assuming Gaussian distribution of the noise in image point measurements. In this case the ML estimate is the one that minimizes the geometric distance :

$$\sum_i d(\mathrm{x_i}, \widehat{\mathrm{x}}_\mathrm{i})^2 + d(\mathrm{x'_i}, \widehat{\mathrm{x}}'_\mathrm{i})^2 \tag{2.13}$$

where $\mathrm{x_i} \leftrightarrow \mathrm{x'_i}$ are the measured correspondences, and $\widehat{\mathrm{x}}_\mathrm{i} \leftrightarrow \widehat{\mathrm{x}}'_\mathrm{i}$ are estimated “true” correspondences satisfying $\widehat{\mathrm{x}}'_\mathrm{i}\ \mathrm{F}\ \widehat{\mathrm{x}}_\mathrm{i} = 0$. The process of the Gold Standard algorithm starts with an initial estimate of the fundamental matrix $\widehat{\mathrm{F}}$ using the normalized 8-point algorithm. The second step is the initial estimation of $\{\widehat{\mathrm{x}}_\mathrm{i}, \widehat{\mathrm{x}}'_\mathrm{i}\}$ based on $\widehat{\mathrm{F}}$ and using the triangulation method to determine the estimated 3D position of the point $\widehat{\mathrm{X}}_\mathrm{i}$ corresponding

[7]Note that this SVD is not the same as in the previous step. Step number 2 used the SVD of a $k \times 9$ matrix, where $k \geq 8$, while this step generates the SVD of $\widehat{\mathrm{F}}$, which is a 3×3 matrix

to $\mathrm{x_i}$. Then the cost function 2.13 should be minimized over $\widehat{\mathrm{F}}$ and $\widehat{\mathrm{X}}_\mathrm{i},\ \ \mathrm{i} = 1, \ldots, \mathrm{n}$ using the Levenberg-Marquart algorithm [WPF92]. Note that we not only find the fundamental matrix, but the 3D position of the points as well. However, this reconstruction is only a projective reconstruction, since no calibration is done. The Gold Standard algorithm is recommended by most of the literature, however this method requires the most effort in implementation.

2.3 Extracting external camera parameters from the fundamental matrix

The external parameters of the camera model are easily deduced from the fundamental matrix. However, a given fundamental matrix defines a pair of camera matrices up to right multiplication by a projective transformation. This means that using only a fundamental matrix while not knowing anything about the internal camera parameters the Euclidean reconstruction of the camera motion – which is necessary to match the virtual camera defined in Euclidean coordinates to the original one – is not possible [FLM92] [RH92].

Given this ambiguity, it is common to define a so called *canonical form* for the pair of camera matrices corresponding to a given fundamental matrix. In canonical form the first camera matrix has always the simple form $\mathrm{P} = [\,\mathrm{I} \mid 0\,]$, where $[\,\mathrm{I}\,]$ is the 3×3 identity matrix and 0 is a null 3-vector, while the second camera matrix has the form $\mathrm{P'} = [\,\mathrm{A} \mid \mathrm{a}\,]$. It is always possible to do, because if P is augmented by one row to to make a 4×4 non-singular matrix $\mathrm{P^*}$ and letting $\mathrm{H} = \mathrm{P^{*-1}}$, one verifies that $\mathrm{PH} = [\,\mathrm{I} \mid 0\,]$ as desired.

While a pair of camera matrices determine a unique fundamental matrix, this mapping is not injective as stated before. Pairs of camera matrices that differ by a projective transformation give rise to the *same* fundamental matrix. Given a fundamental matrix F the canonical matrices of the cameras may be chosen as [HZ00]:

$$\mathrm{P} = [\,\mathrm{I} \mid 0\,] \tag{2.14}$$

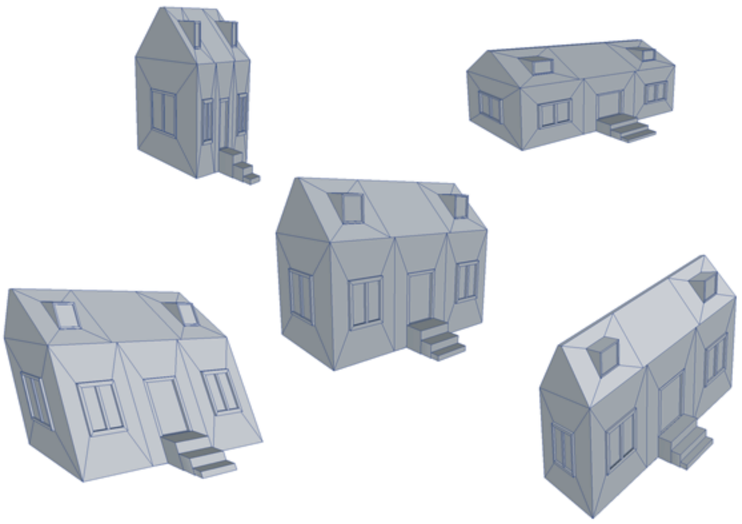

Figure 2.3: The projective reconstruction does not fit the need of computer graphics applications. The reconstructed 3D geometries (corners) are not similar to the original one (center).

and

$$\mathrm{P}' = [\, [\, \mathrm{e}' \,]_\times \, \mathrm{F} \, + \mathrm{e}' \, \mathrm{v}^{\mathrm{T}} \mid \lambda \, \mathrm{e}' \,] \tag{2.15}$$

where

$$[\, \mathrm{e}' \,]_\times = \begin{bmatrix} 0 & -e'_3 & e'_2 \\ e'_3 & 0 & -e'_1 \\ -e'_2 & e'_1 & 0 \end{bmatrix} \tag{2.16}$$

and e' is the epipole such that $\mathrm{e}'^{\,\mathrm{T}} \, \mathrm{F} = 0$, v is any 3-vector, and λ is a non-zero scalar. This kind of reconstruction of camera matrices is not really useful in our case, since the projective reconstruction is not enough for camera matching.

2.4 Extracting internal camera parameters

The process of finding the internal parameters of the cameras is called calibration. It was shown in [FLM92] that this can be achieved based on only images of unknown structure, if we have some constraints on the camera parameters. Usually these constraints are: unit aspect ratio and known principal point. This kind of calibration is called self-calibration or auto-calibration. In this thesis off-line calibration is not considered, since almost all of the applications in computer graphics deal with hand held cameras with no information for calibration. Since the initial publication of Faugeras et. al. many algorithms have been proposed. In [Har93] a non-iterative solution – using singular value decomposition – was given to find the focal lengths, while later a simple, closed-form formula was proposed in [Bou98]:

$$f_2^2 = -\frac{(\mathrm{p}_1^{\mathrm{T}}\,[\mathrm{e}_1]_\times\,\mathrm{I}\,\mathrm{F}^{\mathrm{T}}\,\mathrm{p}_2)(\mathrm{p}_1^{\mathrm{T}}\,\mathrm{F}^{\mathrm{T}}\,\mathrm{p}_2)}{\mathrm{p}_1^{\mathrm{T}}\,([\mathrm{e}_1]_\times\,\mathrm{I}\,\mathrm{F}^{\mathrm{T}}\,\mathrm{I}\,\mathrm{F})\,\mathrm{p}_1} \tag{2.17}$$

where I is the diagonal matrix diag(1,1,0). A similar formula for the focal length of the first camera f_1^2 os obtained by interchanging the roles of the two images. In the implementation related to this thesis the procedure of [KM00] is used, since it introduces a closed-form expression for finding the focal lengths from *only* the fundamental matrix. This procedure is not described here, since it produces the exact same solution as the Bougnoux formula while it is not as compact. All of the methods that extract the focal length from only the fundamental matrix have various deficiencies as shown in [HSA02]:

1. Computation of the focal lengths is not possible in case of some degenerate configurations. This means, if the camera moves in a certain way the reconstruction is not possible.

2. The algorithms require the position of the principal point to be known. Furthermore, the computation of the focal lengths can be very sensitive to the assumed positions of the principal points, and also to the fundamental matrix.

3. In some cases the algorithms fail completely, because the value of the focal length turns out to be negative. In this case the fundamental matrix may be incompatible with any reasonable choice of the principal point.

Researches have shown that the sensitivity of the computation of the focal length based on the fundamental matrix and a guessed principal point position is so severe that the algorithm is of doubtful practical value.

Successful auto calibration algorithms use the previous formulas for the initial estimate of the internal parameters, and find a much more reliable solution with some iterative process like in [HSA02].

2.5 Rebuilding camera motion

If we have the internal calibration data – focal length, principal point position – and the fundamental matrix, the camera matrices may be retrieved up to scale ambiguity. This way the relative motion of the second camera can be deduced. If we have F, f, f', p_x and p_y determined[8] the essential matrix can be calculated. The fundamental matrix and essential matrix are related in the following way:

$$\mathrm{E} = \mathrm{K}'^{\mathrm{T}}\ \mathrm{F}\ \mathrm{K} \tag{2.18}$$

where K and K' are the calibration matrices:

$$\mathrm{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \mathrm{K}' = \begin{bmatrix} f' & 0 & p_x \\ 0 & f' & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

The essential matrix is a 3×3 matrix with one zero and two identical singular values. In other words the essential matrix can be factored in the following way:

$$\mathrm{E} = \mathrm{U}\ diag(1,1,0)\ \mathrm{V}^{\mathrm{T}} \tag{2.20}$$

Using the canonical form of the cameras $\mathrm{P} = [\ \mathrm{I} \mid 0\ ]$ and $\mathrm{P}' = [\ \mathrm{R} \mid \mathrm{t}\ ]$, the matrix R and the vector t must be recovered. Later it will be shown, that the matrix R can be converted to RPY rotational parameters. Using the factorization of 2.20 there are four solutions based on the essential matrix[HZ00]:

[8]We suppose all images are taken with the same camera, thus the principal point is the same for all images.

$$\mathrm{P}' = [\,\mathrm{U}\ \mathrm{W}\ \mathrm{V}^{\mathrm{T}}\ |\ +\mathrm{u}_3\,] \tag{2.21}$$

or

$$\mathrm{P}' = [\,\mathrm{U}\ \mathrm{W}\ \mathrm{V}^{\mathrm{T}}\ |\ -\mathrm{u}_3\,] \tag{2.22}$$

or

$$\mathrm{P}' = [\,\mathrm{U}\ \mathrm{W}^{\mathrm{T}}\ \mathrm{V}^{\mathrm{T}}\ |\ +\mathrm{u}_3\,] \tag{2.23}$$

or

$$\mathrm{P}' = [\,\mathrm{U}\ \mathrm{W}^{\mathrm{T}}\ \mathrm{V}^{\mathrm{T}}\ |\ -\mathrm{u}_3\,] \tag{2.24}$$

where

$$\mathrm{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

and u_3 is the column of U corresponding to the zero singular value, which is the last column using the decomposition of 2.20. The geometrical interpretation of the four solutions is very simple. Since the fundamental matrix F is determined only up to scale, its sign is indeterminate. Also, the vector t can be determined only up to sign. Hence, there exists *four* solutions depending on the choice of the signs of $\pm E$ and $\pm t$. This indeterminacy is due to the fact that the mathematical expression for perspective projection is the same if the object is *behind* the camera. The four solution correspond to the four possible configurations of the relative positions of the object and the two cameras. We have to choose the one for which the object is in front of both cameras (see figure 2.5).

If we want to export the camera motion to an external 3D software the 3×3 rotational matrix is not sufficient. We have to find the 3 RPY rotational angles of roll, pitch and yaw, which is the common way to define orientation (see figure 2.6):

$$R = \mathrm{RPY}(\alpha, \beta, \gamma) = \mathrm{Rot}(z, \alpha)\mathrm{Rot}(y', \beta)\mathrm{Rot}(x'', \gamma) =$$

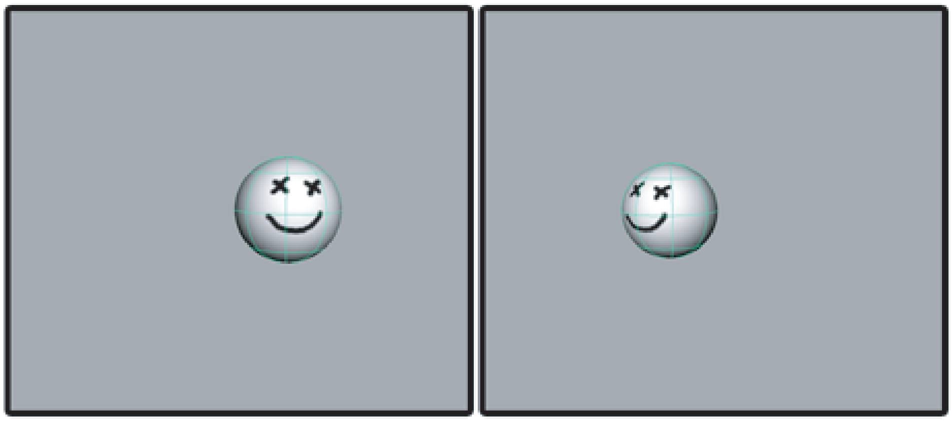

Figure 2.4: Suppose we have these two images taken from the same object.

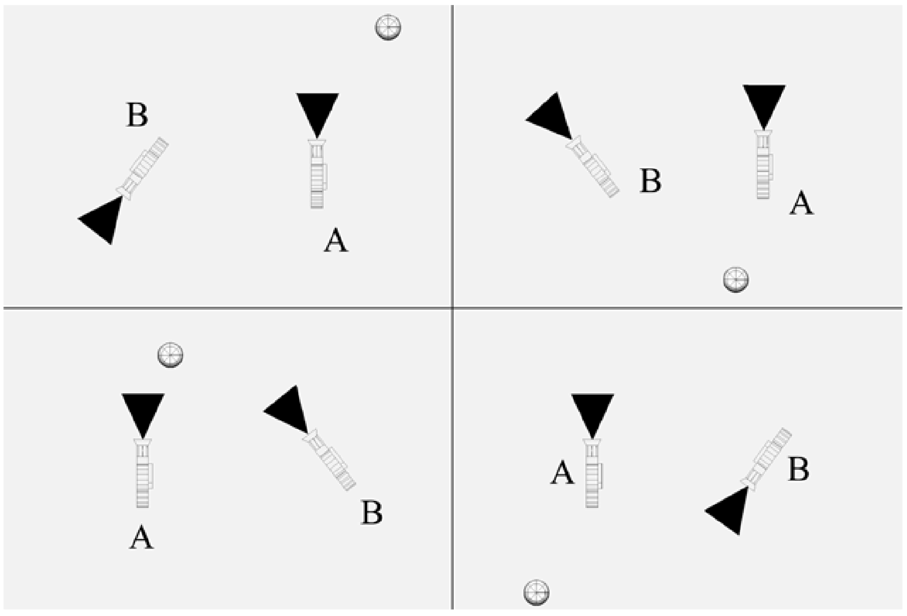

Figure 2.5: There are four camera set-ups, that correspond to the images. Note that there is only one solution, when all 3D points lie in front of both cameras.

$$= \begin{bmatrix} C_\alpha & -S_\alpha & 0 \\ S_\alpha & C_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} C_\beta & 0 & S_\beta \\ 0 & 1 & 0 \\ -S_\beta & 0 & C_\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\gamma & -S_\gamma \\ 0 & S_\gamma & C_\gamma \end{bmatrix} =$$

$$= \begin{bmatrix} C_\alpha C_\beta & C_\alpha S_\beta S_\gamma - S_\alpha C_\gamma & C_\alpha S_\beta C_\gamma + S_\alpha S_\gamma \\ S_\alpha C_\beta & S_\alpha S_\beta S_\gamma + C_\alpha C_\gamma & S_\alpha S_\beta C_\gamma - C_\alpha S_\gamma \\ -S_\beta & C_\beta S_\gamma & C_\beta C_\gamma \end{bmatrix} \quad (2.26)$$

where α, β, γ are the angles we are looking for, and C and S denote the cosine and sine functions. We have to solve the equation 2.27 to get α, β, γ.

$$\mathrm{RPY}(\alpha, \beta, \gamma) = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \quad (2.27)$$

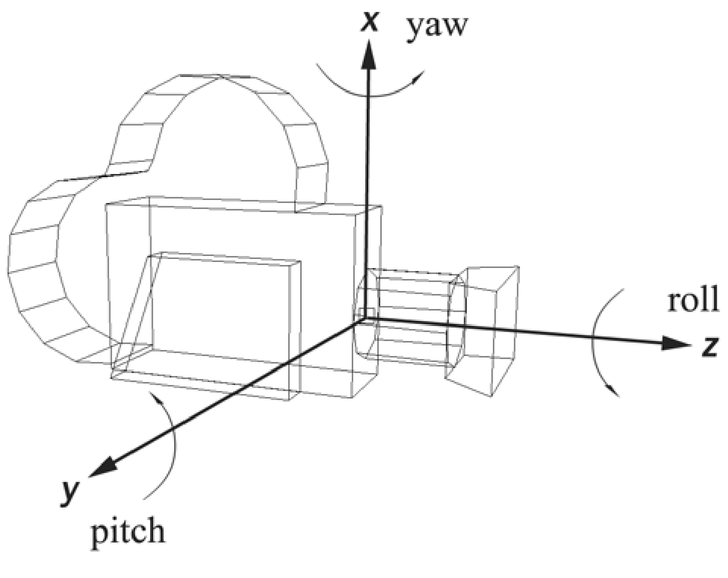

Figure 2.6: The RPY angles describe orientation and can drive the virtual camera of the 3D softwares.

The result of the equations 2.27 are the following [Lan91]:

1. If $R_{11} = R_{21} = 0$ then $C_\beta = 0$. There is two possibility:

   (a) If $R_{31} = -S_\beta = 1$ then $\beta = -\pi/2$, and

$$\alpha + \gamma = \arctan\left(\frac{R_{23}}{R_{13}}\right) + \left\{\begin{array}{c} 0 \\ \pi \\ -\pi \end{array}\right\} \tag{2.28}$$

   (b) If $R_{31} = -S_\beta = -1$ then $\beta = \pi/2$, and

$$\gamma - \alpha = \arctan\left(\frac{R_{12}}{R_{22}}\right) + \left\{\begin{array}{c} 0 \\ \pi \\ -\pi \end{array}\right\} \tag{2.29}$$

2. If $R_{11}^2 + R_{21}^2 \neq 0$ then $C_\beta \neq 0$. In this case:

$$\alpha = \arctan\left(\frac{R_{21}}{R_{11}}\right) + \left\{\begin{array}{c} 0 \\ \pi \\ -\pi \end{array}\right\} \in (-\pi, \pi) \tag{2.30}$$

$$\beta = \arctan\left(\frac{-R_{31}}{S_\alpha R_{21} + C_\alpha R_{11}}\right) \tag{2.31}$$

$$\gamma = \arctan\left(\frac{S_\alpha R_{13} - C_\alpha R_{23}}{-S_\alpha R_{12} + C_\alpha R_{22}}\right) \tag{2.32}$$

Note that in some special cases only sum (or the difference) of two angles can be determined. In those cases there is basically two options:

- Set always one of the angles to 0 and the other to the total sum.
- If we have the previous values of the angles there is another option. For example if $\alpha + \beta = x$ and we know $\alpha_{\text{old}} + \beta_{\text{old}} = x_{\text{old}}$ then

$$\alpha := \alpha_{\text{old}} + \frac{x - x_{\text{old}}}{2}$$

$$\beta := \beta_{\text{old}} + \frac{x - x_{\text{old}}}{2} \tag{2.33}$$

  might be a good choice.

Chapter 3

Lens distortion

3.1 What is lens distortion?

The camera model described in chapter 2 realizes a *pinehole camera* which projects the straight lines of 3D space to straight lines on the image. If the imaging device used to record the images has some significant non-linear lens distortion, the geometric constraints – in this case the fundamental matrix – will not hold. If this distortion is not corrected the elements of the fundamental matrix will be erroneous, thus the solution might differ from the desired data. The most prevalent form of lens distortion is the *barrel* and the *pincushion* distortion. The first is due to the fact that many wide angle lenses have higher magnification in the image center than at the periphery. This causes the image edges to shrink around the center and form a shape of a barrel. The pincushion distortion is the inverse effect, when the edges are magnified stronger (see figure 3.1).

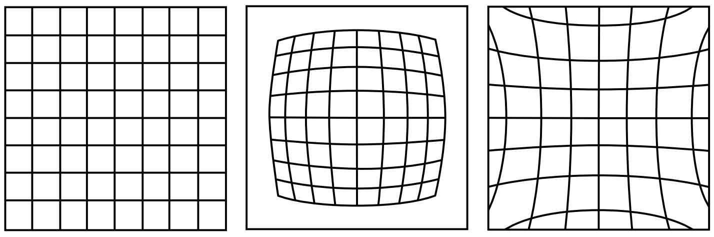

Figure 3.1: The original grid and the effect of barrel (center) and pincushion distortion (right).

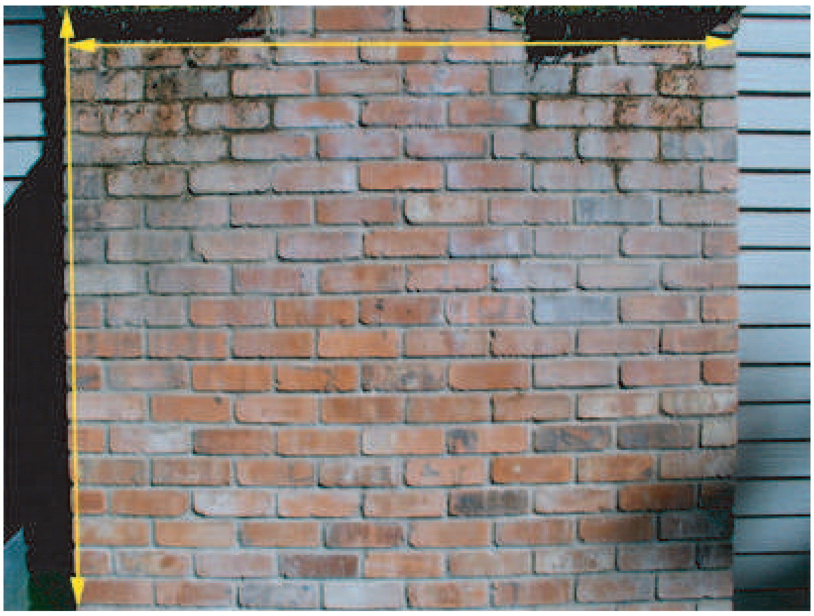

Figure 3.2: Using wide angle lenses recognizable lens distortion occurs. Straight lines become curved.

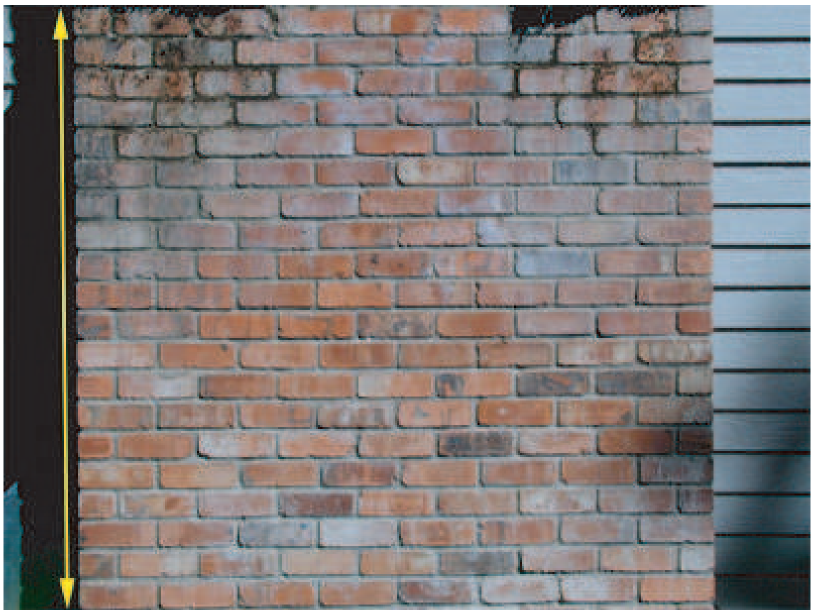

Figure 3.3: Good quality and narrow angle lenses realize correct projection, straight lines remain straight.

The simplest way to model lens distortion is with a shift to the image coordinates. Let $\overline{q} = (\overline{u}\ \overline{v})^T$ denote the undistorted image coordinates, while $q = (u\ v)^T$ represents the observed coordinates. Any radial – symmetric – lens distortion can be approximated with it's Taylor expansion [Ste95]:

$$\overline{q} = f_d(q) = \begin{pmatrix} u \\ v \end{pmatrix} (1 + K_1 r^2 + K_2 r^4 + \ldots) \tag{3.1}$$

where $r^2 = u^2 + v^2$ and $K_1, K_2, \ldots$ are the radial distortion coefficients. In this relation the u, $\overline{u}$, v and $\overline{v}$ coordinates should be not pixel coordinates, since in that case the same image with different resolution would have different coefficients. The image coordinates for the calculations will be *dimensionless coordinates* which are calculated by 3.2.

$$u = \frac{u_{pix} - c_u}{R}$$

$$v = \frac{v_{pix} - c_v}{R} \tag{3.2}$$

where u_{pix} and v_{pix} are the pixel coordinates, c_v and c_v shift the center of the image to $(0, 0)$, and R is the diagonal radius of the image $R = \sqrt{(\frac{u_{pixmax}}{2})^2 + (\frac{v_{pixmax}}{2})^2}$.

There are many different formulas modelling radial lens distortion. The simplest model is based on 3.1:

$$f_d(q) = \begin{pmatrix} u \\ v \end{pmatrix} (1 + K_1 r^2) \tag{3.3}$$

which is a formula with single parameter and manages most of the regular distortions. However, there are some other formulas proposed with one parameter to handle very wide angle lenses, called *fisheye lenses* as well [DF01] [AB95] [PK02]. If more than one coefficient are considered of equation 3.1 we get formulas with two or more parameters applicable for more kinds of radial distortions.

$$f_d(q) = \begin{pmatrix} u \\ v \end{pmatrix} (1 + K_1 r^2 + K_2 r^4) \tag{3.4}$$

3.2 Removing lens distortion

The formula 3.5 used in this thesis is a “production-proven” model, it is currently used in a commercial software package [Sas01]. It is based on the classical distortion model with second order and quartic term (equation 3.4) to handle fisheye lenses, but is not symmetric to work with *anamorphic lenses* as well.

$$\begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix} = g \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u(1 + c_{xx}x^2 + c_{xy}y^2 + \frac{q}{\epsilon}(x^2 + y^2)^2) \\ v(1 + c_{yx}x^2 + c_{yy}y^2 + q(x^2 + y^2)^2) \end{pmatrix} \tag{3.5}$$

where

$$\begin{aligned} c_{xx} &= \frac{\delta}{\epsilon} \\ c_{xy} &= \frac{\delta + \eta_u}{\epsilon} \\ c_{yx} &= \delta + \eta_v \\ c_{yy} &= \delta \end{aligned} \tag{3.6}$$

based on the 5 input parameters:

$\delta =$ primary distortion (default 0)

$q =$ quartic distortion (default 0)

$\epsilon =$ anamorphic squeeze (default 1)

$\eta_u =$ curvature in the u direction (default 0)

$\eta_v =$ curvature in the v direction (default 0)

The equation 3.5 should be evaluated for every pixel on the image to find “source” position of the displaced pixel. All the calculations are in dimensionless coordinates, thus the solution transformed back to pixel coordinates gives a very precise sub-pixel coordinate. In the implemented system bicubic interpolation is used to build the final pixels of the image.

3.3 Applying lens distortion

In computer graphics camera matching is often used to insert computer generated elements into real footage. The pipeline of such effect starts with removing lens distortion from the original plates. After successfully matching the virtual camera to the real one the rendering of the CG images is possible. During the final step the graphics is not inserted into the undistorted footage, but the *distorted* graphics is composited into the *orginal* plates. The reason of this procedure is that the computer generated images can be rendered at higher resolution (super resolution), while the original footage should be used at the best (original) quality.

To apply lens distortion – at high quality[1] – the inverse mapping g^-1 of 3.5 and 3.6 should be found. Since there is no analytic expression of this inverse transformation a numerical algorithm should be used. In the current implementation *Newton's method* was used [WPF92]. The basic goal is to find the distorted $(u, v)^T$ coordinates of the undistorted $(\overline{u}, \overline{v})^T$ coordinates. We know:

$$\begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix} = g \begin{pmatrix} u \\ v \end{pmatrix} \tag{3.7}$$

The iterative algorithm to find $(u, v)^T$ is:

$$\begin{pmatrix} u \\ v \end{pmatrix}^{(j+1)} = \begin{pmatrix} u \\ v \end{pmatrix}^{(j)} - \left(\frac{\partial g((u,v)^{(j)})}{\partial((u,v)^{(j)})} \right)^{-1} \left(g \begin{pmatrix} u \\ v \end{pmatrix}^{(j)} - \begin{pmatrix} \overline{u} \\ \overline{v} \end{pmatrix} \right) \tag{3.8}$$

The success and quality of the solution of this method depends basically on two things: the number of maximum steps and the initial guess. The latter $(u, v)^{(0)^T}$ used in [Sas01] was the first iteration of 3.7, but after testing it it turned out, that with extreme input values it fails to converge. Based on extensive tests the image center $(0, 0)$ turned out to be the

[1]There is a much faster algorithm discussed later with not that good quality

best choice of initial guess. With non-extreme cases it makes the algorithm a little bit slower, but it converges always. The maximum step number at 5 was enough for most of the cases but for the sake of accuracy 30 was set in the final implementation.

3.4 Apply distortion fast

Both the previously described removing and applying methods worked like the code below:

```
for all (x,y) pixels%
{
   (u,v)=dimensionless_coordinate_of(x,y)
   (u_res,v_res)=apply_distortion(u,v)
   (x_res,y_res)=back_to_pixel_coordinate(u_res,v_res)
   result_color(x,y)=source_color(x_res,y_res)
}
```

This method calculates for each pixel in the result image an exact position – at subpixel accuracy – on the source image, where it's color should be sampled. If this source position is exactly the pixel's own position nothing happens. But if the input parameters are set to some arbitrary value, the pixels will be displaced according to the lens distortion model. The Newton iteration runs for each pixel only once – if the parameters are not animated – if we store the results in a displacement buffer.

To apply distortion very fast – and avoid using the slow Newton iteration – a different kind of approach can be used. If we do not require sub-pixel accuracy it is possible to *shift* the pixels according to 3.5, and get the effect of the inverse transformation. In this case the pseudo-code of the program looks like this

```
for all (x,y) pixels%
{
   (u,v)=dimensionless_coordinate_of(x,y)
   (u_res,v_res)=remove_distortion(u,v)
   (x_res,y_res)=back_to_pixel_coordinate(u_res,v_res)
   result_color(x_res,y_res)=source_color(x,y)
}
```

This version goes over all pixels in the original image and shifts the pixel to the required position. This algorithm produces not very good quality results, since some pixels map to the same position while some pixels remain "empty" in the final image. To avoid such empty pixels a cheap trick can be used: if no pixel from the original image shifts to a certain pixel position – eg. it remains empty – we should check it's neighboring pixels and average their color. Using this method the application of lens distortion is extremely fast, while the quality is enough only for previewing. See results on figure 3.4.

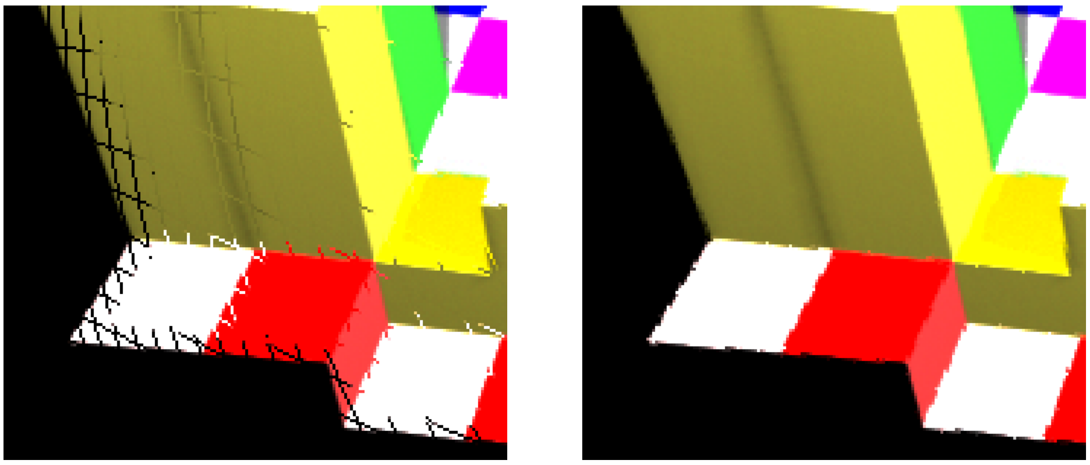

Figure 3.4: Using the fast apply algorithm the pixels are displaced, thus near the edges of the image black lines may occur. On the right image the "empty" pixels are filled based on neighbors. This algorithm is very fast.

3.5 Automatic calibration

It is very uncomfortable and also inaccurate to set the lens distortion parameters by hand. There are several different methods to calibrate these attributes automatically. Some of the proposed methods require an image of a *calibrating grid* or *reference object* of known structure [Zha00] [BB01] [Bai03]. In the production environment these reference images are rarely present.

The newer methods do not require any reference images. Most of the calibration methods search the distortion parameters and the camera parameters – internal and external –

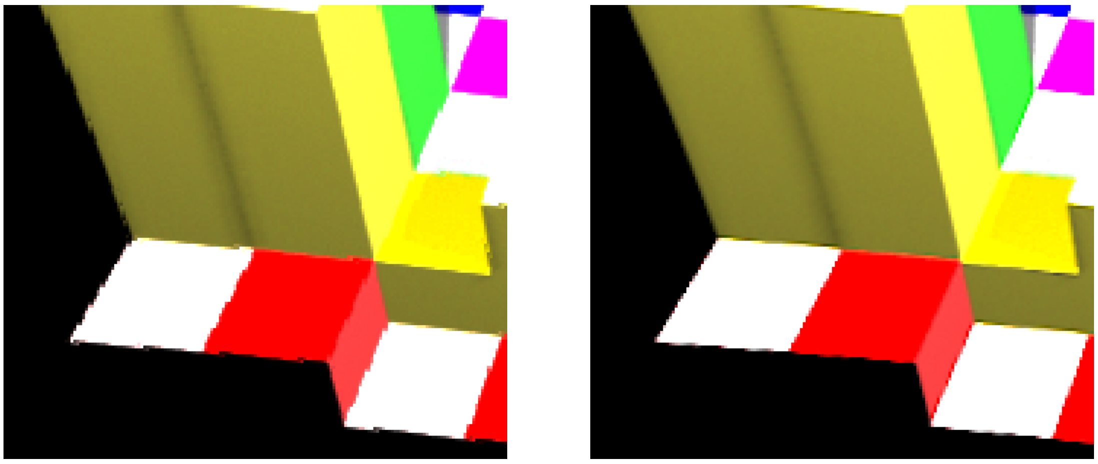

Figure 3.5: Calculation with Newton iteration. On the left image there is no interpolation, the right was generated using bicubic interpolation.

in a separate step. These algorithms use some constraint or error function to successfully find the best parameters to remove distortion. These constraints include:

- Straight lines in the scene should remain straight on the final, undistorted image. The error is proportional to this kind of deviation. This algorithm is usually referred as the *plumb line* method [DF01] [RV02].
- Circles should remain circles (usually needs some calibrating pattern containing a circle).
- Lens distortion introduces specific higher-order correlations in the frequency domain. These correlations can be detected using tools from polyspectral analysis. The amount of distortion is then estimated by minimizing these correlations. [HF01]

Some of the most recently proposed algorithms find the lens distortion and all the camera parameters in one step [Ste97] [Fit01]. These methods use some simple distortion model, thus they often cannot handle fisheye or anamorphic lenses. This is why in this thesis a more sophisticated distortion model and only semi-automatic calibration algorithm is used based on straightening lines. The user has to define lines on the image with several points which are supposed to be straight. The algorithm tries to calibrate one parameter at a time with a very simple algorithm. The pseudo code of the calibration of the parameter called “param” is as follows:

```
stepsize=0.1
while(!good)%
{
   error_minus=test_with(param-stepsize)
   error_here=test_with(param)
   error_plus=test_with(param+stepsize)
   if(error_here is smallest){stepsize/=2}
   if(error_minus is smallest){param-=stepsize}
   if(error_plus is smallest){param+=stepsize}

   if(stepsize<threshold) good=TRUE
}
```

The “test_with” function displaces the points of the lines according to the current distortion parameters and calculates the error[2]. This rather simple method assumes the error function not to have local minima, which is obviously the case[3]. The automatic calibration in the implemented version searches for the primary and the quartic distortion parameters (one after another several times). The algorithm – despite its simpleness – has worked well for every test we made.

[2]The error is the summed squared distance of the inner points of the lines from the line connecting the first and the last points.

[3]It is not possible that the lines become more curved as the parameters approach the ideal set up.

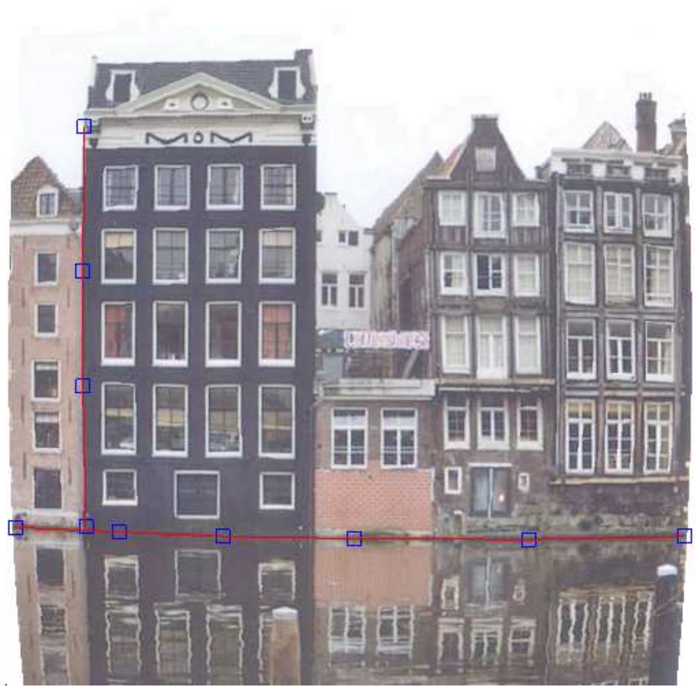

Figure 3.6: The user has to draw lines which should be straight.

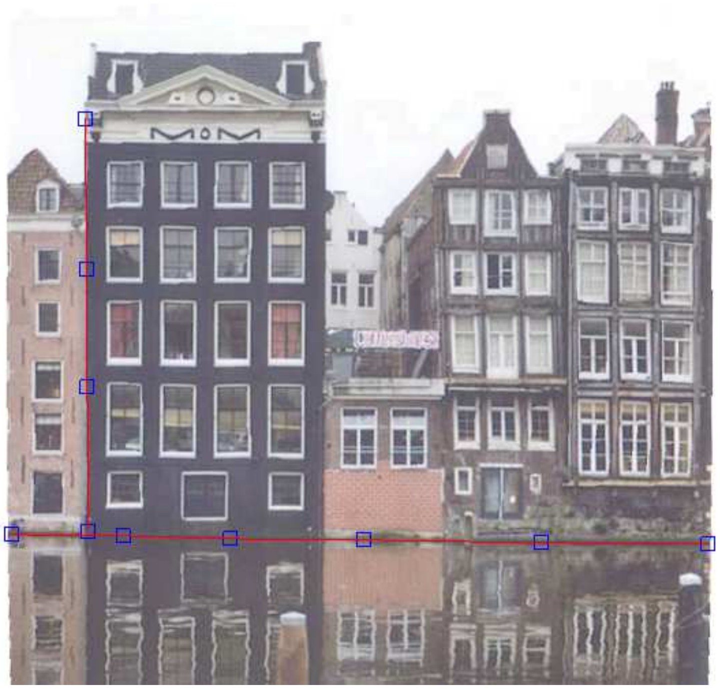

Figure 3.7: The parameters are automatically calibrated and the lines become straight.

Chapter 4

Implementation issues

The software solutions for lens distortion and for camera matching were implemented separately. The first is a plug-in for the image processing and grading system called *Lustre*[1] , while the latter is a stand alone "command line" application with no graphical user interface (GUI). For testing the data of the tracked points were exported from different softwares: a 3D software called Maya was used to generate "perfect" data, while the tracker of Lustre was used to acquire somewhat noisy, measured data.

4.1 Lens distortion plug-In

The plug-in API of Lustre allowed to simply generate GUI elements – push buttons, sliders, radio buttons, labels – and use the image sequence saving/loading capability of the software. All input user interface elements are animatable. Basically the plug-in works on two image buffers provided by the API, one for the input and one for the output. The plug-in has to set the pixel values in the output buffer, based on it's calculations. Since it would be a major waste of processing time to calculate the displacement vectors for every pixel on every frame, the plug-in allocates for itself a 32 bit buffer to store these vectors. Dimensionless coordinates are used to describe the vectors at sub-pixel accuracy. This buffer stores the output of either the removing or the applying method,

[1]Lustre is a product of the software company Discreet, and is developed by Colorfront Ltd. For more details refer to www.discreet.com and www.colorfront.com

thus the rendering of an image sequence does not require the recalculation at every frame (see figure 4.1). It would be certainly very slow, especially the Newton iteration. For each pixel of the output buffer the displacement vector defines a sample position on the source (input) image, where the color information should be read from. This color is calculated with either bicubic interpolation or based on the nearest neighbor.

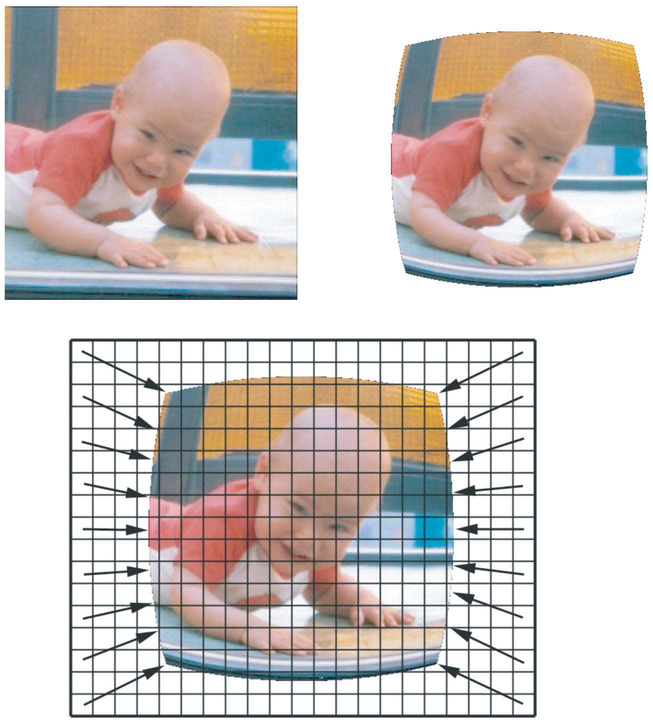

Figure 4.1: Every pixel of the final image is sampled from the source image based on a 32bit buffer of displacement vectors.

All the input parameters of the distortion model and the image parameters can be set by sliders: primary distortion, quartic distortion, anamorphic squeeze, curvature x and y, center x and y of the image (see figure 4.2). However, the user can also define up to 10 lines – consisting maximum 10 points – on the screen, which are supposed to be straight, for automatic calibration. The plug-in sets the primary and the quartic distortion based on these lines.

The automatic calibration is an extremely simple algorithm, still it worked 100% fine.

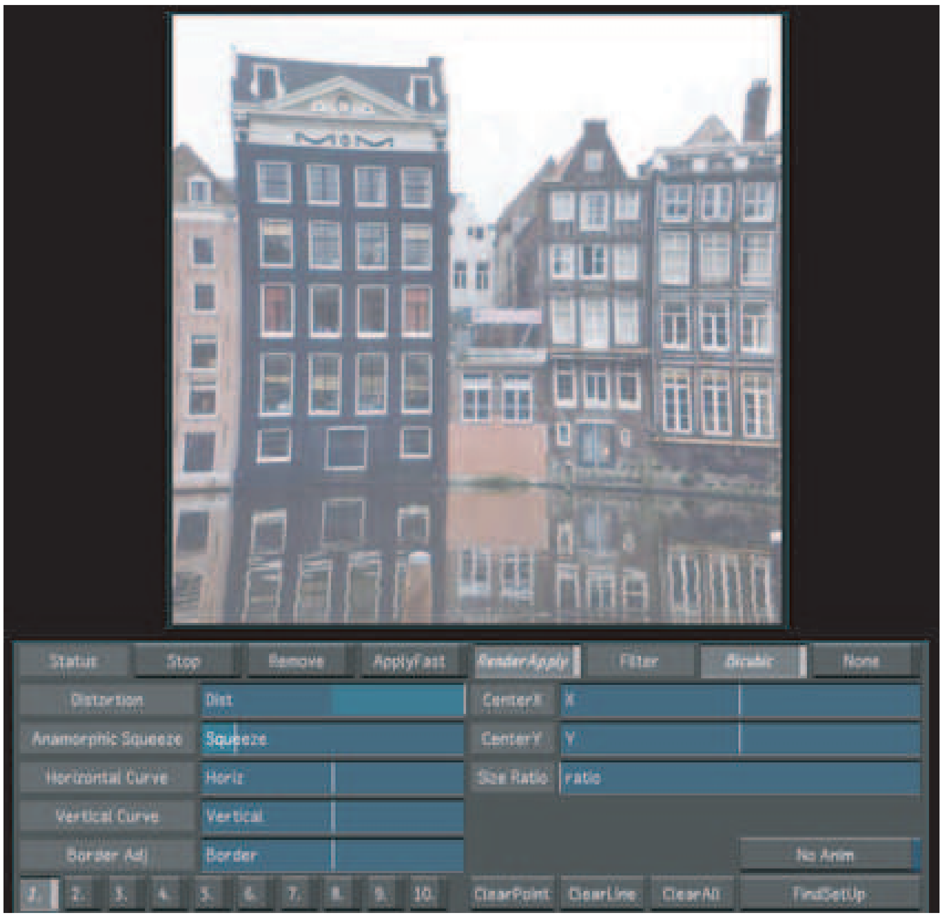

Figure 4.2: The user interface of the lens distortion plug-in.

The search for the primary and the secondary parameters was realized in two separated loops following each other (see figure 4.4). The loops are run until the error reaches a minimum threshold. The primary value is evaluated first, and then the secondary. It is interesting, that the model almost never got to change the secondary distortion parameter, since by setting the first one the lines have become straight. This means that the search was done in most of the cases in one loop. This proves that the mathematical model is sufficient, and the simpler form of the model – involving only one parameter – certainly would be fine for most of the materials in a post production studio.

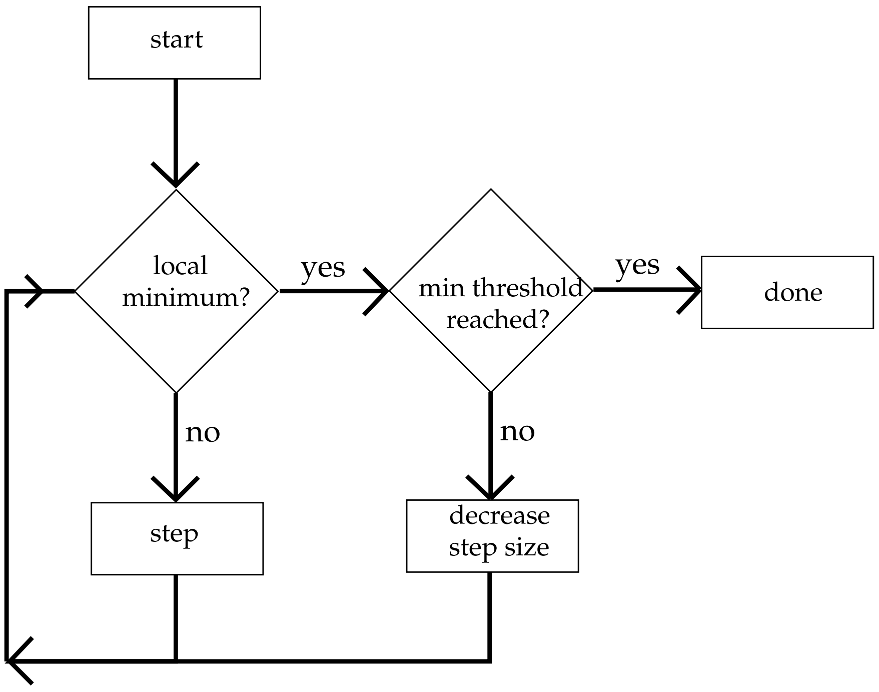

Figure 4.3: The search loop for one distortion parameter.

The loop starts with initial distortion values and the step size set to default. The test for local minimum calculates the error in the positive and negative directions at a distance of the step size. The error is the deviation of the lines from straight. If in the positive or negative direction the error is smaller, the algorithm does a step in that direction. If local minimum is reached the step size is set finer. The initial value of the step size was 0.1 while the minimum threshold of the step size was 0.0001. Each level the step size was divided by 4. The algorithm always converged very fast, since this problem has no other local minimums than the solution. It is not possible that the lines become more curved while setting the lens distortion values closer to the desired numbers.

As mentioned in the previous section it is sometimes necessary to apply distortion to a images with extra border. This means that instead of just applying distortion to the image, we generate a larger version, apply distortion a special way to this large image, and finally crop the inner part to the original size. The resulting image should be distorted the same way the normal image would, and there will be no gaps. An example is shown in figure **??**: Let us suppose digital background has to be inserted into video images. The lens distortion parameters are acquired from the video-sized source. The distorted computer generated grass field – if rendered at regular size – cannot be put in the background because of the visible black area. If larger images are rendered and the application of distortion is handled correctly – *not* the regular way – the cropped result will fit perfectly into the background.

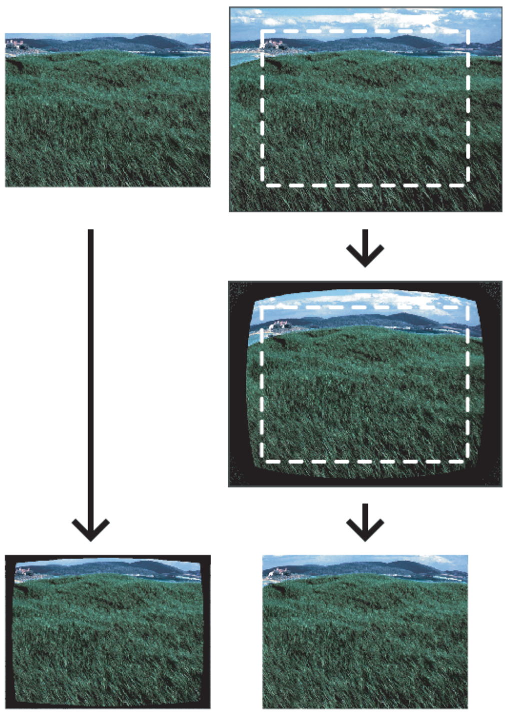

Figure 4.4: Computer generated background (in this case a grass field) should be rendered at larger size to avoid black gaps.

This problem was solved by introducing a new parameter called “scale factor”. This parameter alters the way the pixel coordinates are transformed to dimensionless coordinates. If it is set to 1 no change is made. If the parameter is set to 1.2 – meaning the image is rendered at size 120% – the pixel coordinates are transformed in a way, that diagonal image radius is 1.2 instead of 1. This means that the central window corresponding to 100% will have coordinates as if the it was rendered at regular size. The details of this method will be published at the Budapest conference on Computer Graphics and Geometry 2003.

4.2 Track data

There was no algorithms implemented for feature extraction, external tools were used to generate the data of token matching. For the initial testing “perfect” data was needed to verify, that the algorithm works correctly. A 3D software called Maya was used to build a simple 3D environment and a moving camera. Some 3D points were identified with “locators”, and the screen space positions of these points – calculated based on the camera position and projection parameters – were exported to a text file. These (x, y) positions were more than 4 digit precise, which enabled a very exact reconstruction. After the camera matching software worked well with perfect, calculated data, the testing with acquired data begun.

Some of the camera matching algorithms based on token matching extract tokens from the images independently [PBZ96], and then match them over image pairs or triplets while building the fundamental matrix or the trifocal tensor robustly. This robustness means that mismatched tokens and small, moving objects in the scene – that are obviously not compatible with the constraints of the fundamental matrix – called outliers [Tor95] are filtered out using some robust estimator like RANSAC[2]. The use of these robust estimators is rather slow, since they calculate the fundamental matrix several times for each frame.

The approach followed in this thesis was to track feature-points over the image se-

[2]RANSAC stands for RANdom SAmple Consensus algorithm proposed in [FB81]. A detailed description is also in [HZ00].

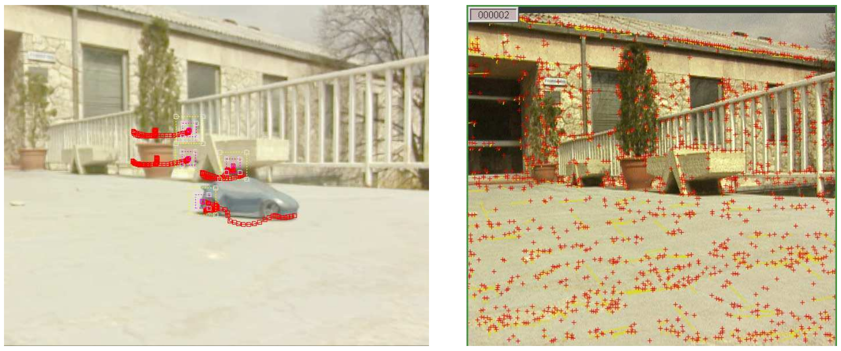

Figure 4.5: The tracker used to test the implemented program was a very fast point-tracker, while other commercial camera matcher softwares track automatically hundreds of points.

quence, thus generate point matches rather fast. However, if there is a moving object in the scene, and some trackers are attached to it, the solution of the fundamental matrix can be useless. Right now it is up to the user to track only points of the static scene. The current version of Lustre tracks user defined points identified by their surrounding region of some specified radius. The algorithm tries to find this region in the search area that is similar to the tracked region on the previous image. If the "look" of the tracked feature slightly changes over time the tracker adjusts itself. Since the tracking algorithm is extremely fast – real time – it is easy to track many features in the sequence. The image coordinates of the tracked points are exported to a text file, which is later processed with the stand alone camera matcher.

4.3 The camera matcher

The camera matcher program is a stand-alone application. It reads an input file containing:

- The resolution of the source images.
- The number of tracked points.
- The number of frames
- The coordinates of every tracker on every frame.

After reading the input file the reconstruction of the relative camera motion is computed for every image pair. The calculation of a single pair starts with the necessary normalization and then the normalized 8-point algorithm is run (refer to section 2.2.1.). The final step of computing the fundamental matrix is the correction with the normalization matrices. The quality of the solution is described with the following formula:

$$\sum_{\mathrm{i}} (\mathrm{x}'_{\mathrm{i}}\ \mathrm{F}\ \mathrm{x}_{\mathrm{i}})^2 \tag{4.1}$$

since the defining equation of the fundamental matrix is:

$$\mathrm{x}'_{\mathrm{i}}\ \mathrm{F}\ \mathrm{x}_{\mathrm{i}} = 0 \tag{4.2}$$

After calculating the fundamental matrix and it's error, the focal distance is calculated using the process of Kanatani *et al.*, which requires only the entries of the fundamental matrix. If the result seems to be wrong – which might be due to noise in F or the wrong estimation of the principal point – the user defined (or guessed) focus distance is used.

Having the fundamental matrix, the focus distance (or distances) and the estimated principal point position – which is supposed to be in the image center – the essential matrix is calculated. Based on the essential matrix the Euclidean reconstruction is possible. The translational vector and the matrix of the camera rotation is calculated. The program exports to a text file the translation and the RPY values of the two rotational matrices. These values can be used in any 3D package – like in Maya – to drive the motion of the virtual camera.

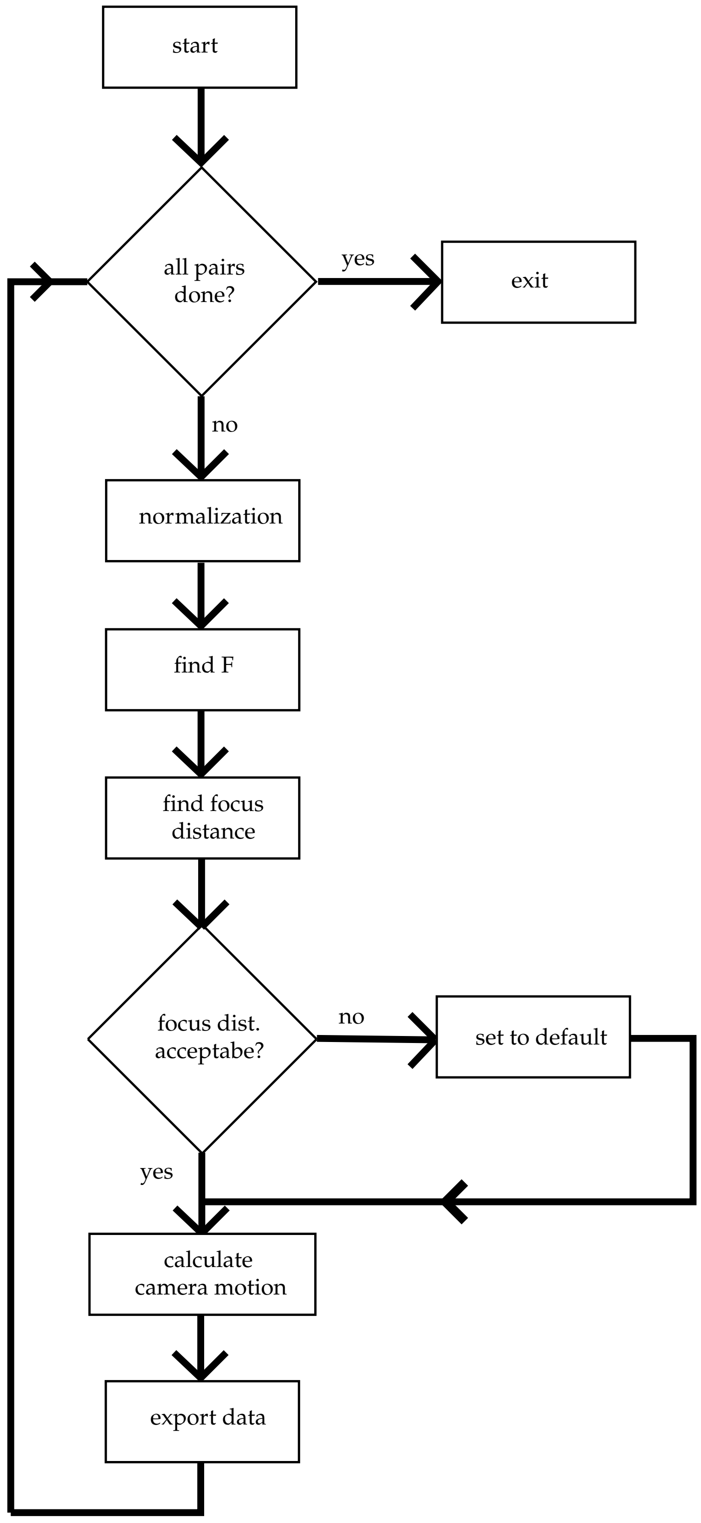

Figure 4.6: The pipe-line of camera reconstruction.

Chapter 5

Camera matching in computer graphics

Since 1996 there are software solutions on the market that are capable to match the virtual camera to the original one based on a image sequence. These solutions are commonly referred as “mach move” softwares, and they make it possible to integrate 3D computer graphics elements seamlessly into live action footage. During the years of development these products have become more and more reliable, and surprisingly while the original products were fully automatic the newer versions include more and more manual or semi-automatic tools. The reason of this trend is that the users – who work mostly in post production facilities – would likely spend some more time in front of their computers to get better results. Although it is known that the mathematical relations behind these programs fail in certain situations – called degenerate configurations – it is not acceptable for a post production studio to work with tools that fail sometimes. This is why tools and algorithms that are not guarantied to converge to the solution are accompanied by some manual or semi-automatic tool.

In the first part of this chapter the typical pipeline of post production studios utilizing match move products is described. After that in the next section the features of the 3 major camera matching products are investigated. Finally the newest and most interesting researches of the field are summarized.

5.1 Integrating computer graphics and live action

The main reason why camera matching is needed in the field of computer graphics is the problem of integrating 3D graphics into live action images. If the virtual camera does not move the exact same way as the original camera the 3D elements that are supposed to stick to a real one will "flow" instead of being static. This error is visible even in those cases, when the 3D objects and the real ones do not touch each other. Typical example is when a floating object has to be put in the air, or an aircraft has to be inserted in a shot, where the background is live action. One may think that the digital aircraft or the floating object is neither attached to the ground nor the camera, so it's movement is not constrained at all. However, if we insert the animated 3D object without matching the virtual camera to the background the result will be not convincing at all[1]. Of course if the camera does not move at all, the camera matching is unnecessary.

Because the matching is a key issue when combining CG elements with real footage this problem had to be solved a long time ago. The first hardware solutions were invented way before 3D computer graphics was used in feature films. These expansive camera rigs – called motion control rigs – were driven by computers and were used to combine different live action footage. It is not surprising that the first method of camera matching was to drive the real cameras with computers, and use their data in the 3D software to generate digital elements. This method was rather expensive and not flexible at all, since for every effect-shot motion control had to be used. The software based solution for camera matching based on only image sequences revolutionized the way 3D special effects were done. The extra 3D elements could be put in any shot, no *a priori* information was needed about the original camera parameters.

[1]The reason of this is the fact, that the 3D object and the camera has to obey the laws of physics. If the camera is not matched to the original one, the relative motion of the 3D element to the background will break these laws.

5.2 Typical pipe-line

The computer vision algorithms used for camera matching today require only images, however, any measured data from the shooting – including distances of objects, camera and lens parameters – improves to solution significantly. This is why all shooting possibly involving special effects start by thorough pre-planning and measuring every useful data on the set. The first step after the images are in the computer is the removal of lens distortion and the tracking. If no image of any calibration grid is available, straight lines may be used to remove the distortion.

Tracking includes not only the – automatic or manual – point tracking but also often some manual segmentation as well. The user has to define which portion of the image is supposed to be the static background and what are the moving objects. For the sake of illustration the camera matching tests made by Digital Dimension studio[2] for the movie Final Destination 2. are used (figures 5.1 to 5.4). On figure 5.1 the tracking of the shot is visible, the road and the moving truck had to be separated manually. If an automatic tracker is used the user has to create *masks*, that define regions on the image to separate the different objects.

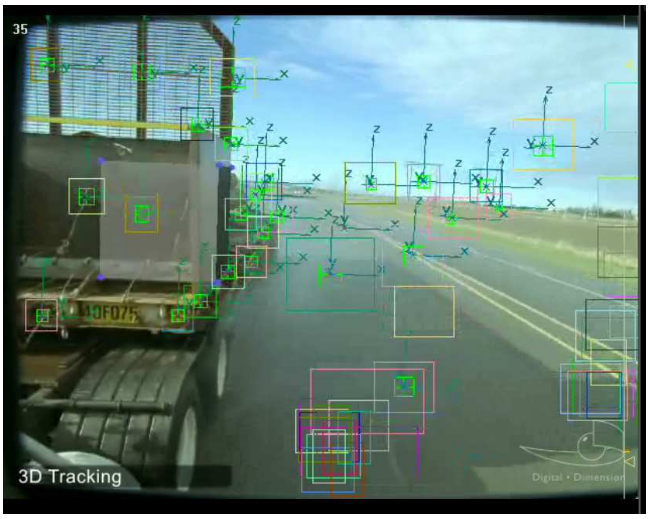

Figure 5.1: Tracking points in the scene. (Digital Dimension Studio, ©New Line Cinema)

[2]For more details about the studio and particularly this project visit www.digitaldimension.com.

After successful tracking the process of camera matching may be started. In scenes with multiple moving objects all elements has to be solved separately, in our case the truck and the road are the two elements. After the camera matching, the relative motion of the camera to these elements will be available. From this data it is easy to reconstruct the camera motion and the truck's motion relative to the ground. The verification of the matching – in post production environment – is almost always qualitative. Simple 3D test objects are inserted in the shot, and the viewers decide wether they seem to be part of the scene or not (see figure 5.2). In some cases the solution seems to be perfect despite of the very bad reconstruction of the focal distance, in other cases even the smallest errors are recognizable. Generally speaking, the human perception is very sensitive to motion – especially high frequency motion – but not that sensitive to distortion in perspective.

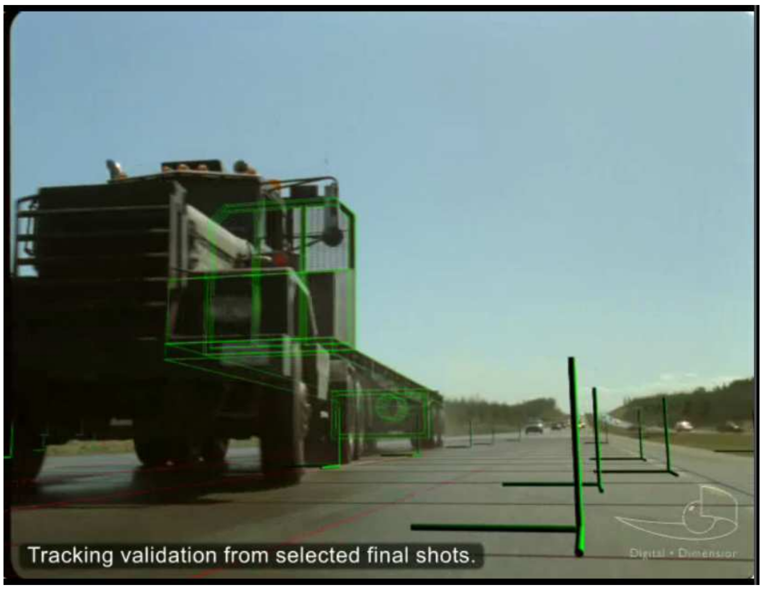

Figure 5.2: The verification of the camera matching with simple 3D objects inserted into the live action footage. (Digital Dimension Studio, ©New Line Cinema)

Following the successful matching of the camera, the virtual copy of the real scene is modelled by graphic artists. This model may be very complex or very simple depending on the requirements of the effect. In our example only coarse models of the truck and the ground were needed for the dynamic simulation of the logs. The modelling of the real scene is not always manual. The match moving products almost always export not only the camera data but also the 3D positions of the tracked points. These points serve as very useful reference for modelling.

After the model of the world and of the 3D objects to be inserted are ready the *interaction* of them is created. There are many different ways 3D objects may interact with the scene

- **Physical interaction**

  The 3D objets of any type – gases, liquids, rigid and soft bodies – may interact with the real world elements via any form of touching or colliding. These interactions – which are more likely *actions* without *inter* since only the 3D objects will react – can be simulated with physical simulations in the 3D software. The simple models of the real elements are used to detect the collisions. In production environment many times the simulations are not physically correct. The animators – based on the directors will – change the parameters like bounciness, weight or gravity to achieve the desired motion.

- **Lighting**

  The digital elements fit into the pictures only if their *look* is believable. This means that the 3D surfaces has to have material properties that are similar to real world materials, and the lighting conditions have to match the original scene. Today the most common way to generate material models or *shaders* is manual like texture painting, shader programming. To match the original lighting conditions the technical directors (TD) of the post production studios often make reference photographs on the set about diffuse and mirror like balls or other reference objects. These pictures can help the TDs to design the shaders and the lighting. The pictures of mirror balls can help to reproduce reflections and place lights manually, but most current techniques allow to extract all lighting information from a HDR[3] image of a mirror ball.

[3]HDR stands for High Dynamic Range image. For more information on that subject refer to Paul Debevec's sire www.debevec.org.

- **Shadows**

  The digital objects always cast shadows to real world elements. To render these shadows the “stand-in” model of all real object has to be created in the virtual environment. Using different rendering techniques the shadows can be computed falling on the stand-in surfaces, and later these shadows can be composited on the original images. If the camera matching was good enough the shadows will look like if they were on the real object.

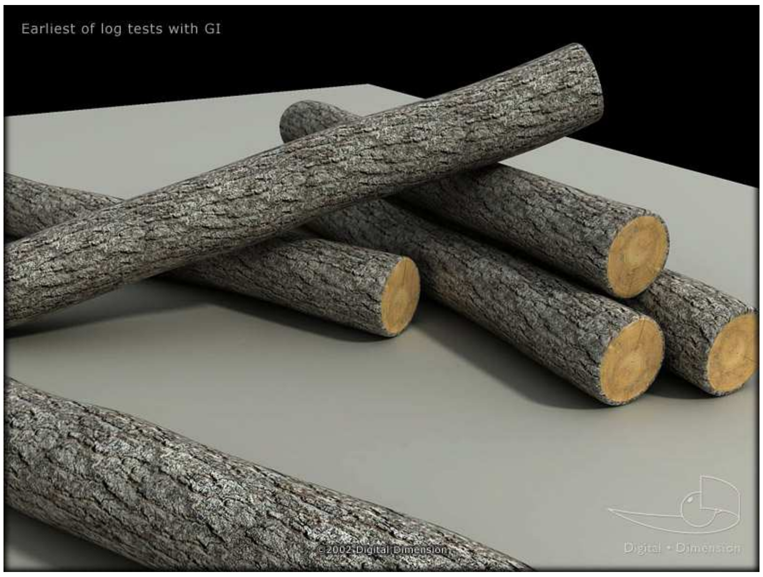

Figure 5.3: The model of the 3D logs that had to be inserted into the shot. (Digital Dimension Studio, ©New Line Cinema)

After the 3D models, the animation, the materials and the lights are set up the rendering of the digital elements can start. Usually the image generation is divided into different passes:

- Diffuse pass: Only diffuse components of the materials.
- Specular pass: Only specular components of the materials.
- Reflection pass

- Shadow pass (sometimes there are different shadow passes: contact shadow, direct shadow).
- Atmospheric effects
- Depth from the camera

These different elements are put together in the stage of compositing. The reason of this is that rendering times are very long, so the re-rendering should be avoided. If everything is rendered separately the compositor – the artist who makes the final pictures from the elements – has the freedom to modify layers, change colors, transparency or apply effects to different layers. The integration of 3D computer graphics and the original images is tricky if there was lens distortion removed prior to tracking. The computer generated elements are rendered with a perfect "virtual" pinhole camera, while the original footage suffers from distortion. One may think that the best solution is to composite the CG elements to the undistorted footage used for tracking. However, the un-distortion process worsens the quality of the live action images. To overcome this problem lens distortion is applied to the CG elements that are later composited on the original footage. The advance of the approach is that the rendered images can be generated at any resolution, thus the quality after applying lens distortion remains excellent.

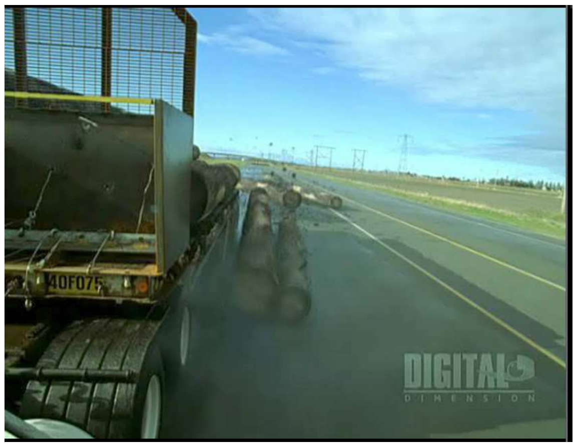

Figure 5.4: The final result, 3D logs fallen off the truck are bouncing on the road. Many layers are composited together. (Digital Dimension Studio, ©New Line Cinema)

Chapter 6

Conclusions and future work

In this chapter the promising new researches are summarized that are related to computer vision and camera matching. In section 6.2 the experiences and conclusions of the work done are collected, while the last section describes the possible future developments.

6.1 Applications in development

The most important use of camera matching in the near future may be the fully automatic model and texture reconstruction at high level. It would be an extremely useful tool for computer graphic artists to be able to generate photo-realistic models with a hand held camera and a software solution.

The 3D reconstruction of a scene is subject of research for years, and this is the field where the most dynamic work have been done recently. For a detailed overview of surface reconstruction refer to [Cro00]. A summary of the reconstruction of feature points and lines are described in the 1996 technical report [PBZ96]. However, the reconstruction of *objects or surfaces* instead of points is not a trivial problem. The 3D reconstruction involves almost always only feature points and sometimes lines, but high quality surfaces cannot be built based on some hundred 3D points. However, if we have very dense information of the object – like when it is rotating in front of the camera – the reconstruction is possible [AWFZ98] (see images 6.1 and 6.2).

In production environment the “rotating table” approach is not always sufficient. Some

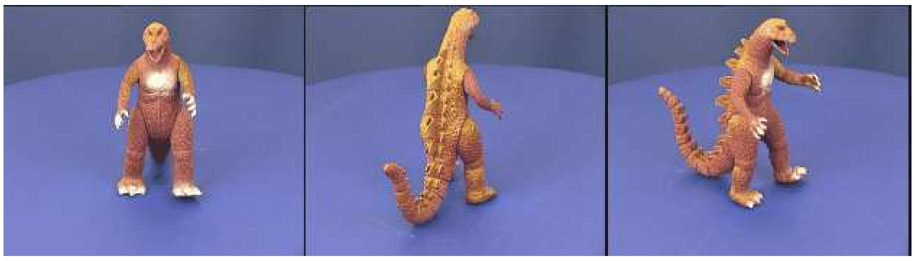
Figure 6.1: The image sequence of the object rotating in front of the camera [AWFZ98].)

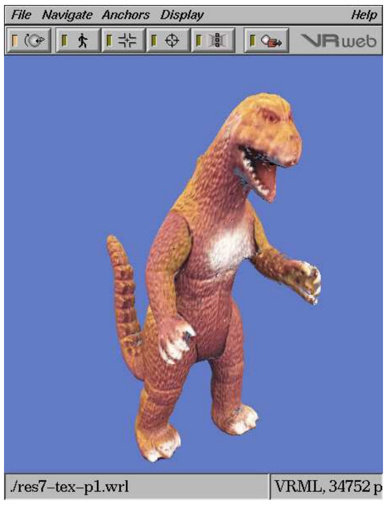

Figure 6.2: The reconstructed and textured 3D model. [AWFZ98])

objects like houses or hills cannot be rotated or filmed in any way from every direction. There are some promising results for such cases as well like the work Mark Pollefeys and Luc Van Gool [MPG00] [PG02]. Their results show that medium quality models with good realistic textures can be acquired with a hand held video camera, and they produced convincing results (see figures 6.3 and 6.3).

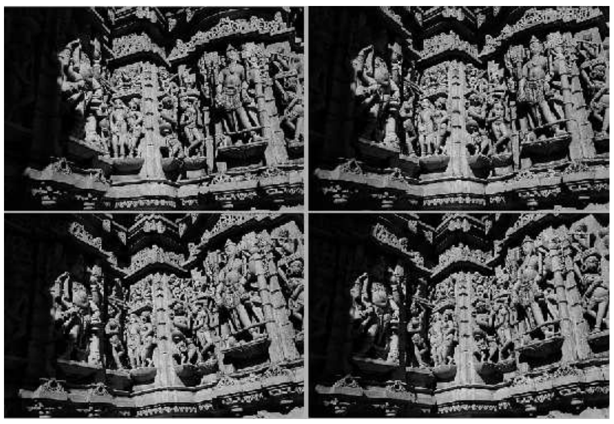

Figure 6.3: The captured images (Mark Pollefeys).

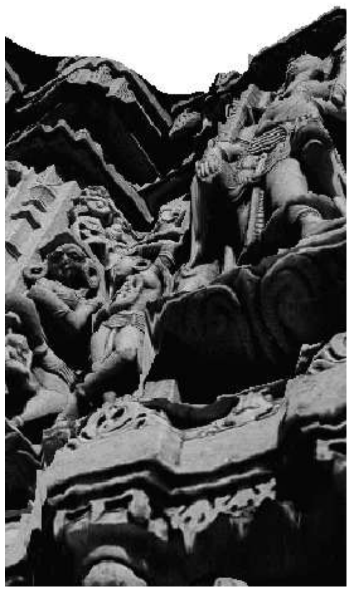

Figure 6.4: The reconstructed object from a different view (Mark Pollefeys).

However, even these models are far from the resolution of the required level for – good quality – computer graphics, thus are used for only a reference in the practice of 3D computer graphics. Some other techniques are semi-automatic like the one proposed by Gibson [GH03]. His approach requires the user to define the 3D objects with the help of the reconstructed points. Very high quality textures are produced for the objects and even the *original lighting environment* may be evaluated.

6.2 Conclusions

A complete tool-set to handle lens distortion for camera matching has been implemented. The mathematical model of the distortion was chosen to handle almost all kinds of lenses that may occur in the post production studios integrating computer graphics into live action images. It is possible to remove and also to apply the very same distortion, thus the CG elements can be distorted to fit into the original images.

The implemented plug-in has a feature that every available lens distortion removing application lacks: it is capable of applying distortion to images with extra border. When the barrel distortion is applied to rendered images the pixels at borders are pulled towards the center generating black gap at the borders. To avoid this larger images are rendered, distorted and cropped. If – for example – a double size image is rendered, the user wants the “window” of the original size in the center to be distorted the same way as expected, not the whole image. With existing tools some tricks had to be used to handle this situation[1]. In the implemented system there is a parameter that sets the relative size of the image to the original one. If this parameter is set to 120% and the user applies distortion to images at the size of 120% of the original and then crops the image back to 100%, it will be distorted the way a normal image would be distorted the regular way.

To calibrate the lens distortion system a semi-automatic algorithm is implemented, which needs only straight lines to be on the images and no calibration grid is needed. The algorithm processes the images very fast, multiple processors are supported as well.

[1]This trick is to calibrate the lens distortion parameters to the original images with extra border. This way the same model can be used for the rendered images with extra border.

For the best quality not only the nearest neighbor but bicubic interpolation is available as well. For the application of lens distortion a new and very fast algorithm was developed and implemented. The advantage of it is that the user might use the plug-in for artistic effects as well, since he or he can set the parameters with almost real-time feedback.

The basic algorithms for camera matching were also successfully implemented. The input of the application is a text file containing the track data of the feature points. The solution is based on the normalized 8-point algorithm, which generates the fundamental matrix from point correspondences. Since based on the fundamental matrix alone the Euclidean reconstruction is not possible (only projective reconstruction). To reconstruct the camera's motion in 3D sufficiently the camera has to be calibrated. The system supposes that the images are taken with conventional cameras, eg. no shear is present and the pixel aspect is 1. The lens center is supposed to be the image center, but this may be altered by the user. The unknown internal parameter of the camera, the focal distance is evaluated from the Fundamental matrix. With these data the program evaluates the "calibrated version" of the fundamental matrix, called essential matrix. From this the rotation matrix and the relative motion is evaluated. The program exports the camera displacement and the RPY values of the rotation to a text file, which can be read in any 3D program to drive a camera. The system worked well for noise free data, however, quantitative analysis has not been done.

6.3 Future work

The current version of the lens distortion plug-in is ready to be used in production, only the automatic calibration needs some improvements. If the user defines the points of the lines not precise enough, the algorithm might give wrong solution. The lines on the images could be automatically detected based on the user's help – to avoid "really" curved lines – which could give a much more precise data to work with. The automatic calibration is also not possible if no lines are on the image, thus some other calibration algorithms might be useful in those cases.

The implemented camera matching algorithm needs much more testing, to have qualitative and quantitative results of it's precision and stability. The response to the so called "degenerate" configurations – like pure translation parallel to the image plane – were not tested either.

After having a stable camera matcher the implementation of the 3D surface reconstruction algorithm can start. To achieve results that can be used in computer graphics semi-automatic approach seems to be the best choice. The help of the user of defining where edges are, what kind of topology the surface has or where smooth regions are could help the reconstruction significantly. Researches did not work on such methods, since in general applications in robotics the requirements are very different. For example in computer graphics the topology of the surface is of key importance while in general computer vision only a dense point-cloud or some triangulated surface can represent the original object well.

Another interesting development could be the use of camera matching in re-lighting any scene. If the camera motion is available a dense depth map can be built from the images, and based on this map the normal vectors can be evaluated. If we have the normal vector information the surfaces can be lit with some local illumination model like the Phong model.

6.4 Acknowledgements

I am grateful to Márk and Áron Jászberényi (Colorfront Ltd.) for supporting this work, and to professor Szirmay-Kalos and Tamás Perlaki for being my consultants.

BIBLIOGRAPHY

[AB95] S. Licardie A. Basu. Alternative models for fish-eye lenses. *Pattern Recognition Letters*, 16:433–441, 1995.

[AWFZ98] Geoff Cross Andrew W. Fitzgibbon and Andrew Zisserman. Automatic 3d model construction for turn-table sequences. In R. Koch and L. VanGool, editors, *Proceedings of SMILE Workshop on Structure from Multiple Images in Large Scale Environments*, volume 1506 of *Lecture Notes in Computer Science*, pages 154–170. Springer Verlag, June 1998.

[Bai03] Donald G. Bailey. A new approach to lens distortion correction. In *Proceedings of Image and Vision Computing, New Zealand*, 2003.

[BB01] Steven S. Beauchemin and Ruzena Bajcsy. Modelling and removing radial and tangential distortions in spherical lenses. *Lecture Notes in Computer Science*, 2032:1–??, 2001.

[Bir98] S. Birchfield. An introduction to projective geometry (for computer vision). Unpublished notes, http://robotics.stanford.edu/˜birch/projective/, 1998.

[Bou98] Sylvain Bougnoux. From projective to euclidean space under any practical situation, a criticism of self-calibration. In *ICCV*, pages 790–798, 1998.

[BP] Jean-Yves Bouguet and Pietro Perona. Motion from points, lines and p-lines on n views. http://www.vision.caltech.edu/bouguetj/.

[Cro00] G. Cross. *Surface Reconstruction from Image Sequences: Texture and Apparent Contour Constraints*. PhD thesis, University of Oxford, 2000.

[DF01] Frederic Devernay and Olivier D. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 13(1):14–24, 2001.

[Fau92] Olivier D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *Proceedings of the European Conference on Computer Vision, Santa Margherita Ligure, Italy*, pages 563–578, 1992.

[FB81] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24:381–385, 1981.

[Fit01] A. Fitzgibbon. Simultaneous linear estimation of multiple view geometry and lens distortion. In *CVPR*, 2001.

[FLM92] Olivier D. Faugeras, Quang-Tuan Luong, and Stephen J. Maybank. Camera self-calibration: Theory and experiments. In *European Conference on Computer Vision*, pages 321–334, 1992.

[FM95] Olivier D. Faugeras and Bernard Mourrain. On the geometry and algebra of the point and line correspondences between n images. In *ICCV*, pages 951–956, 1995.

[GH03] Simon Gibson and Toby Howard. Interactive reconstruction of virtual environments from video sequences. *Computers & Graphics*, 27(2), 2003.

[Har92] R. I. Hartley. Estimation of relative camera positions for uncalibrated cameras. In *Volume 588 of Lecture Notes in Computer Science (Proc. European Conf. Comp. Vision 1992)*, pages 579–587, 1992.

[Har93] R. Hartley. Extraction of focal lengths from the fundamental matrix. Unpublished manuscript, http://www.syseng.anu.edu.au/˜hartley/My-Papers.html, 1993.

[HF01] A.C. Popescu H. Farid. Blind removal of lens distortions. *Journal of the Optical Society of America*, 18(9):2072–2078, 2001.

[HSA02] Richard Hartley and Chanop Silpa-Anan. Reconstruction from two views using approximate calibration. In *Proceedings of the 5th Asian Conference on Computer Vision, Melbourne, Australia*, 2002.

[HZ00] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[KM00] K. Kanatani and C. Matsunaga. Closed-form expression for focal lengths from the fundamental matrix. In *Proceedings of the 4th Asian Conf. Comput. Vision, Taipei, Taiwan*, pages 128–133, 2000.

[Lan91] Béla Lantos. *Robotok irányítása (Controlling Robots)*. Akadémiai Kiadó Budapest, 1991.

[LF96] Q. Luong and O. Faugeras. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision*, 17(1):43–76, 1996.

[LH81] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. In *Nature*, pages 133–135, 1981.

[MPG00] M. Vergauwen M. Pollefeys, R. Koch and L. Van Gool. Automated reconstruction of 3d scenes from sequences of images. *ISPRS Journal Of Photogrammetry And Remote Sensing*, 55(4):251–267, 2000.

[MZ92] J. Mundy and A. Zisserman. *Geometric Invariance in Computer Vision*. MIT Press, 1992.

[PBZ96] P. H. S. Torr P. Beardsley and A. Zisserman. 3d model acquisition from extended image sequences. Technical Report OUEL 2089/96, University of Oxford, 1996.

[PG02] M. Pollefeys and L. Van Gool. From images to 3d models. *Communications of the ACM*, 45(7):50–55, 2002.

[PK02] Janez Pers and Stanislav Kovacic. Nonparametric, model-based radial lens distortion correction using tilted camera assumption. In *Proceedings of the Computer Vision Winter Workshop 2002, Bad Aussee, Austria, February 4-7*, pages 286–295, 2002.

[RH92] T. Chang R. Hartley, R. Gupta. Stereo from uncalibrated cameras. In *Computer Vision and Pattern Recognition (Urbana-Champaign, IL, June 15–18)*, pages 761–764. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[RV02] P.L. Evans R.J. Valkenburg. Lens distortion calibration by straightening lines. In *Proceedings of Imaging and Vision Computing New Zealand*, pages 230–236, 2002.

[Sas01] Uwe Sassenberg. Lens distortion model, 2001. Distortion model of 3DE V3, http://www.3dequalizer.com/sdv_tech_art/paper/distortion.html.

[SK99] L. Szirmay-Kalos. *Számítógépes grafika (Computer Graphics)*. ComputerBooks, Budapest, 1999.

[Spr64] C. E. Springer. *Geometry and Analysis of Projective Spaces*. Freeman, 1964.

[Ste95] Gideon P. Stein. Accurate internal camera calibration using rotation, with analysis of sources of error. In *ICCV*, pages 230–236, 1995.

[Ste97] G. P. Stein. Lens distortion calibration using point correspondences. In *Proceedings of CVPR97 (Computer Vision and Pattern Recognition)*, pages 602–608, 1997.

[Tor95] P. H. S. Torr. *Motion segmentation and outlier detection*. PhD thesis, Dept. of Engineering Science, University of Oxford, 1995.

[WPF92] W. Vetterling W. Press, S. Teukolsky and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[Zha96] Zhengyou Zhang. Determining the epipolar geometry and its uncertainty: A review. Technical Report 2927, Sophia-Antipolis Cedex, France, 1996.

[Zha00] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

SUBJECT INDEX