State of the Art of Free Viewpoint Video

Tim Lenertz

February 20, 2016

Contents

1 Introduction

Free viewpoint television (FTV) is an emerging trend in the development of visual media, which aims to allow the user to interactively change the viewpoint of a captured 3D scene, and view it as they would do naturally in the real world. It represents a next step in virtual reality technology by providing more realism and interactivity than stereoscopic video.

Super-multi view (SMV) light field displays render the FTV content from a dense range of horizontally displaced camera poses, producing a virtual parallax effect. This enables depth perception when viewing the display, and also allows users to walk in front of the display to change the viewpoint. FTV content may also be rendered using , such as *Oculus Rift*. Combined with head and possibly whole body motion tracking, this creates an immersive virtual reality (VR) experience.

While for virtual content traditional computer graphics rendering methods are used, producing FTV video content from a real scene requires a far greater amount of data acquisition and processing. Images are acquired using a number of cameras at different poses, and view synthesis techniques are used to render images from intermediary viewpoints. This process involves scene reconstruction, that is, extracting three-dimensional geometry information from the input camera images. Another issue is efficient encoding and compressing of the images and geometry. [1, 2, 3]

FTV works in a multi-step pipeline: [4]

Acquisition Views from a scene are acquired using multiple cameras. Possibly virtual augmented reality content, or auxiliary data such as depth measurements are added.

Reconstruction From the input data geometry information about the scene is estimated, which will be used to synthesize virtual views.

Transmission The aggregated data is encoded, compressed and possibly transmitted to a front-end. At the other end it is decoded.

Synthesis An image from a virtual viewpoint is rendered using the available data.

Display The content is presented to the end user. A user interface is set up such that the user can change their viewpoint.

FTV is being developed in the context of Moving Picture Experts Group (MPEG) exploration and standardization activities. A recent call for evidence concerns rendering on SMV displays, and scene reconstruction from a sparse set of input cameras. [2, 5] MPEG-FTV also maintains two reference software packages called DERS and VSRS for scene reconstruction and view synthesis.

In this report different state of the art and older methods of *acquisition*, *reconstruction* and *synthesis* are described in more detail. The algorithms employed in DERS and VSRS are also described.

2 Acquisition

This section describes some of the ways input data for FTV is acquired.

2.1 Camera

Camera parameters are generally separated into *extrinsic* and *intrinsic* parameters. The extrinsic parameters are the pose of the camera, that is, its position and orientation in three-dimensional space. Using three-dimensional homogeneous coordinates, this can be expressed as a 4×4 rigid transformation matrix.

The intrinsic parameters describe the way it maps incoming light to the recorded image, including image size, field of view, lens distortion, focus effects.

2.1.1 Pin-hole camera

The simplest camera model is the *pin-hole camera*. The aperture is a single point in space. One ray of light passes through it in each direction and hits the image space in one position, resulting in a perspective projection. No defocus blurring effects occur. Real (approximately) pin-hole cameras are impractical because of the low amount of incoming light resulting in high exposure time.

The projection of 3D scene positions to 2D pixel coordinates on the image frame can also be expressed using a 4×4 projection transformation matrix in homogeneous coordinates. The componentwise division in the conversion from homogeneous to cartesian coordinates accounts for perspective foreshortening. Parallel projections can also be expressed using a transformation matrix.

A planar image space limits the viewing angle to 180°. Imaging devices such 3D scanners may instead use a spherical or cylindrical image surface. Ray projections and back-projections can then no longer be modeled using simple matrix multiplications. With setups such as for instance a slit camera moving attached to a rotating arm, the aperture point may also be different across image pixels. [6]

2.1.2 Lens camera

Light rays emitted from a point source at distance S_1 on one side of the lens, converge at a point at distance S_2 on the other side, governed by the *thin lens formula*

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f} \tag{1}$$

where f is the *focal length* of the lens, which depends on its curvature. When $S_1 = f$ then S_2 diverges to infinity, meaning that parallel light rays entering one side converge at a point in the *focal plane* at distance f on the other side.

Thus an object a scene point at distance S_1 can be captured in with a lens camera by placing the image plane at the corresponding distance S_2. Unlike with the pin-hole camera, an entire beam of light rays emanating from that point is aggregated onto one image space pixel. So the camera needs to be *focussed* in function of the average distance of the object to capture. In real cameras this is done by mechanically moving the lens. In the human eye the lens shape is adjusted.

Because points on scene objects are never at a constant distance from the lens, focussing is always approximate. However, S_2 varies less quickly the further the object point S_1 is: In the most extreme setting, $S_1 \to \infty$ and $S_2 = f$, so a sharp image of far away objects is obtained with this single setting.

For nearer objects, a *defocus blur* effect occurs when S_1 does not match the actual distance of the point source. Then the rays converge either in front of or behind the image plane S_2, and on the image plane the beam of rays is splatted on a disk region. Because this occurs for all neighboring object points, the image is blurred and high-frequency information of the object surface texture is irrevocably lost.

The lens equation 1 is also a simplified model that does not take into account spherical aberration, coma, chromatic aberration and other lens distortion effects affecting the geometry and coloration of the image.

2.2 Epipolar geometry

When two pin-hole cameras are used on the same scene, the relationships between the two images are described by epipolar geometry.

An example is shown in figure 1. Two cameras with centers O_L and O_R are observing a scene point P. Its projections on the camera's image space are denoted p_L and p_R. The plane passing through the points P, O_L and O_R is called the *epipolar plane*. Its intersections with the two cameras' image planes are called the *epipolar lines*, on the cameras' image spaces. Now any scene point that lies on an epipolar plane, necessarily gets projected onto to a point on the epipolar lines of both cameras. This is true regardless of the camera's orientation, though part of the visible content may get clipped by the image bounds.

This fact is used to find *image correspondences*, that is, image points that correspond to the same scene point. Suppose the poses of both cameras are

known, and the goal is to find p_R given p_L, when the scene position P is unknown. The ray $O_L p_L$ which passes through P is constructed, and then the plane spanned by $O_L p_L$ and $O_L O_R$ is the epipolar plane. Now the epipolar line on the right-side camera is constructed, and it is known that p_L lies on that line. This is called the *epipolar constraint*. It reduces image correspondence search to one dimension. Using an image similarity metric p_L is estimated, and from this the three-dimensional position of P can be reconstructed by triangulation.

Stereo reconstruction algorithms operate using this basic principle, as will be elaborated in section 3.3. The camera distance $\|O_L O_R\|$ is called the *baseline*.

For this purpose it is useful to transform the images such that all epipolar lines are horizontal, and corresponding epipolar lines have the same vertical coordinates. This can be expressed using a linear two-dimensional image transformation and is called *rectification*. Each horizontal scanline on a rectified image corresponds to one epipolar plane.

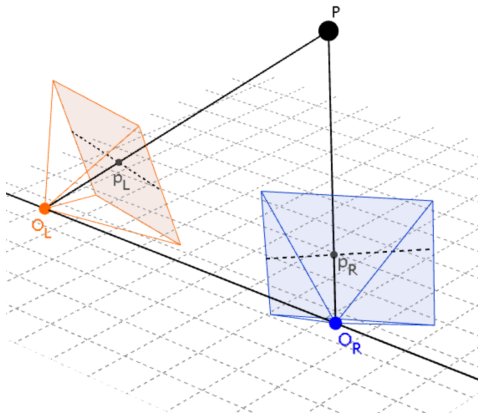

Figure 1: Epipolar geometry

2.2.1 Epipolar plane image

This can be generalized to the case where instead of two cameras, there is an array of cameras $\{O_1, O_2, \cdots, O_n\}$ placed at regular intervals between O_L and O_R. For a given epipolar plane, the epipolar plane image (EPI) is formed by stacking the epipolar lines taken from each camera on top of each other.

2.3 Light field

The goal of FTV is to allow the viewer to freely change the viewing position and orientation, within a given region of space. The view synthesis system should reproduce the image that a real camera would acquire when placed at that pose in the real scene, based on a set of input views collected from cameras with given poses.

A camera does not interact with the physical objects it observes: All information used to record the image exists in the space of the camera's aperture. Using ray optics, light is described as a dense set of rays traveling through space in straight lines, in all directions, and with different wavelengths. There is no interaction between intersecting rays. This *light field* can be modeled with the *plenoptic function*:

$$I = f(x, y, z, \theta_x, \theta_y, \lambda, t) \tag{2}$$

The plenoptic function f expresses the intensity of the ray traveling through the spatial point (x, y, z) with direction given by the angles (θ_x, θ_y), having wavelength λ, at time t.

This is a natural representation of light, also various elements perceived the human vision system correspond to first and second derivatives of f. For example bars, edges, flickers and parallax disparity. [7] It is a full characterization of the light that can be recorded in a given region R of physical space when $(x, y, z) \in R$. Depending on use additional dimensions such as polarization may be added. As cameras do not record full color spectrums, the dimension λ can be removed. Color can instead be represented using 3 or more plenoptic functions for the different color channels.

It is rarely possible to record this full five-dimensional representation of the light field. Cameras record small sections of the plenoptic function. Based on redundancies of the plenoptic functions, and models of the geometry and surface reflectance properties of the three-dimensional objects which generate the light field, *light field rendering* algorithms described in section 4 interpolate a sparse sampling of the light field to estimate the rays that a virtual camera would record.

2.3.1 Redundancies

Rays originating from a light source hit object surfaces. Then from every surface point, they are reflected in all directions. A camera observes the rays which pass through its aperture. This is the simplest description which disregards various other optical phenomena such as specular reflections, translucent objects, refraction, dispersion, or object with no solid surfaces such as fog, smoke or fire.

It is also generally assumed that object surfaces have diffuse lambertian reflectance, that is all rays

emitted from one surface point have the same intensity. Specular reflectance creates additional difficulties for reconstruction and synthesis algorithms.

Within this model the 5D plenoptic function contains several redundancies. Firstly, in an unobstructed region space R, rays remain completely unchanged as they travel in a straight line. For any object O whose convex hull does not intersect with R, all light rays reflected it it described fully using a 4D plenoptic function [8]

$$I = f(x, y, \theta_x, \theta_y) \tag{3}$$

which is also called *lumigraph* [9]. (x, y) will span a plane facing the object. The lumigraph of any plane parallel to to can be constructed knowing the ray angles (θ_x, θ_y). Hence the third spatial dimension z is redundant. This 4D representation of the light field is also referred to as *ray space*. [10, 11]

In addition to this, the lambertian reflectance allows to find image correspondences on camera views where the point are seen from different angles. It also results in a redundant dimension in the epipolar plane image (EPI) that can be exploited for data compression, as will by elaborated later.

Also, object textures are generally piecewise smooth, so that parallel rays have the similar intensities. Image compression is possible because high-frequency variations have lower amplitude.

2.3.2 Parameterization

To represent the section of interest of the lumigraph, most commonly the *two-plane parameterization* is used: Two parallel planes C and F are placed on either side of a scene object. C is called the *camera plane*, and F the *focal plane*. $\vec{c} = (u, v)$ and $\vec{f} = (s, t)$ are 2D cartesian coordinates of points on those planes. The parameterization

$$(u, v, s, t) \mapsto (x, y, \theta_x, \theta_y)$$

maps the pair of points $(\vec{c}, \vec{f})$ to the ray passing from $\vec{c}$ to $\vec{f}$. In [9], the term *full lumigraph* refers to a set of six two-plane pairs, necessary to capture an object from all sides.

With this parameterization, the a 2D section of the lumigraph obtained by fixing $(u, v) = (u_0, v_0)$ corresponds precisely to the rectified image of a pinhole camera placed at (u_0, v_0). Likewise, the 2D section obtained by fixing $(v, t) = (v_0, t_0)$ corresponds to the EPI of an array of cameras moving about the line $v = v_0$ in the camera plane, for the epipolar plane passing through the t_0's image scanline. An example (2D equivalent) is shown in figure 2. The highlighted section $f(u_3, \cdot)$ corresponds to a pin-hole camera image placed at u_3.

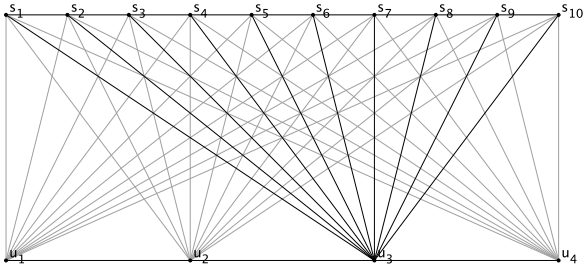

Figure 2: 2D view of two-plane parameterization

This allows for the lumigraph of a scene to be acquired using a 2D array of cameras. [8, 9, 12] In a simple setup, defocus blurring is disregarded, the approximation is made that all scene objects are placed exactly at the focal plane. This is accurate enough when capturing a scene where all objects are relatively far away from the cameras. An array of lens cameras is placed on the camera plane C in a regular lattice. They are focussed on and oriented towards the focal plane F, and the recorded images are rectified so that their view frustum far plane corresponds to F.

Many other lumigraph parameterizations are possible. [8, **?**] For example, by placing the C plane at infinity and replacing (u, v) by ray angles (σ_x, σ_y), 2D sections which fix (σ_x, σ_y) correspond to parallel projection images.

Lens cameras do not correspond to a simple section of the two-plane lumigraph parameterization. (x, y) is allowed to cover the aperture area, and projection to image pixels involves a base change by which the set of rays emanating from a point source at distance S_2 are aggregated to single pixels.

2.4 Plenoptic camera

Another way to directly capture lumigraphs is with a *plenoptic camera*. It creates a lumigraph which can be used for refocussing, depth estimation and synthesis of virtual views having a slightly displayed camera pose.

2.4.1 Principle

The Standard Plenoptic Camera (SPC), depicted in figure 3, consists of a main lens U and a micro-lens array (MLA) s of smaller lenses arranged in a 2D grid. [13] The image plane I is placed behind the MLA at its focal length f_s. So sets of parallel rays (aka collimated light beams) entering a micro-lens are projected onto one point of the image plane.

Each micro-lens projects onto a square section of the image plane, forming a micro-image.

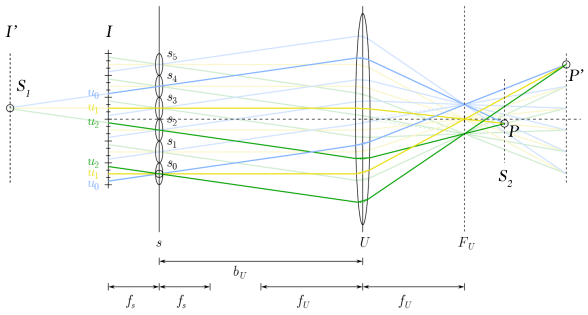

Figure 3: Standard Plenoptic Camera model [13]

The main lens is placed parallel to the MLA as a distance b_U greater than its focal length f_U. As described in section 2.1.2, light rays emanating from a point source P at distance $S_1 > f_U$ will converge behind the lens in a point at distance $S_2 > f_U$. Capturing a sharp image with a normal camera would require placing an image plane I' at S_1. Using the MLA located before S_1, the plenoptic camera is able to instead intercept the rays.

On the example in figure 3, the micro-images have a resolution of 3 pixels. (The figure shows a 2D analogy where images are one-dimensional) A *chief ray* is shown for each micro-image pixel. In reality, pixels form square regions on the image plane. Between lenses s and U, all rays whose orientation is within a given interval, *and* which intersect one of the micro-lenses s_i, are projected onto one pixel region u_j of that micro-image. Rays emanating from P, after passing through U, are not parallel, and so they pass through different micro-lenses at different angles. Each micro-lens s_i now makes all of the rays passing through it in a given angle converge onto a much smaller image pixel u_j.

As shown on the figure, scene points P and P' at different distances get projected onto different pixels in different micro-images. An artificial image for a given focus can be constructed by summing different groups of micro-image pixels in a certain pattern. [13]

2.4.2 Availability

Plenoptic cameras provide a cheaper and more integrated way to capture light fields. Unlike a full 2D camera array, the lens system also allows for refocussing. However, the virtual images synthesized from it are of much lower resolution.

In recent years the first consumer plenoptic cameras have been commercialized, for by *Lytro* and *Raytrix*. The first generation *Lytro* camera released in 2012 uses a 11 megapixel sensor, a MLA with 328 × 328, and produces an output image of 1 megapixel. The news *Lytro Illum* model released in 2014 has a 40 megapixel sensor and produces 4 megapixel output images.

Raytrix cameras are more targeted towards industrial use. They use an interleaved MLA where the micro-lenses have different focal lengths, in order to be able to extract more accurate depth information. [12]

2.5 Depth sensor

For the purpose of view synthesis, depth estimates will be made from the acquired images and light fields. It can be useful to include direct depth measurements.

Time of flight (TFL) laser scanners directly probe the physical environment using a *range finder*, consisting of a laser beam directed in different angles in the field of view. The distance of each point is measured using the time delay between the beam being sent out and its reflection from the object arriving back at a sensor attached to the scanner.

As it is impractical to generate the very short laser pulses needed for this, range finders may instead measure phase differences between the outgoing and incoming beams.

Using the direction of the laser beam and the measured distance, spherical coordinates of the scene point are obtained, which are then converted into cartesian coordinates and stored as a *point cloud*. It does not hold connectivity information about the object surfaces and can contain noise and outliers. Several techniques exist to post-process point cloud data, for example surface estimation (meshing), registration, filtering, and others.

Other methods to measure depth include triangulation laser scanners and structured light cameras. RGB-D cameras also exist, which directly capture a visible image and depth image at the same time, with the help of infrared emitters. [14]

3 Scene Reconstruction

Most view synthesis techniques require information about the 3D geometry of the scene as input, in addition to the camera images. Depth information can be measured directly using 3D scanning or other auxiliary data sources, as described in the previous chapter. However generally this is obtained using photogrammetric techniques which extract depth information from the images.

This depth information is always an estimation, based on hypothesis made about the physical scene: Firstly, it is assumed that the objects have well-defined boundaries. It should be possible to assign a meaningful depth value to each pixel in an image. Translucency, refraction, and non-lambertian surface reflectance also require more advanced methods to be modeled correctly. It can be shown that for stereoscopic methods, constant intensity[1] regions are the only inherently ambiguous case when attempting to extract depth information [15].

3.1 Representations

Different representations of geometry are used for the estimated scene geometry.

depth map An additional depth component d is added to each pixel in the images, which indicates the distance from the camera's center of vision to the object surface point depicted by that pixel. Depending on the method used, d can also be a projected depth value, for example the result of applying a homogeneous projection matrix to 3D coordinates. Depth maps are a 2.5D view of the geometry containing only the part of the scene visible from one camera. It is a *view-dependent* representation relative the camera view to which it is tied.

layered depth map If there are multiple cameras facing the same side of an object, but none facing other sides of it, then most of the geometry information missing on a single 2.5D view is due to occlusions. A layered depth map resolve this by allowing multiple depth values at each pixel, which indicate the distances of multiple surface points along that ray of sight. [16] It can be a good compromise between multiple redundant depth maps, and an incomplete global geometry representation.

voxel space A bounded range of 3D space is divided into an uniform three-dimensional lattice of cubic voxels. Each voxel can be occupied or empty, and possibly be attributed with a color and opacity value. The occupied voxels approximate the volume of the reconstructed object. Unlike depth maps, this is a view-independent global representation. Voxels are easy to process algorithmically, but require a lot of memory for an acceptable resolution. [17]

The voxel space can also be represented using an octree or other spatial tree, where fully occupied or free cells need not be further subdivided. [18]

point cloud The 3D shape is represented only using a set of points laying on object surfaces, given by three-dimensional coordinates. No information on surface connectivity is stored. This corresponds to the raw recorded data from 3D scanners. A depth map or layered depth map can easily be converted into a point cloud when camera parameters are known. However, unlike with laser scans, pixels on the depth map do not correspond to thin rays, but to square bundles of light rays. This information can be included into the point cloud by assigning a *stamp size* to the points, depending on the resolution of the depth maps and the point's depth. [19]

mesh The classical method in computer graphics to represent surfaces in 3D is to use a set of points called *vertices*, connected using *edges* to form a mesh of triangular *faces*. This removes the problems with loss of connectivity and low resolution that exist for point clouds and voxels. But creating and modifying meshes can be highly complex, considering that there are many possible point arrangements and triangulations which form same surface. Holes, overlapping or too acute triangles can pose problems. High-quality 2D projections of textured meshed surfaces can be rendered efficiently using established techniques.

3.2 Cues

Since images do not contain explicit measured depth information, several *cues* are used to estimate scene geometry. [20]

dense correspondences For two of more views depicting the same part of the scene, image points that correspond to the same 3D scene

[1] for non-monochromic images, intensity refers to color, which is represented usually using 3 scalar values

point on object surfaces are put into correspondence. This is done for each pixel in the images, leading to a set of dense correspondences. [21]

Stereo matching methods can be extended to work with more than two views of the scene as input. The term multi-view stereo (MVS) encompasses scene reconstruction methods that use stereo disparity estimation as their main cue, but operate on multiple views in a pairwise manner. [22, 23]

When a multiple of rectified input views are available with the cameras placed as regular intervals along a straight line, depth can instead by estimated using line detection on the EPI. [24] Similar approaches for circular or arbitrary camera movement known as image cube trajectory (ICT) analysis has also been developed. [25, 26]

sparse correspondences When the input views are farther apart, image feature detectors such as sift can be used. By cross-checking feature matches over several views, reliable multi-view correspondences can be determined. [27] They can also be used for camera calibration.

The obtained sparse set of correspondences can then be used to aid disparity estimation [20] as well as EPI analysis [28].

It is also possible to start with a sparse set of correspondences, and gradually extend it to dense correspondences in a match, expand and filter procedure. [29]

silhouette Another important cue for scene reconstruction is the silhouette of objects in the images. After the silhouette of the same 3D object has been detected in multiple images, for each image a three-dimensional cone is formed from the set of rays emanating from the silhouette. The intersection volume of those cones constitutes an approximation of the object's shape called its *visual hull*.

others Additional cues for depth reconstruction include motion of objects and lens defocus [30]. Also, low-level scene reconstruction can be revised with higher-level cues based on reasoning about the scene geometry. This mimics the way human vision involves understanding of the scene. [31]

3.3 Stereo

Stereo methods use only two views to estimate depth and scene geometry. The main cue is *disparity*.

Image pixels on the two input views that correspond to the same 3D scene point on object surfaces are put into correspondence. When the cameras are placed relatively close to each other, the scene points are depicted at similar angles are *photo-consistent*, that is, they look visually similar on the images. To find correspondences, stereo reconstruction methods use image comparison metrics on small windows of the views. For example pixel-wise metrics such as difference of squares, correlation, or mutual information can be used. [21, 32].

Knowledge of the camera's relative poses restricts the search of corresponding pixels to the one-dimensional epipolar line. *Rectification* turns corresponding epipolar lines into same horizontal scanlines. Then the horizontal displacement of corresponding points is called the *disparity*, and is inversely proportional to the depth. This horizontal disparity estimation is analogous to human stereoscopic vision. [21] Without rectification, a definition of disparity must take into account the differing orientations of the epipolar lines in both images.

The set of all possible values (x, y, d), i.e. a pixel on the reference image and a hypothetical disparity putting it into correspondence with a pixel from the other image, is called the *disparity space*. Stereo algorithms effectively use different methods to traverse it and find the *disparity map* $d(x, y) = \arg\min_{d \in \mathbb{R}} C(x, y, d)$, and convert it into the depth map.

They generally operate using four steps, namely computation of pixel-wise and aggregated matching costs, optimization by cost minimization, and refinement of the estimated disparity map. [21]

3.3.1 Matching cost

One of the two input images is denoted the *reference* image. To find the dense correspondences, the algorithm will need to estimate the disparity d corresponding to each reference image pixel (x, y).

A computable *matching cost* function $C(x, y, d)$ is defined which should be minimal when d is the correct disparity of pixel (x, y). Given rectified input views, this means the pixel $p_1 = (x, y) \in I_1$ is matched to the pixel $p_2 = (x + d, y) \in I_2$. The cost function compares the intensity values of a group of pixels in both images, centered around p_1 and p_2. Different algorithms use pixel-wise color intensity differences, normalized cross-correlation, mu-

tual information and various other metrics. [21]

When the group of pixels to compare spans a window $w(x, y, d)$ of (x, y) values, keeping d constant, correct results are produced only for fronto-parallel surfaces, which have a constant disparity (and depth) in this image window. To support slanted or curved surfaces, a three-dimensional support window is instead used which also spans over a range of disparity values. For methods based on a pixel-wise matching cost function $C_0(x, y, d) = f(I_1(x, y), I_2(x + d, y))$, this can be expressed as a 2D or 3D convolution:

$$C(x, y, z) = w(x, y, z) * C_O(x, y, d) \qquad (4)$$

Generally, larger support windows increase robustness against noise and produce a smoother depth map, but decrease accuracy at sharp edges. Metrics which directly compare color intensity values require accurate color calibration. Correlation-based metrics are robust against affine transformation between the intensity values, that is, a correct depth map is still found when the intensity values in one image are scaled and offset. The mutual-information metric is also robust against any non-linear transformation. [33, 34]

3.3.2 Cost minimization

To minimize the cost functions and find the estimated disparity map, one can distinguish between *local* and *global* methods.

The *uniqueness constraint* asserts that each pixel corresponds to one object surface point, and hence disparities must be estimated such that pixels p_1 and p_2 are put in bijective correspondence. That is, there must not be multiple pixels p_2 that correspond to the same p_1, and vice versa. As an exception, if an area corresponding to a range of sequential pixels $p_1 \in I_1$ is occluded in I_2, this may be modeled by putting those pixels in correspondence with the same pixel p_2. Some methods instead explicitly detect and mark occlusions. [21, 35]

The constraint can be formulated using an *inhibition area* $\Sigma(x, y, d)$. The function Σ assigns to each match (x, y, d) a set of forbidden matches $(x' \neq x, y' \neq y, d')$ which may not be true at the same time. [35]

The *ordering constraint* is the requirement that left-right relations, that is, the relative ordering of pixels along the scanlines is preserved. Let (p_1, p_2) and (q_1, q_2) be two pairs of corresponding pixels in the two images I_1, I_2. If $x(p_1) < x(q_1)$ then the constraint asserts that $x(p_2) < x(q_2)$. However this is not generally the case. [36, 21, 37]

Finally, the depth map is generally supposed to be piecewise smooth. The algorithms must produce a smooth depth map on object surfaces, while preserving discontinuities on object edges.

Local methods treat each pixel (x_0, y_0) in the reference image separately. A "winner-take-all" optimization is performed: $C(x_0, y_0, d_0)$ is evaluated over a range of hypothetical disparities d_0, and the one for which C is lowest is retained. While this approach is the most simple, it does not satisfy the uniqueness constraint.

Global methods formulate the entire disparity estimation as minimization of an energy function: [21]

$$E(d) = E_{\text{data}}(d) + \lambda E_{\text{smooth}}(d) \qquad (5)$$

where d is the disparity *function* $d(x, y)$. Some methods include additional terms, but generally this framework is retained. This energy function E is a computable function that attains a minimal value when d is a good disparity map.

The data term E_{data} measures photo-consistency of the supposed matches, and may be defined using the matching cost function as

$$E_{\text{data}}(d) = \sum_{(x,y)} C(x, y, d(x, y)) \qquad (6)$$

The smoothness term E_{smooth} measures piecewise smoothness of d, which can be expressed as

$$E_{\text{smooth}}(d) = \sum_{(p,q)\in\mathcal{N}} V_{\{p,q\}}(d(p), d(q)) \qquad (7)$$

where $\mathcal{N}$ is the set of pairs of adjacent pixels. $V_{\{p,q\}}$ is a function that serves as distance metric for the disparities $d(p)$ and $d(q)$. It is defined such that its value is low when d retains similar values over neighboring pixels, but allows for disparity jumps at object edges, which correspond to high frequency variations of the pixel intensity values.

It is also possible to include terms that favor disparity maps satisfying the uniqueness and ordering constraints. [38]

A finite set of possible depth values $D = \{d_1, d_2, \cdots, d_m\}$ is defined, and as a result there is a finite number of possible disparity map functions d. Minimization becomes a labeling problem. In general, finding the global minimum of E is NP-hard [20]. However different approaches exist to efficiently compute a disparity map d which approximately minimizes E. [21] One possibility is *graph-cut optimization*. [39]

It is also possible to include segmentation into the cost minimization. In this case an energy function $E(d_i, l_j)$ is minimized, assigning to each pixel both

a depth label d_i and a layer label l_j. Additional energy terms are then included to differentiate layers. [20]

Also, when sparse feature correspondences can be included: The energy function is formulated such that same depth (and layer) labels are forced at matching feature pixels. The smoothness term the ensures that the surrounding areas adapt accordingly. [20]

3.3.3 Graph-cut optimization

The graph-cut algorithm is one way to approximatively minimize the described global energy function. [39] It begins with an arbitrary labeling $d^{(0)}$. Now iteratively new labelings $d^{(i+1)}$ are computed from $d^{(i)}$. An α-β *swap* move may set some pixel labels from α to β, or from β to α, but does not modify pixels whose label is neither α nor β. A similar algorithm using instead α *expansion* moves also exists.

For any input partition $\mathbf{P}^{(i)}$, the optimal α-β *swap*, for which $E(d^{(i+1)})$ is minimal, can be found by computing a minimal cost graph cut:

Labels α and β are chosen arbitrarily. Then a weighted graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ is constructed dynamically based on α, β and the current partition $\mathbf{P}(i)$. $\mathcal{V}$. $\mathcal{V}$ consists of two terminal nodes called α, β, and one node for each pixel (x, y) that currently has the label α or β. $\mathcal{E}$ consists of three types of edges: t-links connect the pixel nodes to both the terminals α and β, and n-links connect pixel nodes corresponding to neighboring pixels.

On this graph, any cut $\mathcal{C}$ that separates the terminals α and β severs exactly one t-link for each pixel node. This defines a labeling $d_{\mathcal{C}}$ corresponding to $\mathcal{C}$, which is one optimal α-β *swap* away from $d^{(i)}$.

Weights are associated to these edges, using the functions $C(x, y, d)$ and $V_{\{p,q\}}(d(p), d(q))$ from the energy term expressions in 6 and 7. These weights and the labeling $d_{\mathcal{C}}$ are defined in a way that the minimum cost cut corresponds to the optimal α-β *swap*.

Based on this the full algorithm performs the following steps shown in figure 4.

Executions of the inner loop are called *iterations*, and those of the outer loop *cycles*. The algorithm stops after the first unsuccessful cycle in which it could not further reduce $E(d)$. The disparity map d converges to a labeling such that $E(d)$ is a local minimum, at a bounded maximal distance from the global minimum.

It can be shown that the algorithm terminates after $O(N)$ cycles, with N the number of pixels in the image. The graph cuts can be computed in low-order polynomial, and in practice near-linear time with respect to the number of nodes and edges in the graph. Thus the entire algorithm runs in polynomial time with respect to N. [39]

```
d ← arbitrary initial labeling
repeat
    success ← 0
    for all {(α, β)} ∈ D² do
        construct graph G using α, β, d
        compute minimal cost cut C
        d̂ ← d_C
        if E(d̂) < E(d) then
            d = d̂
            success ← 1
        end if
    end for
until success = 0
```

Figure 4: Graph-cut optimization algorithm

This graph-cut optimization algorithm is employed in the MPEG DERS software for disparity estimation.

3.3.4 Dynamic programming

Another approach to minimize the global energy function 5 in polynomial time works by processing scanlines independently. Two corresponding scanlines $y_1 = y_2$ from the two views are represented on a 2D square matrix M_y.

Each cell $M_y[x_1, x_2]$ corresponds to the matching cost of $I_1(x_1, y)$ in the reference image and $I_2(x_2, y)$ in the other image. That is, $M_y[x_1, x_2] = C(x_1, y, x_2 - x_1)$ unless another mapping for the disparity values is used. Disparity for all the points on the scanline y is estimated by computing the minimum-cost path through this matrix. This path is diagonal where the disparity remains constant along the scanline, and horizontal or vertical when the scene point is occluded in one of the two views. Because disparity is assumed to be piecewise smooth, the path consists of near-diagonal curves separated by horizontal or vertical segments. So it can be computed incrementally by looking only at the matrix cells in a local neighborhood of the current cell, and so the disparity of a whole scanline can be estimated in linear time. It is described as a form of dynamic programming because previous results affect the progression of the algorithm.

By repeating this process for each scanline a full disparity map d is computed in polynomial time. However because scanline are processed independently, inter-scanline consistency is not enforced,

resulting in a horizontal "streaking" artifacts on the disparity map [33]. Also, this method is only applicable when the *ordering constraint* is met on the input views.

3.3.5 Semi-global matching

Semi-global matching [33, 40, 32] resolves the lack of inter-scanline consistency using the realization that although matching pixels on rectified images are located on the same horizontal scanlines, this dynamic programming method can also cross the image in arbitrarily oriented lines.

For a given pixel $\vec{p}$ of the reference image, and a given direction vector $\vec{r} \in \mathbb{R}^2$, the algorithm traverses the line $\vec{p} + t\vec{r}$ in both images. The matrix $M[t, d]$ is considered, where the column indicates the position along this line t, and and row is a disparity d value for this pixel. So $M[t, d] = C(\{\vec{p}+t\vec{r}\}, d)$, where corresponding pixels with disparity d are still taken with a horizontal offset.

Because $d(x, y)$ is still piecewise smooth when moving across that line, disparity values can be computed by traveling through the minimum-cost path of this matrix. The semi-global matching algorithm chooses 8 directions r: four straight and four diagonals. This computation is performed for all pixels on the image border, so that 8 different disparity estimates $d_i(\vec{p})$ get computed for each pixel. Each of these disparity maps contains the streaking errors along their direction $\vec{r}$. The final disparity map $d(\vec{p})$ is constructed by taking for each pixel the value $d_i(\vec{p})$ whose cost $C(\vec{p}, d(\vec{p}))$ was minimal. Errors are thereby averaged out.

The semi-global matching method also defines a global error function similar to formula 5 which included a smoothness term, insuring the disparity map becomes piecewise smooth. It it effectively minimized during the minimum-cost path traversal, through an additional term included in the recursive path cost function. [33]

3.3.6 Mutual information

Semi-global matching also uses a mutual information error metric for the computation of pixel matching costs. As already stated it is robust against non-linear color calibration difference between the two views. The mutual information $I(I_X, I_Y)$ between two images I_X and I_Y can be interpreted as a measure of the certainty by which X can be predicted knowing Y, and vice versa. For semi-global matching, I_X and I_Y are windows of the image around given pixel, taken by convolution with a gaussian kernel.

Let X and Y be a random variables which give the intensity of a random pixel in the images I_X and I_Y, respectively. Let $\mathcal{I}$ be their range of possible intensity values, for example $\mathcal{I} = [0, 255]$. The probability distribution $P(x)$ is the histogram of the image, and X is invariant of the pixel's positions (same for Y). The joint probability distribution $P(x, y)$ gives the probability that a pixel *at same coordinates* in I_X takes value x and in I_Y value y.

The *entropy* of X is defined as

$$H(X) = -\sum_{x \in \mathcal{I}} P(x) \log_2(P(x)) \tag{8}$$

It indicates the average number of bits per pixel needed to transmit I_X over a binary channel using an optimal binary coding. For instance the Huffman coding satisfies this constraint, by associating longer binary sequence codewords to less probable intensity values. As such entropy can be regarded as a measure of the amount of information contained in I_X, in the context of signal coding.

The *joint entropy* between random variables X and Y is defined as

$$H(X, Y) = -\sum_{x \in \mathcal{I}} \sum_{y \in \mathcal{I}} P(x, y) \log_2(P(x, y)) \tag{9}$$

Its interpretation is the same, for a coding that assigns binary sequences to the possible pairs (x, y).

If I_X and I_Y are sent sequentially over a channel with optimal coding, $H(X) + H(Y)$ bits are required per pixel. If instead corresponding pixel pairs (x, y) are sent together, $H(X, Y)$ bits are required, which is always a lower value. If there is *any* function $y = f(x)$ that maps pixel values from I_X to the ones in I_Y, then $P(x, f(x))$ gets a higher value and all other $P(x, \cdot) = 0$. Even when this mapping applies only approximately, it allows for a more efficient coding: shorter codewords can be associated to values (x, y) that are more likely to come together. Sending the images sequentially would not incorporate any information about which values come together.

Mutual information is defined as this difference

$$I(X, Y) = H(X) + H(Y) - H(X, Y) \tag{10}$$

Originally developed in the context of registering medical images [34], it it a measure of similarity between two images, that is robust against illumination or modality differences between the images. It only requires similarity in the sense that some pairs (x, y) are more probable on corresponding pixels, than on different pixels.

In the context of stereo matching, it provides robustness for example against different camera calibration, scene illumination and noise.

3.4 Shape from silhouette

When cameras are placed around a 3D object, at possibly more different view poses that do not permit disparity estimation, different techniques can be used to directly reconstruct the volumetric shape of the object.

A cue for 3D scene reconstruction in this case is to extract object silhouettes from the input views. Silhouettes can be detected relatively easily using segmentation and corner detection methods. Each silhouette provides some information about the three-dimensional shape of a scene object. Using knowledge of the camera parameters, a cone of rays in 3D scene space is constructed which emanate from the camera center and cover the object. The intersection of these cones from different camera poses produces an approximation of the object's shape. [41, 17]

3.4.1 Visual hull

The precision of this approximation depends on the number of cameras. At best, the *visual hull* of the object can be obtained, which in general is a superset of the object's actual volume, but a subset of its convex hull. [42]

Computing the shape of this cone intersection is done most easily using a voxel scene representation. Constructing a high quality mesh from the silhouette cone intersections would be very complicated. However an intermediary constructive solid geometry (CSG) representation can be used when the 3D model is to be rendered using ray-tracing. [17]

The process of *space carving* starts with a fully occupied slab of voxel space. Then for each view, all voxels falling outside the silhouette cone are freed. The resulting volume is a quantized representation of the visual hull according to the given volumetric grid. It can be further improved and converted into a mesh by surface extraction, surface smoothing and mesh complexity reduction. [30]

Several early approaches to space carving use an octree representation of the volume. Either an octree representation of for each silhouette cone is generated and then their intersection is computed, or a single octree is successively refined for each view. [43]

If the virtual camera using which the reconstructed volume is going to be rendered is known beforehand, one way to avoid the quantization artifacts is to build a *view-dependent* visual hull representation, by using its image space coordinates for the voxel grid. [41]

3.4.2 Photo hull

After a visual hull has been constructed from silhouette cone intersections, it can be refined by carving out inconsistent voxels. Under the assumption of lambertian reflectance, for any voxel laying on the object surface which is visible in at least two views, its corresponding pixels should have approximatively the same intensity. The voxel is a false surface voxel it it protrudes from the real object surface, and is projected onto different features in the images. [44] If this *photo-consistency* test fails, the voxel is freed, because it was not part of the object volume. Space carving algorithms generally operate by repeating this procedure until no inconsistent voxels remain. [44, 17, 45]

Visibility of the voxels in the different views needs to be determined for this: as space carving progresses, the remaining voxels may occlude each other in complex and changing patterns. [17] Different methods have been developed to do this efficiently.

The best approximation of the actual object shape that can be obtained by testing photo-consistency of the voxels' projections is called the *photo hull*. [23] One has

$$\text{object} \subset \text{photo h.} \subset \text{visual h.} \subset \text{convex h.}$$

3.4.3 Ordinal visibility constraint

One is to arrange the cameras such that the *ordinal visibility constraint* is fulfilled: It requires that there be a single near-to-far norm $\|\cdot\|$ for the voxels which is valid for each view. [46] So for any two voxels p, q, and for every view, the projection of p occludes that of q only if $\|p\| < \|q\|$. It is satisfied whenever no scene point is contained within the convex hull of the input camera centers. [46] One such camera arrangement is to place all cameras on a common plane, with the entire scene on one side of it. Then $\|\cdot\|$ is the point's distance to the plane.

With this constraint met, a voxel reconstruction V of an object can be build using the simple algorithm in figure 5. [46]

A projection of a voxel (cube) v onto an image I_i corresponds to a set of pixels $\text{proj}(v, I_i)$ on the image. If v is part of the object surface, and assuming the object surface has a relatively constant coloring region, then all the pixels $\text{proj}(v) = \bigcup_i \text{proj}(v, I_i)$ should have consistent coloring. λ_v is a measure of this consistency.

If v is (partially) occluded, $\text{proj}(v, I_i)$ contains fewer pixels. Because of the ordinal visibility con-

$V \leftarrow$ all voxels free
for all $v \in V$, by increasing $\|v\|$ **do**
  project v into all views I_i
  compute λ_v
  if $\lambda_v <$ threshold **then**
    set v to average pixel color
  end if
end for

Figure 5: Voxel coloring algorithm with ordinal visibility constraint [46]

straint, it is possible to keep track of occlusions using a binary mask attached to each image. The masks are initialized with 0 at each pixel.

if voxel v occludes voxel v' (in any view), then v is necessarily visited before v'. After v was projected to a view I_i, the mask is set to 1 for all pixels $\text{proj}(v, I_i)$. When a new voxel v' is projected, the masked pixels are ignored, because it is known in this image region v' is occluded by v, or another preceding.

In addition to carving out inconsistent voxels, this method also allows to associate colors to the voxels based on the projections $\text{proj}(v)$.

3.4.4 Generalized voxel coloring

The ordinal visibility constraint is a severe restriction on camera arrangement, and makes it impossible to construct a full 3D model of the objects from all directions. A generalized voxel coloring algorithm for the arbitrary camera poses performs the steps shown in figure 6. [17]

$V \leftarrow$ all voxels occupied
repeat
  all voxels consistent $\leftarrow 1$
  for all $v \in V$ where v is occupied **do**
    project v into all views I_i
    compute λ_v
    if $\lambda_v <$ threshold **then**
      set v to average pixel color
    else
      set v to free
      all voxels consistent $\leftarrow 0$
    end if
  end for
until all voxels consistent $= 1$

Figure 6: Generalized voxel coloring algorithm [17]

Computing the projection pixels $\text{proj}(v, I_i)$ also becomes harder because pixel visibility can no longer be determined using a simple binary mask. Several techniques have been developed to implement generalized voxel coloring efficiently. [17]

A multi-view photo-consistency measure λ_v needs take into account that unlike in stereo methods, cameras are placed further apart and a voxel englobes a three dimensional region of space. Abrupt color boundaries on object surfaces for instance may lead to false negatives. [17]

A more recent algorithm enables voxel reconstruction of objects with specular reflective surfaces. [47] Along with the voxel carving iterations, normal directions of the object surface are estimated, and used to estimate the expected coloring given an environment map.

3.5 Epipolar plane image (EPI)

As described in section 2.2.1, for an array of cameras placed in regular intervals along a straight line, an EPI can be constructed for each scanline of their rectified images. Each row of the EPI corresponds to the same scanline of another camera. As explained the EPI can also be interpreted as a 2D section of the lumigraph. [24, 48, 49, 50, 51]

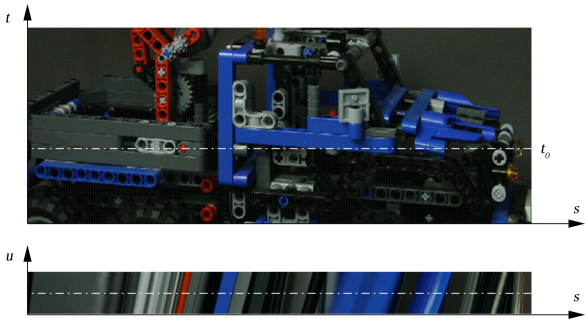

Figure 7: Example of EPI [49]

An example is shown in figure 7. The shown coordinates correspond to the two-plane light field parametrization. The upper view shows the rectified image of one of the cameras. One scanline $t = t_0$ is highlighted. The bottom view the shows the EPI for that scanline. Each row is equal to that same scanline from cameras with different horizontal positions u. The highlighted middle camera corresponds to the upper view.

EPI images can also be constructed for non-lateral camera motion. For example when the camera moves towards the scene, that is, the camera plane C is perpendicular to F, hyperbolic curves occur instead in the EPI. [24] Most of the more recently developed EPI-based techniques assume lateral camera motion, though. There are also generalizations for rotational or arbitrary movements [25, 26], and for non-planar image projections [52].

3.5.1 Observations

It can be observed that the EPI is formed entirely of straight line segments: The same object points are depicted, but disparities increase as the camera position u varies. Instead of two views as in stereo, now an entire array of views across the baseline is available. Instead of comparing image windows to estimate disparity, an *increase* of disparity can be measured, by estimating the slope of the line segments in the epi. This way information from all cameras is implicitly taken into account. The slope is directly proportional to the depth of the scene point.

With a two-dimensional camera array, different EPIs for one same scanline, but different camera heights v can also be constructed, as well as vertical EPIs where instead the t axis is fixed in the images and the camera moves along the v axis. This can be used to cross-check and improve the result of depth estimation.

Oblique line segments form in the EPI under the assumption of lambertian surface reflection: Object surface points need to reflect light of the same color in all directions. One oblique line segment represents the same scene points seem from different angles. When surfaces have specular reflection, color gradients occur in the line segments, as can for instance be observed on the righthand side in figure 7. This complicates line detection, but also represents additional information about the object.

Occlusions are visible in the EPI in the same way they are for the camera images. For any line of sight a nearer object can never be occluded by a farther object. For EPI analysis this means that a line segment with lower slope can never be interrupted by one with higher slope. This is called the *occlusion constraint*. Also, line segments in EPIs always have a positive slope.

3.5.2 Line detection

The problem of EPI-based depth estimation is to detect line segments as accurately as possible, by exploiting the mentioned constraints and redundancies.

As the camera moves along u, complex occlusion patterns can occur and hence the lines on the EPI line segments can be interrupted several times and by several layers of occlusion. While not visible everywhere, the underlying lines always cross the entire height of the EPI. But a 1D sweep through the EPI is not sufficient to find all line segments and associate depth to all scene points.

EPI analysis algorithms generally proceed by first locally estimating the most likely slope for each point (s, u) in a given EPI. Then results are integrated using other EPIs depicting the same scene points [49] or the same EPI at different scales [50], yielding a single depth map for each view.

One way to detect lines segments in a given EPI is via *Hough transform* [48]: Lines in the EPI are parameterized using two variables, a slope m (or an angle) and and offset p. So the line space (m, p) constitutes a dual space to the EPI space (s, u). Each element (m, p) is set to a consistency measure of that hypothetical line in the EPI. Then for each (s, u), the pencil of possible lines (with positive slope) passing through that point corresponds to a 1D section in line space. Along that curve, the element with maximum consistency value is searched, and its slope is attributed to (s, u).

Another is to use a *structure tensor* J [49]. J is a 2×2 matrix formed for a given EPI, point (s, u) and window function $w(\Delta s, \Delta u)$. One of its eigenvectors corresponds to the direction that is maximally aligned with the gradient within the window, which is an estimate of the line direction. The window w can be set to a Gaussian kernel. The *coherence* of the structure tensor also constitutes a reliability measure for this estimated direction. [49]

By discretizing the depth into a finite set of possible depth labels d_i, local line detection for a given EPI, can be expressed using any cost function $C(s, u, d_i)$ which is minimized by comparing its value for all labels d_i. [50]

Each pixel in the EPI corresponds to a scene point, and to a pixel in (at least) one image. Instead of relying only on the EPI for line detection, using image feature detectors such as sift in the views is possible. With the method used in [28], lines are instead fitted through EPI pixels attributed with matching features.

3.5.3 Fine-to-coarse method

The method described in [50] employs a *fine-to-coarse* methodology to get reliable depth estimates, still on a single EPI. The method starts with the original ("fine") EPI.

As a pre-processing step prior to local line detection, an *edge confidence* is used to sort out pixels (s, u) of the *epi* that seem promising for depth estimation. It measures the difference of (s, u) and its horizontal neighbors (s', u), where s' varies in a 1D small window centered at s. The sum of squares is thresholded, and pixels where it is too large are discarded from further processing. This removes most of the pixels corresponding to homogeneous image regions, that is, where high-frequency variation is

low.

For the remaining pixels (s, u), depth is estimated by minimizing a cost function with a finite set of possible depth labels. A *depth confidence* measure is also calculated for each pixel (s, u), and pixels whose reliability is deemed too low are discarded. This include pixels with strongly view dependent appearance, typically due to specular reflectance.

The now remaining pixels form a sparse set of *line segment tuples* $l = \langle d, s, u, \bar{r} \rangle$. d is the estimated depth, and $\bar{r}$ the mean radiance, that is, the mean intensity value along the line, with some additional filtering.

In a *depth propagation* step, these line segments are then extended along the line's direction to the free pixels of the EPI, without overwriting existing depth estimates and while respecting the occlusion constraint. (Line segments with greater slopes are not overwritten.)

These depth estimates are considered reliable, but parts of the EPI still remain free due to either insufficient edge confidence or depth confidence. In order to get a full depth map, the entire EPI is now down-sampled to half its size, and the whole process is repeated. The algorithm becomes sensitive to lower frequency variations in the image. The existing depth estimates are used only as bounds for the possible depth estimated on the lower resolution EPI. This *fine-to-coarse* multi-scale scheme is repeated iteratively. Finally the depth estimates are combined, such that lower resolution estimates are used to fill in holes from higher resolution estimates.

3.5.4 Consistent labeling

Another approach to compute a consistent depth map respecting the occlusion constrained is described in [49]. This is still in the context of a single EPI.

Again a finite set of depth labels d_i is used, but the depth map estimation is formulated as a energy function minimization problem. A binary indicator function $b_i : (s, u) \mapsto \{0, 1\}$ is defined for each depth label d_i. It takes the value 1 if and only of d_i is the depth estimate for (s, u). $\mathbf{b} = (b_1, b_2, \cdots, b_n)$ is the vector of indicator functions.

The global energy function is defined as

$$E(\mathbf{b}) = R(\mathbf{b}) + \sum_{i=1}^{n} \sum_{(s,u)} C(s, u, d_i) b_i(s, y) \qquad (11)$$

The second term uses the local line detection cost function $C(s, u, d_i)$, multiplied by the binary indicator, meaning that after global minimization only one depth d_i is selected per pixel. For any u_i which assigns 1 to multiple depth hypothesis, the cost function would only increase.

The first term $R(\mathbf{u})$ is called the regularizer, which insures that the occlusion constraint is respected. It is constructed based on a penalty function $e(d_i, d_j, \vec{r})$ which assigns costs to transitions from depth label d_i to d_j in direction $\vec{r}$.

3.5.5 Global integration

These methods operate on a single EPI, and return a , that is, a depth value for each EPI pixel (s, u). The next step is to integrate all of the one single depth map per camera view.

In the case of a 1D camera array there is one per scanline. A trivial method to create depth maps consists of a remapping of the pixels. [28] For 2D camera arrays, multiple horizontal and vertical EPIs can be constructed for different and for same scanlines. This allows for global integration methods that further refine consistency.

The method described in [49] uses an energy minimization scheme for this, during which piecewise smoothness of the resulting depth maps is encouraged, allowing for discontinuities at image edges.

3.5.6 Redundancy and encoding

As mentioned in section 2.3.1, the regular structure of EPIs is a result of a light field redundancy that occurs with lambertian reflectance. Rays emitted from an object point have the same intensity in all directions.

So the EPIs representation can be exploited to compress light fields. As described before the it can be approximated as a set of line segment tuples $l = \langle d, s, u, \bar{r} \rangle$. This constitutes a sparse representation of the light field section, where this redundancy is removed.

The difference between the original EPI, and the one reconstructed from the sparse representation, called the *Delta-EPI* in [50], contains effectively only information about view-dependent effects such as specularities (and about errors in the slope estimations). This Delta-EPI is has low entropy and can be compressed efficiently.

EPIs is also suitable for sparse encoding. In [11], the EPI is subdivided into small 8×8 pixel blocks. Similarly to the discrete cosine transform employed by JPEG, the blocks can be compressed by separating different frequency components. It is separated into a set of *Gabor atoms*.

The *Gabor function* is a Gaussian window on a sine wave. In 2D, it is parameterized by a tuple $\langle \theta, f, \phi, u, \sigma \rangle$ consisting of the direction $\theta \in [0, 2\pi]$, frequency f, phase ϕ, and position u and window size σ. Convolving an image with a Gabor function extracts components of a certain frequency and direction, and only at a certain position. This is called a *Gabor filter*.

The image can be reconstructed as a linear combination of several Gabor functions with different intensities, then called Gabor atoms. In [11] it is shown that keeping only a small number of Gabor atoms with highest intensities is sufficient to get a good reconstruction of the EPI block. This can be applied for compressive sensing of the light field, using a sensor system that only records a sparse set of Gabor atoms.

3.6 MPEG Depth Estimation Reference Software

The Depth Estimation Reference Software (DERS) software package is an implementation of depth estimation. It is developed and used by MPEG. Along with the view synthesis software vsrs is was initially developed at Nagoya University in 2008, and is now being maintained by the MPEG-FTV group.

The software uses stereo methods, and takes *three* views as input: a left, center, and a right image (I_L, I_C, I_R). The cameras are assumed to be arranged approximatively on a straight line, with a relatively low baseline. The center camera needs to be equidistance from the left and right cameras. The output is one depth map D_C, relative to the center camera. Also, extrinsic and intrinsic parameters for the three cameras are taken as input, in the form of pairs of 4×4 transformation matrices.

3.6.1 Algorithm

The initial version of DERS implements the following algorithm: [53]

For each pixel in the center image $(x, y) \in I_C$, a disparity d is estimated. The pixel $(x, y) \in I_C$ has disparity d when it corresponds to $(x + d, y) \in I_L$ and $(x - d, y) \in I_R$. Error functions $E_L(x, y, d)$ and $E_R(x, y, d)$ are defined as the absolute intensity difference between those hypothetical correspondences.

$E_{\text{sim}}(x, y, d) = \min\{E_L(x, y, d), E_R(x, y, d)\}$ is defined as their minimum. It is a disparity cost function robust against occlusion in one of the two views.

A global error function E is defined as $E(x, y, d) = E_{\text{sim}}(x, y, d) + \lambda E_{\text{reg}}(x, y, d)$. The term λE_{reg} is added to encourage piecewise smoothness. A disparity map which minimizes E is now computed using the graph-cut algorithm described before in section 3.3.3.

This disparity map is then directly converted into the depth map D_C. Because d takes only a number of discrete values (256 in this version), and the depth is inversely proportional to it, the discretization can be significant.

3.6.2 Development

Several improvements to the basic algorithm have been added to DERS.

A *segmentation mode* and a *semi-automatic mode* were added. The segmentation mode performs segmentation into different layers by color in addition to depth estimation. Semi-automatic mode takes auxiliary data as input to improve the results. Different matching cost functions have been implemented which compare pixels on in 2D windows. A *temporal regularization* step was added for use with video frames, which distinguishes moving and static elements. [54]

4 View Synthesis

The next stage of the FTV pipeline is *view synthesis*, the rendering of views from a virtual camera. This virtual camera can have arbitrary position, orientation, field of view, and possible focussing, within a limited range. A goal of free navigation (FN) is to get a large and uninterrupted range, so as to create a VR system where the user can freely move through a scene.

In general a camera image varies smoothly when camera's parameters change. That is, the *optical flow* is continuous, with the exception of discontinuities and occlusions at the boundaries of scene objects. So except for very small movements, scene geometry information is an essential ingredient for view synthesis.

In general one distinguishes between *IBR* and *model-based rendering (MBR)*. IBR methods interpolate the light field, without knowledge of the scene geometry. This requires a lot of input data, and synthesis is generally possible only for a small range of motion. However the accuracy and efficiency is completely independent of the geometric complexity of the scene. At the other extreme, MBR methods render the scene based on the reconstructed geometry, typically using a textured mesh and classical computer graphics techniques. This poses no restriction on the virtual camera, but requires accurate modeling of the geometry. Hybrid techniques combine elements from both techniques. [55, 30, 56, 57]

4.1 IBR techniques

As stated IBR techniques take only a sample of the light field as input, typically acquired using camera arrays or plenoptic cameras. As shown in section 2.3.2 a (virtual) pin-hole camera corresponds to a 2D section of the plenoptic function f. *Light field rendering* methods use the sparse sample of f known from the input views to estimate intensities of new rays with different positions and orientations. This interpolation is done in 5D (or 4D) space: there is no continuity between light rays that just intersect in the 3D space.

4.1.1 Lumigraph interpolation

An example of light field interpolation on a lumigraph (two-plane light field parameterization) is shown on figure 8. The figure shows a 2D equivalent.

A fifth, virtual camera with 5 pixel image is shown on the bottom, with center u', and one of its virtual rays is highlighted. It needs to be interpolated from the known rays from the lumigraph. Four rays that have similar positions and orientations are highlighted. An intensity value for the virtual ray can be interpolated as a weighted mean of those.

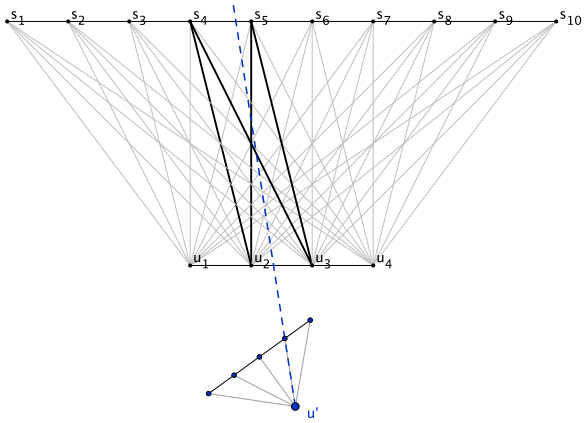

Figure 8: Light field interpolation

On the figure the lumigraph is discretized with 4 camera poses u_i, each of which sample 10 pixels s_j. The shown rays (u_i, s_j) in the lumigraph actually correspond to a beam of light rays in the real light field. In the 3D case, each *grid point* [9] $x_{u,v,s,t}$ corresponds to a beam of light. To model this a base function $B_{u,v,s,t}$ is associated to each grid point such that

$$f(u', v', s', t') = \sum_{(u,v,s,t)} x_{u,v,s,t} B_{u,v,s,t}(u', v', s', t')$$

This allows any ray (u', v', s', t') to be estimated using the existing grid points $x_{u,v,s,t}$. Geometrically, the intersections between the virtual ray with the camera plane C and focal plane F are taken, a kernel is placed the planes C and F at the intersection points, which defines the weights at which rays are taken. Doing this on both planes constitutes a 4D ray filter. [8] A simple box basis would be a constant function, which always takes the value 1 when (u, v, s, t) are closest to (u', v', s', t'). It produces discontinuities in the virtual image. Better results are obtained using a quadralinear basis: The 4 neighboring points on F and C are taken, and the ray is interpolated against 16 known rays (aka grid points). [9]

4.1.2 Depth-based lumigraph interpolation

The described method assumes that the entire object surface is located on the focal plane F. It leads to incorrect results if this is not the case, even if the input views are pin-hole camera images.

When a Gaussian or similar kernel is used as 4D ray filter, a blurring defocus effect occurs in the virtual image, which simulates that from a lens camera. The size of the kernel used on the camera plane C corresponds to a virtual aperture size. A wider kernel creates more blur, reduces the range of motion, but improves the precision of the interpolation. (A greater number of different virtual poses become possible) [58]

The goal is to interpolate between rays that emanate from the object surface. Knowing its location, it is possible to choose more appropriate grid points by intersecting the object surface plane instead of the focal plane. This is shown on figure 9. Instead of a single surface plane, it is also possible to take a non-flat surface which approximates the actual object's surface. [58]

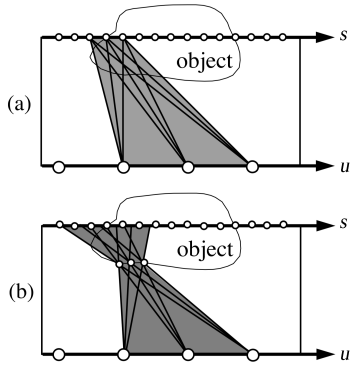

Figure 9: Light field interpolation (a) without, (b) with depth correction [9]

4.1.3 Plane sweeping

Another simple method for view synthesis without geometry information is *plane sweeping*. [59, 51]. The main observation the rays pointing out of corresponding pixels on different input views and also of the virtual view (when placed in the 3D space), will intersect and because of this the pixels will have the same color.

Multiple planes D_j are placed in 3D space parallel to the image plane of the virtual camera, as depicted in figure 10. Now each view I_i is reprojected onto each plane D_j, forming a set of *transformed images* $I_{(i,j)}$. Reprojection means that for each pixel in I_i, rays are extended and the pixel is mapped into the ray's intersection with D_j. On the figure the set of rays emanating from the same feature (the soccer player's head) is shown. They intersect on one of the depth planes D_z, that is, the one where the feature is located in 3D space. Therefore all the projections, $\forall i, I_{(i,z)}$ have the same pixel value at that pixel.

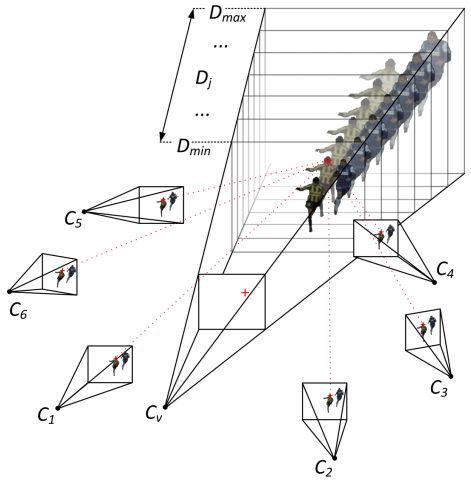

Figure 10: Plane sweeping [51]

For view synthesis, the value of each pixel in the virtual image $p_v \in I_v$ needs to be determined. A ray is extended from I_v, and intersected which each depth plane D_j. Let p'_j be this intersection point. For each depth plane D_j, the variance of $I_{(i,j)}[p'_j]$ for the different images i is computed.[2] It should be minimal at the correct D_j because there the rays from the real images intersect.

Therefore p_v is assigned the mean value of $I_{(i,j_0)}[p'_{j_0}]$, where j_0 is the index of the depth plane for which the variance was minimal. At a side product, the depth j_0 was also estimated for p_v.

The method can be related to stereo techniques, light field rendering and space carving, as it implicitly does similar calculations. [59] It can be improved when the depth is known (to some degree), by eliminating some depth hypothesis j beforehand.

4.2 DIBR techniques

Depth image-based rendering (DIBR) refers to techniques that are mostly IBR, but require depth maps for each view as additional input data.

When the relative camera poses are known, *dense correspondences* can be directly derived from the depth maps. The 2D vector field which associates to each pixel in $\vec{p}_1 \in I_1$ a vector pointing to the coordinates of the corresponding pixel in $\vec{p}_2 \in I_2$ is also called the *optical flow* $F_{1,2} : p_1 \mapsto p_2$. With pin-hole cameras, it can be computed using matrix multiplication. That is $p_2 = \mathbf{M}_{1,2}p_1$, where $\mathbf{M}_{1,2}$ is a 3×3 homogeneous 2D transformation matrix derived from the relative camera poses of views I_1 and I_2.

[2] The brackets indicate that the pixel p'_j is taken from the transformed image $I_{(i,j)}$.

If dense correspondences are known, but the pixels depths are not (for example as a result of disparity estimation without precise knowledge of camera poses), the information is also called *implicit geometry*.

4.2.1 Correspondence interpolation

The optical flow $F_{1,2}$ can be interpolated to that $F_{1,v}$ which would correspond to a virtual camera I_v placed in-between I_1 and I_2. When only implicit geometry is available, the simplest approach is by linear interpolation $F_{1,v} = kF_{1,2}$ with $k \in [0, 1]$. When the difference $\|C_1 - C_2\|$ between the two camera centers is small, this approximates the optical flow of a camera placed at $C_1 + kC_2$. [60]

If C_v is not between C_1 and C_2 in a straight line, then this interpolation would alter image shapes, because corresponding image pixels also no not move in a straight line when C_v varies from C_1 to C_2. It can be improved by applying a transformation to I_1 before the morphing step, and the inverse transformation on I_v afterwards. [**?**]

When depth maps are known, the accurate optical flow can be computed by back-projecting each p_1 into scene space and then reprojecting it on p_v. That is, the point cloud is rendered. Errors in the depth map can also be alleviated by linearly interpolating scene point positions instead. [61]

4.2.2 Morphing

The virtual image I_v is now generated using a real image I_1 and an estimated optical flow $F_{1,v}$. This is called *image morphing* or *warping*. The basic technique is to copy the intensity of each pixel $p_1 \in I_1$ to the corresponding $p_v = F_{1,v}(p_1)$ in the new image.

The transformation does not preserve the regular pixel grid: $\mathbf{M}_{1,2}p_1$ generally does not take integer coordinates, so p_v is not a single pixel on the image I_v. When depths are known, one solution is to use a splatting kernel, whose size is adapted to the depth. However it does not handle excessive stretching. [52]

A pixel p_1 in the image actually corresponds to a square region, which maps to a quadrilateral for p_v in I_v. So another approach is to instead transform the lattice points between pixels p_1, and fill in the quadrilateral in I_v. [60] This can be implemented efficiently using texture mapping.

Because the optical flow is constant for fronto-parallel surfaces, it can be optimized by moving patches in the image together. In [60] a quadtree decomposition of the images is employed for this.

Due to occlusions and camera bounds, the optical flow is not a one-to-one mapping: $\mathbf{M}_{1,2}p_1$ may fall outside the range of I_2. If the scene point depicted by p_1 is occluded in I_2, then p_2 falls in the same position as pixels from the occluding surface. The same is true with the interpolated $F_{1,v}$, and so holes and overlaps will occur. Back-to-front ordering needs to be respected when rendering I_v, and remaining holes need to be filled.

Since at least two views I_1 and I_2 are available, results are improved by computing two virtual images I_v^1, I_v^2 using $F_{1,v}$ and $F_{2,v}$, and then combining them using *image blending*: A (weighted) mean of the pixel intensities is taken, $I_v = kI_v^1 + (1-k)I_v^2$. For holes in I_v^1 or I_v^2, only one of the values is taken. [61] In practice this fills in most larger holes due to occlusion, so that inpainting algorithms can be used for remaining holes. [62]

In [63] an image interpolation technique based on human visual perception is described. By focussing on correct correspondence of edges, homogeneous regions and coherent motion instead of on pixel-wise correspondence, perceived visual quality is improved.

4.3 MPEG View Synthesis Reference Software

The View Synthesis Reference Software (VSRS) software package implements an algorithm based on image morphing. Like its counterpart the DERS, it is designed for use with a set of linearly displaced views facing in the same direction, with a relatively low baseline.

The first versions of VSRS, originally called *ViewSynthesis*, were developed at Nagoya University in 2008. It is programmed in C++, and uses the *OpenCV* library. It is used by MPEG as defacto standard method for view synthesis, and new algorithms are evaluated by comparing the quality of their output to that of VSRS. It has been extensively tested for linear and arc camera arrangements. [64]

The synthesized virtual images are compared using the peak signal-to-noise ratio (PSNR).

4.3.1 Input data

The synthesis algorithm takes the following data as input:

- left-side camera image I_L
- left-side camera depth map D_L
- parameters of left-side camera

- right-side camera image I_R
- right-side camera depth map D_R
- parameters of right-side camera
- parameters of virtual camera

Camera parameters are expressed in the form of two 4×4 matrices for the extrinsic (pose) and intrinsic (projection) transformation.

The output is the synthesized virtual image I_v.

4.3.2 Algorithm

In VSRS version 2.0, the view synthesis algorithm proceeds according to the following steps: [62, 65]

1. First, the *depth maps* are morphed into the virtual view. Like with image morphing described before, pixels are moved to new coordinates, *without changing their values.*

   For the case where images are rectified and the three cameras are on the horizontal plane, morphing only displaces pixels on horizontal scanlines. Otherwise 3D reprojection is used. At first the left- and right-side depth maps are mapped independently into the target view.

   Small holes are fixed using a median filter. The resulting virtual image depth maps are here denoted D_v^L and D_v^R.

2. Now using texture mapping, the left-side image I_L is morphed into D_v^L yielding I_v^L. And the right-side image I_R is morphed into D_v^R yielding I_v^R.

   The two resulting images I_v^L and I_v^R have different occlusions. Using the previously generated depth maps (instead of the input depth maps and 3D transformations) takes advantage of the applied median filter.

3. Holes in I_v^L are identified by looking at missing pixels in D_v^L. Then they are then filled by copying over pixels from I_v^R. The same is done in the other direction.

4. Now I_v^L and I_v^R are blended together. The weighted mean value of all pixels is computed with

   $$I_v = \frac{b_R}{b_L + b_R} I_v^L + \frac{b_L}{b_L + b_R} I_v^R$$

   where b_L and b_R are the distances of the virtual camera position to the left-side and right-side camera.

5. Remaining holes in I_v are filled in using an inpainting algorithm from *OpenCV*.

4.3.3 Extensions

Since version 2.0, depth maps can be morphed using 3D reprojection, instead of only horizontal displacement.

Version 2.1 adds a mode for which blending is disabled and replaced with a nearest-neighbor approach. It simply takes only the image (I_v^L or I_v^R) from the camera which is closer to the virtual position. [65]

For version 3.0 several improvements are added. One is an additional blending method, which operates per-pixel and always chooses the pixel from the view I_v^L or I_v^R which is, according to D_v^L or D_v^R, closer to the virtual camera position. This adds some robustness against occlusion. [66]

In version 4.0 the depth maps are extended from 8 to 16 bit, because 256 possible depth labels have proven insufficient for circular camera arrangements. [64]

References

[1] C.-C. Lee, A. Tabatabai, and K. Tashiro, "Free viewpoint video (fvv) survey and future research direction," *APSIPA Transactions on Signal and Information Processing*, vol. 4, October 2015.

[2] "Call for evidence on free-viewpoint television: Super-multiview and free navigation," tech. rep., MPEG, October 2015.

[3] G. Lafruit, M. Domański, K. Wegner, T. Grajek, T. Senoh, J. Jung, P. T. Kovács, P. Goorts, L. Jorissen, A. Munteanu, B. Ceulemans, P. Carballeira, S. García, and M. Tanimoto, "New visual coding exploration in mpeg: Super-multiview and free navigation in free viewpoint tv," 2016.

[4] G. Lafruit, K. Wegner, T. Grajek, T. Senoh, K. P. Tamás, P. Goorts, L. Jorissen, B. Ceulemans, P. C. Lopez, S. G. Lobo, Q. Wang, J. Jung, and M. Tanimoto, "Ftv software framework," tech. rep., MPEG, July 2015.

[5] M. Tanimoto, "Proposal of new study for mpeg-ftv," tech. rep., MPEG, October 2015.

[6] H.-Y. Shum and L.-W. He, "Rendering with concentric mosaics," *Proc. SIGGRAPH*, 1999.

[7] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," *Computational Models of Visual Processing*, 1991.

[8] M. Levoy and P. Hanrahan, "Light field rendering," *ACM SIGGRAPH*, July 1996.

[9] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," *SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996.

[10] T. Fujii, T. Kimoto, and M. Tanimoto, "Ray space coding for 3d visual communication," *Proc. Picture Coding Symposium*, vol. 2, pp. 447–451, March 1996.

[11] Q. Yao, K. Takahashi, and T. Fujii, "Compressed sensing of ray space for free viewpoint image (fvi) generation," *ITE Transactions on Media Technology and Applications*, vol. 2, no. 1, pp. 23–32, 2014.

[12] D. Doyen, A. Schubert, L. Blondé, and R. Doré, "Light field uses cases and workflows," tech. rep., MPEG, February 2016.

[13] C. Hahne, A. Aggoun, S. Haxha, V. Velisavljevic, and J. C. J. Fernández, "Light field geometry of a standard plenoptic camera," *Opt Express*, vol. 22, pp. 26659–73, Nov 2014.

[14] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, and M. Stamminger, "Real-time non-rigid reconstruction using an rgb-d camera," *ACM Transactions on Graphics*, 2014.

[15] S. Baker, T. Sim, and T. Kanade, "A characterization of inherent stereo ambiguities," *8th International Conference on Computer Vision*, July 2001.

[16] J. Shade, S. Gortler, L. wei He, and R. Szeliskiz, "Layered depth images," *Computer Graphics (SIGGRAPH'98) Proceedings*, July 1998.

[17] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer, "A survey of methods for volumetric scene reconstruction from photographs," *International Workshop on Volume Graphics*, June 2001.

[18] M. Potmesil, "Generating octree models of 3d objects from their silhouettes in a sequence of images," *Computer Vision, Graphics, and Image Processing*, vol. 40, pp. 1–29, 1987.

[19] C.-F. Chang, G. Bishop, and A. Lastra, "Ldi tree: A hierarchical representation for image-based rendering," *SIGGRAPH*, 1999.

[20] J.-Y. Guillemaut and A. Hilton, "Joint multi-layer segmentation and reconstruction for free-viewpoint video applications," *International Journal of Computer Vision*, 2011.

[21] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, pp. 7–42, April 2002.

[22] Y. Furukawa, "Multi-view stereo: A tutorial," *Foundations and Trends® in Computer Graphics and Vision*, vol. 9, no. 1-2, pp. 1–148, 2015.

[23] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," *CVPR*, vol. 1, pp. 519–526, 2006.

[24] R. C. Bolles, H. H. Baker, and D. H. Marimont, "Epipolar-plane image analysis: An approach to determining structure from motion," *International Journal of Computer Vision*, vol. 1, 1987.

[25] I. Feldmann, P. Eisert, and P. Kauff, "Extension of epipolar image analysis to circular camera movements," *Proc. International Conference on Image Processing*, pp. 697–700, 2003.

[26] I. Feldmann, P. Eisert, and P. Kauff, "Towards arbitrary camera movements for image cube trajectory analysis," *Proc. International Conference on Image Processing*, 2005.

[27] P. Goorts, S. Maesen, Y. Liu, M. Dumont, P. Bekaert, and G. Lafruit, "Self-calibration of large scale camera networks," 2014.

[28] L. Jorissen, P. Goorts, S. Rogmans, G. Lafruit, and P. Bekaert, "Multi-camera epipolar plane image feature detection for robust view synthesis," tech. rep., 2015.

[29] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multi-view stereopsis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 1362–1376, August 2010.

[30] A. Smolic, "3d video and free viewpoint video - from capture to display," *Pattern Recognition*, vol. 44, pp. 1958–1968, 2011.

[31] S. Hadfield and R. Bowden, "Exploiting high level scene cues in stereo reconstruction," *ICCV*, pp. 783–791, 2015.

[32] Z. Moratto, "Semi-global matching," September 2013.

[33] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, February 2008.

[34] P. Viola and W. M. W. III, "Alignment by maximization of mutual information," *International Journal of Computer Vision*, vol. 24, no. 2, 1997.

[35] C. L. Zitnick and T. Kanade, "A cooperative algorithm for stereo matching and occlusion detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, July 2000.

[36] A. L. Yuille and T. Poggio, "A generalized ordering constrains for stereo correspondence," A.I. Memo 777, AI Lab, MIT, 1984.

[37] C. Verleysen, *3D estimation and view synthesis in wide-baseline stereo.* PhD thesis, Université Catholique de Louvain, École Polytechnique de Louvain, ICTEAM Institute, November 2015.

[38] G. Pajares, P. J. Herrera, and J. M. de la Cruz, "Combining stereovision matching constraints for solving the correspondence problem," *Advances in Theory and Applications of Stereo Vision*, 2011.

[39] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *PAMI*, vol. 23, no. 11, pp. 1222–1239, 2001.

[40] H. Hirschmüller, "Semi-global matching - motivation, developments and applications," *Photogrammetric Week*, 2011.

[41] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan, "Image-based visual hulls," in *SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000.

[42] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, February 1994.

[43] R. Szeliski, "Rapid octree construction from image sequences," *CVGIP: Image Understanding*, vol. 58, pp. 23–32, July 1993.

[44] T. Fromherz and M. Bichsel, "Shape from multiple cues: Integrating local brightness information," *Fourth International Conference for Young Computer Scientist, ICYCS*, pp. 855–862, 1995.

[45] G. Slabaugh, R. Schafer, and M. Hans, "Image-based photo hulls," in *First International Symposium on 3D Data Processing Visualization and Transmission, 2002. Proceedings*, 2002.

[46] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *Computer Vision and Pattern Recognition Conference*, 1997.

[47] C. Godard, P. Hedman, W. Li, and G. J. Brostow, "Multi-view reconstruction of highly specular surfaces in uncontrolled environments," *International Conference on 3D Vision*, 2015.

[48] T. Yasuno and T. Hamano, "Three-dimensional reconstruction using homocentric spherical spatiotemporal image analysis," *Journal of Visual Communication and Image Representation*, vol. 2, pp. 365–372, December 1991.

[49] S. Wanner and B. Goldluecke, "Globally consistent depth labeling of 4d light fields," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[50] C. Kim, H. Zimmer, Y. Pritch, A. Sorkine-Hornung, and M. Gross, "Scene reconstruction from high spatio-angular resolution light fields," *ACM Transactions on Graphics, Proceedings of Siggraph*, vol. 32, no. 4, 2013.

[51] P. Goorts, *Real-time, Adaptive Plane Sweeping for Free Viewpoint Navigation in Soccer Scenes*. PhD thesis, Hasselt University, Transnationale Universiteit Limburg, 2014.

[52] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," *Proceedings of SIGGRAPH '95*, August 1995.

[53] M. Tanimoto, T. Fujii, and K. Suzuki, "Multi-view depth map of rena and akko kayo," tech. rep., MPEG, October 2007.

[54] "Report on experimental framework for 3d video coding," tech. rep., MPEG, October 2010.

[55] H.-Y. Shum and S. B. Kang, "A review of image-based rendering techniques," tech. rep., Microsoft Research, 2000.

[56] M. Zwicker, M. H. Gross, and H. Pfister, "A survey and classification of real time rendering methods," tech. rep., Eidgenössische Technische Hochschule Zürich, 1999.

[57] S. B. Kang, R. Szeliski, and P. Anandan, "The geometry-image representation tradeoff for rendering," *International Conference on Image Processing*, September 2000.

[58] A. Isaksen, L. McMillan, and S. J. Gortler, "Dynamically reparameterized light fields," *SIGGRAPH '00*, 2000.

[59] R. Yang, G. Welch, and G. Bishop, "Real-time consensus-based scene reconstruction using commodity graphics hardware," 2002.

[60] S. E. Chen and L. Williams, "View interpolation for image synthesis," *SIGGRAPH'93 Proceedings*, July 1993.

[61] C. Lipski, F. Klose, and M. Magnor, "Correspondence and depth-image based rendering," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 2014.

[62] M. Tanimoto, T. Fujii, and K. Suzuki, "View synthesis algorithm in view synthesis reference software 2.0 (vsrs2.0)," February 2009.

[63] T. Stich, C. Linz, C. Wallraven, D. Cunningham, and M. Magnor, "Perception-motivated interpolation of image sequences," *ACM Transactions on Applied Perception*, vol. 8, pp. 1–25, February 2011.

[64] A. K. Wegner, O. Stankiewicz, M. Tanimoto, and M. Domański, "Enhanced view synthesis reference software (vsrs) for free-viewpoint television," tech. rep., MPEG, October 2013.

[65] M. Tanimoto, T. Fujii, and K. Suzuki, "View synthesis method without blending," tech. rep., MPEG, February 2009.

[66] K. Wegner, O. Stankiewicz, and M. Domański, "Depth based view blending in view synthesis reference software (vsrs)," tech. rep., ISO MPEG, October 2015.