*Real-Time Imaging* **6**, 433–448 (2000)
doi:10.1006/rtim.1999.0199, available online at http://www.idealibrary.com on IDEAL®

# *Real-Time Camera Calibration for Virtual Studio*

In this paper, we present an overall algorithm for real-time camera parameter extraction, which is one of the key elements in implementing virtual studio, and we also present a new method for calculating the lens distortion parameter in real time. In a virtual studio, the motion of a virtual camera generating a graphic studio must follow the motion of the real camera in order to generate a realistic video product. This requires the calculation of camera parameters in real-time by analyzing the positions of feature points in the input video. Towards this goal, we first design a special calibration pattern utilizing the concept of cross-ratio, which makes it easy to extract and identify feature points, so that we can calculate the camera parameters from the visible portion of the pattern in real-time. It is important to consider the lens distortion when zoom lenses are used because it causes nonnegligible errors in the computation of the camera parameters. However, the Tsai algorithm, adopted for camera calibration, calculates the lens distortion through nonlinear optimization in triple parameter space, which is inappropriate for our real-time system. Thus, we propose a new linear method by calculating the lens distortion parameter independently, which can be computed fast enough for our real-time application. We implement the whole algorithm using a Pentium PC and Matrox Genesis boards with five processing nodes in order to obtain the processing rate of 30 frames per second, which is the minimum requirement for TV broadcasting. Experimental results show this system can be used practically for realizing a virtual studio.



**Seong-Woo Park, Yongduek Seo and Ki-Sang Hong**[1]

*Dept. of E.E., POSTECH, San 31, Hyojadong, Namku, Pohang, Kyungbuk , 790-784, Korea*

## Introduction

In the broadcasting area, composing real objects with a graphic background has long been realized using Chromakeying [1]. Chromakeying systems, despite their usability, impose a very strong constraint upon the system: the camera cannot move. This is because the graphic background cannot be changed according to the camera's movement, due to the difficulties in computing the parameters of the real camera.

In recent years, the improvement of computing power and graphics technology has stimulated people to try and decorate the real scene with a graphic virtual background — the virtual studio [3–9]. They mix real scenes from a real camera with virtual scenes generated

[1]Corresponding author. E-mail: hongks@postech.ac.kr, tel. +82-54-279-2216

1077-2014/00/120433+16 $35.00/0

from a graphics machine, for which the extrinsic motion parameters — rotation and translation — of the real camera and intrinsic parameters like focal length are necessary to make the virtual views follow exactly the same motion as the real camera. Thus, a camera tracking system is needed to get the intrinsic and extrinsic parameters of the real camera. Especially, the camera tracking systems used in the virtual studio can be divided into two main categories: electromechanical systems and optical systems [10]. Electromechanical systems were developed first and are still more commonly used. However, even though they can yield highly accurate parameters from the usage of electromechanical sensors, they have several disadvantages:

- Calibration — accurate registration requires intimate knowledge of lens characteristics. For example, a mathematical function of zoom and focal length must be given to determine the horizontal field of view for the camera lens [10].
- Alignment — the initial state of the camera must be accurately aligned.
- Number of systems — every camera needs its own system.
- Operability of a camera — operators cannot arbitrarily move the camera freely in three-dimensional space. In order to do that, a huge mechanical setup is required.

The optical system — the alternative to electromechanical tracking — is based on pattern recognition. Despite the fact that this method can overcome the constraints of electromechanical tracking, it is not widely used because of the difficulties in achieving camera parameters in real time with the accuracy needed for the virtual studio. In this paper, we address our optical camera tracking system.

Figure 1 shows the overall image flow of our method from the input image (a) to the resultant composed image (e). In this flow, the input image (a) is passed to the camera tracking system where the camera parameters are calculated from the extracted special pattern we designed (see (b)). Then, using this information, the graphics background is rendered (see (c)). Meanwhile, the input video is delayed to compensate for the delay of the camera tracking system, and then it is composed into the graphics background. Figure 1(d) shows the objects in the foreground scene.

The first thing we have to do to get camera parameters in real time is to obtain the 2D–3D pattern matches in order to measure the camera parameters from the pattern in the image. Thus, we design a planar pattern so that the feature extraction, image-model matching and identification are fast and reliable for the real time implementation. The pattern is designed to have a series of the cross-ratio, which is invariant under 2D projective transformation [11,12], to automatically identify the feature points wherever the pattern is seen and even when only a part of it appears. In this paper we will introduce how to design the pattern by applying the concept of cross-ratio and how to identify the pattern automatically.

After this process of identifying the image and pattern, camera calibration is performed using the identified feature points. The calibration algorithm we

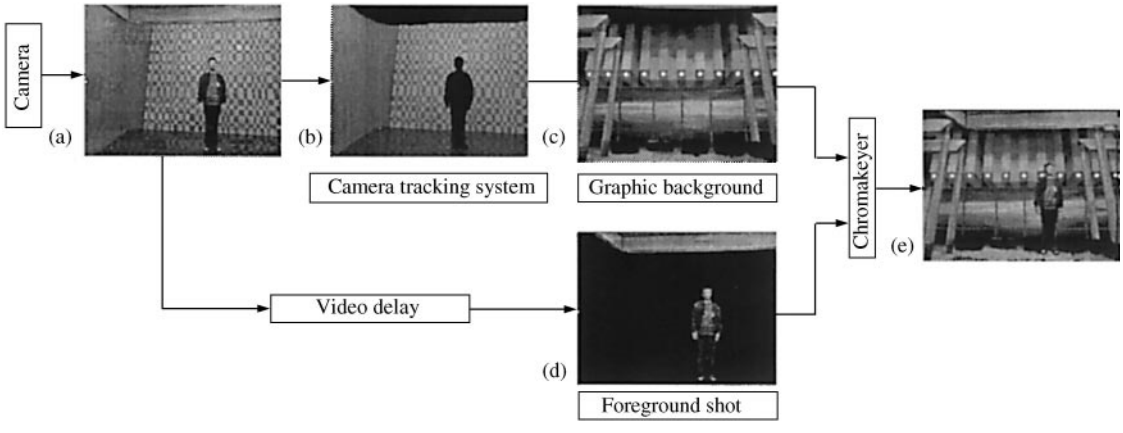


**Figure 1.** Total image flow: (a) Input image, (b) extracted pattern we designed, (c) graphics background, (d) real objects extracted from an input image, and (e) resulting composed image.

adopted is based on Tsai's method, in which camera parameters are calculated through nonlinear optimization [13]. Although exact camera parameters can be obtained by this method, it requires a large number of calculations due to the optimization and thus, it is not appropriate for a real-time implementation. To reduce the computational burden, the lens distortion parameter must be calculated independently so that the other parameters involved in the nonlinear optimization may be obtained with a linear method. Accordingly, we need a method for the calculation of the lens distortion. Camera calibration techniques considering the lens distortion have long been studied [13–22] utilized was the known motion of the camera [14,18] or the feature correspondences of a few images [17,19,20]. Recently, the direct image mosaic method has been applied instead of obtaining correspondences [22]. According to the spatial distribution of the features, calibration techniques may be classified into two categories: planar [13,16,21] and nonplanar [15,17,20]. Basically, the methods of Willson [16] and Batista [21] are grounded on the calibration technique of Tsai [13]. However, in order to compute the lens distortion parameters, they rely on the iterative optimization methods with respect to all the camera parameters, which are not appropriate for a real-time system. In this paper, we propose two practical methods for calculating lens distortion independently of other calibration parameters, which will give a linear calibration algorithm for our real-time implementation. One method uses a focal length to lens distortion look-up-table (or LUT) that can be constructed in the initialization process. The other method finds lens distortion in real time without any initialization process using the relationship between feature points in an image. The performance of our methods for computing lens distortion parameters is then evaluated through the comparison of our results with the results of Tsai's optimization method.

Figure 2 depicts the overall flow for generating virtual studio. The processes of the camera tracking system will be explained in the following sections. In the next section we will review Tsai's camera calibration model and explain why lens distortion has to be calculated independently of other parameters, and also describe the coordinate transformation between various coordinates involved for generating a graphics studio. The following sections explain the procedure for making the pattern by applying cross-ratio, give our real-time algorithm for automatically finding feature points in an image and identifying them, present our new methods for calculating lens distortion, explain temporal filtering and

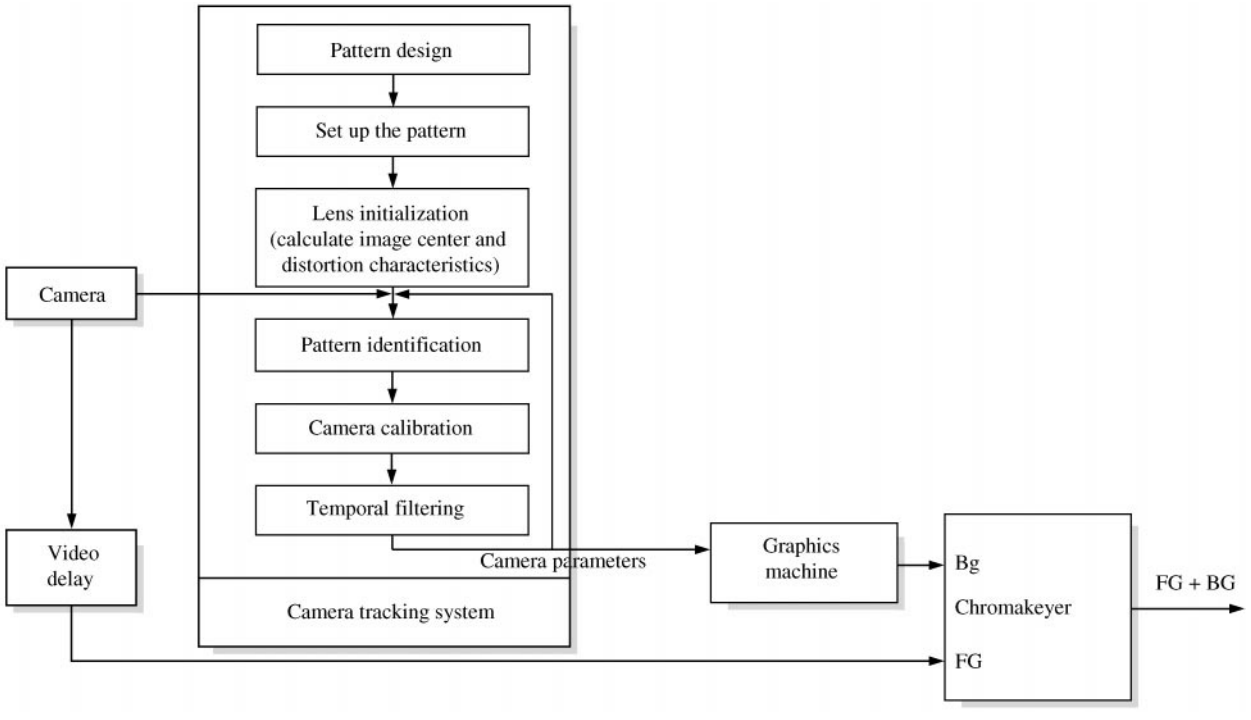


**Figure 2.** Total flowchart for generating virtual studio.

describe our system from the implementation point of view. Experimental results, including calculation of lens distortion, and conclusions are then presented.

## Camera Calibration Model

*Tsai's calibration model*

The calibration model of this paper is based on Tsai's model for a set of coplanar points [13]. Figure 3 illustrates Tsai's camera model.

The transformation from the 3D world coordinate $(x_w, y_w, z_w)$ to frame coordinate $(X_f, Y_f)$ consists of the following four steps.

**Step 1:** Rigid body transformation from the object world coordinate system $(x_w, y_w, z_w)$ to the camera 3D coordinate system (x, $y$, $z$):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T, \tag{1}$$

where $R$ is a $3 \times 3$ rotation matrix about the world coordinate axes $(x_w, y_w, z_w)$,

$$R = Rot(R_x)Rot(R_y)Rot(R_z) = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}, \tag{2}$$

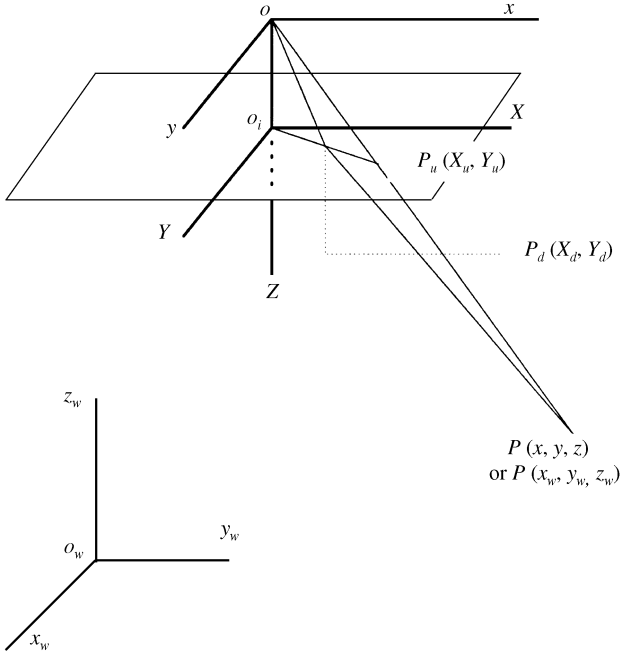


**Figure 3.** Tsai's camera model.

and $T$ is a translation vector,

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}. \tag{3}$$

**Step 2:** Projection of the 3D camera coordinate to the undistorted image coordinate $(X_u, Y_u)$:

$$X_u = f\frac{x}{z},$$

$$Y_u = f\frac{y}{z}, \tag{4}$$

where $f$ is the effective focal length.

**Step 3:** Calculating the distorted image coordinate $(X_d, Y_d)$ with a lens distortion coefficient, $\kappa_1$:

$$X_d(1 + \kappa_1 r^2) = X_u,$$

$$Y_d(1 + \kappa_1 r^2) = Y_u, \tag{5}$$

$$r^2 = X_d^2 + Y_d^2. \tag{6}$$

**Step 4:** Transformation from the distorted image coordinate $(X_d, Y_d)$ to the frame coordinate $(X_f, Y_f)$:

$$X_f = X_d \cdot s_x^{-1} + C_x,$$

$$Y_f = Y_d \cdot s_y^{-1} + C_y, \tag{7}$$

where the scale factor $s_x$, $s_y$, and image center $(C_x, C_y)$ are presumed to be known. In the later experiment we use the center of expansion [24] as a constant image center which can be found in the initialization process.

In **Step 3**, we considered only the first radial distortion coefficient, $\kappa_1$, that is the most significant factor and neglected the higher order terms.

Tsai proposed a two-stage calibration method. In the first stage, he calculated the extrinsic camera parameters $[T_x, T_y, R_x, R_y, R_z]$ using RAC (Radial Alignment Constraint) representing the relationship of $\overline{O_iP_u} // \overline{O_iP_d}$. That is, if we presume that the lens has only radial distortion, the direction of a distorted point is the same as the direction of an undistorted point. In the next stage, by minimizing the error function with $f$, $T_z$, and $\kappa_1$ as unknowns, using a standard optimization scheme such as steepest descent, the optimum $f$, $T_z$, and $\kappa_1$ are found. The error function for the optimization is defined as

$$error = \sum [(X_u - X_u')^2 + (Y_u - Y_u')^2], \tag{8}$$

where $(X_u,\ Y_u)$ is calculated from $(X_f,\ Y_f)$ using equations (5)–(7) and $(X_u^{'},\ Y_u^{'})$ is a projected point from the world coordinate to the image using the already calculated parameters $[T_x,\ T_y,\ R_x,\ R_y,\ R_z]$, and $f,\ T_z$ as variables. This opitmization process requires a large number of computations. If, in this stage, one can calculate $\kappa_1$ independently of other camera parameters, the undistorted image coordinate $(X_u,\ Y_u)$ can be calculated from equations (5)–(7). Then, the linear equation involving $f$ and $T_z$ can be derived from equations (1)–(4) as

$$[y_i - Y_{ui}]\begin{bmatrix} f \\ T_z \end{bmatrix} = w_i Y_{ui}, \tag{9}$$

where

$$y_i = r_4 x_{wi} + r_5 y_{wi} + r_6 z_{wi} + T_y, \tag{10}$$

$$w_i = r_7 x_{wi} + r_8 y_{wi} + r_9 z_{wi} \tag{11}$$

Since rotation matrix $R$, and translations $T_x$ and $T_y$ have all been determined by this point, $y_i$ and $w_i$ are fixed so that $f$ and $T_z$ can be lineary calculated from Eqn (9).

*Transformation between Tsai's camera model and the graphics coordinate system*

Figure 4 shows the coordinates involved in implementing virtual studio. Rotation matrix $R$, and translation matrix $T$, calculated from Tsai's camera model, are the orientation and position of the camera in the world coordinate $(x_w,\ y_w,\ z_w)$. In order to generate virtual studio, the graphics coordinate $(X_g,\ Y_g,\ Z_g)$ should be defined as illustrated in Figure 4, in which graphics objects and a virtual graphics camera will be placed. Now we have to convert the camera parameters obtained from Tsai's calibration, represented in the world coordinate, to those represented in the graphics coordinate in order for the virtual camera to generate a virtual background of graphic objects represented in the graphics coordinate. The transformation between the world coordinate and the camera coordiante is described as

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R_1 \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T_1, \tag{12}$$

where the rotation matrix $R_1$ and the translation $T_1$ represent the orientation and position of the camera in the world coordinate $(x_w,\ y_w,\ z_w)$. Let $(R_2,\ T_2)$ be a transformation between the world coordinate and the graphics coordinate, then the transformation between the two coordinates becomes

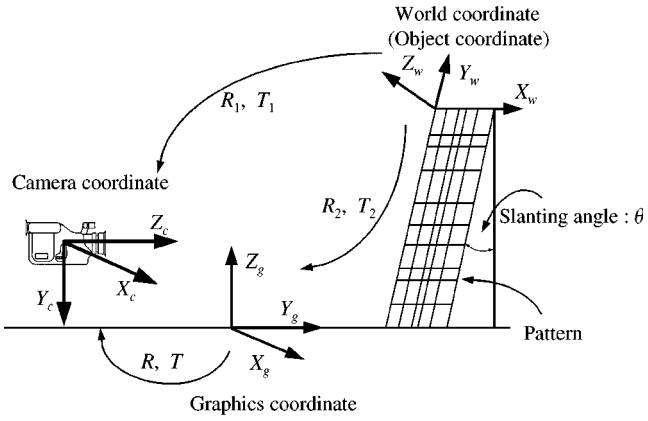


**Figure 4.** Coordinate transformation between the graphics coordinate and the camera coordinate.

$$\begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix} = R_2 \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T_2, \tag{13}$$

Once the origin of the graphics coordinate is fixed in the real studio, $R_2$ and $T_2$ can be given by measurements. Particularly, if the x-axes of the two coordinates $X_g$ and $x_w$ are made parallel, $R_2$ can simply be expressed as

$$R_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}, \tag{14}$$

where $\theta$ denotes a slant angle of the pattern against the wall. From the above two transformations, Eqns. (12) and (13), we can derive the transformation between the graphics coordinate $(X_g,\ Y_g,\ Z_g)$ and the camera coordinate $(X_c,\ Y_c,\ Z_c)$:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R_1 R_2^{-1} \begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix} - R_1 R_2^{-1} T_2 + T_1. \tag{15}$$

From this transformation we can find the orientation and position of the camera in the graphics coordinate. The graphics generator will use $R{=}R_1 R_2^{-1}$ as a reference orientation and $T{=}-R_1 R_2^{-1} T_2 {+} T_1$ as a reference position of the camera.

## Pattern Design

Camera calibration requires a special pattern consisting of feature points whose locations in the world

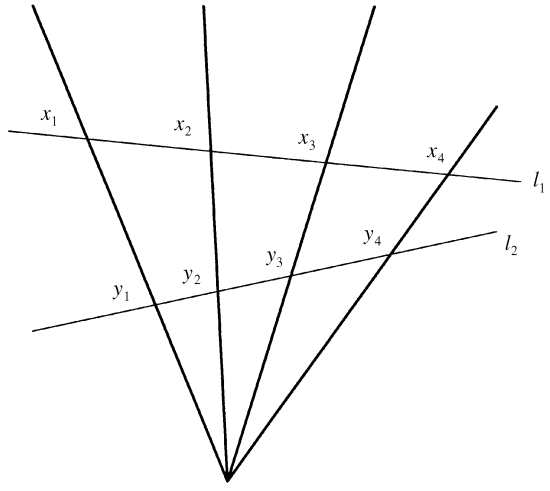


**Figure 5.** Cross-ratio of a pencil of lines $= \frac{(x_2-x_1)(x_4-x_3)}{(x_4-x_2)(x_3-x_1)} = \frac{(y_2-y_1)(y_4-y_3)}{(y_4-y_2)(y_3-y_1)}$.

coordinate system are known *a priori*. Since the camera moves around in a studio with changing zoom level, the pattern should be made so that feature points on it can be identified easily wherever it is seen from and even when just a portion of it is visible. In order for the feature points in the image to be identifiable, we apply the concept of planar projective geometric invariance—cross-ratio. In this section, we explain how cross-ratio is applied to the pattern design and in the next section we explain how the pattern can be identified in an image using this invariant.

Cross-ratio is defined for four points on a line as

$$C(x_1, x_2, x_3, x_4) = \frac{(x_2 - x_1)(x_4 - x_3)}{(x_4 - x_2)(x_3 - x_1)}. \qquad (16)$$

Figure 5 shows four lines meeting at a point (Pencil of Lines) and two lines intersecting them. In this geometry, the cross-ratio of the four lines is constant for the lines that intersect them, e.g. cross-ratio $C_{l1}$ for the intersections of line $l_1(x_1, x_2, x_3, x_4)$ and cross-ratio $C_{l2}$ for those of line $l_2$ $(y_1, y_2, y_3, y_4)$ are the same as

$$C_{l1} = C_{l2} = \frac{(x_2 - x_1)(x_4 - x_3)}{(x_4 - x_2)(x_3 - x_1)} = \frac{(y_2 - y_1)(y_4 - y_3)}{(y_4 - y_2)(y_3 - y_1)}. \qquad (17)$$

Note that when the four lines meet at infinity (ideal point) — (in this case the four lines become parallel), they also have the same property . When these lines are projected through a pin-hole camera with any orientation and zoom, the cross-ratios obtained from the image exhibit the same values as those in the real world. Figure 6 shows a part of the pattern designed. In order to extract the lines easily, the pattern is made in a grid shape. In the figure the grid is black and white, but in the real pattern, the white and black parts are actually light blue and dark blue, respectively, for chromakeying [23]. Figure 8(a) shows an image of our designed pattern used in the real implementation. Note that the vertical lines and horizontal lines have consecutive cross-ratios, that is to say, the vertical lines have cross-ratio $C_1$ of x-coordinates $(x_1, x_2, x_3, x_4)$ of the first four lines, and the second cross-ratio $C_2$ is defined for the next four lines $(x_2, x_3, x_4, x_5)$ and so on. Consequently, $N_V$ vertical lines give $(N_V-3)$ consecutive cross-ratios and $N_H$ horizontal lines give $(N_H-3)$ consecutive cross-ratios. Ideally, the cross-ratio of Eqn (16) has a value from 0 to 1. However, the extreme values are practically not feasible due to the actual distance between consecutive lines. For example, the cross ratio will become 1 when $x_4 \to \infty$ with the other values fixed, and it will become 0 when $x_4 \to x_3$ or $x_2 \to x_1$. Hence, we limited the range of cross-ratios in the practical implementation. From the experiments with our images, whose feature points had about a half-pixel accuracy, the standard deviation of the cross-ratios was found to be around 0.03. This means that we have at most nine independent cross-ratios or equivalently twelve lines, when the actual range of the cross-ratios is limited to 0.2 — 0.7. However, a small number of cross-ratios (lines) will confine the actual range of the camera work, and, therefore, we compare more than two consecutive cross-ratios, utilizing combinations of the cross-ratios in the identification

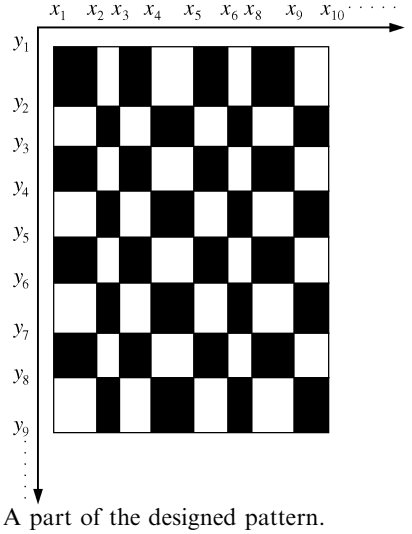


**Figure 6.** A part of the designed pattern.

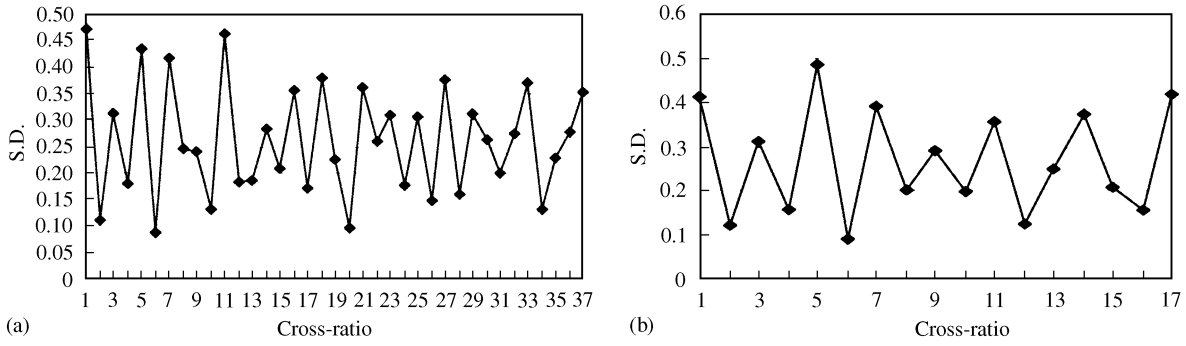


**Figure 7.** Cross-ratios for the designed pattern: (a) Vertical lines ($N_V$=40) and (b) Horizontal lines ($N_H$=20).

step. The two plots of Figure 7 show the values of the cross-ratio of the designed pattern in practice. The real pattern had 20 horizontal lines and 40 vertical lines and thus the plots have 37 and 17 consectutive cross-ratios, respectively.

## Pattern Identification

In order to calibrate a camera, first we must find the feature points in an image and identify them so as to find their correspondence to feature points in the object space (the world coordinate). In real-time applications, these feature points need to be extracted and identified automatically. In the initial identification process, we first extract and identify vertical and horizontal lines of the pattern by comparing their cross-ratios, and then we compute the intersections of the lines. Theoretically with this method, we can identify feature points in every frame automatically, but several situations cause problems in the real experiments.

(1) When the camera is maximally zoomed-out: the gaps of adjacent lines become too small to distinguish adjacent lines. In our experiment it became less than five pixels for the lens we used (Figure 8(a)).
(2) When the camera is maximally zoomed-in: in this case there are not enough lines in the image. To apply cross-ratio, at least five consecutive lines (in the case of comparing two consecutive cross-ratios) have to be visible in the image. But as the camera is zoomed-in, the number of lines becomes less than five (Figure 8(b)).
(3) When the pattern is occluded: line identification becomes harder when the lines are broken or hidden by an object, because calculation of cross-ratios requires consecutive lines of the pattern. If one line in the middle of the pattern is hidden by an object, we cannot calculate cross-ratios containing the line.

To overcome these problems, we first find intersections of the pattern in the initial identification process and then track them thereafter.

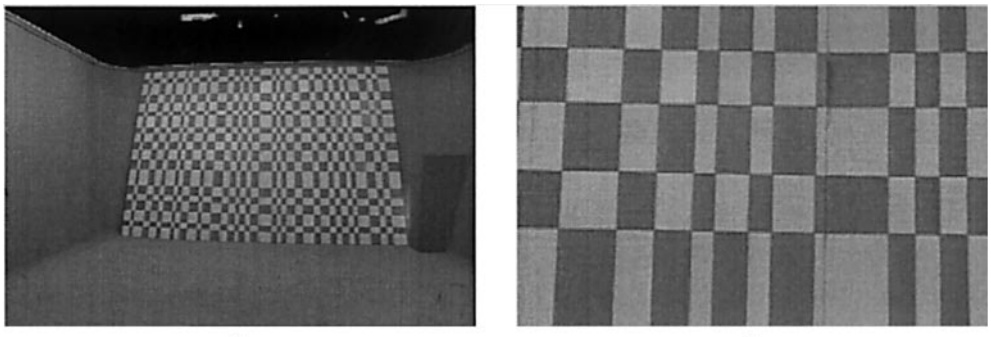


**Figure 8.** Situations that make identification process difficult: (a) When the camera is maximally zoomed-out and (b) When the camera is maximally zoomed-in.

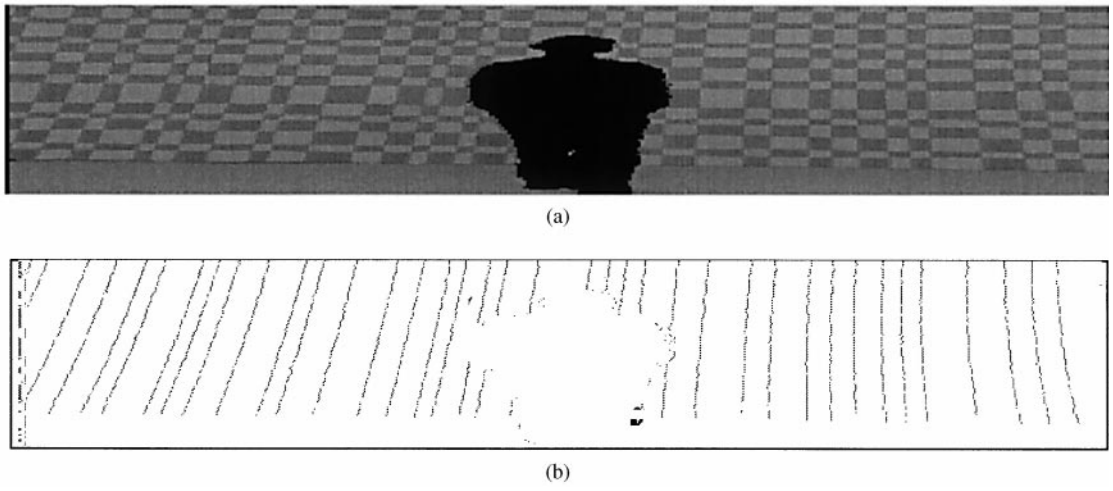


**Figure 9.** (a) An image ¼-subsampled in the *y*-direction. (b) Positions of the local maxima of the *x*-directional convolution of (a).

*Initial identification process*

In the initial identification process, we set the camera lens in the middle level of its complete zoom range to find the initial positions of the intersections in an image. Identification of the intersections is accomplished as follows:

*Extracting the pattern in an image*

When we made the pattern, it was painted with two colors, light blue and dark blue, so that we can extract the pattern from an image using the blue color as a key for chromakeying [23]. For extracting the blue color only, we first convert RGB to YUV and then pick up a chrominance region corresponding to the blue color.

*Gradient filtering*

To find the edge of the grid, a first-order Derivative of Gaussian (DoG) filter with a kernel of $h = [-1, -7, -15, 0, 15, 7, 1]$ is used. One might use an other type of kernel for the gradient operation, but in our case this filter showed a good performance. We apply the filter to every fourth line in an image to reduce the computation time. *x*-directional convolution of an image with the filter has local maxima at vertical edges, and *y*-directional convolution has local maxima at horizontal edges. Figure 9(a) shows an image 1/4-subsampled in the *y*-direction, and (b) shows positions of the local maxima of the *x*-directional convolution of (a). Detected local maxima in the $x$ and $y$ directions are then connected and fitted to lines, as can be seen in Figure 10(b).

*Line fitting*

For an ideal pin-hole lens having no distortion, a line in the pattern projects to a line in an image. However, if a lens has a distortion the line will not look like a line, but instead a curve. Hence, the cross-ratio will change from image to image. Note that the cross-ratio is not preserved for the frame coordinate $(X_f, Y_f)$-positions of the feature points in an image — or for the distorted image coordinate $(X_d, Y_d)$. Cross-ratio is invariant only for the undistorted coordinate $(X_u, Y_u)$. Figure 11 explains this situation. In this figure $a$ and $b$ denote undistorted points, and $a'$ and $b'$ their distorted points. If there is no lens distortion, a line connecting them will appear as line $l_3$, but the line in an image would have a shape like $l_2$ and be fitted to line $l_1$. As shown in the figure, $l_1$ is located at a position different from $l_3$. This

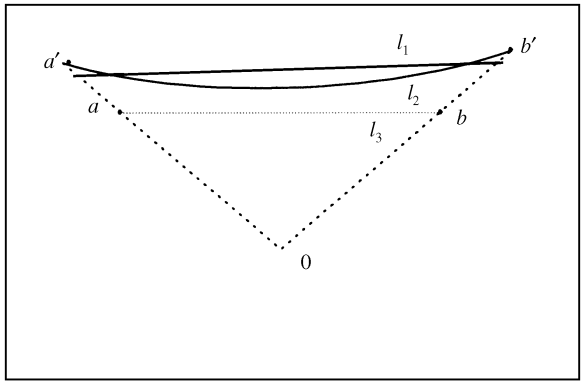


**Figure 10.** The effect of lens distortion: due to the distortion, line $l_3$, in the object space is shown as curve $l_2$, and fitted to line $l_1$, located at a position different from line $l_3$.

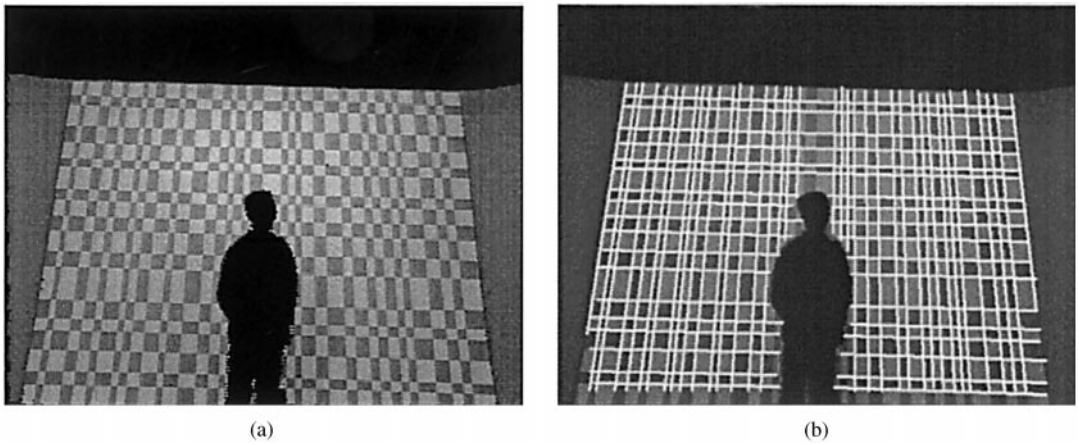


**Figure 11.** (a) Pattern extracted from an input image and (b) the vertical and horizontal lines extracted.

effect of lens distortion can be an obstacle in identifying the lines in the image. However, we can derive the line equation of the undistorted coordinate $(X_u, Y_u)$, i.e., line $l_3$ in Figure 11, when we have the lens distortion parameter $\kappa_1$. Let the line equation of undistorted points $(X_u, Y_u)$ be

$$Y_u = aX_u + b. \tag{18}$$

Then we can derive a distorted line equation (actually it is a curve) using Tsai's camera model. Using the relationship between $(X_u, Y_u)$ and $(X_d, Y_d)$, the line equation for the distorted points can be written as a quadratic form of $X_d$ and $Y_d$ involving $\kappa_1$:

$$\begin{aligned} Y_d &= aX_d + b(1+\kappa_1^2)^{-1} \\ &= aX_d + b(1+\kappa_1(X_d^2+Y_d^2))^{-1}, \end{aligned} \tag{19}$$

where the distorted point $(X_d, Y_d)$ can be calculated from the frame coordinate $(X_f, Y_f)$ using Eqn (7). After fitting the curve in the image using Eqn (19), we can calculate the line equation for the undistorted points of Eqn (18) by setting $\kappa_1$=0. By comparing cross-ratios of the undistorted lines with the known cross-ratios of the pattern, we can identify the curves in the image and compute the feature points by calculating intersections of the vertical lines and the horizontal lines for the distorted points of Eqn (19).

*Feature point tracking*

In the initial identification process, after the resulting camera parameters become stabilized for several continuous frames, the identification method is changed to tracking the intersection points. In implementing virtual studio, camera parameters are calculated at the rate of 30 frames/s so that the differences between the positions of two continuous frames is very small. Consequently, we can track the feature points by referring the previous positions of the points. In this tracking process, we first find intersections of the pattern in the neighborhood of the previous positions using the intersection-filter $H$ that we designed:

$$H = \begin{bmatrix} -1 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 \end{bmatrix}. \tag{20}$$

The result of the 2D convolution of the intersection-filter with an image shows the local maxima or minima at intersections of the pattern. Figure 12 shows the result obtained for a part of an image.

When the camera is maximally zoomed-out, the gap between two neighboring intersections becomes so close that the intersections are likely to be misidentified. To reduce the possibility of misidentification we separate the intersections into two classes by the sign of the resulting value of filtering. When we convolve an image with the intersection-filter $H$, two neighboring intersections have a different sign from each other, intersection (1) in Figure 12, for example, has a local minimum (having a minus sign) and (2) has a local maximum (having a plus sign). By classifying the intersections into two classes we can prevent the feature points from being misidentified as its neighbor.

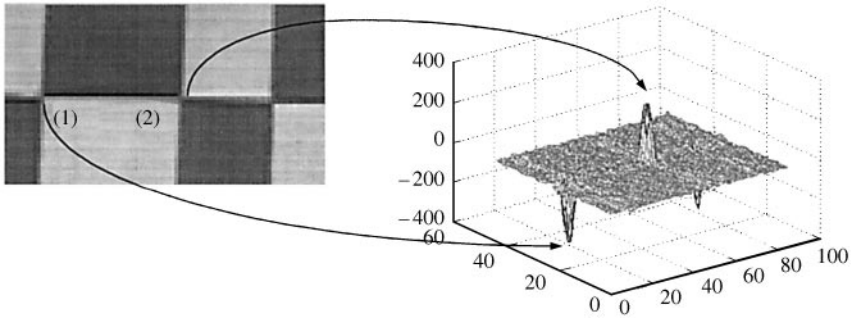


**Figure 12.** Result of the convolution of the intersection-filter, $H$.

To recognize new points that were not shown in the previous frame, we project points of the object space into an image using the camera parameters calculated from the previous frame, and then use the projected points as reference points for the current frame.

## Real-time Camera Parameter Extraction

Once the feature points at intersections are obtained, camera parameters can be calculated using Tsai's algorithm as explained previously. As discussed in that section, if lens distortion is obtained independently, calibration can be achieved using a linear method, which will speed up the calibration a great deal, enabling the real-time camera parameter calculation required in virtual studio. Therefore, in this section, after discussing the method of calculating the image center, we present new practical methods for calculating lens distortion independently.

*Determining image center by zooming*

The image center has to be found in advance because it is needed to calculate the lens distortion, as will be explained later. We will use the center of expansion as a constant image-center. This image-center is found in the initialization process. To find the center of expansion, the camera is operated from its maximum zoom-out end to maximum zoom-in end, storing the feature points which we find and identify in images. The center of expansion can be calculated as the common intersection of the line segments connecting corresponding feature points found in the images. For zoom lenses, the image-centers vary as the camera zooms because the zooming operation is executed by a composite combination of several lenses. However, when we examined the location of the image-center, its standard deviation was about 2 pixels; thus we ignored the effect of the image-center change.

*Calculating lens distortion coefficient*

In this section, we explain two practical methods for calculating lens distortion independently of other parameters. One method uses a look-up table (LUT), which is constructed in the initialization process, relating focal length to lens distortion. The other uses the invariance between feature points in an image without any initialization process. The invariance we adopted is the collinearity which represents the property where lines in the pattern should be shown as lines in an image if there is no lens distortion.

*LUT-based method*

Lens distortion is not an important factor for constant-parameter lenses, which have a constant distortion coefficient. But zoom lenses are zoomed by a complicated combination of several lenses so that the effective focal length, $f$, and distortion coefficient, $\kappa_1$, vary during zooming operations. In this section, to calculate lens distortion, we construct a look-up table (LUT) using the relationship between $f$ and $\kappa_1$ and then refer to this LUT during real-time operations.

In the initialization process, we operate the camera from its maximum zoom-out end to maximum zoom-in end, storing the feature points which we find and identify in images. To make the $f/\kappa_1$ LUT we use the nonlinear optimization method proposed by Tsai [13]. We can find the optimum $f/\kappa_1$ LUT because the nonlinear optimization process is executed off-line. When using the coplanar pattern with small depth variation, it turns out that focal length $f$ and $z$-translation $T_z$ cannot be separated exactly and reliably even with small noise. This can be easily understood

when considering that we cannot tell zooming from $z$-directional motion by watching input video even with human eyes. So here we use an alternative that uses $T_z/f$ as an index presuming a camera does not move in the $z$-direction. In the real-time process, camera parameters are calculated at the rate of 30 frames/s so that the change of camera parameters between two adjacent frames is very small. Consequently we can use the $T_z/f$ of the previous frame as an index for the current frame, and through iterative references we can refine them, i.e. we can refer $\kappa_1$ again using the index $T_z/f$ that was calculated from equation (9) using the $\kappa_1$ referred from the $T_z/f$ of the previous frame.

*Calculating lens distortion using collinearity*

The basic idea of finding lens distortion using collinearity is a kind of searching for $\kappa_1$ which maximally preserves collinearity. In other words, the value of $\kappa_1$ we want to find is the one that makes distorted feature points on a line the most collinear if we are correcting them into undistorted points. Collinearity represents a property where the line in the world coordinate is also shown as a line in the image. This property is not preserved when the lens has a distortion.

To find the lens distortion for a certain frame, we first take an initial lens distortion coefficient $\kappa_1$ and compensate frame coordinate $(X_f, Y_f)$ using the distortion coefficient in order to calculate the undistorted image coordinate $(X_u, Y_u)$. Then we find the optimum $\kappa_1$ that meets the collinearity.

Practically, we first identify feature points found in an image, which are located on a number of straight lines in the pattern used for camera calibration. Then for each line (here we assume lines are horizontal), we find three points (the center point and two end points) on the same line like the small rectangles in Figure 13. The dotted lines of Figure 13 represent lines of a pattern we found in the image. Due to the lens distortion these lines do not look straight. We break a line (a dotted line) into two lines (solid lines) and calculate the slope of each line. The error function is defined by the difference of the slopes between the two lines:

$$E(\kappa_1) = \frac{1}{N}\sum_{n=1}^{N}\left|\frac{Y_{un1} - Y_{un2}}{X_{un1} - X_{un2}} - \frac{Y_{un2} - Y_{un3}}{X_{un2} - X_{un3}}\right|^2, \qquad (21)$$

where $N$ is the number of straight lines used, and undistorted points $(X_{uni}, Y_{uni}; i = 1,2,3)$ are calculated from the distorted image points $(X_{dni}, Y_{dni})$ that are calculated from the feature point position in image $(X_{fni}, Y_{fni})$ using equations (5)–(7). For vertical lines, the error function we use is

$$E(\kappa_1) = \frac{1}{N}\sum_{n=1}^{N}\left|\frac{X_{un1} - X_{un2}}{Y_{un1} - Y_{un2}} - \frac{X_{un2} - X_{un3}}{Y_{un2} - Y_{un3}}\right|^2. \qquad (22)$$

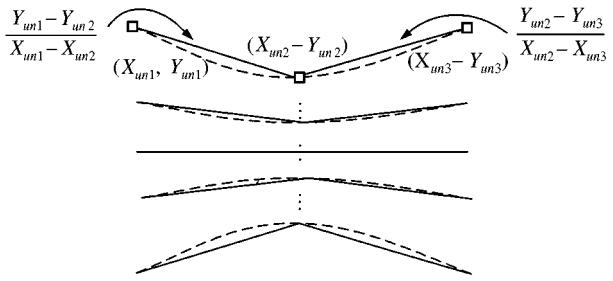


**Figure 13.** $N$ vertical lines (dotted lines) and two lines (solid lines) separating a line.

Then the lens distortion coefficient minimizing the error function is what we want to obtain, i.e.

$$\kappa_1 = \min_{\kappa_1} E(\kappa_1). \qquad (23)$$

Though this method calculates lens distortion through nonlinear optimization, the number of calculations is very small since it is an optimization in a 1D parameter space.

Furthermore, when we use the $\kappa_1$ of the previous frame as an initial value of this optimization, we can minimize the number of iterations. Once the lens distortion is calculated, we can execute camera calibration using linear methods.

## Temporal Filtering

Detected locations of feature points have noisy components even when the camera is stationary, thus resulting in noisy camera parameters. To reduce the effect of the noise, an averaging filter is applied to the parameters calculated, which is simple to implement and has the advantage of having constant delay. There is a tradeoff in determining the filter size $N$ since frame delay increases as it increases while reducing noise more. So it is better to choose the smallest $N$ that reduces noise up to the point where the resulting graphic studio with stationary camera does not give the feeling of trembling. Of course, achieving a moderate value of $N$ requires

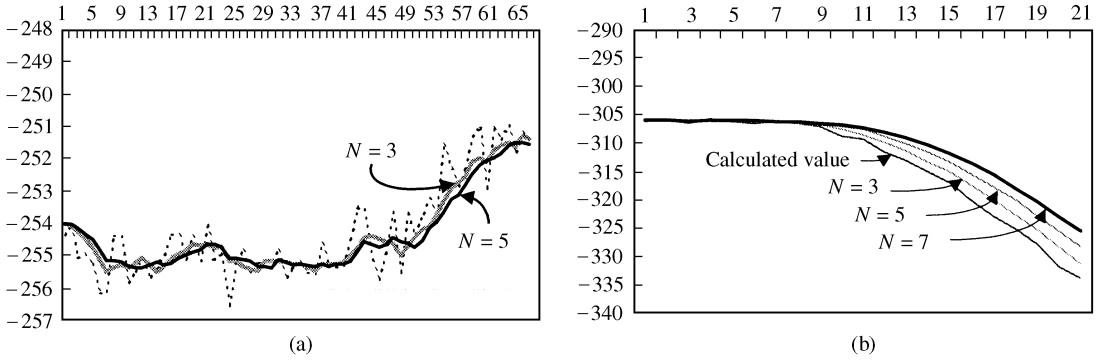


**Figure 14.** (a) Examples of noise reduction and (b) the time delay for filters with various lengths.

high accuracy of feature point locations. Through experiments, our system turned out to have a half-pixel accuracy and $N$ is determined to be five. Figure 14 shows an example of noise reduction and time delay for filters with various lengths.

## Real-time Implementation

In order to make a realistic virtual studio, graphics must be generated at a rate of 30 frames/s, requiring computation of camera parameters at the same rate. Coping with the tremendous computation required, we use a Pentium-200 PC together with three DSP boards (Genesis board of Matrox Co.), one main board with one TMS320C80 processor and one grab module, and two processor boards with two TMS320C80 processors in each. Consequently, a total of five processing nodes perform independent work in parallel, synchronized by the host PC. A TMS320C80 processor contains one master processor (MP), four parallel processors (PP) and a transfer controller (TC). The MP has a floating-point unit (FPU) and controls the behavior of the PP's. The TC is in charge of transferring data between external memory and the on-chip memory of each TMS320C80. Inside each TMS320C80 processor, in order to process an image, the MP divides the image into four sections and makes each PP process one of the sections. Each PP can work independently of the other PPs. Images grabbed by the grab module attached to the main board are broadcast through a grab channel and the five processing nodes can grab images into their memory simultaneously. Data transfer between nodes can be done through the VM Channel or PCI bus. The host PC also transfer data to nodes through the PCI bus.

Five processing nodes work in parallel, taking frames in turn, processing them, and sending the results to the host. Therefore, to process 30 frames/s, each node must complete its work within 165 ms (33 ms × 5) and the host PC within 33 ms. Processing nodes perform tasks from extracting pattern features to finding lines and intersections, while the PC takes charge in all the tasks thereafter. The border line between the tasks of the processing nodes and the PC is whether a task requires floating point operations or not because the FPU in the C80 processor is too slow to do those kinds of operations. Table 1 lists tasks performed by the processing nodes and PC and the processing time they expend.

Though each node ideally needs to process each frame in 165 ms, actually, however, in order not to lose grab-timing they should complete their tasks in less than 160 ms. Also, the PC, in order to exactly synchronize each

**Table 1.** Processing time in each processor

| Processor | Work | Processing time |
|---|---|---|
| DSP processor | Extracting pattern | 7.69 ms/frame |
| | Image subsampling | 5.11 ms/frame |
| | Gradient filtering | 22.03 ms/frame |
| | Finding lines and intersections | 75.00 ms/frame |
| | Transfer results to PC | 3.00 ms/frame |
| PC CPU | Line identification (in initial process) | 11.00 ms/frame |
| | Identification intersections (in real operation) | 12.00 ms/frame |
| | Camera calibration | 3.2 ms (300 points) |
| | Predicting next positions of intersections | 4.00 ms/frame |

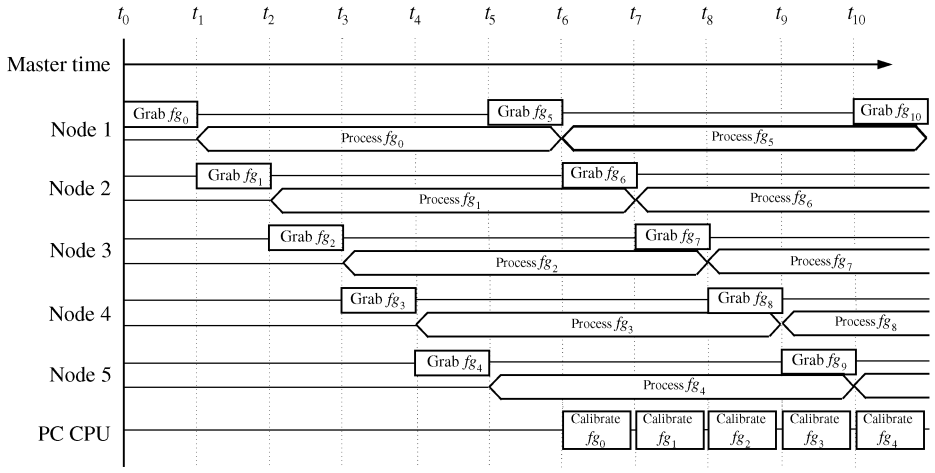


**Figure 15.** Parallel processing of five nodes with a PC.

processor, must complete its own tasks in less than 30 ms.

The timing diagram in Figure 15 shows how the five nodes and the PC process frames in parallel. Each time interval in this table is 33 ms (frame rate is 30 frames/s). As can be seen in the diagram, each processing node grabs and processes a frame in turn. Using the capability that the grabbing and processing can be executed independently in each node, we use a double-buffering technique, which allocates two different buffers and makes alternate use of them for grabbing and processing. Therefore, at time $t_5$, node 1 can grab the next frame $fg_5$ while still processing frame $fg_0$.

This parallel processing of frames inevitably causes time-delay. The five processors and PC working in parallel cause a time delay of six frames. Temporal filtering with a filter length of five adds two another frame delay so that the total delay becomes eight frames. In actual operation, to compensate for this time delay, real objects separated by chromakeying should be delayed by eight frames before being composed with graphics.

The YUV color coordinate is more efficient than the RGB color coordinate in extracting the blue pattern from an image. Color conversion from RGB to YUV takes about 40 ms by one DSP processor and thus requires two additional processors. These days, new digital cameras can produce YUV format output eliminating such a process. In Table 1 the color conversion time is omitted assuming that the YUV color format is supported.

## Experiments on Calculating Lens Distortion

Here we present experimental results on the proposed methods that calculate lens distortion. In this experiment, we use the center of expansion as a constant image-center, which can be found in the initialization process. The image-center should be calculated for every lens because the image-center is a specific feature of lenses. The variation of the image-center during zooming is less than two pixels in both $x$ and $y$ directions, which seems to have practically negligible effects on the later calculations.

Lens distortion coefficient $\kappa_1$ varies according to the zooming operation, and the focal length uniquely represents the zooming level so that there is a unique relationship between the focal length and the lens distortion coefficient. Using this basic observation we make a LUT of the focal length $f$ and the lens distortion. In real-time processes, we presume that the difference between the camera parameters of two continuous frames is not large. Therefore, we can find the lens distortion coefficient by referring to the LUT using the previous $f$ as an index of the current frame.

In the initialization process we use Tsai's nonlinear optimization method to calculate the focal length and lens distortion LUT. But in the case where a pattern with a set of coplanar points with small depth variation

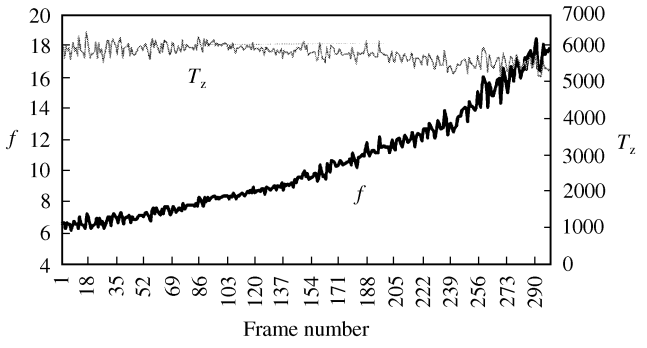


**Figure 16.** $f$ and $T_z$ calculated by Tsai's nonlinear optimization for a camera with monotonously zooming.

is used, it is very difficult to reliably separate focal length and $z$-directional translation $T_z$ because they are strongly coupled to each other. Actually, in that case, it is not easy to tell zooming from $z$-directional camera motion even by the human eye. Figure 16 shows this phenomenon. In this figure, $f$ and $T_z$ seem somewhat noisy, while their ratio, $T_z/f$, does not, as can be seen in Figure 17.

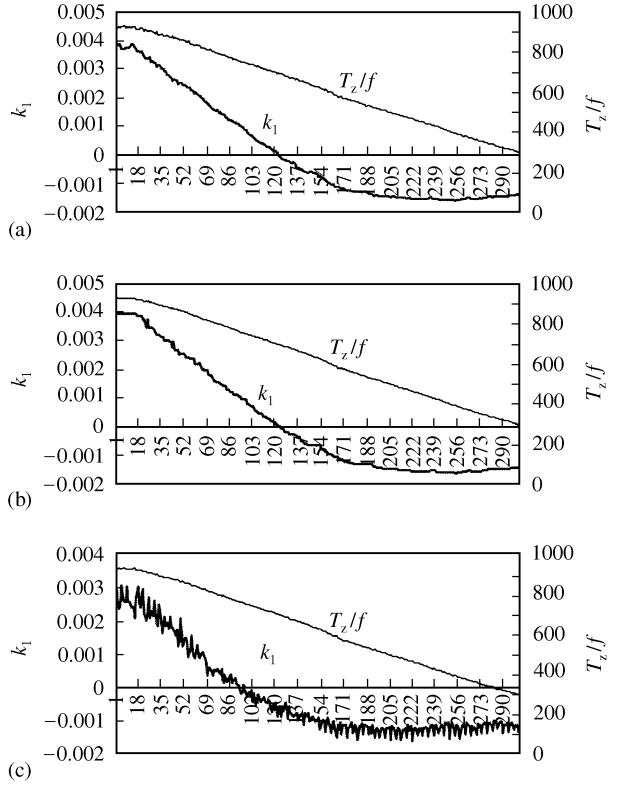


**Figure 17.** $T_z/f$ and $\kappa_1$ calculated with (a) the three-variable ($f$, $T_z$, $\kappa_1$) optimization method, (b) the $T_z/f - \kappa_1$ LUT-based method and (c) the collinearity method. Horizontal axis represents frame number.

Therefore, in this experiment, assuming a camera with fixed $T_z$ we make a LUT using the ratio of $T_z/f$, instead of $f$, as an index of the LUT. Figure 17(a) shows the $T_z/f$ and $\kappa_1$ calculated in the initialization process using Tsai's optimization method. We use 300 frames for 10 s (30 frames/s). As described above, we can presume that the difference in $T_z/f$ between two continuous frames is so small that we can find $\kappa_1$ of the current frame by looking up the $T_z/f - \kappa_1$ LUT using the previous $T_z/f$ as an index of the current frame. As described in a previous section, for more accurate lens distortion you can iterate this look-up. Figure 17(b) shows the results of $\kappa_1$ and $T_z/f$ calculated without an iterative look-up.

In this paper we define the undistorted projection error (UDPE) as a measure of the accuracy of the calibration. The UDPE is defined by the error between undistorted points, which are projected from the world coordinate using the camera parameters calculated and the points calculated from the frame coordinate. To obtain the UDPE, we first calculate the undistorted model point ($X_{um}$, $Y_{um}$) that is calculated from the world coordinate, and another undistorted frame point ($X_{uf}$, $Y_{uf}$) from the frame coordinate ($X_f$, $Y_f$) using equations (5)–(7), that is,

$$UDPE = \frac{1}{N}\sum_{n=1}^{N}((\Delta X_u)^2 + (\Delta Y_u)^2)^{1/2}, \qquad (24)$$

where

$$\Delta X_u = (X_{um} - X_{uf}) \cdot s_x^{-1},$$
$$\Delta Y_u = (Y_{um} - Y_{uf}) \cdot s_y^{-1}. \qquad (25)$$

Figures 18(a) and (b) show the UDPE's of the three variable ($f$, $T_z$, $\kappa_1$) optimization of Tsai's calibration model and the referring LUT-based method, respectively. Comparing the results, these two methods show nearly the same UDPE's. The UPDE increases as the frame number increases, because the number of feature points decreases as the camera zooms in.

Figure 17(c) shows the result of the experiment of the collinearity method for the continuous 300 frames. When the previous lens distortion coefficient $\kappa_1$ is used as an initial value of the optimization process, we can find lens distortion in less than 20 iterations. Using 20 lines to find the lens distortion, we can complete the optimization process in less than 1.2 ms with a Pentium 200 MHz, while the three-variable optimization of the

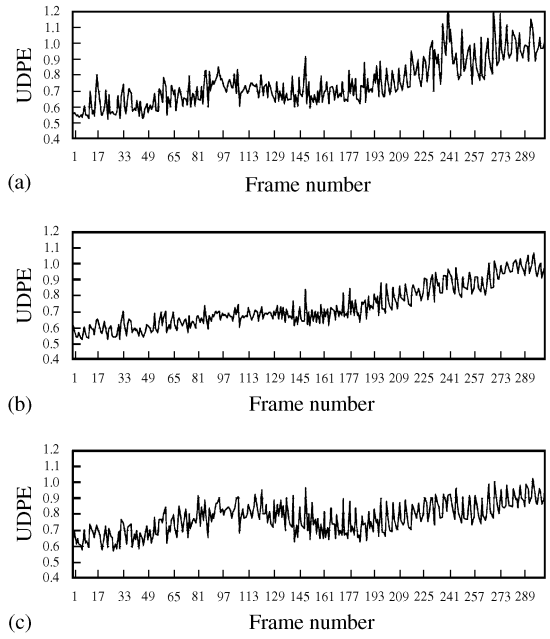


**Figure 18.** UDPE of (a) the three-variable ($f$, $T_z$, $\kappa_1$) optimization method, (b) the $T_z/f - \kappa_1$ LUT-based method and (c) the collinearity method.

Tsai's model takes longer than 30 ms. Therefore it can be favorably applied to the real-time process.

Figure 18(c) also shows the UDPE of the collinearity method. The feature points we found have an accuracy of about a half-pixel.

Judging from the resulting UDPE, the two methods, LUT method and collinearity method, are nearly comparable with Tsai's nonlinear optimization method. The higher UDPE can cause greater variance of camera parameters and consequently, the graphic background trembles more severely. In real operation, jittering of the graphic background (we used the LUT method in real operation) is not noticeable if we use a temporal filtering with a length of five.

## Conclusion

In this paper, we introduced a real-time camera parameter extraction system developed for virtual studio. In order to make a very realistic graphics studio generated from a graphics machine, camera parameters needed to be calculated very accurately in real-time, i.e. 30 frames/s. Toward this goal, we first designed a special pattern which makes it easy to identify feature points using the concept of cross-ratio. For the identification of feature points on the pattern, we applied two modes of operation. In the initialization mode, every feature point in an image was detected and identified first. Then in the tracking mode, feature points were tracked by searching in the neighborhood of every point of the previous frame. Newly appearing feature points were tracked by projecting the pattern using camera parameters calculated for the previous frame.

As for calculation of camera parameters, we applied a modified Tsai's algorithm in which all the parameters can be calculated by a linear method by obtaining the lens distortion independently of other camera parameters, while the original one involves a nonlinear optimization, which usually requires a large amount of computation. In this paper, we have proposed two methods for obtaining lens distortion separately. One was based on a look-up table (LUT) and the other on the collinearity of feature points. Experiments showed around a 30 factor improvement in computation time so that a rather inexpensive computer like a Pentium-200 PC without any DSP board can perform the computation.

All the algorithms were implementated with a PC and PC-based DSP boards. Five processing nodes in three DSP boards worked in parallel, synchronized by the host PC, computing camera parameters at the rate of 30 frames/s.

This system has been tested in connection with a real-time graphics machine (SGI Onyx-2). We could not find any noticeable trembling in the composite video generated from a stationary camera even when it was maximally zoomed-in, which means that the calculated camera parameters were sufficiently accurate.

## References

1. Grob, B. & Herndon, C.E. (1998) *Basic Television and Video Systems* (5th edition), Glencoe: McGraw Hill.
2. Gibbs, S. & Baudisch, P. (1996) Interaction in the Virtual Studio. *Computer Graphics*, 29–32.
3. Blond, L. *et al.* (1996) A Virtual Studio for Live Broadcasting: The Mona Lisa Project. *IEEE Multimedia*, pp. 18–29.
4. Hayashi, M. (1998) Image Compositing Based on Virtual Cameras. *IEEE Multimedia*, pp. 36–48.
5. Wojdala, A. (1998) Challenges of Virtual Set Technology. *IEEE Multimedia*, pp. 50–57.
6. CYBERSET, Orad, http://www.orad.co.il

7. 3DK: The Virtual Studio, GMD, http://viswiz.gmd.de/DML/vst/vst.html
8. ELSET: Accom, http://www.studio.sgi.com/Features/VirtualSets/accom.html
9. E & S Mindset, Evans & Sutherland, http://www.es.com./Products/DStudio/index.html
10. Gibbs, S. *et. al.* (1998) Virtual Studios: *The State of the Art*, *Eurographics'96 State of the Art Reports*.
11. Semple, J.G. & Kneebone, G.T. (1952) *Algebraic Projective Geometry*. Oxford University Press.
12. Stolfi, J. (1991) *Oriented Projective Geometry: A Framework for Geometric Computations*. Academic Press.
13. Tsai, R.Y. (1987) A Versatile Camera Calibration Technique for High Accuracy 3-D Maching Vision Metrology Using Off-the-shelf TV Cameras and Lenses. *IEEE Journal of Robotics & Automation*, **3**: pp. 323–344.
14. Basu, A. (1990) Active Calibration: Alternative Strategy and Analysis. *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 127–140.
15. Faugeras, O. (1993) *Three-Dimensional Computer Vision A Geometric Viewpoint*, MIT Press.
16. Willson, R.G. (1994) Modeling and Calibration of Automated Zoom Lenses, Ph.D. thesis, CMU-RI-TR-94-03, CMU.
17. Devernay, F. & Faugeras, O. (1995) Automatic Calibration and removal of distortion from scenes of structured environment. *Proc. SPIE conference*, *San Diego*, *CA*, *July 1995*.
18. Stein, G.P. (1995) Accurate Internal Camera Calibration using Rotation with Analysis of Sources of Error. *Proc. 5th Int'l Conf. Computer Vision (ICCV)*, *Boston*, *MA*, *June 1995*.
19. Stein, G.P. (1997) Lens Distortion Calibration Using Point Correspondences. *Proc. Int'l Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 602–608.
20. Zhang, Z. (1995) On the Eipiplar Geometry Between Two Images With Lens Distortion. *Proc. Int'l Conf. Pattern Recognition (ICPR)*, *Vienna*, Aug. 1996, pp. 407–411.
21. Batista, J. Araujo, H. & de Almeida, A.T. (1998) Iterative Multi-Step Explicit Camera Calibration. *Proc. IEEE Int. Conf. Computer Vision*, pp. 709–714.
22. Sawhney, H.S. & Kumar, R. (1999) True Multi-Image Alignment and Its Application to Mosaicing and Lens Distortion Correction. *IEEE Trans. Pattern Analysis and Machine Intelligence* **21**: 325–243.
23. Jack, K. (1996) *Video Demystified*, *2nd Edition*, *A Handbook for the Digital Engineer*, High text Interactive.
24. Willson, R.G. & Shafer, S.A. (1993) What is the Center of the Image. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 670–671.