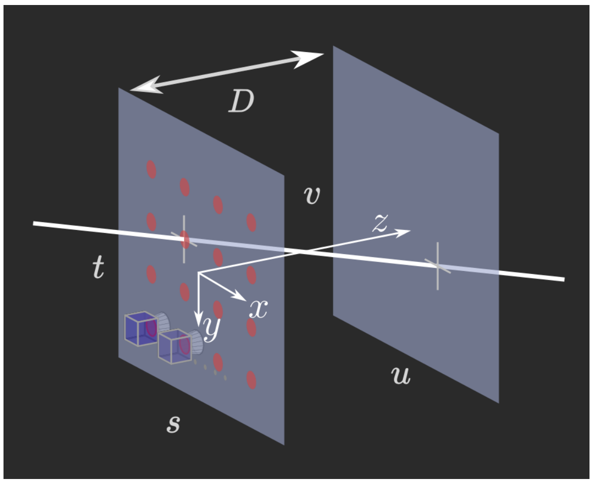
Exercise 1b: Interpolating Renderer



In this exercise you will write a simple interpolating light field renderer.

The input is a 4D sampled light field L(*i,j,k,l)* and an intrinsic matrix H. The output is a 2D perspective (pinhole camera) image showing the scene from an arbitrary camera pose.

Input

Download the sample light field from http://http://dgd.vision/LF2018/Lego.256.mat . This is an adapted version of the light field taken from http://lightfield.stanford.edu . It has been downsampled and reorganized in i,j to match the axis alignment depicted above.

Load the light field into matlab and inspect the variables contained within.

LF is a sampled light field indexed in the order [*j,i,l,k*], with each index corresponding to the [*t,s,v,u*] dimensions, respectively. The light field also has three colour channels. For this exercise you may choose to use only one colour channel, e.g. by taking the green colour channel using `LF = squeeze(LF(:,:,:,:, 2));`

H is a plenoptic intrinsic matrix describing the mapping from sampled indices ***n***=[*i,j,k,l*] to rays ***Φ***=[*s,t,u,v*], as in ***Φ*** = H***n****.*

- H is in the absolute two-plane parameterization
- Indices start at 1

D is the plane separation used by H.

Inspect the light field, check its size and visualize it in slices. Inspect D, H, and H inverse. From H and the light field's size you can deduce the scales of the s,t and u,v planes *(3 and 16, respectively)*.

Renderer Overview

Write a 2D renderer following the rough procedure below. Write the code from scratch, or refer to the more detailed procedure in the following section.

Camera: choose the pose and parameters of the virtual 2D pinhole camera: position, rotation, field of view (FOV), and pixel count.

Rays in 3D: describe the 3D ray corresponding to each pixel of the 2D camera. The description can be in any 3D line parameterization.

Rays in 4D: convert your 3D rays to continuous light field coordinates. Do this by intersecting each ray with each of the s,t and u,v planes.

Rays to indices: convert your 4D rays to indices using the plenoptic intrinsic matrix H. These are “continuous” indices in that they are not whole numbers, but fractional.

Interpolate: use the continuous indices to interpolate pixel values from the light field.

Display: that’s it!

Renderer In Detail

Camera

A 3D position vector and 3D rotation matrix are reasonable choices. Start at the origin looking down the z axis. Also choose an output resolution, start small so that testing is fast, e.g. 128x128 pixels. Finally, decide on the field of view of your camera. ~60 degrees is a good start.

Rays in 3D: point + direction

Describe the ray corresponding to each pixel of your camera. It is probably easiest to do this using a 3D point and direction for each ray. Since each ray passes through your camera’s pinhole, the position is easy, it’s the camera’s 3D position vector. For direction you can start with an axis-aligned camera, then rotate each vector by the camera’s rotation.

If you’re stuck: build up a matrix of ray directions by fixing the z coordinate and varying the x and y coordinates over a grid. For a 128x128 output image, this will be a 3 x (128x128) = 3 x 16384 matrix. This is a single long list of ray directions, which you can rotate using a rotation matrix. See the minimal example below.

Rays in 4D: 2pp

Now convert your point + direction rays to continuous-domain light field coordinates. Do this by finding the point of intersection of each ray with the s,t and u,v planes. Assume s,t is centered at the origin, and u,v is at distance D along the z axis, as in the reference image above. The intersections are easy to find, fix z and solve for x and y, for each of the two planes. Stack the s,t,u,v coordinates into a matrix. For a 128x128 output, this will be a 4 x (128x128) matrix.

Rays to Indices

The plenoptic intrinsic matrix makes this easy, but get the order right. $\boldsymbol{\Phi}$ = H$\boldsymbol{n}$ but we want to go from rays to indices, so use $\boldsymbol{n}$ = $H^{-1}\boldsymbol{\Phi}$. Remember H is in homogeneous coordinates, so you will need to stack a 1 at the end of each of your ray matrices.

Interpolate

Matlab's `interpn` function will come in handy here. Watch out for the ordering of parameters, though. Remember the light field is indexed in the order [*j,i,l,k*] corresponding to the ray coordinates [*t,s,v,u*].

Display

Interpn doesn't know the rays you've asked to interpolate correspond to an image. Reorganize the rays into an image using `reshape` to yield a 128x128-pixel image.

Questions

1. What determines the limits on the range of motion of the virtual camera?
2. What happens if you change the interpolation to nearest-neighbour?
3. How could you replace the virtual 2D camera with a light field camera?
4. You were given an approximate plenoptic intrinsic matrix H which is not metric or calibrated. What happens to the rendered output if you change the components of H, and D? e.g. if D is halved, or H_{11} is doubled, and so on?
5. What physical measurements would you need to derive the plenoptic intrinsic matrix H correctly and to scale? *Hint: consider starting with H^{-1}. How does s vary with i?*
6. Try turning up the resolution (pixel count) on your virtual camera. What limits the maximum effective resolution? Is it higher or lower than the resolution of the light field slices in k,l, i.e. what's displayed by LFDispMousePan?

Minimal Example

```
%---Load the light field---
load('Lego.256.mat', 'LF', 'H', 'D', 'about');
LF = squeeze(LF(:,:,:,:,2));  % monochrome demo
LF = single(LF) ./ 255; % interpn wants floats

%---Set camera position and rotation---
OutRes = 512;
CamPos = [0,0,0];
CamRot = eye(3);

%---Build a set of rays for the rendered image---
RayVec = linspace(-0.6,0.6, OutRes);
[RayY, RayX] = ndgrid(RayVec, RayVec);
RayDir = [RayX(:), RayY(:), ones(size(RayX(:)))]';
RayDir = CamRot*RayDir;
RayDir = RayDir(1:2,:) ./ RayDir(3,:);

%---Express the rays as s,t,u,v coordinates---
STDist = 0 - CamPos(3);
UVDist = STDist + D;
ST = CamPos(1:2)' + STDist .* RayDir;
UV = CamPos(1:2)' + UVDist .* RayDir;
RayVec = [ST;UV; ones(1,size(ST,2))];

%---Convert from ray to index using the plenoptic intrinsic matrix---
NCont = H^-1 * RayVec;

%---Interpolate from the light field---
OutFrame(:) = interpn(LF, NCont(2,:),NCont(1,:),NCont(4,:),NCont(3,:));
OutFrame = reshape(OutFrame, [OutRes,OutRes]);

%---Display---
imshow(OutFrame);
```