Computational Photography
Project 6, Fall 2014

27. November 2014

This project is about light fields, ray space analysis and reconstruction filters. Upload your solution on ILIAS. The deadline is December 11, 14:00.

Ray Space Analysis

The goal of this assignment is to open a light field and to analyze the properties of the scene in ray and in Fourier space.

1. You can get light field data from `http://graphics.ucsd.edu/datasets/lfarchive/lfs.shtml`. These are 3D light fields where the cameras are distributed along a straight line. Open one of the image sequences with more than one hundred views and find a good way to show the sequence. The following matlab code shows how to open an image sequence out of a folder.

```
% find the folder
folder = uigetdir();
if ~(folder==0)
    % get the names of all image files
    searchFor = strcat(folder, '/*.png');
    dirListing = dir(searchFor);
    % get number of images
    numFrames = length(dirListing);
    % open images and store it in array

    % first image
    % use full path because the folder may not be the active path
    fileName = fullfile(folder,dirListing(1).name);
 end
```

2. (**1 point**) Create the EPIs for at least two scanlines (i.e., horizontal rows). Try to use no for loops (hint: use the Matlab function *permute*). Show the EPIs and the position of the scanlines in the image.

   Analyze two or more EPIs. What do you observe, what do the different parts of the image show?

3. (**1 point**) Calculate the Fourier transform of two EPIs and visualize the power spectrum (according to Project 3). What can be observed on the images? How can the images be interpreted?

Turn-in: Runable Matlab code organized in cells, a pdf file containing example images and your observations.

View Reconstruction

In this assignment you will render new camera views using different approaches. The goal is to compare the different methods and to analyze their advantages and disadvantages.

1. (**2 points**) **Linear interpolation:** Render a new camera view by linearly interpolating two existing views. Implement the interpolation in ray space. You can do this as follows: To interpolate between the cameras i and $i+1$ choose the corresponding rows in the EPIs and interpolate them. Then reassemble the new view.

   Perform this experiment by interpolating between camera views i and $i+k$ for different k. Analyze your result: What do you observe? What happens when k increases, i.e., you interpolate between views that are further apart? What are the disadvantages of this approach?

2. (**2 points**) **Sheared interpolation:** The result can be improved when the object(s) at a certain depth are aligned with the reconstruction filter. The existing camera views are translated such that the desired objects are in the same place in ray space, i.e., they form vertical lines in the EPIs. The effect is that we virtually focus on the objects at that depth. Translating each view such that desired objects align corresponds to a shear in ray space, which is a shear of the EPIs. Equivalently, this can also be interpreted as a shear of the reconstruction filter.

   - Analyze the EPIs to define the shearing angle. The slanted lines of the object(s) you want to have in focus need to be vertical after the shear.
   - Shear the EPIs.
   - Render interpolated camera views as in the assignment above. Use different values of k again. Because of the shear you may have to remove the black border after reassembling the new view.
   - Try different shearing angles and analyze the results. What are the improvements of this approach? What are the drawbacks?

3. (**2 points**) **Interpolation after filtering with Gauss filter:** Apply a 2-dimensional elliptical Gaussian filter to the EPI before rendering the new view. You can implement the elliptical Gaussian as a *separable* filter. This means that you apply a 1D filter in one direction first, then you filter the intermediate result in the other direction again with a 1D Gaussian. You can select the standard deviation in both directions independently to obtain an elliptical filter.

Render the new views again with different shearing angles and for different k. What is the effect when using bigger or smaller Gauss kernels? What are the effects of the vertical and horizontal extent of the elliptical Gaussian? Use at least two different kernels.

Turn-in: Runable Matlab code organized in cells, a pdf file containing example images and your observations.

Autostereoscopic images

The goal of this assignment is to produce autostereoscopic images, i.e., images that can be viewed in 3D without wearing glasses. They are implemented using view dependent pixels that show a different color based on the viewing angle. Since both eyes look from a slightly different angle onto each pixel, both eyes may see a different image, which leads to 3D depth perception.

View dependent pixels can be constructed, for example, using lenticular sheets. The idea is simply to attach a printed image to the back of the lenticular sheet. Each pixel in the printed image produces a light ray in a certain direction, where the ray has the color of the printed pixel. Hence, we obtain a "light field display". The number of pixels behind each lens gives the number of angular samples, or the angular resolution. The number of lenses gives the spatial resolution. In this assignment, we work with pixels that have a horizontal view dependency only. This means that the lenticular sheets consist of vertical parallel stripes of small lenses (as opposed to "fly's eye" lens arrays).

Your goal is to produce an autostereoscopic output from a sequence of input images. The problem here is to *resample* the light field appropriately: we want to reproduce the light field represented by pixels of the input images, but sampled on a new grid, i.e., the pixel grid of the printed image placed behind the lenticular sheet. To do this, we need to define a mapping that relates rays from the input images (i.e., rays through image pixels) to rays of the lenticular sheet (i.e., rays defined by printed pixels).

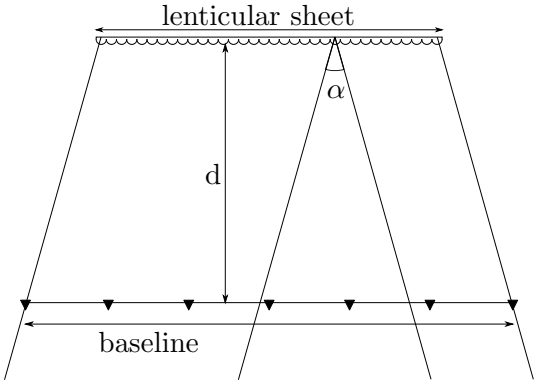
We define this mapping by specifying the geometry of the lenticular sheet and the input images relative to each other. This is shown in the figure below. We assume the cameras are placed on a horizontal line parallel to the lenticular sheet. Note that because we use lenticular sheets consisting of vertical stripes of lenses, we can work on each horizontal scanline independently, simplifying the geometry to 2D. The geometry is given by the following parameters:

- The width of the lenticular sheet.
- The number of lenses (i.e., the number of vertical stripes, or the spatial resolution of the lenticular sheet).
- The opening angle of the bundle of rays corresponding to one lenticular stripe.
- The number of printed pixels per lens (i.e., the angular resolution of the lenticular sheet).
- The distance between the lenticular sheet and the camera array. Note that this distance should be chosen roughly as the viewing distance of an observer in front of the lenticular sheet.

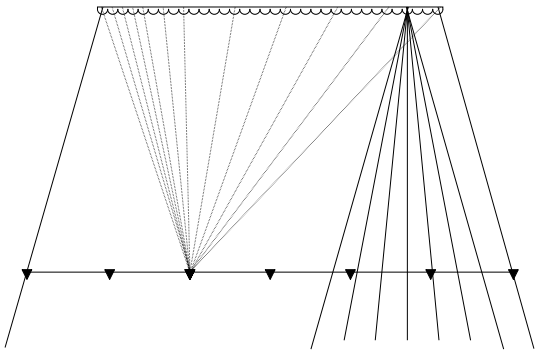
- The number of cameras.
- The camera baseline, i.e., the width spanned by the cameras.

You may assume that the horizontal resolution of the camera images is the same as the number of lenses (if this is not the case with your images, simply downsample them appropriately). Also assume that there is a one-to-one correspondence of rays through camera pixels and lenses as shown on the right in the figure.

To perform resampling, you need to represent the camera rays and the rays of the lenticular in the same coordinate system. We propose that you use a two plane parameterization (more precisely, a two line parameterization in our 2D case) where the first plane is the camera plane and the second plane goes through the centers of the lenses. The coordinates of the camera rays in this parameterization are trivial to determine. To perform resampling and determine the colors of the lenticular rays, you need to compute the coordinates of these lenticular rays and then interpolate their colors from the camera rays, for example using bilinear interpolation.



The triangles on the baseline are the cameras.



Some of the light rays from a camera (dotted) resp. a lenticular lens.

1. **(2 point)** Write Matlab code that calculates the coordinates of all the camera and lenticular rays in the two plane parameterization. Your code should take the parameters of the geometry (see above) as an input to determine the mapping of lenticular rays to the common coordinate system. Plot a 2D ray space diagram indicating the position of the camera and lenticular rays. Show plots for different parameter settings. To produce a reasonable output, you should make sure that the camera and lenticular rays overlap as much as possible (although there will always be some camera rays that go "unused").

Bonus

1. **(2 point)** Write Matlab code to interpolate the camera rays and determine the color of the lenticular rays. You may use bilinear interpolation. Create output images using the example data we provide and print them.

   We provide a lenticular sheet in the lab where you can test your results. The lenticular sheet has 20.196 vertical lens stripes per inch (or 7.951 lenses per centimeter), 220 lenses

in total and a viewing angle of 29 degrees. We recommend to use about ten samples for the angular resolution of your output, but you should try out several different numbers.

For example, if you use a spatial resolution of 200 lenses and an angular resolution of 10 samples per lens, your output image will have a 2000 pixels horizontally. Using our lenticular sheet, 200 lenses cover a width of 25.154 centimeters. We recommend to print your image using Photoshop or a similar tool to make sure it has the appropriate physical size.

Turn-in: Matlab code, the images you produced, and the information needed to print them (mainly the resolution).

2. **(1 point)** Create your own image set to create a light field. Capture image data of your own static scene and construct a light field of it. To acquire the data, put your camera on a tripod. Fix the focus, the aperture size and the shutter speed. Then take a series of pictures while moving the camera parallel to the scene after each shot. Make sure not to rotate the camera and to move the camera by the same distance after each shot. Once the images are taken, you need to shear and crop your image stack such that the objects you want to have in the focal plane appear at the same horizontal pixel location on each image. To find out which part you have to crop, create the EPI of a convenient scanline. Shear it until the objects in the focal plane form vertical lines in the EPI.

3. **(1 point)** The goal of this part is to use camera calibration and image rectification to improve the accuracy of your light field. The idea of camera calibration is to recover the 3D position and orientation of the cameras based on the captured images themselves. You can use this information to compensate for small errors in the physical placement of the cameras.

   We recommend to use the camera calibration toolbox for Matlab, which you can download at `http://www.vision.caltech.edu/bouguetj/calib_doc/`. Study the example to learn how to use it. To support camera calibration, you will need to include a printed checkerboard mounted on a flat surface in your scene. Add this to the scene such that it is visible in each shot. Using the camera calibration toolbox you can find the intrinsic and extrinsic camera parameters for each of your images.

   After obtaining the calibration data, you will use it to rectify the captured images. Rectifying means projecting the images onto a common plane, i.e., in our case the plane of the lenticular sheet. This is implemented using a suitable homography of each image that you can compute based on the calibration data. After rectification, you can perform resampling as above.

   To get good results, you should still avoid vertical camera movements and place the cameras equidistantly. Small rotations of the camera, however, should be no problem because they are compensated by the rectification.

   Demonstrate a result with and without calibration and rectification. Deliberately rotate the cameras by a small amount to show the benefits of calibration.