Implement an algorithm for rendering a virtual view

You shall implement a Depth Image-Based Rendering (DIBR) algorithm in Matlab or C/C++. Your algorithm shall generate a new (virtual) 2D image that depicts the scene from a perspective different from the original image, when the program is provided with a 2D original image (V) and a corresponding depth map (D). The generated image shall exhibit a natural perspective change, i.e. the new view shall present scene objects that have been translated with different amounts depending on how far away they are positioned from the camera.

A 3D point in the scene, described by homogenous coordinates **M** = $(x,y,z,1)^T$, is projected onto the cameras image plane at homogenous coordinate **m** = $(fx/z, fy/z, 1)^T$, where f is the focal length of the camera. This perspective projection can be expressed as the following matrix multiplication:

$$\begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

This equation can be simplified into:

$$z\boldsymbol{m} = P\boldsymbol{M}$$

where *P* is called the camera projection matrix. The above projection matrix is the simplest case as it only contains the camera's focal length *f*. In the general case P is a 3x4 full rank homogenous matrix, having 11 degrees of freedom. Often P is factorized into three matrices:

$$P = K[R|\boldsymbol{t}]$$

where K is an upper triangular matrix called camera calibration matrix, *R* is a rotation matrix describing the orientation of the camera, and $\boldsymbol{t}$ is a translation vector describing the camera center relative to a world coordinate systems common for all cameras. The camera calibration matrix encodes the transformation from camera coordinates to pixel coordinates within the image plane, also known as the intrinsic parameters of the camera:

$$K = \begin{bmatrix} f/s_x & 0 & o_x \\ 0 & f/s_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

where *f* is the focal length, s_x and s_y are the pixel width and height, respectively, and $(o_x, o_y)^T$ are the image center coordinates in pixels.

A 3D point **M** in the scene captured by two cameras (original and virtual) produce corresponding points $\boldsymbol{m}_o$ and $\boldsymbol{m}_v$ in the two camera's respective image plane. Combined these translate into

$$\zeta_o \boldsymbol{m}_o = P_o \boldsymbol{M}$$
$$\zeta_v \boldsymbol{m}_v = P_v \boldsymbol{M}$$

By solving for $\boldsymbol{m}_v$ one can derive at what position in the virtual image color from a specific pixel in the original image should be transferred.

A more detailed description of the multiple view geometry concepts that shall be used in this task can be found in the attached excerpt from the book *3D VIDEO COMMUNICATION – Algorithms, concepts and real-time systems in human centered communication, Edited by O. Schreer, P. Kauff, and T. Sikora, John Wiley and Sons, Ltd.*

Given

Camera metadata - includes extrinsic $[R|\mathbf{t}]$ and intrinsic matrices K, and depth map normalization factors Z_{near} and Z_{far}. Z_{near} and Z_{far} are needed to transform the dimensionless depth map D integer-based value ranges into the floating point geometrical distance map Z. Z_{near} corresponds to the brightest gray scale value of the depth map and Z_{far} the darkest.

Image V – contains color, also known as texture, information.

Depth map D – contains the normalized depth information.

Deliverable

The task has a set of deliverables defined below in order of priority. The more deliverables you finish in the defined time, the better.

1. Make a program (Matlab function or C/C++ source code with compiled binary) that takes as input arguments the original 2D image V, the original depth map D, and the original and virtual cameras' intrinsic and extrinsic matrices**.** As output it shall produce a synthesized 2D virtual image located at the positioned defined by the two cameras' projection matrices.

   ```
   function [ V_v] = DIBR( V_o, D_o, K_o, Rt_o, K_v, Rt_v)
   ```

2. Make a script that synthesizes an arbitrarily number of N virtual views using the above program, with the views corresponding to camera positions evenly spread out on the line connecting the original and virtual camera centers respectively. Evaluate the script by setting N to 5.

3. As you will notice, a synthesized view will exhibit different types of artifacts. *Cracks* may occur in the virtual image due to rounding of pixel positions $\boldsymbol{m}_v$ to integer pixel positions. Empty areas, so called *Disocclusions*, appear due to parts of the scene being revealed when seen from the virtual camera view point. (These areas have no texture information present in the original camera image.) Try to reduce these artifacts by filling these empty pixels employing whatever method you find suitable.

The solution shall include

- a well-structured and commented source code
- an executable version of the program (if Matlab is used source code and executable version is the same thing)
- the resulting images in PNG format
- a short readme text file that includes a description of how the program shall be used[1], clear presentation of any assumptions that you have made, elaboration on any design choices you have felt obliged to make in order to solve the task.

[1] Note that this should really not be necessary. Abiding by the above function definition should make running the solution be self-explanatory.