Advanced Graphics and Image Processing - Lecture notes

Rafał Mantiuk

Lent term 2018/19

1 Light field rendering using homographic transformation

This section explains how to render a light field for a novel view position using a parametrization with a focal plane. The method is explained on a rather high level in [1]. These notes are meant to provide a practical guide on how to perform the required calculations and in particular how to find a homographic transformation between the virtual and array cameras.

The scenario and selected symbols are illustrated in Figure 1. We want to render our light field "as seen" by camera K. We have N images captured by N cameras in the array (only 4 shown in the figure), all of which have their apertures on the camera array plane C. We further assume that our array cameras are pin-hole cameras to simplify the explanation. The novel view is rendered assuming focal plane F. The focal plane has a similar function as the focus distance in a regular camera: objects on the focal plane will be rendered sharp, while objects that and in front or behind that plane will appear blurry (in practice they will appear ghosted because of the limited number of cameras). The focal plane F does not need to be parallel to the camera plane; it can be titled, unlike in a traditional camera with a regular lens. Because we have a limited number of cameras, we need to use reconstruction functions A_0, ..., A_1 (only two shown) for each camera. The functions shown contain the weights in the range 0-1 that are used to interpolate between two neighboring views.

To intuitively understand how light field rendering is performed, imagine the following hypothetical scenario. Each camera in the array captures the

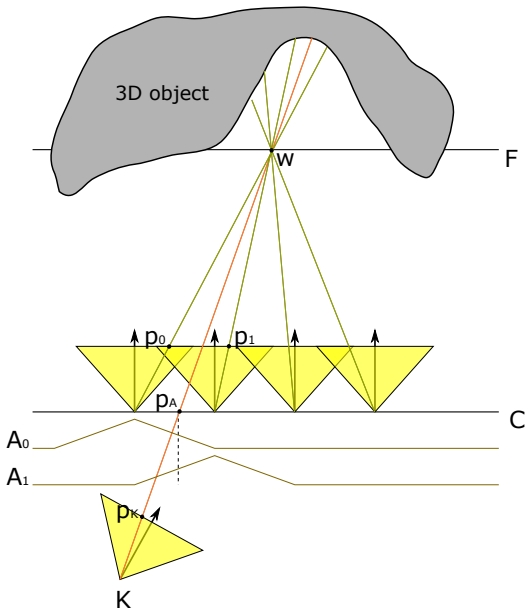

Figure 1: Light field rendering for the novel view represented by camera K. The pixels P_K in the rendered image is the weighted average of the pixels values p_1, ..., p_N from the images captured by the camera array.

image of the scene. Then, all objects in the scene are removed and you put a large projection screen where the focal plane F should be. Then, you swap all cameras for projectors, which project the captured images on the projection screen F. Finally, you put a new camera K at the desired location and capture the image of the projection screen. The projection screen (focal plane) is needed to form an image. Ideally, to obtain a sharp image, we would like to project the camera array images on a geometry. However, such a geometry is not readily available when capturing scenes with a camera array. In this situation a single plane is often a good-enough proxy, which has its analogy in physical cameras (focal distance). More advanced light field rendering methods attempt to reconstruct a more accurate proxy geometry using multi-view stereo algorithms and then project camera images on that geometry [3].

This simplified scenario misses one step, which is modulating each projected image by the reconstruction function A, as such modulation has no physical counterpart. However, this scenario should give you a good idea what operations need to be performed in order to render a light field from a

Data: Camera array images J_1, J_2, ..., J_N
Result: Rendered image I
for *each pixel at the coordinates* $\boldsymbol{p}_K$ *in the novel view* **do**
  $I(\boldsymbol{p}_K) \leftarrow 0$;
  $w(\boldsymbol{p}_K) \leftarrow 0$;
  for *each camera* i *in the array* **do**
    Find the coordinates $\boldsymbol{p}_i$ in the i-th camera image corresponding to the pixel $\boldsymbol{p}_K$;
    Find the coordinates $\boldsymbol{p}_A$ on the aperture plane A corresponding to the pixel $\boldsymbol{p}_K$;
    $I(\boldsymbol{p}_K) \leftarrow I(\boldsymbol{p}_K) + A(\boldsymbol{p}_A)\, J_i(\boldsymbol{p}_i)$;
    $W(\boldsymbol{p}_K) \leftarrow W(\boldsymbol{p}_K) + A(\boldsymbol{p}_A)$;
  end
  $I(\boldsymbol{p}_K) \leftarrow I(\boldsymbol{p}_K)/W(\boldsymbol{p}_K)$;
end

Algorithm 1: Light field rendering algorithm

novel view position.

Now let us see how we can turn such a high-level explanation into a practical algorithm. One way to render a light field is shown in Algorithm 1. The algorithm iterates over all pixels in the rendered image, then for each pixel it iterates over all cameras in the array. The resulting image is the weighted average of the camera images that are warped using homographic transformations. The weights are determined by the reconstruction functions A_i. The algorithm is straightforward, except for the mapping from pixel coordinates in the novel view p_K to coordinates in each camera image p_i and the coordinates on the aperture plane p_A. The following paragraphs explain how to find such transformations.

1.1 Homographic transformation between the virtual and array cameras

The text below assumes that you are familiar with homogeneous coordinates and geometric transformations, both commonly used in computer graphics and computer vision. If these topics are still unclear, refer to Section 2.1 in [4] (this book is available online) or Chapter 6 in [2].

We assume that we know the position and pose of each camera in the

array, so that homogeneous 3D coordinates of a point in the 3D word coordinate space w can be mapped to the 2D pixel coordinates p_i of camera i:

$$\boldsymbol{p}_i = \boldsymbol{K}\,\boldsymbol{P}\,\boldsymbol{V}_i\,\boldsymbol{w}\,. \tag{1}$$

where $\boldsymbol{V}$ is the view transformation, $\boldsymbol{P}$ is the projection matrix and $\boldsymbol{K}$ is the intrinsic camera matrix. Note that we will use bold lower case symbols to denote vectors, uppercase bold symbols for matrices and a regular font for scalars. The operation is easier to understand if the coordinates and matrices are expanded:

$$\begin{bmatrix} x_i \\ y_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} . \tag{2}$$

The view matrix $\boldsymbol{V}$ translates and rotates the 3D coordinates of the 3D point $\boldsymbol{w}$ so that the origin of the new coordinate system is at the camera centre, and camera's optical axis is aligned with the z-axis (as the view matrix in computer graphics). This matrix can be computed using a *LookAt* function, often available in graphics matrix libraries.

The projection matrix $\boldsymbol{P}$ may look like an odd version of an identity matrix, but it actually drops one dimension (projects from 3D to 2D) and copies the value of Z coordinate into the additional homogeneous coordinate w_i. Note that to compute Cartesian coordinates of the point from the homogeneous coordinates, we divide x_i/w_i and y_i/w_i. As w_i is now equal to the depth in the camera coordinates, this operation is equivalent to a perspective projection (you can refer to slides 88–92 in the Introduction to Graphics Course).

The intrinsic camera matrix $\boldsymbol{K}$ maps the projected 3D coordinates into pixel coordinates. f_x and f_y are focal lengths and c_x and c_y are the coordinates of optical center expressed in pixel coordinates. We assume that the intrinsic matrix is the same for all the cameras in the array.

Besides having all matrices for all cameras in the array, we also have a similar transformation for our virtual camera K, which represents the currently rendered view:

$$\boldsymbol{p}_K = \boldsymbol{K}_K\,\boldsymbol{P}\,\boldsymbol{V}_K\,\boldsymbol{w}\,. \tag{3}$$

Our first task is to find transformation matrices that could transform from pixel coordinates $\boldsymbol{p}_K$ in the virtual camera image into pixel coordinates $\boldsymbol{p}_i$

for each camera i. This is normally achieved by inverting the transformation matrix for the novel view and combining it with the camera array transformation. However, the problem is that the product of $\boldsymbol{K}_K \boldsymbol{P} \boldsymbol{V}_K$ is not a square matrix that can be inverted — it is missing one dimension. The dimension is missing because we are projecting from 3D to 2D and one dimension (depth) is lost.

Therefore, to map both coordinates, we need to reintroduce missing information. This is achieved by assuming that the 3D point lies on the focal plane F. Note that the plane equation can be expressed as $\boldsymbol{N} \cdot (\boldsymbol{w} - \boldsymbol{w}_F) = 0$, where $\boldsymbol{N}$ is the plane normal, and $\boldsymbol{w}_F$ specifies the position of the plane in the 3D space. Operator $\cdot$ is the dot product. If the homogeneous coordinates of the point $\boldsymbol{w}$ are $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$, the plane equation can be expressed as

$$d = \begin{bmatrix} n_x & n_y & n_z & -\boldsymbol{N} \cdot \boldsymbol{w}_F \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{4}$$

where d is the distance to the plane and $\boldsymbol{N} = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}$. We can introduce the plane equation into the projection matrix from Equation 2:

$$\begin{bmatrix} x_i \\ y_i \\ d_i \\ w_i \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & c_x \\ 0 & f_y & 0 & c_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ n_x^{(c)} & n_y^{(c)} & n_z^{(c)} & -\boldsymbol{N}^{(\boldsymbol{c})} \cdot \boldsymbol{w}_F^{(c)} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{5}$$

The product of the matrices in above is a full 4×4 transformation matrix, which is not rank-deficient and can be inverted. Note that the pixel coordinates $\boldsymbol{p}_K$ and $\boldsymbol{p}_i$ now have an extra dimension d, which should be set to 0 (because we constrain 3D point w to lie on the plane).

It should be noted that the normal and the point in the plane equation have superscript $^{(c)}$, which denotes that the plane is given in the *camera* coordinate system, rather than in the world coordinate system. This is because the point $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$ is transformed from the world to the camera coordinates by the view matrix $\boldsymbol{V_i}$ before it is multiplied by our modified projection matrix. This could be a desired behavior for the virtual camera, for example if we want the focal plane to follow the camera and be perpendicular to the camera's optical axis. But, if we simply want to specify the focal plane in the

world coordinates, we have two options: either replace the third row in the final matrix (obtained after multiplying the three matrices in Equation 5) with our plane equation in the world coordinate system; or to transform the plane to the camera coordinates:

$$\boldsymbol{w}_F^{(c)} = \boldsymbol{V}_i \boldsymbol{w}_F \tag{6}$$

and

$$\boldsymbol{N}^{(c)} = \overline{\boldsymbol{V}}_i \boldsymbol{N} \,. \tag{7}$$

$\overline{\boldsymbol{V}}_i$ is the "normal" or direction transformation for the view matrix $\boldsymbol{V}_i$, which rotates the normal vector but it does not translate it. It is obtained by setting to zero the translation coefficients w_{14}, w_{24}, w_{34}.

Now let us find the final mapping from the virtual camera coordinates $\hat{\boldsymbol{p}_K}$ to the array camera coordinates $\hat{\boldsymbol{p}_i}$. We will denote the extended coordinates (with extra d) in Equation 5 as $\hat{\boldsymbol{p}_K}$ and $\hat{\boldsymbol{p}_i}$. We will also denote our new projection and intrinsic matrices in Equation 5 as $\hat{\boldsymbol{P}}$ and $\hat{\boldsymbol{K}}$. Given that, the mapping from $\boldsymbol{p}_K$ to $\boldsymbol{p}_i$ can be expressed as:

$$\hat{\boldsymbol{p}_i} = \hat{\boldsymbol{K}}_i \hat{\boldsymbol{P}} \boldsymbol{V}_i \boldsymbol{V}_K^{-1} \hat{\boldsymbol{P}}^{-1} \hat{\boldsymbol{K}}_K^{-1} \hat{\boldsymbol{p}_K}. \tag{8}$$

The position on the aperture plane $\boldsymbol{w}_A$ can be readily found from:

$$\boldsymbol{w}_A = \boldsymbol{V}_i^{-1} \hat{\boldsymbol{P}}_A^{-1} \hat{\boldsymbol{K}}_i^{-1} \hat{\boldsymbol{p}_K} \,, \tag{9}$$

where the projection matrix $\hat{\boldsymbol{P}}_A$ is modified to include the plane equation of the aperture plane, the same way as done in Equation 5.

1.2 Reconstruction functions

The choice of the reconstruction function A_i will determine how images from different cameras are mixed together. The functions shown in Figure 1 will perform bilinear-interpolation between the views. Although this could be a rational choice, it will result in ghosting for the parts of the scene that are further away from the focal plane F. Another choice is to simulate a wide-aperture camera and include all cameras in the generated view (set $A_i = 1$). This will produce an image with a very shallow depth of field. Another possibility is to use the nearest-neighbor strategy and a box-shaped reconstruction filter (the width of the boxes being equal to the distance between the cameras). This approach will avoid ghosting but will cause the views

to jump sharply as the virtual camera moves over the scene. It is worth experimenting with different reconstruction startegies to choose the best for a given application but also for the given angular resolution of the light field (number of views).

References

[1] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *Proc of SIGGRAPH '00*, volume 7, pages 297–306, New York, New York, USA, 2000. ACM Press.

[2] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics*. A K Peters/CRC Press, 4 edition edition, 2016.

[3] Ryan S. Overbeck, Daniel Erickson, Daniel Evangelakos, Matt Pharr, and Paul Debevec. A system for acquiring, processing, and rendering panoramic light field stills for virtual reality. *ACM Transactions on Graphics*, 37(6):1–15, dec 2018.

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York Inc, 2010.