# Assignment 1 Part 1: Traditional Light Field Rendering

NAME: CHENQIHE
STUDENT NUMBER: 2020533088
EMAIL: CHENQH@SHANGHAITECH.EDU.CN

## 1 INTRODUCTION

In this assignment, we implement novel-view synthesis results based on traditional light field rendering.

There are more concrete problems that have been solved.

- Implement both bilinear and quadra-linear interpolation schemes to interpolate views along $x$ and $y$ directions.
- Implement focussing and defocussing by variable focal plane- and variable aperture size.
- Implement the z-directional motion of the camera by expanding field of view.

## 2 RELATED WORK

In this section, we review related work to provide context for our own work.

**Plen-optic Function** Traditional plen-optic functionis defined by 7D that $(x, y, z, \theta, \phi, \lambda, t)$. In this assignment, we assume that camera receives the light rays. $(x, y, z)$ is where light hits, i.e.3D coordinates of pixels on the camera's focal plane. $(\theta, \phi)$ is the direction of light ray, i.e. how light rays hit the focal plane. $(\lambda)$ is the wavelength of light ray, i.e. the information of illumination. $(t)$ is the time spot, i.e. describes how the radiance changes during the timespan.

**Light field** It describes a simple and robust method for generating new views from abitrary(but actually limited) camera positions without depth information or feature matching, simply by combing and resampling the available images.

The key to this technique lies in interpreting the input images as 2D slices of a 4D function. The solution it proposes is to parameterize lines by their intersections with two planes in arbitrary position. In other words, it simplifies the plen-optic function from 7D into 4D.

## 3 METHODS

### 3.1 Representation of rays: light slab

In traditional light field, we use a light slab to parameterize rays by $(s, t, u, v)$, where $(s, t)$ is the index of camera, $(u, v)$ is the index of pixel aof $(s, t)$ camera. Thus, we store all the images information or color of rays into a 4D matrix.

In light slab, an oriented line is defined by a directional line from a point on $uv$ plane to a point on $st$, where $st$ plane is camera plane, $uv$ plane is focal plane.

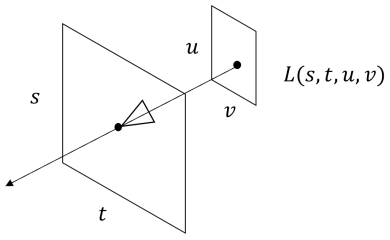



Fig. 1. light slab

In this way, we compress all the input images into a 4D expression $(s, t, u, v)$.

### 3.2 Rendering

*3.2.1 Disparity.* A point $(u, v)$ on focal plane can be exactly mapped back to a 3D point in world space. To render an image at the view of a virtual camera, any points on focal plane should be consistent in the existing cameras. Please notice that the locations should be consistent on 3D world space, but may not be on the 2D focal plane. The reason why 2D focal plane may not be consistent can be explained by below:

Consider a light slab $(s, t, u, v)$ and a 3D point $P$ $(x, y, z)$ lies inside the light slab. A problem is how to get all the light rays in light slab that pass throught $P(x, y, z)$.

Know conditions: 3D point P$(x, y, z)$, $st$ plane $\{(s, t, 0)|(s, t) \in [0, 1]^2\}$, $uv$ plane $\{(u, v, 1)|(u, v) \in [0, 1]^2\}$

Assume the target light ray intersects $st$ plane at point $(s, t, 0)$ and $uv$ plane at point $(u, v, 1)$, then the origin of light ray $\dot{o} = (u, v, 1)$, the direction of light ray (not normalized) $\vec{d} = (s - u, t - v, -1)$, i.e. $r(\lambda) = \dot{o} + \lambda\vec{d}$

$$r(\lambda) = (u, v, 1) + \lambda(s - u, t - v, -1)$$

since $P(x, y, z)$ is on the light ray, we have

$$\begin{cases} x &= (1 - \lambda)u + \lambda s \\ y &= (1 - \lambda)v + \lambda t \\ z &= 1 - \lambda \end{cases}$$

Using $\lambda = 1 - z$, we have

$$\begin{cases} zu - (1 - z)s &= x \\ zv - (1 - z)t &= y \end{cases}$$

Thus, we construct equations w.r.t $u \sim s$ and $v \sim t$, then assume we have a camera array $st$ plane, and each integer point corresponds to a single camera. If each cameras have the same interval distance

along $s, t$ directions respectively, the transform of $st$ focal plane is contant. And we define the constant by $\delta$, and we regard $\delta$ as disparity, which reflects how $uv$ changes w.r.t $st$.

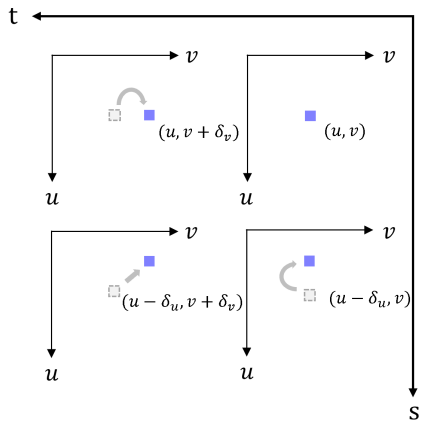



Fig. 2. disparity

In Fig.2, we show four neighbor cameras and describe how to align pixel w.r.t disparity. Generally, for a camera array, the disparities along $s$ and $t$ direction are the same. However, we'd better check it manually. And we check that it is exactly true for dataset in this assignment.

### 3.3 Interpolation

In general, there are two steps to complete interpolation. First, we interpolate the images of neighbor cameras w.r.t. disparity. Second, we interpolate pixel value for each image, respectively.

We implement two ways for interpolation, i.e. bilinear and quadra-linear. Bilinear interpolation only interpolates on $st$ plane i.e. multi-camera interpolation. Quadra-linear interpolation interpolates both $st$ plane and $uv$ plane, that is two bilinear interpolation actually.

To implement bilinear interpolation, we display a virtual camera at $(s, t)$ and $s, t$ can be floating number, which implies that we can not directly get the results from existing $(s, t, u, v)$. Because all cameras in $(s, t, u, v)$ light field are all at integer points.Thus, we apply bilinear interpolation w.r.t $(s, t)$'s neighbor cameras.

In fact, bilinear interpolation calculates the weights to sum up the images of neighbor camera. For a virtual camera, we get four neighbor cameras' views that are shifted by disparity. And we first interpolates along $s$ axis and second interpolates along $t$ axis.

Assume virtural camera is at $(s', t')$, then the righttop camera is at $(s, t)$, where $s = floor(s'), t = floor(t')$. For simplifity, we denote righttop camera by $C(0, 0)$. The other three cameras are $C(0, 1), C(1, 1), C(1, 0)$ in anticlockwise order.

First, interpolate along $s$ axis, we have

$$C_0 = pC(1, 0) + (1 - p)C(0, 0)$$
$$C_1 = pC(1, 1) + (1 - p)C(0, 1)$$

Second, interpolate along $t$ axis, we gave

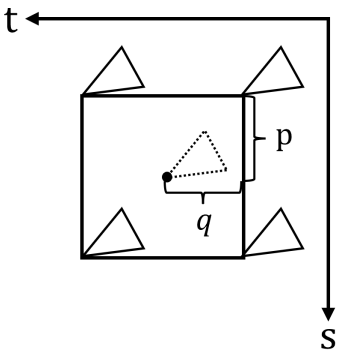



Fig. 3. bilinear interpolation

$$\begin{aligned} C(s', t') &= qC_1 + (1 - q)C_0 \\ &= (1 - p)(1 - q)C(0, 0) + (1 - p)qC(0, 1) \\ &\quad + p(1 - q)C(1, 0) + pqC(1, 1) \end{aligned}$$

Therefore, we complete the bilinear interpolation w.r.t neighbor cameras. Generally, we can also adapt it to calculating the interpolation of pixels on $uv$ plane, which is needed in quadra-linear interpolation method.

Fig. 4. bilinear result

Fig. 5. quadra-linear result

What do you observe if you use biliner interpolation on undersampled light field?

Aliasing artefacts.

Fig. 6. undersampled result by bilinear

### 3.4 Variable focal plane

We achieve the effect of variable focal plane by mapping the depth $z$ of the focal plane to the disparity. Just like what we show in the disparity part, we conclude the equation w.r.t disparity and depth by $\delta = \frac{1-z}{z}$.

What happens when you move your focal plane from far to near?

When the focal plane is far, the scene becomes very blurry. As focal plane gets closer and closer, we can see the background of scene clearly and the objects clearly soon. However, when focal plane gets too close, the scene becomes blurry and has some invisible black areas.

The reason behind this phenomenon is that the area that we can see clearly in the scene depends on disparity. One value of $z$ corresponds to one value of disparity and one focal plane. The closer the focal plane is, the bigger the disparity is. Thus, we can see part of the scene clearly only at an appropriate range of disparity or depth of focal plane.

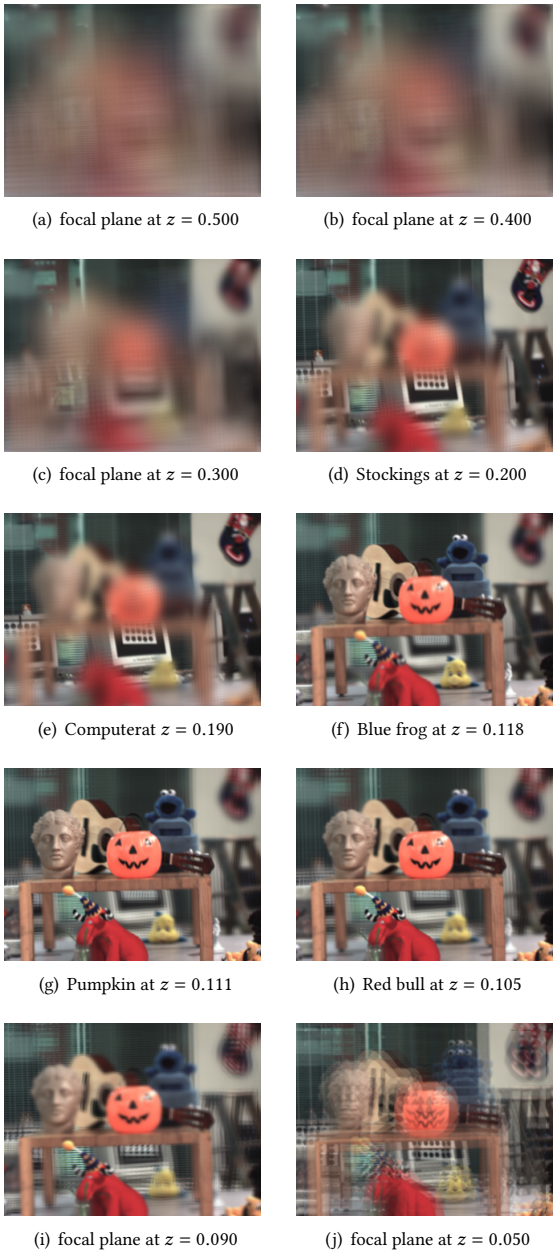
Which focal plane gives yo the optimal results(least aliased reconstruction)?

$z = \frac{1}{9}$

### 3.5 Variable aperture size

We implement a wide aperture filter to show the effect of changing aperture size. In fact, when aperture size = 1, it corresponds to bilinear or quadra-linear interpolation. However, when aperture size > 1, we have to deal with with Gaussian weigh. Let aperture size be $a$ and the weight for a neighbor camera is that

$$w(s,t) = \exp(-\|\delta(s,t)\|_2^2/(2a^2))$$



(a) focal plane at $z = 0.500$ (b) focal plane at $z = 0.400$

(c) focal plane at $z = 0.300$ (d) Stockings at $z = 0.200$

(e) Computerat $z = 0.190$ (f) Blue frog at $z = 0.118$

(g) Pumpkin at $z = 0.111$ (h) Red bull at $z = 0.105$

(i) focal plane at $z = 0.090$ (j) focal plane at $z = 0.050$

Fig. 7. Variable focal plane from far to near

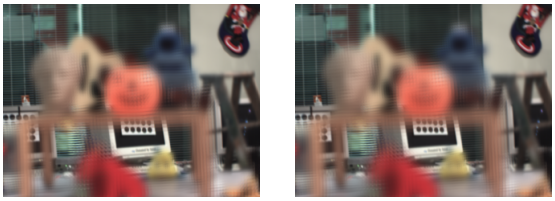
After calculating all weights of neighbor camera, we need to normalize the sum to 1.

$$w'(s,t) = \frac{w(s,t)}{\sum w(s,t)}$$

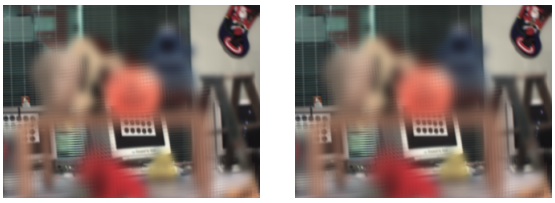
We show the results of stockings at $z = 0.200$ by variable aperture size. And we find that we can see the stockings clearly while seeing other things unclearly with a big aperture.

(a) aperture size = 1 (b) aperture size = 2

(c) aperture size = 3 (d) aperture size = 4



(e) aperture size = 5 (f) aperture size = 6



(g) aperture size = 7 (h) aperture size = 8

Fig. 8. Variable aperture size with focal plane $z = 0.200$

### What happens when you increase the size of the aperture?

With the size of aperture increasing, we implement the focussing effect and we can only see the objects at the specific depth clearly.

## 3.6 Expand field of view

We implement the z-directional motion of the camera w.r.t the depth of virtual camera. When the virtual camera is on the camera array plane, the depth is 0. However, the depth of the virtual camera increases when it goes towards the focal plane or the scene.

Assume that $z >= 0$ when the virtual camera gets closer to the scene. Then we can calculate the projection region onto the image captured by the virtual camera on the camera array plane. Denote the focal plane by $f$.

$$U' = U * (f - z)/z$$
$$V' = V * (f - z)/z$$

Then, we can get the new image by upsampling the original image in the projection region.

(a) depth = 0.00 (b) depth = 0.02

(c) depth = 0.03 (d) depth = 0.04

(e) depth = 0.05 (f) depth = 0.07

Fig. 9. Expand field of view with focal plane $z = 0.111$