The Visual Computer

Surround video: a multihead camera approach

Frank Nielsen

Sony Computer Science Laboratories, Tokyo, Japan
E-mail: Frank.Nielsen@acm.org

Published online: 3 February 2005

We describe algorithms for creating, storing and viewing high-resolution immersive surround videos. Given a set of unit cameras designed to be almost aligned at a common nodal point, we first present a versatile process for stitching seamlessly synchronized streams of videos into a single surround video corresponding to the video of the multihead camera. We devise a general registration process onto raymaps based on minimizing a tailored objective function. We review and introduce new raymaps with good sampling properties. We then give implementation details on the surround video viewer and present experimental results on both real-world acquired and computer-graphics rendered full surround videos. We conclude by mentioning potential applications and discuss ongoing related activities.

Video supplements:
http://www.csl.sony.co.jp/person/nielsen

Key words: Virtual reality – stitching – environment mapping

1 Introduction

Ever since ancient times, society has used images as a rich communication medium to transmit knowledge. Image depiction, the art of representing scenes, is of primal importance to painters. Renaissance painting is seen as a key period where many projection methods were discovered to allow artists to simulate or construct the appearance of three dimensional space on a two dimensional surface. Historically, painting wide views of landscapes and battlefields were already solicited by emperors and kings. Depicting a surround image, later called a panorama, was first popularized as an art form by Barker[1] in the mid-18th century. The word *panorama* stems from the Greek word combination of *pan*, meaning all, and *horama*, meaning sight. Although computers were already being used to stitch images into seamless pictures in the mid 1970s (see [13]), it was certainly Chen [5] who remarkably expanded their widespread use in the commercial Apple Quicktime VR player. Nowadays, spherical photo panoramas are commonplace on the Internet. In computer graphics, digital panorama techniques were precursors of the *image-based rendering* (IBR) field that advocates warping source images to create compelling virtual environments. IBR systems avoid processing detailed geometric models by synthesizing output images directly from real-world input images, and thus provides fast photorealistic walkthroughs [1]. However, IBR systems are often restricted to static scenes and have limited exploration abilities.

In this paper, we extend the still panorama workflow[2] to that of producing high quality spherical movies (say, on the order of a few million pixels at 60 frames per second). We describe our system for capturing, authoring, storing and viewing full (i.e., covering the complete[3] field of view) spherical videos. We particularly emphasize problems we faced in an industrial context and how our approach (and underlying algorithms) differs from previous techniques. We propose several novel processes to stitch, store and view immersive videos. Section 2 dis-

[1] Robert Barker, an Irish painter and fine arts teacher was granted a "panorama" patent in London for his painting method in 1787. He reportedly invented the process while being jailed for debt as he was struck by the light effect of his cell bars [2].

[2] We recommend the *panoramic vision* book [3] for a survey of major techniques related to this discipline.

[3] That is 4π steradian in terms of solid angle. Also called $360^\circ \times 180^\circ$ panoramas by reference to the latitude-longitude mapping, or sometimes abusively called $360^\circ \times 360^\circ$ mosaics [15].

Digital Object Identifier (DOI) 10.1007/s00371-004-0273-z

cusses the pros and cons of several camera systems to image panoramic videos and details our built-in-house multihead cameras. Section 3 describes a generic algorithmic framework for stitching multihead camera videos into a seamless spherical video. Section 4 gives detail on commonly used environment maps, that we call *raymaps*, as well as present new ones with good sampling properties. In Sect. 5, we devise fast algorithms to synthesize views from a virtual camera from surround videos (environment maps). Finally, Sect. 6 concludes the paper, mentions possible improvements, and discusses perspectives.

We believe that within a decade or so, surround videos will successfully enter the mass-market as high-fidelity interactive content will be broadcast to digital settop boxes and next-generation game consoles.

Fig. 1. *Top*: A hyperbolic mirror (courtesy of Accowle Corporation) mounted on top of a high-definition camera. *Bottom*: (Cropped) equirectangular map

2 Acquiring panoramic videos

Imaging panoramic videos can be done in various ways[4] depending on the number of block cameras and their attached lenses, and whether reflecting surfaces are used or not. A common approach is to use catadioptric systems (i.e., using mirrors) to acquire a large vertical field of view (VFOV) over a full 360-degree horizontal field of view (HFOV) (See Fig. 1). Although the monolithic recording system (only one image sensor is used so that an off-the-shelf recording pick-up system can be used) is convenient to record videos, we noticed several undesirable drawbacks such as blurred imagery, heterogeneous image densities, and most importantly partial view acquisition (that is, not full 4π steradian, but this can be overcome by using two back-to-back devices). Theoretically, even a single lens may capture close to full 4π angular view, but for practical reasons, it is not used as image quality degrades significantly as the FOV grows.

Multihead camera[5] design is about combining a set of block cameras, lenses, and eventually mirrors so that the imaging units share, as closely as possible, the same nodal point. On one hand, using several sensor devices allows the use of per-camera exposure settings to capture well-balanced sceneries (like sunsets, live concerts with strobe flashes, etc.) and control the density of the imagery. But on the other hand, it complicates the recording process as we need to synchronize video streams and design a specific recording unit. Moreover, since it is desirable for cost reasons to minimize the number of camera units, often lenses with inherent large distortions are used as we seek to minimize the overlapping areas of their individual video streams. This implies in turn that the stitching process becomes more delicate than in the still imaging case, where pictures are obtained at *different times* from a *single digital camera* precisely mounted at its nodal point on a graduated pan head tripod. Because focal planes are located in front of their respective sensors, another drawback of full spherical multihead cameras (see Fig. 2) is *parallax*[6] phenomenon. Nevertheless, parallax can be ignored, or ideally hidden, during video shootings if objects are past a given threshold distance to the camera, called *parallax clearance* [19]. Using reflecting surfaces like mirrors come in handy as it allows one to simulate a unique virtual nodal point (e.g., the non-parallax camera design of Nalwa [14] patented by #5,793,527 and #5,745,305 US patents; see Fig. 2, right). Although *deghosting* techniques [20] based on dispar-

[4] Daniilidis maintains a Web page of main camera designs. See http://www.cis.upenn.edu/~kostas/omni.html.

[5] Camera *Dodeca*, patented by US patent #6,141,034, from Immersive Media Corp. is one of the first prototypes [8]. Swaminathan and Nayar [19] called them *polycameras* (1996).

[6] The apparent displacement, or difference of position, of an object, as seen from two different camera positions (points of view).

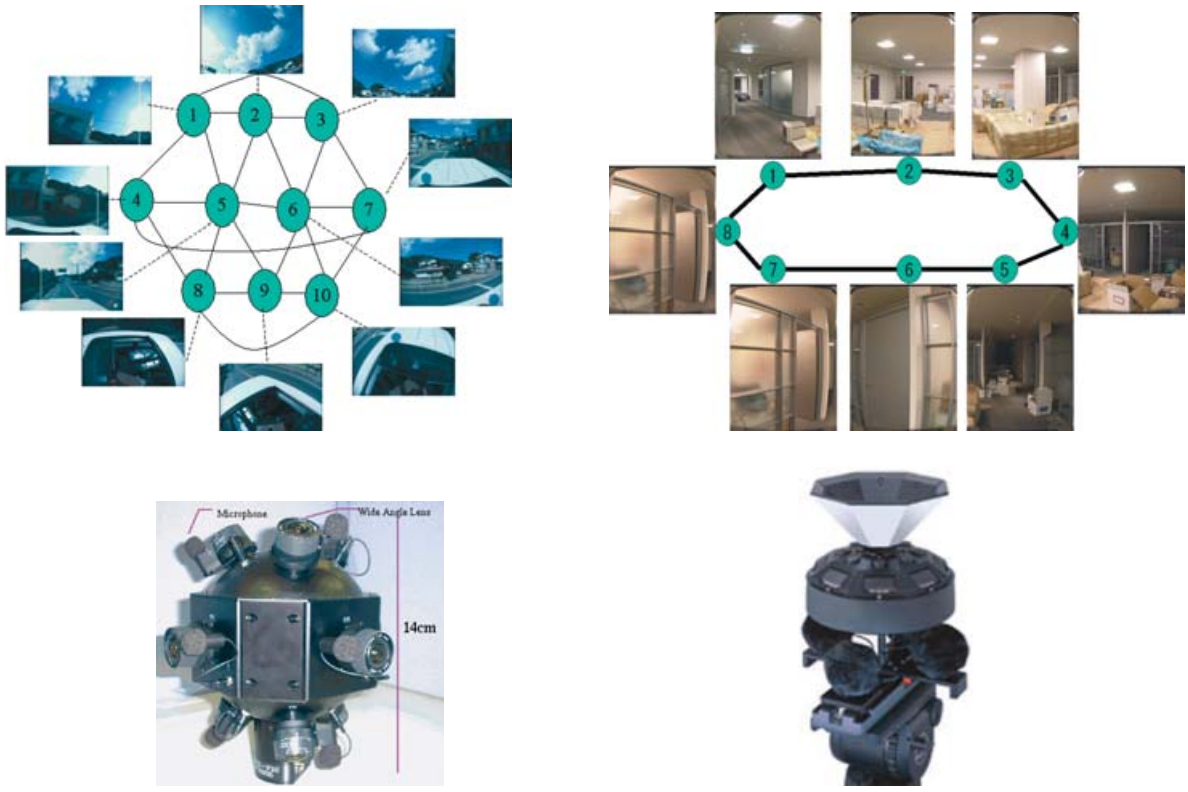


Fig. 2. *Bottom*: A 10 1-CCD multi-head camera and an 8 3-CCD multi-head camera using mirrors to align virtual nodal points. *Top*: FOV graph. Each block camera is associated with a node and two nodes are linked with an edge if and only if their respective fields of views overlap

ity maps have been proposed to compensate parallax effects, the results in practice are quite poor, especially for videos that need to enforce time consistencies. Therefore, we took special care at the camera design level to minimize parallax problems as much as possible. We opted for multihead camera designs that have good flexibility in controlling the resolution of the imagery and good individual exposure capabilities. Figure 2 shows two built-in-house multihead cameras. The left 10 1-CCD multihead camera fully covers the sphere but exhibits some amount of parallax while the right 8 3-CCD multihead camera delivers sharp high-quality imagery with almost undistinguishable parallax for most shootings. All block cameras are hardware genlocked and deliver RGB/SDI/NTSC signals (60 interlaced frames per second) that we digitize offline or directly record online onto hard disks at resolution 720×480@10-bit/channel.

3 Multihead camera

3.1 Preliminary notations

For a given camera model $\mathcal{C}$, let $\vec{r}(x, y)$ denotes the geometric ray associated to a pixel of coordinates (x, y). If the focal and iris settings are set up to remain unchanged during the video shooting, the geometric attributes of the ray does not change over time (that is, say, only its incoming luminance Y and chrominance UV vary). The relationship between $\boldsymbol{r}$ and (x, y) is often modeled in computer vision for a pinhole camera as follows:

$$(x, y) = RayToImage(\boldsymbol{r}, \mathcal{C}) = L(\bar{x}, \bar{y}),$$

where $(\bar{x}\ \bar{y}) \sim \boldsymbol{K}\boldsymbol{R}\boldsymbol{r}$, with $\boldsymbol{K}$ being a 3×3 matrix representing the camera intrinsic parameters of $\mathcal{C}$ and $\boldsymbol{R}$ the rotation matrix encoding for the extrinsic parameters: roll, pitch, and yaw of $\mathcal{C}$. We use homo-

geneous coordinates so that the operator $\sim$ means $(x\ y) = (x'/w\ y'/w) \sim \begin{pmatrix} x' & y' & w \end{pmatrix}$. $\boldsymbol{K}$ is treated as the following 3×3 matrix:

$$\boldsymbol{K} = \begin{pmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{pmatrix},$$

where (p_x, p_y) is the camera principal point (usually the image center), and f_x and $f_y = a_y f_x$ are the x-focal and y-focal lengths, respectively (in pixel units; obtained, respectively, from the horizontal and vertical fields of view) and s is the skewness parameter (here, without loss of generality, we assume it to be 0). Parameter a_y represents the image aspect ratio. $L(\cdot)$ is a bijective mapping function used to go from the ideal pinhole to the original (after applying lens distortions) image. In other words, we work in the $\bar{x}$ space of ideal pinhole images and we look up corresponding x pixels in the sensor image via a lens distortion function $L(\cdot)$. A ray $\boldsymbol{r}$ can be coded in various ways depending upon its processing context. Conventional representations include an anchor point O and either (1) a unit vector $\boldsymbol{v}$, (2) two spherical angles (θ, ϕ) coding for the direction $\boldsymbol{r} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\phi \\ \cos\theta\cos\phi \end{pmatrix}$, or (3) a unit quaternion $\begin{pmatrix} q_1 & q_2 & q_3 & q_4 \end{pmatrix}$. Quaternions are well-suited for numerical optimization [7] while the user can easily understand and edit interactively Euler angles (i.e., roll, pitch, and yaw angular attributes) using a GUI.

We next introduce the notion of what we define as *raymap* (popularly known as *environment map* in computer graphics). Informally, raymaps store the color attributes of the surrounding environment given a fixed viewpoint. More precisely, a raymap (environment map) is defined by a mapping function $m(\cdot, \cdot)$ that associates to each pixel coordinate (x, y) (indexing for a ray) of a picture its spherical coordinates $(\theta, \phi) = m(x, y)$ (examples are given in Sect. 4).

3.2 Versatile camera model

We introduce two abstract bijective mapping functions that create the basis of our stitching algorithm. Let $RayToImage(\theta, \phi, \mathcal{C})$ denote the image Cartesian coordinates (x, y) of a ray with spherical directions (θ, ϕ) anchored at the origin O for a camera model $\mathcal{C}$. Reciprocally, let $ImageToRay(x, y, \mathcal{C})$ return the spherical direction (θ, ϕ) for the image coordinates (x, y) of a camera model $\mathcal{C}$ (See Fig. 3). Below, we describe how to calculate those functions for Tsai's camera model [22], which only considers radial distortions. Using this generic framework, we directly map pixels into ray elements (also called *raxels* by Grossberg and Nayar [10]) and do not need to undistort and therefore resample the input images (which would introduce, in practice, a significant loss of resolution). Also, we can handle several camera models at the same time, making the stitching system flexible.[7] (Note that this unified framework allows us to deal with nonpinhole camera models, like fisheye cameras, having fields of view larger than 180 degrees.) Here, to solidify these ideas, we describe these mapping functions for Tsai's radial distortion lens camera model. For the sake of conciseness, we only consider the simplest radial distortion, lens center (c_x, c_y), which has only one radial distortion coefficient parameter κ. In practice, it is often required to take into account several coefficients $\kappa_1, \kappa_2, \ldots$ (at least 2 or 3 coefficients depending on the aberration of the lens [9]). We describe the operation pipeline below:

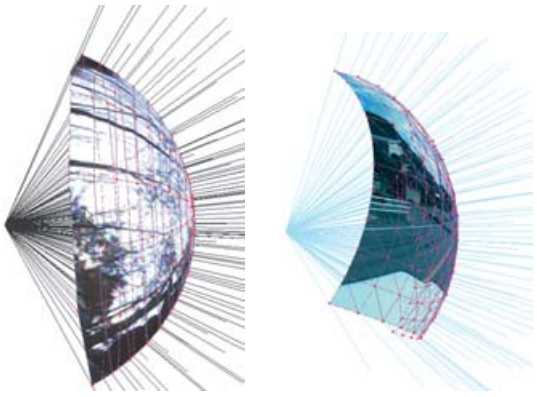


Fig. 3. The function $ImageToRay(\cdot)$ plotted for a fisheye lens (*left*) and for a wide-angle distortion lens (*right*)

$ImageToRay(x, y, \mathcal{C})$:

1. Compute ideal pinhole coordinates:
   $\bar{x} = x + (x - c_x)\kappa r^2$,
   $\bar{y} = y + (y - c_y)\kappa r^2$,
   where $r^2 = (x - c_x)^2 + (y - c_y)^2$.

[7] For example, we can stitch a configuration where we have horizontally four wide-angle lens cameras and vertically two fisheye lens cameras for the ground and sky parts.

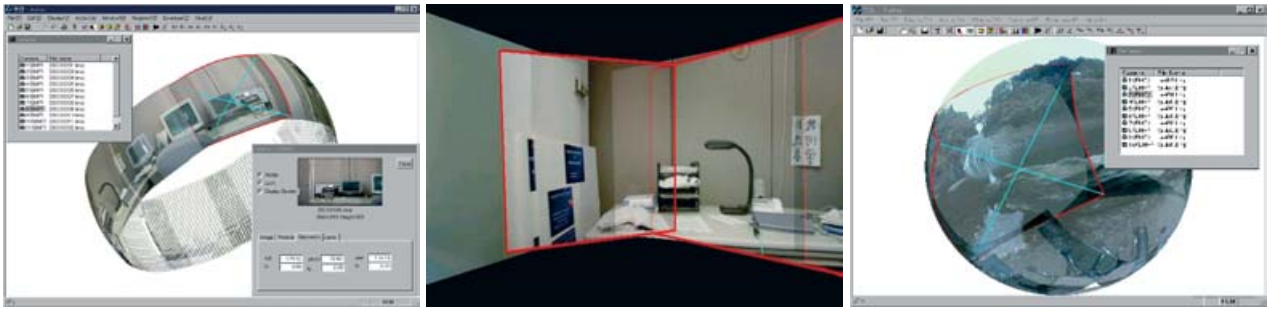


Fig. 4. Screen capture of the authoring and editing software. *Left*, *right*: project images displayed on the unit sphere. *Middle*: images displayed from the nodal viewpoint. Radiometric corrections are not displayed

2. Obtain a 3d vector:

$$P = \begin{pmatrix} P_X \\ P_Y \\ P_Z \end{pmatrix} = \boldsymbol{R}^{-1}\boldsymbol{K}^{-1}\begin{pmatrix} \bar{x} \\ \bar{y} \\ 1 \end{pmatrix}.$$

3. Obtain the spherical coordinates:

$$\theta = \arctan\frac{P_X}{P_Z},$$
$$\phi = \arctan\frac{P_Y}{\sqrt{{P_X}^2 + {P_Z}^2}}.$$

RayToImage$(\theta, \phi, \mathcal{C})$:

1. Obtain a 3d point on the unit sphere (vector direction):

$$P_{\theta,\phi} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\phi \\ \cos\theta\cos\phi \end{pmatrix}.$$

2. Get ideal Cartesian pinhole coordinates.

$$\begin{pmatrix} \bar{x} & \bar{y} \end{pmatrix} \simeq \begin{pmatrix} \frac{x}{w} & \frac{y}{w} \end{pmatrix} = \begin{pmatrix} x & y & w \end{pmatrix} = \boldsymbol{K}\boldsymbol{R}P_{\theta,\phi}.$$

3. Compute radial distortions by letting $\bar{r} = \sqrt{(\bar{x}-c_x)^2 + (\bar{y}-c_y)^2}$. Solve the polynomial system $\kappa r^3 + r - \bar{r} = 0$ and *choose* the root solution r' that is the closest to r.
4. Obtain Cartesian image coordinates:

$$x = \frac{\bar{x} + c_x\kappa r'^2}{1+\kappa r'^2},$$
$$y = \frac{\bar{y} + c_y\kappa r'^2}{1+\kappa r'^2}.$$

For large field of view lenses (bigger than 90 degrees), tangential distortions (also called decentering distortions resulting from the nonorthogonality of the lens and image sensor with respect to the optical axis) become quite noticeable. In that case, Conrady's model [6] should be used (see [9] for a robust estimation algorithm).

Using these abstract functions, we can easily add new camera models in our stitching system without changing the registration method. For fine precision stitching, an engineering strategy consists in tabulating these functions using more precise optical simulations delivered by optical design software (e.g., Zemax optical packages). This approach allows one to take into account numerous physical aspects of light propagations through the lens, like refractive, reflective, and diffractive surfaces, lens surface coatings, etc. Another interesting feature of this generic framework is that there is no more distinction between camera images and environment maps.[8] A camera image being a (partial) set of rays parameterized by intrinsic parameters while an environment map is an absolute set of rays (with no intrinsic parameters). Therefore, once an environment map is computed, we can further register high-resolution still images and output another environment map.

3.3 Initializing parameters

We first start by manually initializing the parameters of each camera using our graphical user interface (GUI) (that, is we explicitly give intrinsic and rotational parameters obtained from the multihead camera design; see Fig. 4 for several GUI snapshots). Then, by selecting manually corresponding feature points of overlapping image pairs (see the FOV graph of Fig. 2), we coarsely refine those parameters

[8] Recall that an environment map is an image representation of a discretization of all rays surrounding a given viewpoint.

using numerical optimization (e.g., gradient methods like steepest-descent or Newton–Raphson techniques). Our objective function is then to minimize the maximum difference of angles of corresponding feature points. Say, in image I_1, feature $f_1 = (x_1, y_1)$ corresponds to feature $f_2 = (x_2, y_2)$ in image I_2. Their difference angle is computed as follows:
Let $\boldsymbol{r_i} = ImageToRay(x_i, y_i, \mathcal{C}_i)$ for $i \in \{1, 2\}$. Then,

$$\alpha = \arccos \frac{\boldsymbol{r_1} \cdot \boldsymbol{r_2}}{\|\boldsymbol{r_1}\| \, \|\boldsymbol{r_2}\|},$$

where $\cdot$ denotes the dot product. In practice, we input for each edge of the overlapping image graph, around ten feature pairs (see Fig. 2). We then refine parameters using a fine (per-pixel) optimization.

3.4 Objective function

In this section, we explain the design of our objective function used to retrieve the block camera parameters by minimization. Intrinsic camera parameters are first recovered on an individual basis using image-based camera calibration techniques as depicted in Fig. 5. Then, registration is the process of refining these intrinsic parameters and finding the extrinsic rotational parameters (roll, pitch, and yaw angular attributes) of each block camera so that all camera rays emanating from a common[9] nodal point O match. For an n-head camera, we need to find n unit orthogonal 3×3 matrices $\boldsymbol{R}_i$ so that for all directions $\boldsymbol{r}$, the luminance/chromacies of the rotated rays match well. Our objective function differs from previous approaches [7, 11, 20, 24] in two ways:

1. First, the *atomic score* of the matching of ray $\boldsymbol{r}$ is defined according to the respective ray density functions $w_i(\cdot)$ (it is quite significant for wide-angle/fisheye lenses that we use in our multihead cameras; see Fig. 6). Denoted by $l_i(\boldsymbol{r})$ is the luminance attribute of a ray $\boldsymbol{r}$ captured from the ith camera image. After registration of the camera ray bundles, a ray $\boldsymbol{r}$ of the synthesized environment map is chosen as the sum of weighted attributes of the corresponding camera image rays.

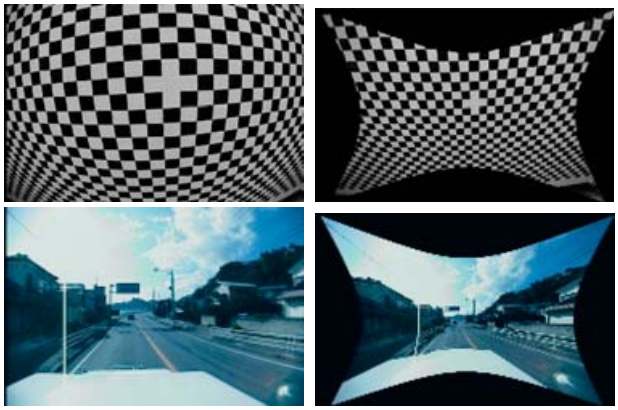


Fig. 5. Lens parameters of the 5th camera (from the 10-head camera) obtained by calibration: *center* = (360.8, 224.6), aspect ratio 1.083, radial coefficients $(2.29 \times 10^{-6}, 1.65 \times 10^{-11})$. *Left column* depicts the sensor images. *Right column* shows the undistorted pinhole images

Namely,

$$\widehat{l(\boldsymbol{r})} = \sum_{j=1}^{n} w_j(\boldsymbol{r}) l_j(\boldsymbol{r}), \text{ such that} \sum_{j=1}^{n} w_j(\boldsymbol{r}) = 1.$$

We define the scoring atom function as below:

$$s_{\boldsymbol{r}} = \sum_{i=1}^{n} w_i(\boldsymbol{r}) |l_i(\boldsymbol{r}) - \widehat{l(\boldsymbol{r})}|^2. \quad (1)$$

Loosely speaking, $w_i(\cdot)$ denotes the reliability of the ray $\boldsymbol{r}$ captured in the original camera images. In the case of Tsai's lens distortion model, $w_i(\cdot)$ is related to the density of the mapping $L_i(\cdot)$ from the ideal to original image (see Fig. 6). More precisely, we define $w_i(\boldsymbol{r})$ as the inverse of the solid angle spanned[10] by the pixel $(x_i(\boldsymbol{r}), y_i(\boldsymbol{r}))$ of camera image i. Notice that we need an interpolation scheme for computing $l_i(\boldsymbol{r})$ (see Sect. 5).
2. Second, since we store the result of the registration in a ray map $\mathcal{R}$ (environment map), we want to obtain the less "visually" defective raymap (when displayed later on in the viewer). Therefore, we ask for the minimization of:

$$f = \sum_{\boldsymbol{r}_{i,j} = m(i,j) | (i,j) \in \mathcal{R}} \Omega_{i,j} s_{\boldsymbol{r}_{i,j}},$$

[9] This is not true in the absolute because of camera misalignments and complex optical lens systems made of several units that create a caustic surface [10]. Also, because of the dual wave/ray nature of light, diffraction occurs at the boundary of the aperture and the light is spread all over the image. Moreover, refraction laws are defined given a refractive index, say the air, that may change, for example, with different temperature conditions, etc.

[10] Images are considered as four-connected meshes on the unit sphere. Solid angles can be approximated by $A \cos \alpha$, where A is the surface area spanned by a "pixel" and α is the incidence angle from O to the center of quadrilateral A.

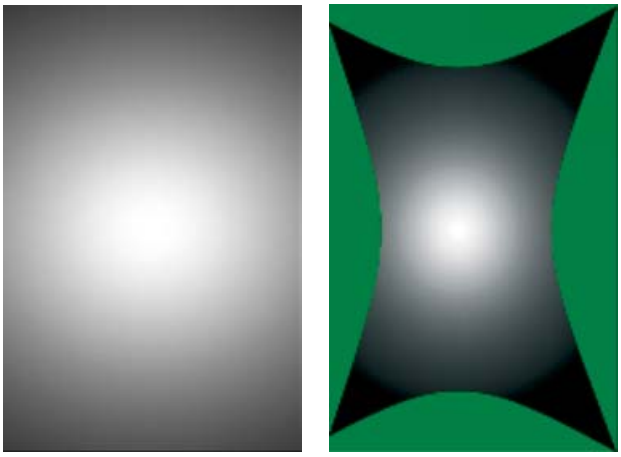


Fig. 6. Ray densities for an ideal pinhole camera (with 90° FOV) and for a wide-angle camera (FOV $\simeq 110°$). The brighter the sampling, the better the sampling

where $\Omega_{i,j}$ is the solid angle subtended by the pixel with coordinates (i, j) of the raymap $\mathcal{R}$. For example, using a traditional cubic map, it is known that $\frac{\max_{i,j}\Omega_{i,j}}{\min_{i,j}\Omega_{i,j}} = 3\sqrt{3}$. (For a dual paraboloid it is 4 and for an equirectangular map it tends to infinity.[11]) The bigger the solid angle, the more important it is to pay attention to finely optimizing this portion. Informally, this amount gives mismatch penalties according to the local resolution of the environment map. In practice, from an engineering point of view, we also tabulate the $w_i(\cdot)$ and Ω functions.

3.5 Local numerical optimization

Once all parameters have been properly initialized either by calibration, bundle adjustment methods on manually selected feature pairs, or by user input (using the 3d GUI depicted in Fig. 4), we perform per-pixel local numerical optimization to approach the global optimum. (Unfortunately, because of the high-dimensionality of the objective function, say on the order of 100 variables, we likely obtain a local optimum.) Loosely speaking, it is required to compute Jacobians and Hessians. Those are obtained from the derivatives of the objective function f according to the parameters. Numerous numerical recipes have been proposed so far. Starting from the early work of McMillan and Bishop [12], and Szeleski and Shum [20] (see also [24]), to the more recent work of Corg and Teller [7], which appropriately uses quaternions for handling rotations in the registration process. Our registration is not absolute and contrasts with the aforementioned approaches. Key differences with those methods are that: (1) each camera possibly has different parameters and, most importantly, (2) our atomic score is weighted according to both ray densities ($w_i(\cdot)$) and the density of the output raymap (function $\Omega(\cdot)$). This means that the parameter results of the registrations are slightly different according to the chosen raymaps (environment maps). We do not take into account radiometric correction at this level (for example, Turkowski and Xiong [24] considered linear intensity correction inside the registration process) because we found that this should be done offline using imaging tools. Camera engineers are familiar with robust camera color matching techniques to analyze and set the so-called camera shadings.

We locally optimize the objective function f using a standard Levenberg–Marquardt process since its use of second derivatives accelerates the convergence rate (i.e., a small number of iterations; the steepest-descent algorithm, although simple to implement, converges too slowly). The gradient $\boldsymbol{G}$ and Hessian $\boldsymbol{H}$ are computed by summing over all pixels of the raymap. Then, the optimization loop starts from an estimate $\boldsymbol{P}$ of the parameters (in a vector column) and update incrementally the solution as $\boldsymbol{P} \leftarrow \boldsymbol{P} + \Delta\boldsymbol{P}$, where $\Delta\boldsymbol{P} = -(\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\boldsymbol{G}$ ($\boldsymbol{I}$ denotes the identity matrix). We set λ (the stabilization parameter) initially at 10^{-4} and update it according to the variations of $\Delta\boldsymbol{P}$ (we keep the parameters $\boldsymbol{P}$ that have so far yielded the best result in memory). Since we need to compute $(\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}$, we encounter unstable numerical matrix inverts. Therefore, we use singular value decompositions[12] (SVD) or pseudo-inverse matrices. Once registered, we compute the raymap $\mathcal{R}$ by blending the pixels according to their weight function $w_i(\cdot)$. Notice that during the registration process, pixel intensity/color channel value is computed at noninteger positions. This requires one to implement interpolants. Figure 7 shows our raymaps obtained from several optimizations (different $\Omega(\cdot)$ functions).

[11] Oversampling at the poles.

[12] The invert of an SVD is easy to obtain by transpose and diagonal matrix invert operations.

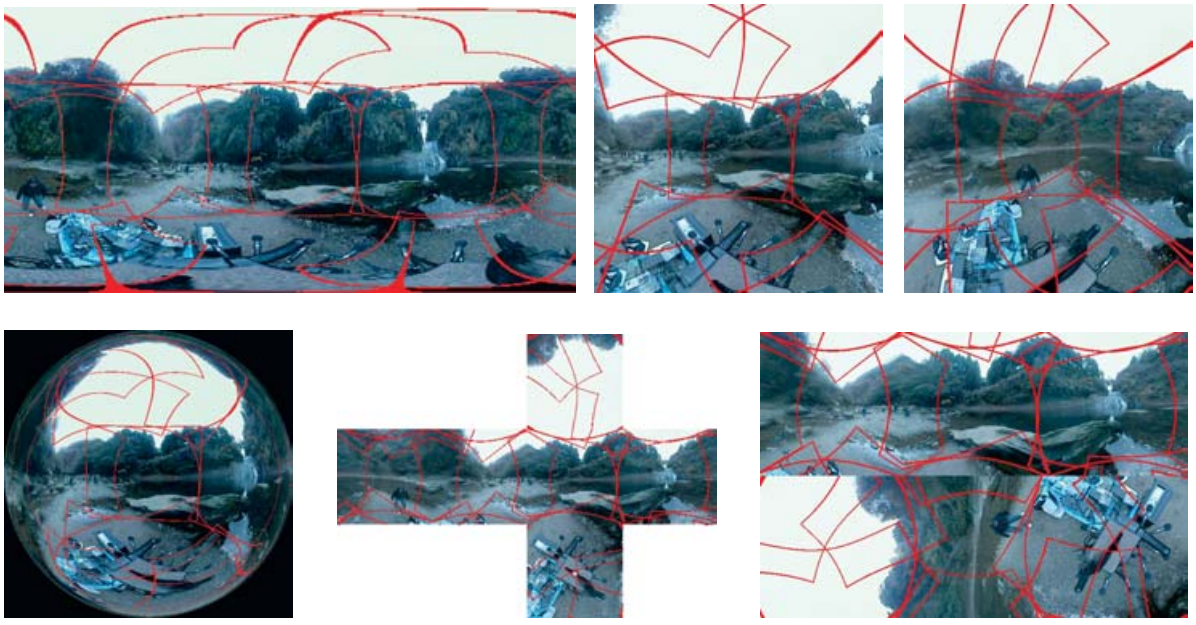


Fig. 7. Optimization results for various environment maps (from *top left* to *bottom right*): Equirectangular, front and back dual paraboloid, spherical, cubic, and rearranged cubic maps. *Red* borders are delimiting the respective fields of view of the camera units of the 10-head camera (see Fig. 2)

4 Environment maps

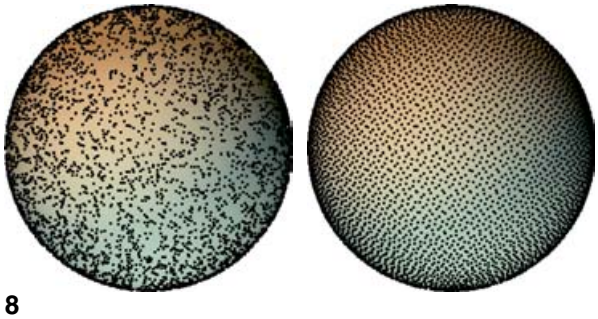
Raymaps (say, of size $W \times H$) store the color attributes of the surrounding environment in an image array. A raymap (environment map) is defined by a mapping function $m(\cdot, \cdot)$ that associates with each pixel coordinate (x, y) (indexing for a ray) a picture of its spherical coordinates $(\theta, \phi) = m(x, y)$. Conventional raymaps include equirectangular (also known as latitude-longitude), cubical, cylindrical, and dual paraboloidal maps (see Fig. 7). We present alternatives below that are particularly well suited to the case of immersive videos where bandwidth, and not so much image size, is the limiting factor. We detail sampling properties of those conventional environments as well as our proposed ones in Table 1. Raymaps can be interpreted as a special case of UV-texture maps, where the texture coordinates have spherical coordinate meanings. In cartography, unflattening the Earth has a long history [21] that has yielded to numerous map drawings with characteristics such as conformity, equal area, equidistance, etc. However, in our surround video setting, we do not need the map to be interpreted visually by humans but rather be efficiently processed by computers. Mappings are discrete (that is, stored as an image array) and possibly dynamic (that is, they change with time).

Table 1. Sampling properties of some environment maps. Discrepancy is defined as $\frac{\max_{i,j} \Omega_{i,j}}{\min_{i,j} \Omega_{i,j}}$

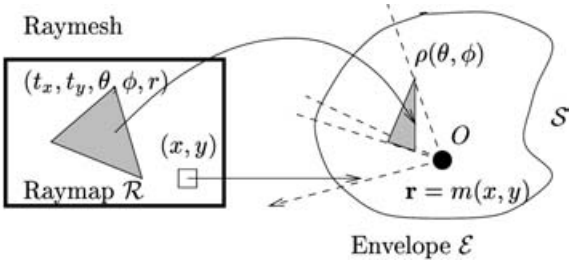
Raymap	Aspect	% Pixels	Discrepancy
Lat.-long.	2:1	100%	∞
Sinusoidal	2:1	$\frac{200}{\pi}\% \sim 63.7\%$	
Cube	3:2	100%	$3\sqrt{3}$
Paraboloid	2:1	$25\pi\% \sim 78.5\%$	4
Mirror ball	1:1	$25\pi\%$	∞
Angular	1:1	$25\pi\%$	
Hammersley	Any	100%	$\simeq 1$

Raymesh. The raymesh format is a 3d mesh, where each vertex has five components: (ρ, θ, ϕ) its spherical coordinates[13] and (t_x, t_y), the corre-

[13] For viewers standing at the nodal point, we set $\rho = 1$.



8



9

Fig. 8. *Left*: 10 000 points statistically uniformly distributed on the sphere using pseudo-random generators. *Right*: first 10 000 points of the incremental Hammersley sequence

Fig. 9. Raymesh: radial envelope $\mathcal{S}$ and its corresponding raymap $\mathcal{R}$

sponding texture coordinates, inside the raymap (see Fig. 9).

For example, to obtain a simple polygonal approximation of the sphere, we use an icosahedron or a surface refinement of it. Fuller's map (also called Dymaxion) is a special unfolding of an icosahedron onto a raymap and can be considered as a raymesh (see Fig. 10, right). Given the geometry of our sensors, we can define a raymesh that preserves as much as possible the original quality of the multihead imagery. This polygon encoding can easily include picture-in-picture mechanisms and can be used with legacy audio and video codec systems such as MPEG-2/4 (see also the 3D MPEG standardization document [18]).

Compressed spherical. One drawback of the equirectangular map is the nonuniformity of the sampling. Indeed, at the equator, we have for each pixel an increment of $\Delta\theta = \frac{2\pi}{W}$. But at height h, it becomes $\Delta\theta = \frac{2\pi\sqrt{1-(\frac{2h-H}{H})^2}}{W}$. We can partially overcome this over amount of information at the poles by sampling proportionally the latitude circles. Let $\phi_h = \pi\frac{h}{H}$, then the width at row h is $w(h) = W\sin(\phi_h)$, and we sample proportionally at row h by $\Delta\theta_h = \frac{2\pi}{w(h)}$ increments. Doing so, we *reorganize* the standard equirectangular map without loosing quality. We save a factor[14] of $1 - \frac{2}{\pi} \simeq 0.36$.

Also, we keep the advantage that pixels are stored in a *lexicographic order* so that ray interpolation and texture operations can be used without any indexing problem (See Fig. 10, middle). In cartography, this is known as the *sinusoidal equal area map*. However, a difference here is that we do not require the maps to be readable by humans, but rather seek for efficient encoding/decoding computer processes. For example, by aligning to the left the sinusoidal map, we halved the number of filled macroblocks cut in an MPEG-2/4 encoding.

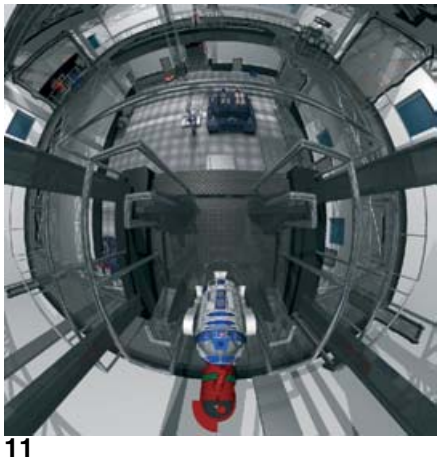
Hammersley sequence. This is a deterministic sequence of points distributed on the sphere with low discrepancy [23]. It has been used mainly in computer graphics for Monte Carlo algorithms. We use the Hammersley sequence with basis $p = 2$ (i.e., the so-called Van Der Corput sequence). One inconvenience is that we do not have a easy 2d lexicographically indexing order of the rays anymore (see Fig. 10, left). However, there are several possibilities for implementing those point sequence maps efficiently when computing the views of a virtual camera (see Sect. 5). Since the construction is incremental, we can increase the resolution progressively by streaming point-by-point (see Fig. 10, top left).

We implemented these formats as well as traditional ones used in cartography in a ray tracing software[15] to compare on a ground truth basis those

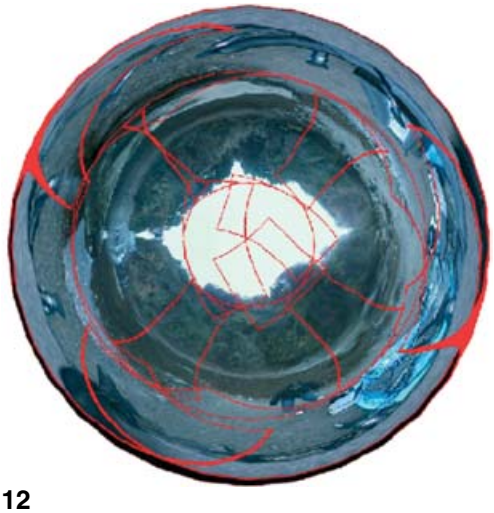
[14] In practice this means that we obtain better image quality from the video at a given image size for a fixed bit rate.

[15] We modified the Povray software package (`http://www.povray.org`) because of the wide availability of public domain script animations (e.g., `http://www.irtc.org`).

10



11



12

Fig. 10. (From *left* to *right*) Hammersley's, compressed spherical, and Fuller's maps
Fig. 11. Front paraboloidal map rendered by ray tracing
Fig. 12. A 360-degree fisheye view synthesized directly from a raymap

representations (that is, without taking into account stitching defects such as parallax or blending artifacts). Figure 11 shows a front paraboloidal map of a computer graphics animation. In practice, from an engineering perspective, we opted for the raymesh to be able to control the density into interesting areas (like people's faces). For still imagery, Hammersley's maps guarantee evenly distributed surround imagery but make the viewer algorithm more cumbersome since we need to initially triangulate points on a unit sphere. Our current video encoding is based on the legacy MPEG-2/4 framework. Since vector quantization and entropy coding are general purpose compression schemes, the maps can be encoded as is. (For example, using a cube as a raymesh is a simple and convenient solution since each of the six faces of the cube can be seen as an ideal pinhole camera with a field of view 90 degrees.)

5 Viewer

We describe in this section the implementation of our viewer. We consider two methods to synthesize novel views of a virtual camera: (1) the per-pixel method and (2) the per-triangle (texture mapping) method. On one hand, per-pixel drawing allows us to precisely control the rendering pipeline like the interpolation scheme, but rendering is fully handled by software and is therefore quite CPU demanding. On the other hand, texture mapping allows us to reduce the processor load by delegating triangle rasterization to the graphic processor unit (GPU). In this case, filtering operations are often limited (often, at best a linear interpolation) and fixed in the Large Scale Integration circuitry. We implemented both methods, described below, on Windows/Linux/PlayStation 2 platforms. Surround videos are encoded in MPEG-2/4 at 13 Mbps and burned onto DVDs. Some ad-

ditional XML tags containing information related to the videos are added (e.g., song titles, places, durations, suggested viewing angles, etc.) Users interact with surround videos either by using a joystick or by wearing a head-mounted display equipped with a gyroscope for retrieving gazing directions.

5.1 Per-texture method

Nowadays, widely available hardware capabilities allow us to render views of a virtual camera using 2d textured triangles rasterized on the GPU. For each triangle $t = \{p_1, p_2, p_3\}$ with p_i associated to its spherical coordinates (θ_i, ϕ_i) of the raymap, we compute the position of the corresponding triangle in the 2d XY-screen as $x_i \sim \boldsymbol{K}\boldsymbol{R}\boldsymbol{r}_{\theta_i,\phi_i}$. Using matrix formulations, we intersect a unit sphere with a line (instead of a ray), so that we remove the ambiguity (that is to eliminate the antipodal point) by checking whether $\boldsymbol{r}_{\theta_i,\phi_i} = \boldsymbol{r}'$ or not, where $\boldsymbol{r}' \sim (\boldsymbol{K}\boldsymbol{R})^{-1}x_i$.
We can choose to view the map not through an ideal pinhole camera but from any kind of camera (e.g., wide-angle or fish-eye lenses). We simply need to define a camera model $\mathcal{C}$ and the $RayToImage(\theta, \phi, \mathcal{C})$ function that, given spherical coordinates (θ, ϕ), returns the position, if it exists, on the XY-screen (see Sect. 3.2 for more details). For example, $RayToImage(\theta, \phi, \mathcal{C}_F)$ can be written as $(x, y) = r(\phi)(\cos\theta, \sin\theta)$, where $r(\phi) = f_1\phi + f_2\phi^2 + ...$ for a fisheye camera $\mathcal{C}_F$ using the equidistance projection model (see Fig. 12).
Using the GPU allows us to rasterize views from a virtual camera fast but lacks flexibility since interpolation and filtering operations are fixed in the hardware. Since the algorithm is *forward-mapping* (that is from the $\theta\phi$-texture coordinates to the XY-screen coordinates), we can split the raymap into several independent textures and send them one-by-one to the GPU (multipass rendering). This is specially useful for game consoles or set top boxes that have a small memory footprint.[16] Some examples of polygonal approximations of the sphere related to surround video quality are detailed in the 3D MPEG document [18].a

5.2 Per-pixel method

Per-pixel drawing executes all operations on the CPU and is therefore computationally heavier than its per-texture element counterpart. But we can freely choose the interpolation (for example, a sinc interpolant) and filtering algorithms. The rendering algorithm is *backward-mapping*, that is, from the XY-screen to the $\theta\phi$-texture map. We compute an image that we directly blit to the XY-screen. For each integer pixel coordinates x_i of the virtual camera image, we first compute the real-valued spherical coordinates of the corresponding ray as $\boldsymbol{X} = (\boldsymbol{K}\boldsymbol{R})^{-1}x_i$. We then transform the Cartesian coordinates of $\boldsymbol{X}$ to its spherical coordinates. Since the spherical coordinates may have noninteger values, we interpolate the color from neighbor rays (i.e., neighbor pixels in the environment map). For Hammersley raymaps that do not exhibit an implicit triangulation defining neighborhood, we initially build a triangulation of points on the unit sphere and call more sophisticated localization and interpolation procedures based on spherical barycentric coordinates [4].

6 Conclusion and perspectives

In this paper, we described algorithms for stitching, representing, and viewing high-quality surround videos obtained from built-in-house multihead cameras. We presented algorithms for stitching, representing, and viewing full spherical videos. Our immersive video system is currently in use in an industrial setting and opens up new content creation possibilities (e.g., real-time interactive theater play broadcasted on Internet). We believe that within a decade or so, cost-reduction will eventually make affordable the use of surround videos to both the consumer and professional markets.
Full surround video, as a primary imagery input, is also interesting for new computer vision techniques. Indeed, since the imagery has no intrinsic parameters, more robust or efficient algorithms are expected for core vision problems, such as structure-from-motion [16] or long video sequence tracking.
Moreover, interactive image-based rendering photorealistic walkthroughs can be obtained by massively acquiring high-resolution still panoramas from a moving panoramic head [1, 17] in a static environment. We expect this line of work to extend the *panorama* technologies to a new digital media: *interactive 3d picture*.

Acknowledgements. The author would like to acknowledge Sony Corporation for fruitful R&D cooperation.

[16] Sony PlayStation 2 has only 4 MB of embedded RAM.

References

1. Aliaga DG, Funkhouser T, Yanovsky D, Carlbom I (2003) Sea of images: a dense sampling approach for rendering large indoor environments. IEEE Comput Graph Appl 23(6):22–30
2. Australian Center for the Moving Image (2005) Magic Machines: A History of the Moving Image from Antiquity to 1900. http://www.acmi.net.au/AIC/MAGIC_MACHINES.html
3. Benosman R, Kang SB (eds) (2001) Panoramic vision: sensors, theory and applications. Springer, Berlin Heidelberg New York
4. Cabral B, Olano M, Nemec P (1999) Reflection space image based rendering. Proceedings of the 26th annual conference on computer graphics and interactive techniques, pp 165–170. DOI 10.1145/311535.311553
5. Chen SE (1995) QuickTime VR: an image-based approach to virtual environment navigation. Proceedings of the 26th annual conference on computer graphics and interactive techniques, pp 29–38. DOI 10.1145/218380.218395
6. Conrady A (1919) Decentering lens systems. Mon Not R Astron Soc 79:385–390
7. Coorg S, Teller S (2000) Spherical mosaics with quaternions and dense correlation. Int J Comput Vis 37(3):259–273. DOI 10.1023/A:1008184124789
8. Dodeca^{TM} (2005) camera system. http://www.immersivemedia.com/technology.php
9. El-Melegy MT, Farag AA (2003) Nonmetric lens distortion calibration: closed-form solutions, robust estimation and model selection. IEEE International Conference on Computer Vision, pp 554–559
10. Grossberg MD, Nayar SK (2001) A general imaging model and a method for finding its parameters. IEEE International Conference on Computer Vision, pp 108–115
11. Hsu S, Sawhney HS, Kumar R (2002) Automated mosaics via topology inference, IEEE Comput Graph Appl 22(2):44–54. DOI 10.1109/38.988746
12. McMillan L, Bishop G (1995) Plenoptic modeling: an image-based rendering system. Proceedings of the 22nd annual conference on computer graphics and interactive techniques, pp 39–46. DOI 10.1145/218380.218398
13. Milgram DL (1975) Computer methods for creating photomosaics. IEEE Trans Comput C-24:1113–1119
14. Nalwa VS (1996) A true omnidirectional viewer. Technical report, Bell Laboratories. http://www.fullview.com
15. Nayar SK, Karmarkar A (2000) 360×360 Mosaics. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 2388–2395
16. Neumann J, Fermuller C, Aloimonos Y (2003) Polydioptric camera design and 3D motion estimation. IEEE Comput Vis Pattern Recogn 2:294–301
17. Nielsen F (2003) Plenoptic path and its applications. IEEE International Conference on Image Processing 1:793–796
18. Smolic A, Kimata H (2004) AHG on 3DAV Coding. ISO/IEC JTC1/SC29/WG11, MPEG04/M10795, Redmont, WA, USA, http://www.chiariglione.org/mpeg/
19. Swaminathan R, Nayar SK (2000) Non-metric calibration of wide-angle lenses and polycameras. IEEE Trans Pattern Anal Mach Intell 22(10):1172–1178
20. Szeliski R, Shum HY (1997) Creating full view panoramic image mosaics and environment maps. Proceedings of the 24th annual conference on computer graphics and interactive techniques, pp 251–258. DOI 10.1145/258734.258861
21. Snyder JP, Tobler WR, Yang OH, Yang QH (2000) Map Projection Transformation: Principles and Applications. CRC Press
22. Tsai RY (1987) A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. IEEE J Robot Automat 3(4):323–344
23. Wong TT, Luk WS, Heng PA (1997) Sampling with Hammersley and Halton points. J Graph Tools 2(2):9–24
24. Xiong Y, Turkowski K (1998) Registration, calibration and blending in creating high quality panoramas. IEEE Workshop on Algorithms of Computer Vision, pp 69–74

FRANK NIELSEN received BSc and MSc degrees from Ecole Normale Superieure (ENS) of Lyon (France) in 1992 and 1994, respectively. He defended his PhD thesis on adaptive computational geometry prepared at INRIA Sophia-Antipolis (France) under the supervision of Professor J.-D. Boissonnat in 1996. As a civil servant at the University of Nice (France), he gave lectures at the engineering schools ESSI and ISIA (Ecole des Mines). In 1997, he served in the army as a scientific member of the computer science laboratory at Ecole Polytechnique. In 1998, he joined Sony Computer Science Laboratories, Inc., Tokyo (Japan), as a researcher. His current research interests include geometry, vision, graphics, learning, and optimization.