CSCI 3280 Introduction to Multimedia Systems

Spring 2020, Assignment 2 – View Synthesis from Light Field

Deadline: Mar. 13, 2020 (11:59pm)

Submission via Blackboard.cuhk.edu.hk

Late submission penalty : 10% deduction per day (max: 30% deduction)

PLAGIARISM penalty : whole course failed

Introduction

As a 3D scene representation method, the light field records both the radiance and direction information of light rays coming from the scene. Generally, a 4D light filed is parameterized by two 2D plane, as illustrated in **Fig. 1**, where any point pair from different planes denotes a light ray. We can use two typical devices, i.e. camera array and light-field camera, to capture the light field of a scene. For example, the camera array shown in **Fig. 2** consists of multiple cameras that are regularly installed on a plane, each of which observes the scene from a unique viewpoint. Due to the rich scene information recorded, light fields enable many exciting applications. In this assignment, you are required to implement a program on **synthesizing novel views from a recorded light filed.**

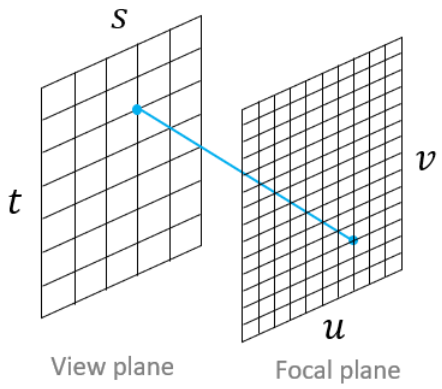

Fig. 1. Two-plane representation of light field

Fig. 2. Camera array

Implementation Guideline

Data description. The provided light field data, consisting of 81 RGB sub-images (each of image resolution: 512×512), is captured by a camera array of 9×9 views. All these cameras

of the same imaging setting, are regularly installed on a plane and share the orientation perpendicular to the plane. The detailed parameters, *baseline=30mm, image size=35x35mm^2, focal length (i.e. distance between viewpoint and image plane)=100mm*, are marked in **Fig. 3**.

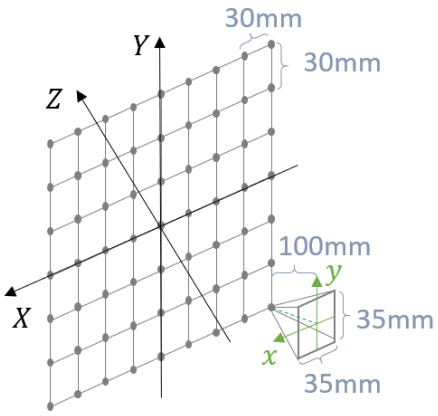

Fig. 3. Camera array parameters

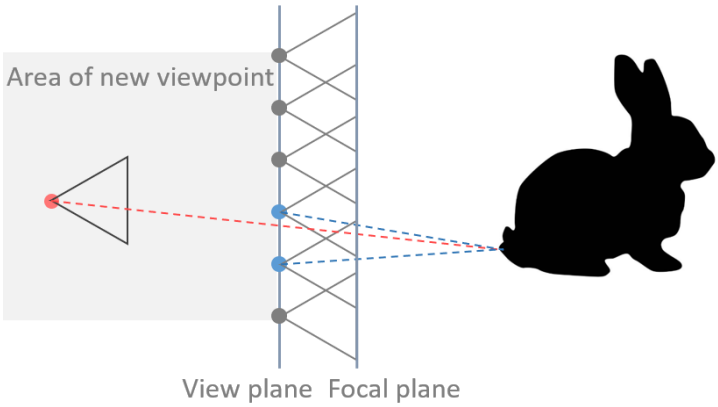

Fig. 4. Target light ray interpolation from neighbour light rays

Implementation steps. A light field can be regarded as a set of light rays with known radiance and direction values. For example, the pixel $p(x_i, y_i)$ on the sub-image of viewpoint $V(s_j, t_j)$, is representing a light ray passing 3D point $P(X_j, Y_j, Z_j)$ with direction of $D(x_i, y_i, -f)$, where P is the position of the viewpoint $V(s_j, t_j)$ in the global 3D coordinate system (as defined in **Fig. 3**), f is the focal length and (x_i, y_i) is the coordinate of p on image plane. So, **to synthesize an image of new viewpoint, we only need to retrieve/resample the radiances (pixel values) from the collection of pixel values (or image array)**. In summary, the algorithm can be implemented by the following steps:

1. **Loading light field data** – Read all the sub-images of different viewpoints in order, which correspond to the grid viewpoints $\{V(s,t) \mid s = 0,1,\dots,8; t = 0,1,\dots,8\}$ on the view plane. Specifically, the provided sub-images of viewpoint $V(s,t)$ are named as "*cam_no.bmp*", where $no = t * 9 + s$. For the corresponding 3D coordinate, we have that $V(0,0)$ locates at $P(-120,120,0)$ and $V(8,8)$ locates at $P(120,-120,0)$.
2. **Building global coordinate system** – Before any computation, we need to build a 3D global coordinate system to describe the relative position of viewpoints and the directions of light rays. Although any 3D cartesian coordinate system is feasible, in this assignment, we require the global coordinate system used in your program to follow the one defined in **Fig. 3** (just for a uniform testing standard).
3. **Defining the target view** – To synthesize a new image, we need to define the parameters of the virtual camera, consisting of intrinsic and external parameters. Here, intrinsic

parameters include focal length f, image size $\mathrm{w} \times h$, image resolution $\mathrm{c} \times r$; external parameters include the viewpoint position (X, Y, Z) and orientation vector (v_x, v_y, v_z) of the virtual camera. To simplify the problem, you are only required to implement the feature of variable viewpoint position, with other parameters fixed as that of the cameras array: $\mathrm{f} = 100\mathrm{mm}, \mathrm{w} \times h = 35mm \times 35mm$, $\mathrm{c} \times r = 512 \times 512$, $(v_x, v_y, v_z) = (0,0,-1)$.

4. **Resampling new light rays** – Once the target view is defined, we can compute the pixel values by retrieving/resampling their corresponding 3D light rays.
   - In **Fig. 4**, the new light ray (red dash line), passes through the two parameterization plane, where we can first find the neighbour light rays (blue dash lines) from known viewpoints (blue nodes). (**Hint**: considering the tiny scalar of baseline w.r.t the distance between scene and camera, the blue dash lines could be regarded as parallel to the red dash line in your implementation.)
   - In **Fig. 5**, for a light ray (e.g. blue dash line) from a known viewpoint, we can bilinearly interpolating its value from the 4-neighborhood pixels on the image of this viewpoint.
   - When all the four neighbour light rays obtained, we finally compute the value of the new light ray from the target view via bilinear interpolation from them.

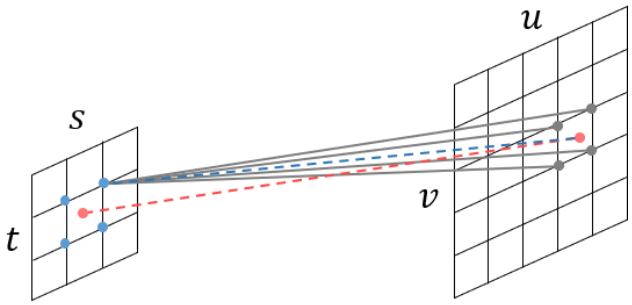

Fig. 5. The known neighbour light rays of a target light ray

5. **Saving resultant image** – When all the pixel values of the new target view are obtained, save this synthesized image into the user specified path.

Basic Requirements (80 point)

1. You are required to implement a special case of the algorithm described above: the target viewpoint locates on the camera array plane, i.e. the viewpoint coordinate is $= (X, Y, 0)$ with $X \in [-120,120], Y \in [-120,120]$. With this assumption, we can easily find the four neighbour known viewpoints of the target viewpoint, of which all the light rays can be interpolated from the light rays of the four known viewpoints. So, there is no need to locate neighbour viewpoints for each individual new light ray.

2. Program must be coded in ANSI C/C++ and uses standard libraries only
3. The compiled program must run in Windows 10 command prompt as a console program and accepts source bitmap (.bmp format) with the following syntax and save generated images to the current directory.

   `C:\> veiwSynthesis <LF_dir> <viewpoint_coord> <focal_length>`

   veiwSynthesis is the executable file of your program, e.g. the command: `veiwSynthesis light_filed_views 200 200 0 100` is to synthesize an image from the viewpoint position $(200, 200, 0)$ and with a focal length 100.

   <LF_dir> is the directory where the light field sub-images are located.

   <viewpoint_coord> and <focal_length> are the 3D coordinate of the target viewpoint and the focal length of the virtual camera respectively. Note that the coordinate and scalar conform to the definition in **Fig. 3**.
4. You are required to submit source code only. We will use Visual Studio 2015 C++ compiler and have your program compiled via Visual Studio command prompt (Tools Command Prompt) with the command line: `C:\> cl viewSynthesis.cpp bmp.cpp` (**Please ensure your source code gets compiled well with it**).

Enhanced Features (20 point)

You are required to implement the full version of the algorithm described above, which means that the target viewpoint can locate at any 3D position $P(X, Y, Z)\ where\ X \in [-120,120], Y \in [-120,120], Z \in [0, \infty)$. Besides, your implementation is also encouraged to support a variable focal length $f \in (0, \infty)$ Note that, when a query light ray falls out of the range of recorded light field, it is assigned with the background black color, i.e. rgb=(0,0,0).

Submission

We expect the following files zipped into a file named by your SID (e.g. s1234567890.zip) and have it uploaded to the Blackboard by due date: **Mar. 13, 2020 (11:59pm)**

- **README.txt** (Tell us anything that we should pay attention to)
- **bmp.h & bmp.cpp** *(No need to change)*
- **viewSynthesis.cpp** (write your code in this file only)