Light Field Reconstruction Using Sparsity in the Continuous Fourier Domain

Code Documentation

Tianwei Huagn

tracyhuangtw@gmail.com

Table of Contents

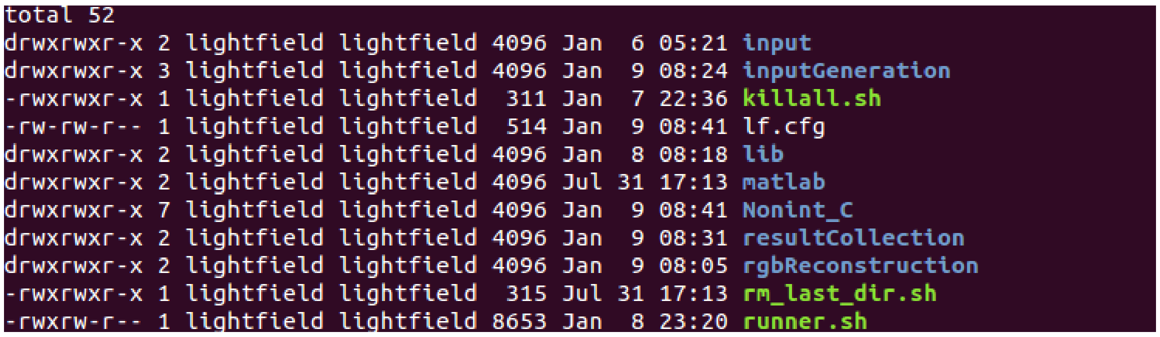
1. Program Overview

This program is used to do 4D light field reconstruction by method of sparse fast fourier transform (SFFT). It can be generally divided into two parts: voting and gradient descent. In the first part, namely voting, we use powers of sample positions to do voting on each single slice and by learning a threshold, we can get a set of estimated peaks. In the second part, we will improve our estimation by gradient descent, which involves the use of pseudo inverse matrix and shadow buckets. For more details about the algorithm, please read the paper ***Light Field Reconstruction Using Sparsity in the Continuous Fourier Domain*** on our website.

2. Running Procedure

2.1 Download Procedure

After downloading the program, unzip it and it will look like something below:



```
total 52
drwxrwxr-x 2 lightfield lightfield 4096 Jan  6 05:21 input
drwxrwxr-x 3 lightfield lightfield 4096 Jan  9 08:24 inputGeneration
-rwxrwxr-x 1 lightfield lightfield  311 Jan  7 22:36 killall.sh
-rw-rw-r-- 1 lightfield lightfield  514 Jan  9 08:41 lf.cfg
drwxrwxr-x 2 lightfield lightfield 4096 Jan  8 08:18 lib
drwxrwxr-x 2 lightfield lightfield 4096 Jul 31 17:13 matlab
drwxrwxr-x 7 lightfield lightfield 4096 Jan  9 08:41 Nonint_C
drwxrwxr-x 2 lightfield lightfield 4096 Jan  9 08:31 resultCollection
drwxrwxr-x 2 lightfield lightfield 4096 Jan  9 08:05 rgbReconstruction
-rwxrwxr-x 1 lightfield lightfield  315 Jul 31 17:13 rm_last_dir.sh
-rwxrw-r-- 1 lightfield lightfield 8653 Jan  8 23:20 runner.sh
```

2.2 Overall Structure

- The "input" and "matlab" folder contains some matlab codes which is not part of the main program, but you can use them to manipulate and check on results.
- The “lib” folder contains some shared link libraries used by the main program.
- The “inputGeneration” folder contains the program responsible for generating the input. The program takes colored images to do FFT and store the results in .dat files as input to the main program.
- The “Nonint_C” folder contains the main program and related configuration files.

- The “resultCollection” folder contains the program responsible for collecting results from one or multiple machines, and transfer results back to image, specifically one channel of the output image (r, g, b).
- Note that the main program is running on one channel (r, g, b) for one pass, so we actually run the main program 3 rounds to handle the colored image. Thus we have a program in “rgbRecunstruction” to combine the results of three channels and generate the final colored result.
- The "lf.cfg" file is the main configuration file for the whole program.
- The "runner.sh" script is used to run the whole program; it will go through each executable in the order of inputGeneration -> Main Program -> Result Collection -> RGB Reconstruction.
- The "rm_last_dir.sh" script is used to remove folders on remote machines.
- The "killall.sh" script is used to terminate execution on remote machines, namely kill processes on remote machines.

2.3 Compiling Procedure

2.3.1 Install Libraries

We have three programs that need to be compiled. Before compiling them you need to make sure you have the following libraries installed on your machine. (Note that all these libraries are only required on the machine that you use to compile the program. Say if you have machine 0-3 to use. Then you can install below libraries on machine 0 and compile all three programs on machine 0. For machine 1-3, if you are only using them to parallel run the program, then you don’t need to install libraries on them).

Library	**Install Instruction**
openCV	http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html
FFTW	http://www.fftw.org/fftw2_doc/fftw_6.html
Libconfig	http://packages.ubuntu.com/trusty/libconfig-dev

The Libconfig library is a bit annoying here, as it has several different versions for different versions of Ubuntu releases.

For the Libconfig library, the most important package is the “libconfig++-dev” (again for Ubuntu users). Another requirement is the libconfig++9/8 (libconfig++8 is for Ubuntu versions lower than Trusty and libconfig++9 is for Ubuntu versions higher than Trusty).

You can check the installation of libconfig related libraries by running the following command:

```
dpkg --get-selections | grep -v deinstall | grep libconfig
```

On the machine I used for testing it shows the following result:

```
libconfig++-dev:amd64          install
libconfig++8-dev               install
libconfig++9:amd64             install
libconfig-dev:amd64            install
libconfig-doc                  install
libconfig-file-perl            install
libconfig9:amd64               install
```

If you have trouble compiling the program in the “Nonint_C” folder (like header file not found), then you need to check the installation and install packages that are missing.

2.3.2 Generate ssh public key

In some cases, you may wish to run the program on several remote machines to save time. The shell script use ssh and scp to connect and send files to remote machines. In order to avoid the case where you need to input password a million times, it is highly suggested that to use ssh public key.

Please read instructions in this link: https://macnugget.org/projects/publickeys/

A more specific instruction can be found here: http://www.linuxproblem.org/art_9.html

2.3.3 Compile the Code

For compilation, you need to go to three different folders and "make", a list of commands is listed below as reference:

```
cd inputGeneration | make
cd ../Nonint_C | make
cd ../resultCollection | make
cd ../rgbReconstruction | make
```

2.4 Example Run

2.4.1 Parameter Setting

In most cases, as we have a very large input dataset, we have to run the program paralleled by multiple processes. Thus we split the input dataset into small pieces, and let one process take charge of one split. If multiple machines are used, these processes are on different machines and they will be on same machine if only one machine is used. The number of process on one machine is called the "load" of that machine. It is recommended that the load of one machine should be smaller than the number of CPU/cores that machine has.

If multiple machines are used, we need to setup SSH connection so that we don't need to input passwords forever. Please refer to section 2.3.2 for instruction on this part.

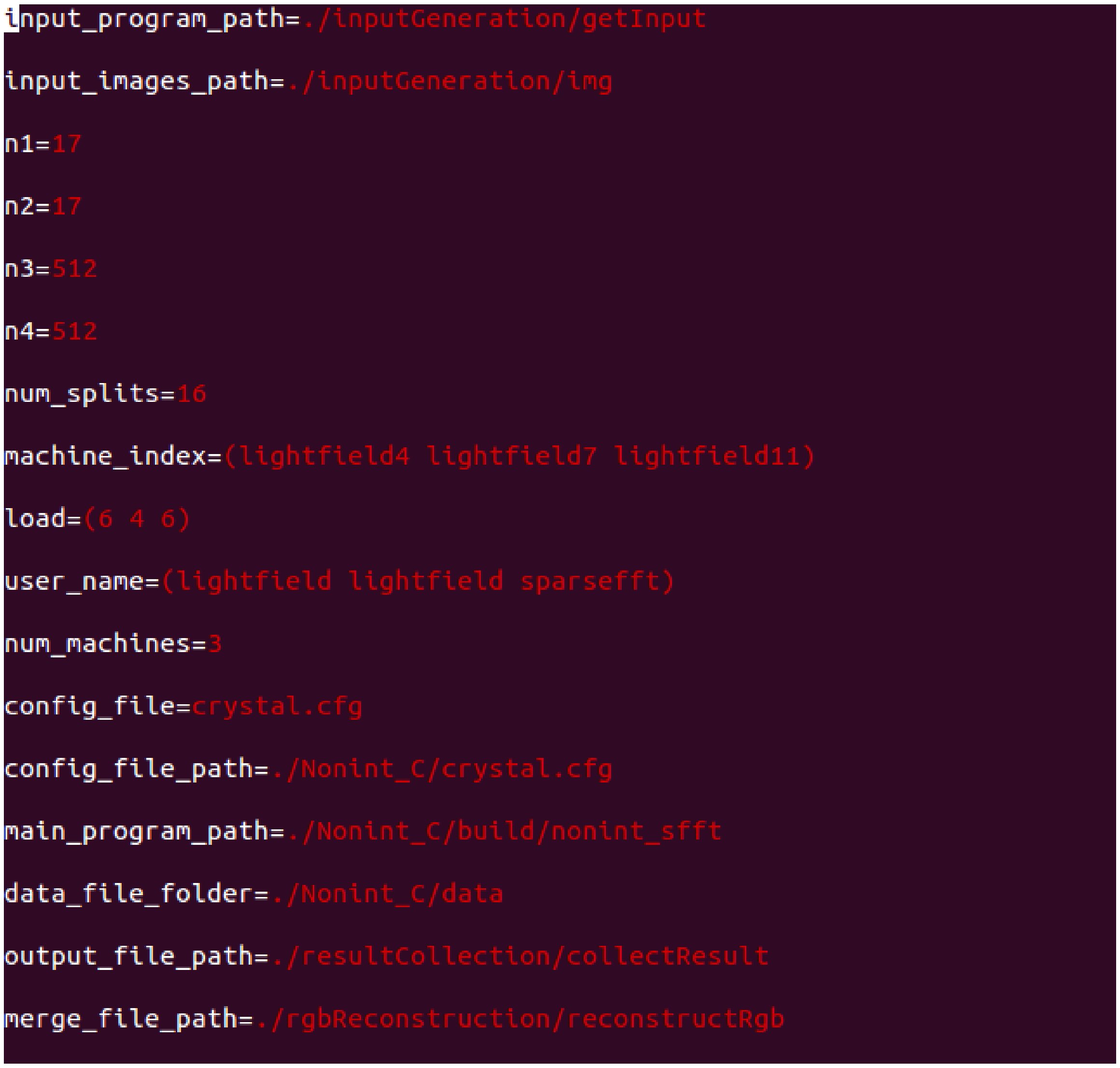
Before running the program, we need to setup the configuration file "lf.cfg" first. In most cases, we only need to set **num_splits, machine_index, load, username, num_machines**.

parameter name	usage	example
input_program_pat	Path to the input generation program	**./inputGeneration/getI**
input_images_path	Path to the raw image input	**./inputGeneration/img**
n1, n2, n3, n4	n1/n2 are the x/y dimension for the camera plane. n3/n4 are the x/y dimension for the scene plane.	**n1=17** **n2=17** **n3=512** **n4=512**
num_splits	The total number of splits we want to use. It should be equal to the sum of all elements of "load" array.	**16**
machine_index	Array containing the IPs of all machines used. that are going to be used.	**(ip1 ip2 ip3)**
Load	Array containing how many processes will be on each single machine. The indexing of this array should be in accordance to machine_index.	**(4 6 4)**
user_name	Array containing the user names on all machines that are going to be used. The indexing of this array should be in accordance to machine_index	**(username1 username2 username3)**
num_machines	Number of machines that are going to be used. num_machines= length(machine_index) = length(load) = length(user_name)	**3**
config_file	**Name of the configuration file that is going to be used by main program**	**crystal.cfg**

2.4.2 Sample Run with Multiple Machines

This sample run uses three Ubuntu 12.04 machines.

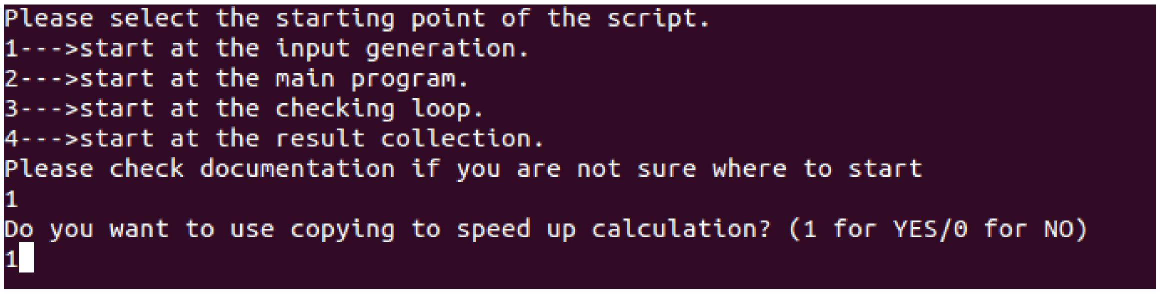
The parameters in the lf.cfg file will be the same as the example values in the chart above. Like this:



```
input_program_path=./inputGeneration/getInput

input_images_path=./inputGeneration/img

n1=17

n2=17

n3=512

n4=512

num_splits=16

machine_index=(lightfield4 lightfield7 lightfield11)

load=(6 4 6)

user_name=(lightfield lightfield sparsefft)

num_machines=3

config_file=crystal.cfg

config_file_path=./Nonint_C/crystal.cfg

main_program_path=./Nonint_C/build/nonint_sfft

data_file_folder=./Nonint_C/data

output_file_path=./resultCollection/collectResult

merge_file_path=./rgbReconstruction/reconstructRgb
```

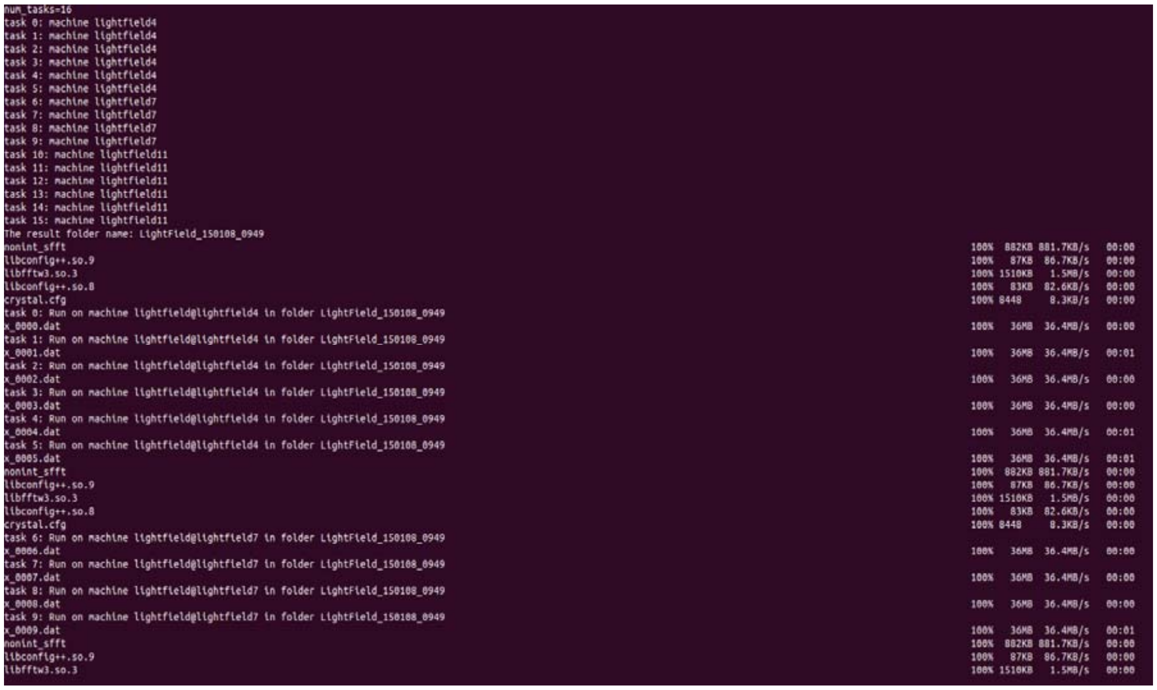
After setting the configuration file, we will start the program by running:

./runner.sh ./lf.cfg



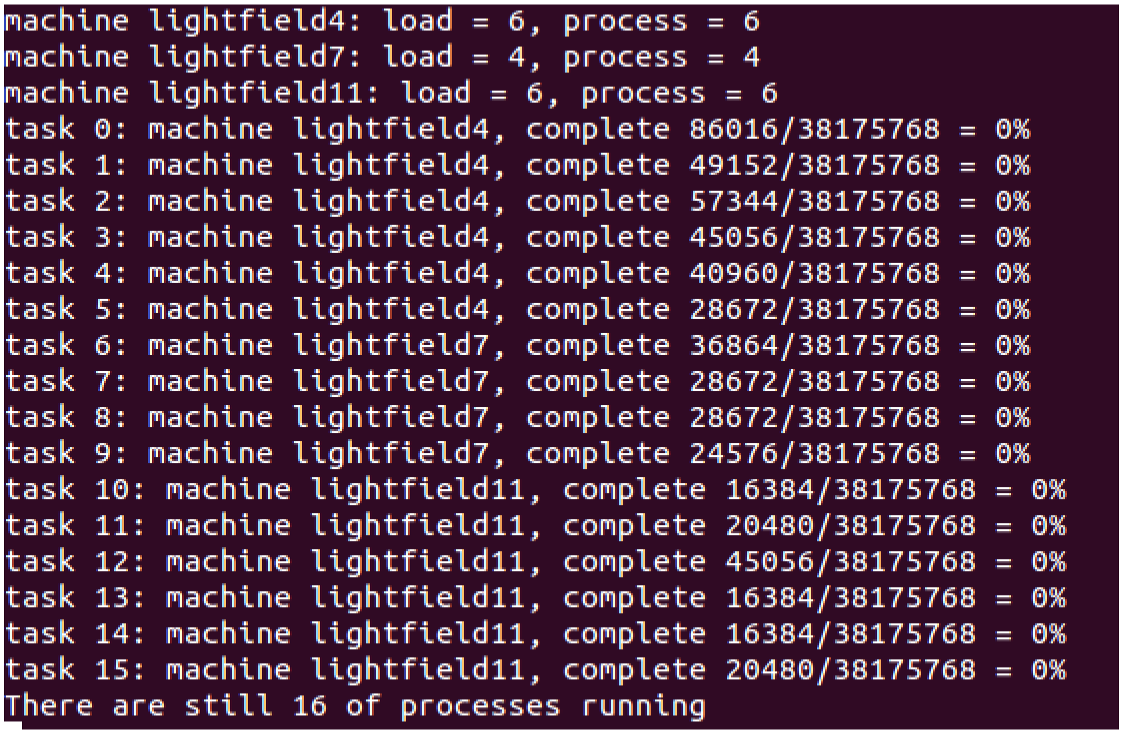
```
Please select the starting point of the script.
1--->start at the input generation.
2--->start at the main program.
3--->start at the checking loop.
4--->start at the result collection.
Please check documentation if you are not sure where to start
1
Do you want to use copying to speed up calculation? (1 for YES/0 for NO)
1
```

First you need to select the starting point of the program. In most cases you want to start at the beginning, which is input generation. And then you will be prompted to select to use copying to speed up execution or not. Thus just type 1 + <enter> twice.



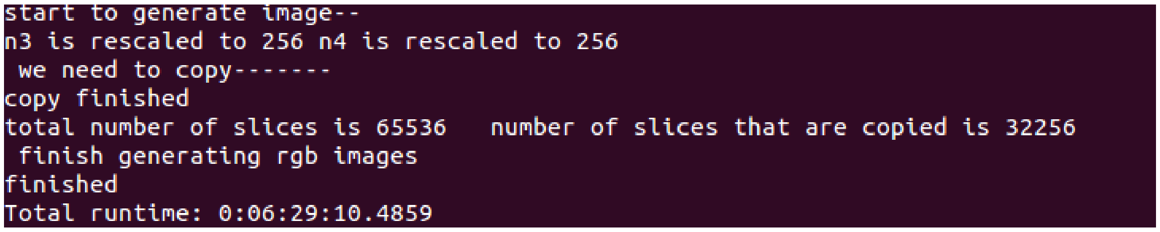
```
num_tasks=16
task 0: machine lightfield4
task 1: machine lightfield4
task 2: machine lightfield4
task 3: machine lightfield4
task 4: machine lightfield4
task 5: machine lightfield4
task 6: machine lightfield7
task 7: machine lightfield7
task 8: machine lightfield7
task 9: machine lightfield7
task 10: machine lightfield11
task 11: machine lightfield11
task 12: machine lightfield11
task 13: machine lightfield11
task 14: machine lightfield11
task 15: machine lightfield11
The result folder name: LightField_150108_0949
nonint_sfft                                                                                      100%  882KB 881.7KB/s   00:00
libconfig++.so.9                                                                                 100%   87KB  86.7KB/s   00:00
libfftw3.so.3                                                                                    100% 1510KB   1.5MB/s   00:00
libconfig++.so.8                                                                                 100%   83KB  82.6KB/s   00:00
crystal.cfg                                                                                      100% 8448     8.3KB/s   00:00
task 0: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0000.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 1: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0001.dat                                                                                       100%   36MB  36.4MB/s   00:01
task 2: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0002.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 3: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0003.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 4: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0004.dat                                                                                       100%   36MB  36.4MB/s   00:01
task 5: Run on machine lightfield@lightfield4 in folder LightField_150108_0949
x_0005.dat                                                                                       100%   36MB  36.4MB/s   00:01
nonint_sfft                                                                                      100%  882KB 881.7KB/s   00:00
libconfig++.so.9                                                                                 100%   87KB  86.7KB/s   00:00
libfftw3.so.3                                                                                    100% 1510KB   1.5MB/s   00:00
libconfig++.so.8                                                                                 100%   83KB  82.6KB/s   00:00
crystal.cfg                                                                                      100% 8448     8.3KB/s   00:00
task 6: Run on machine lightfield@lightfield7 in folder LightField_150108_0949
x_0006.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 7: Run on machine lightfield@lightfield7 in folder LightField_150108_0949
x_0007.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 8: Run on machine lightfield@lightfield7 in folder LightField_150108_0949
x_0008.dat                                                                                       100%   36MB  36.4MB/s   00:00
task 9: Run on machine lightfield@lightfield7 in folder LightField_150108_0949
x_0009.dat                                                                                       100%   36MB  36.4MB/s   00:01
nonint_sfft                                                                                      100%  882KB 881.7KB/s   00:00
libconfig++.so.9                                                                                 100%   87KB  86.7KB/s   00:00
libfftw3.so.3                                                                                    100% 1510KB   1.5MB/s   00:00
```

Then the script will SCP all needed files (main program, input file, link libraries) to remote machines and send commands to have the program running on them. After sending files, the script will go into the phase of "checking loop", namely checking the running progress on remote machines, the running information will be displayed in the format below:



```
machine lightfield4: load = 6, process = 6
machine lightfield7: load = 4, process = 4
machine lightfield11: load = 6, process = 6
task 0: machine lightfield4, complete 86016/38175768 = 0%
task 1: machine lightfield4, complete 49152/38175768 = 0%
task 2: machine lightfield4, complete 57344/38175768 = 0%
task 3: machine lightfield4, complete 45056/38175768 = 0%
task 4: machine lightfield4, complete 40960/38175768 = 0%
task 5: machine lightfield4, complete 28672/38175768 = 0%
task 6: machine lightfield7, complete 36864/38175768 = 0%
task 7: machine lightfield7, complete 28672/38175768 = 0%
task 8: machine lightfield7, complete 28672/38175768 = 0%
task 9: machine lightfield7, complete 24576/38175768 = 0%
task 10: machine lightfield11, complete 16384/38175768 = 0%
task 11: machine lightfield11, complete 20480/38175768 = 0%
task 12: machine lightfield11, complete 45056/38175768 = 0%
task 13: machine lightfield11, complete 16384/38175768 = 0%
task 14: machine lightfield11, complete 16384/38175768 = 0%
task 15: machine lightfield11, complete 20480/38175768 = 0%
There are still 16 of processes running
```

The script will check the progress every 5 minutes. If it notices that all processes finish running, it will ask the remote machines to send back result data files. Note that we are operating on the colored image which has 3 channels (r, g, b). So the above process (send input to remote machines→running program→collect result) will repeat 3 times. After getting all the results, the script will use the **./rgbReconstruction/rgbReconstruct** program to reconstruct the colored image.



```
start to generate image--
n3 is rescaled to 256 n4 is rescaled to 256
 we need to copy-------
copy finished
total number of slices is 65536   number of slices that are copied is 32256
 finish generating rgb images
finished
Total runtime: 0:06:29:10.4859
```

This is the end of the script, and all the output files (with name like **result_<x index>_<y index>.png**) will be stored under the current directory.

2.4.3 Sample Run with One Machine

Running with one machine is very similar to running with multiple machines. The only difference is that the **machine_index, user_name, load** parameters in the **lf.cfg** file will be a single value instead of an array. The **num_machine** should be set to 1, and the **num_splits** should equal to the value of **load** parameter.

Another thing needs attention is that even if only one machine is used, we still need to be able to SSH localhost without password. The setup procedure is similar to the multiple machine scenario.

3. Additionals

3.1 ./Nonint_C/crystal.cfg

This is the configuration file for the main sfft program. Most of the parameters here are relatively stable. If you want to run a new data set, you may wish to change following parameters:

Parameter Name	Usage
n_x	**length of x dimension of scene plane**
n_y	**length of y dimension of scene plane**
n_v	**length of x dimension of camera plane**
n_h	**length of y dimension of camera plane**
num_projection_lines	**number of lines that we used in step one**
projection_matrices	**parameters related to each single projection line**
sample_pos	**array containing sample positions of all sample we are using**
num_samples	**number of samples that are used**

3.2 Parallelization

The reason why we want to use parallelization is simple the long running time (usually takes more than 6 hours for a reasonable dataset, like 17 * 17 colored images with 512 * 512 pixels for each one).

The most important step for parallelization is the setup of SSH login without password. As the program uses SSH and SCP to communicate with remote machines (send/fetch files, send commands, check running progress), it needs to setup large number of SSH connections in total. If we are forced to type in the password every time it would be a disaster. The SSH public key feature can help us achieve login without password. For instructions on how to do this, please refer to section 2.3.2.

Another preparation of parallelization is to modify the lf.cfg file. Different from using multiple machines, we will only have entry in **machine_index, user_name, load** (instead of an array). The value of load parameter will also be the same as the num_splits.